Theoretical Computer Science 412 (2011) 1958–1978

Contents lists available at ScienceDirect

Theoretical Computer Science

journal homepage: www.elsevier.com/locate/tcs

A linear algorithm for MLL proof net correctness and sequentialization

Stefano Guerrini [1]

Dipartimento di Informatica, Sapienza Università di Roma, Via Salaria, 113, 00198 Roma, Italy

ARTICLE INFO

Keywords:
Multiplicative linear logic
Proof nets

ABSTRACT

The paper presents in full detail the first linear algorithm given in the literature (Guerrini (1999) [6]) implementing proof structure correctness for multiplicative linear logic without units. The algorithm is essentially a reformulation of the Danos contractibility criterion in terms of a sort of unification. As for term unification, a direct implementation of the unification criterion leads to a quasi-linear algorithm. Linearity is obtained after observing that the disjoint-set union-find at the core of the unification criterion is a special case of union-find with a real linear time solution.

© 2010 Elsevier B.V. All rights reserved.

1. Introduction

This paper is the long and complete version of [6], in particular, it contains a detailed description of the algorithm and the proofs of the statements given there.

The first linear time algorithm for verifying the correctness of proof nets for multiplicative linear logic without units was presented in [6]. Starting from the result in [6], another linear algorithm has been given by Murawski and Ong [13,14]. The above algorithms are linear when executed on sequential RAMs. If one takes Turing machines instead, a recent and surprising result by Jacobé de Naurois and Mogbil has shown that correctness of multiplicative proof structures is NL-complete [8]. (We recall that NL is the class of the decision problems which can be solved by a non-deterministic log-space Turing machine.)

Since the goal of the paper is to give an efficient (and practical) sequential implementation of proof net correctness, the only computational model taken into account is the sequential RAM. Indeed, while there is no direct connection between the polynomial degree of a polytime problem on RAMs and its degree on Turing machines (a problem solvable with a linear time algorithm by a RAM may be P-complete on Turing machines), finding the exact computational complexity of a polytime problem on (non-)deterministic Turing machines may be of great interest when looking for efficient parallel implementations. In particular, it may help in finding the exact position of the problem inside the Boolean circuit hierarchies NC or AC (for a definition of these classes see [17]). For instance, in the case of proof nets, the result by Jacobé de Naurois and Mogbil gives the Boolean circuit characterization of the problem also, since $AC^0 \subset NL \subseteq AC^1$.

1.1. Proof nets

We shall consider only multiplicative linear logic without constants. In the case with constants, proof net correctness is NP-hard [9].

A multiplicative proof net is a graph representation of a multiplicative linear logic derivation [5]. The proof net N corresponding to the derivation Π is a (directed) hypergraph with a link (i.e. a hyperedge) for each rule and a vertex for each formula occurrence s.t. every link of N connects the active formulas of the corresponding rule of Π. For instance, let Π be a cut-free derivation using atomic axioms only and ending with the sequent $\vdash A$. The corresponding proof net N is

E-mail address: stefano.guerrini@univ-paris13.fr.

[1] Current address: Laboratoire d'Informatique de Paris-Nord (LIPN, UMR CNRS 7030), Institut Galilée, Université Paris 13 Nord 99, avenue Jean-Baptiste Clément 93430 Villetaneuse, France.

0304-3975/$ – see front matter © 2010 Elsevier B.V. All rights reserved.
doi:10.1016/j.tcs.2010.12.021

formed of a part isomorphic to the syntax tree of the formula A, plus a set of connections between pairs of occurrences of dual atoms, i.e. between pair of leaves of the syntax tree; in particular, each node of the syntax tree corresponds to a $\parr$ or $\otimes$-link, while each pairing connection corresponds to an axiom link.

The derivation Π establishes an ordering, say a *sequentialization*, among the links of the corresponding proof net N. In general, the sequentialization of a proof net is not unique. For instance, let us assume that Π ends with $\vdash A\parr B, C\parr D$ and that the last two rules of Π introduce the principal connectives of $A\parr B$ and $C\parr D$; then the proof net N corresponding to Π does not depend on the actual order of that pair of rules.

1.2. Proof structures and correctness criteria

A proof structure is a hypergraph built in accord with the syntax of proof nets, but without following any sequentialization order. A *correctness criterion* is a test that, given a proof structure N, answers *yes* when N is a proof net and *no* when there is no sequentialization of N. The *Danos–Regnier switching condition* is the most known correctness criterion: let a *switch* of N be the graph obtained by disconnecting one of the premises of each $\parr$-link; then N is a proof net iff every switch of N is a tree [3].

If n is the number of $\parr$-links in N, the Danos–Regnier criterion checks 2^n graphs. Moreover, there is good evidence that correctness cannot be inferred by the inspection of a fixed subset of the switches of N. For instance, for any n, we can construct a proof structure s.t. only one (or two, or three, etc.) of its switches is not a tree.

The situation is different for the multiplicative proof nets of non-commutative (and cyclic) linear logic. In that case, the Danos–Regnier criterion does not suffice for correctness; a proof net of commutative linear logic with only one conclusion is a non-commutative proof net iff it is a planar graph. Because of this additional requisite, verifying the correctness of a non-commutative proof structure requires the inspection of two switches only [15]. Unfortunately, planarity does not play any role in the commutative case; therefore, in order to obtain a linear algorithm, we must resort to something else.

Danos *contractibility* was the first step towards an efficient correctness criterion. In his thesis [2], Danos gave a set of shrinking rules for proof structures, characterizing proof nets as the only proof structures that contract to a point. Each shrinking rule removes a link (which corresponds to a constant time operation), then the shrinking of a proof net with n links requires n shrinking steps; but, since at each step we may need a linear search on the links of the net in order to find the next link to shrink, Danos contractibility can be implemented in quadratic time.

The idea of Danos was improved and extended by showing that it could be presented as a *parsing* algorithm [9,7].

1.3. The first linear time algorithm

The first algorithm implementing a correctness criterion for multiplicative proof structures in linear time was that presented in [6]. This solution, which will be the main topic of this paper as well, is essentially an efficient implementation of parsing as a sort of *unification*.

Like term unification, the unification criterion can be formulated as a disjoint-set union-find problem. The use of any union-find α-algorithm leads to a quasi-linear implementation of the criterion (that is, linear up to a factor α that is a sort of inverse of the Ackermann function). Although that kind of algorithm is to all extents and purposes linear (there is no feasible experiment showing their non-linearity) and behaves very well in practice (their constants are very small), a more detailed analysis of the union-find required by the unification criterion shows that this is a special case with a real linear time solution. Therefore, getting rid of the α factor, we can conclude that checking the correctness of a multiplicative proof net is linear.

We stress that all the efficient algorithms derived from contractibility and parsing verify correctness by constructing a sequentialization. Therefore, we have the particularly surprising result that forgetting the sequentialization of a proof net does not imply the loss of any information, not even in terms of the computational resources required to recover a sequentialization. The reconstruction of a sequentialization can be done in linear time, that is in the minimal time required for reading the whole proof net.

1.4. Another linear time solution

After the publication of the first linear time algorithm, another linear time algorithm for correctness of multiplicative proof structures was given by Murawski and Ong [13,14]. Their solution implements a correctness criterion for Lamarche's essential nets [10,11]. Essential nets are a polarized version of proof nets for intuitionist multiplicative linear logic. Polarization induces an orientation on the edges of proof nets that allows one to view them as DAG's (directed and acyclic graphs), for which one can give a direct formulation of the correctness criterion. But, since any proof net can be transformed into an essential net in linear time, Murawski and Ong's algorithm gives a linear time solution for correctness of proof nets too.

The solution given in [6], which we shall see in the rest of the paper as well, rests on efficient solutions for the union-find disjoint set problem. As already stated, the general solution for union-find leads to quasi-linear or α-algorithms, but in some special cases – and that used by the algorithm that we shall analyze in the following is one of these cases – we can eliminate the α-factor to obtain a true linear solution. Remarkably, Murawski and Ong's solution verifies (computes) the so-called

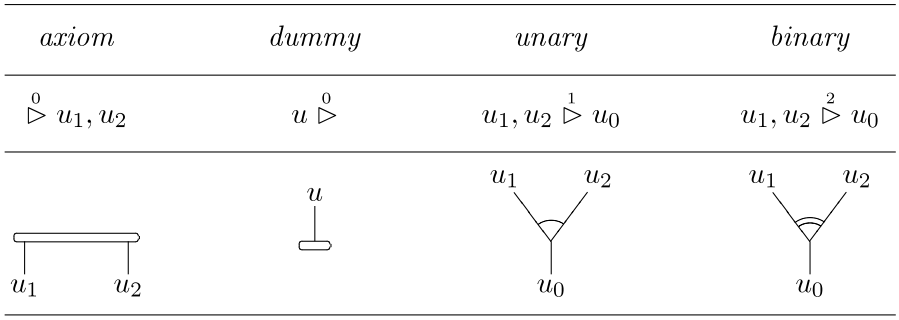


Fig. 1. Abstract proof structure links.

dominator trees associated with an essential net, a problem whose efficient (linear) implementation requires a special case of union-find.

1.5. Structure of the paper

In Section 2, we shall define the hypergraph structures that underlie proof structures, the so-called abstract proof structures.

In Section 3, we shall define proof nets, the Danos–Regnier correctness criterion, and present the contractibility and the parsing approaches to proof structure correctness.

In Section 4, we shall see the sequentialization theorem, namely that a direct consequence of the equivalence between the Danos–Regnier and parsing criterion is that every Danos–Regnier correct proof structure can be sequentialized into a proof of multiplicative linear logic without units.

In Section 5, we shall discuss the computational complexity of the correctness criteria presented above.

In Section 6, we shall reformulate the parsing criterion in terms of a sort of unification.

In Section 7, we shall present an efficient implementation of unification that we shall name sequential unification. Sequential unification exploits the fact that the rules of the algorithm for the unification criterion can be implemented by following a particular order determined by the topology of the structure.

In Section 8, we shall analyze the cost of sequential unification. In particular, we shall see that using a standard α-algorithm implementation of union-find we get a quasi-linear cost. However, by noticing that the actual union-find required by sequential unification is a special case for which there is a true linear solution, we show that sequential unification is indeed linear.

2. Abstract proof structures

The correctness of a proof structure depends only on its topology. The actual value of the formulas plays a role in restricting the valid proof structure that we may consider, in particular, in controlling how proofs can be combined through cuts.

In this section we shall define the hypergraph structures associated with every proof structure, the so-called abstract proof structures.

Definition 1 (*Link*)**.** A *link* is a pair $\beta \triangleright \alpha$ in which β and α are two disjoint sets of vertices that are not simultaneously empty, namely $\alpha \cap \beta = \emptyset$ and $\alpha \cup \beta \neq \emptyset$. A vertex u is a *premise* of the link when $u \in \beta$, or a *conclusion* when $u \in \alpha$.

Each premise or conclusion of a link has a distinct name (e.g., in a link with two premises, the names might be left and right; in a link with n conclusions, we might distinguish the 1st, the 2nd, ..., the nth conclusion). However, since we shall not consider cut-elimination, names will not play any relevant role in the following.

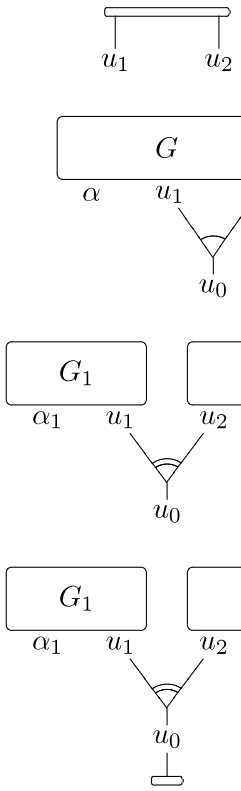
A link without premises, *source* link, or without conclusions, *target* link, is a *root* link and is denoted by $\overset{0}{\triangleright}$.

In an abstract proof structure, or aps for short (see Definition 2), there are two kinds of root links: *axiom* links (source links with two conclusions) and *dummy* links (target links with one premise). The other links that can appear in an aps are of two kinds: *unary links* and *binary links*, denoted by $\overset{1}{\triangleright}$ and $\overset{2}{\triangleright}$, respectively. Both these kinds of links have two premises and one conclusion. A summary of aps links with the corresponding pictorial representation is given in Fig. 1.

Definition 2 (*Aps*)**.** An *abstract proof structure* (aps) G over the vertices $\mathcal{V}(G)$ is a set of links $G = \beta_1 \triangleright \alpha_1; \beta_2 \triangleright \alpha_2; \ldots; \beta_k \triangleright \alpha_k$ s.t.

(1) the shape of every link in G is one of the four given in Fig. 1;
(2) every vertex in $\mathcal{V}(G)$ is a conclusion of exactly one link;
(3) every vertex in $\mathcal{V}(G)$ is a premise of at most one link and at least one vertex in $\mathcal{V}(G)$ is not premise of any link of G;
(4) the premise of every dummy link is the conclusion of a binary link.

We shall say that a vertex that is not a premise of any link of G is a *conclusion* of G, and we shall write $G \vdash \alpha$ to denote that α is the (non-empty) set of the conclusions of G.



$$\frac{}{\overset{0}{\triangleright}\, u_1, u_2 \in \mathsf{Seq}}\ \mathsf{axiom}$$

$$\frac{G \in \mathsf{Seq} \qquad G \vdash \alpha, u_1, u_2}{G\,;\, u_1, u_2 \overset{1}{\triangleright} u_0 \in \mathsf{Seq}}\ \mathsf{par}$$

$$\frac{G_1, G_2 \in \mathsf{Seq} \quad G_1 \vdash \alpha_1, u_1 \quad G_2 \vdash \alpha_2, u_2}{G_1\,;\, G_2\,;\, u_1, u_2 \overset{2}{\triangleright} u_0 \in \mathsf{Seq}}\ \mathsf{tensor}$$

$$\frac{G_1, G_2 \in \mathsf{Seq} \quad G_1 \vdash \alpha_1, u_1 \quad G_2 \vdash \alpha_2, u_2}{G_1\,;\, G_2\,;\, u_1, u_2 \overset{2}{\triangleright} u_0\,;\, u_0 \overset{0}{\triangleright} \in \mathsf{Seq}}\ \mathsf{cut}$$

Fig. 2. Sequentialization.

Let us say that a *premise* of a set of links G is a vertex that is not a conclusion of any link of G. By definition, the set of the premises of an aps is empty, while the set of its conclusions cannot be empty. In the following it will be useful to consider substructures, namely sets of links contained in some aps, with a possibly non-empty set of premises and a possibly empty set of conclusions.

Definition 3 (*Asl*)**.** A non-empty set of links G' is an *abstract structure of links* (asl) if $G' \subseteq G$, for some aps G. A *premise* of G' is a vertex that is not the conclusion of any link in G'. We shall write $G' : \alpha_1 \vdash \alpha_0$ to denote that α_1 is the set of the premises of G' and α_0 is the set of its conclusions.

Given an aps G, every asl $G' \subseteq G$ is a sub-asl (or an abstract substructure of links) of G. When G' is an aps, G' is a sub-aps (or an abstract proof substructure) of G also.

2.1. Concrete proof structures

Definition 2 does not postulate anything about the intended meaning of G. The concrete examples corresponding to that abstraction are the proof structures of the multiplicative constant-free fragment of linear logic. Therefore, G is a *concrete proof structure* (cps) when $\mathcal{V}(G) = F \cup C$ is formed of a set F of occurrences of multiplicative linear logic formulas and a set C of occurrences of the reserved symbol cut, s.t. every link of G matches one of the following patterns:

$$\overset{0}{\triangleright}\, A, A^{\perp} \qquad A, B \overset{1}{\triangleright} A \parr B \qquad \mathsf{cut} \overset{0}{\triangleright} \qquad A, A^{\perp} \overset{2}{\triangleright} \mathsf{cut} \qquad A, B \overset{2}{\triangleright} A \otimes B$$

with $A, B \in F$.

Remark 4. A cps differs from a usual proof structure because of the representation of cuts. In a cps, tensors and cuts collapse into the same type of link (with the minor technicality of a dummy below every cut). In fact, for checking correctness, there is no difference between a tensor and a cut (compare the tensor and cut rules in Fig. 2).

3. Proof nets

3.1. Sequentializable proof structures

The class Seq of the sequentializable aps is the smallest set of aps inductively defined by the rules in Fig. 2, with the proviso that in each of those rules, u_0 is a fresh vertex, and that, in the rules tensor and cut, $\mathcal{V}(G_1) \cap \mathcal{V}(G_2) = \emptyset$. (These side conditions are implicit if one assumes that Seq contains well-formed aps only.)

The rules in Fig. 2 are the graph theoretic abstraction of the derivation rules of multiplicative linear logic. In particular, any sequentialization of a cps G (i.e. any derivation of $G \in \mathsf{Seq}$) corresponds to a linear logic derivation.

3.2. Danos–Regnier correctness criterion

We shall represent graphs by means of their incidence relation. An (undirected) graph is a pair $(V, \frown)$, where $\frown$ is a symmetric and anti-reflexive binary relation over the set of vertices V. We shall use $\smile$ to denote the negation of $\frown$. Therefore, $u \frown v$ will mean that there is an edge between u and v, while $u \smile v$ will mean that there is no edge between u and v.

Definition 5 (*Switch*)**.** A *switch* S of the asl G is a graph $(\mathcal{V}(G), \frown)$ whose edges are defined by the following rules:

(1) each axiom link of G is replaced by an edge between its conclusions, that is $u_1 \frown u_2$ for every $\overset{0}{\triangleright} u_1, u_2 \in G$;
(2) each unary link is replaced by an edge (only one) between its conclusion and one of its premises, that is $(u_1 \frown u_0) \wedge (u_2 \smile u_0)$ or $(u_2 \frown u_0) \wedge (u_1 \smile u_0)$ for every $u_1, u_2 \overset{1}{\triangleright} u_0$;
(3) each binary link is replaced by a pair of edges between its conclusion and its premises, that is $(u_1 \frown u_0) \wedge (u_2 \frown u_0)$ for every $u_1, u_2 \overset{2}{\triangleright} u_0$.

The edges introduced by the previous items are the only edges in a switch, that is $u_1 \smile u_2$ for every pair of vertices u_1, u_2 to which we cannot apply one of the above items.

Let us remark that a dummy link does not introduce any edge in a switch.
If G is an asl, $\mathsf{Sw}(G)$ is the set of its switches.

Definition 6 (DR-*Correctness, apn*)**.** An asl G is DR-*correct* when every $S \in \mathsf{Sw}(G)$ is connected and acyclic, namely S is a tree. A DR-correct aps is said an *abstract proof net* (an apn).

We shall say that an asl G is DR-acyclic when every switch of G is acyclic, or is DR-connected when every switch of G is connected.
The number of connected components of any switch of an aps G is related to the number of axioms and binary links in G.

Lemma 7. *Let G be a* DR*-acyclic aps. Let n_2 be the number of binary links and n_a be the number of axiom links in G. Then, every switch of G has*

$$k(G) = n_a - n_2$$

connected components.

Proof. We exploit the fact that the number of connected components of an acyclic graph is equal to $n_e - n_v$, where n_e is the number of the edges of the graph and n_v is the number of its vertices. Let n_1 be the number of unary links in G. By definition, the number of the vertices of G is equal to the number of the conclusions of the links of G, namely $n_v = 2n_a + n_1 + n_2$; while the number of edges in any switch of G is $n_e = n_a + n_1 + 2n_2$. Therefore, every switch of G has $n_v - n_e = n_a - n_2$ connected components. □

Lemma 8. *An aps G is an apn iff $n_a = n_2 + 1$ and it is* DR*-acyclic or* DR*-connected.*

Proof. The only if direction and the fact "G is DR-acyclic with $n_a - n_2 = 1$ implies G is DR-correct" are immediate corollaries of Lemma 7. Then, let us assume that G is DR-connected with $n_a - n_2 = 1$. As seen in the proof of Lemma 7, for every switch S of G, $n_a - n_2 = n_v - n_e$, where n_v is the number of vertices in the switch and n_e is the number of edges. Therefore, since S is connected by hypothesis, it cannot contain cycles. Summing up, G is DR-acyclic. □

The previous lemma shows that in order to prove the correctness of an aps, first of all, one can verify that the number of axioms is one more than the number of its binary links, then it suffices to verify either that the aps is DR-correct or that it is DR-acyclic.

Remark 9. If G is an asl, the statements in Lemmas 7 and 8 can be easily reformulated by adding the number n_p of the premises of G to the number of the vertices of G. Thus, a DR-acyclic asl G has $k(G) = n_p + n_a - n_2$ connected components and G is DR-correct iff $n_p + n_a = n_2 + 1$ and it is DR-acyclic or DR-connected.

3.3. The vertical partial order

The vertices of an apn (or of a DR-correct asl) can be ordered according to the vertical layout in Fig. 1. In fact, given an asl G, let us say that u_1 is immediately below u_0 (or that u_0 is immediately above u_1) when G contains a link $\alpha_0, u_0 \triangleright \alpha_1, u_1$.

Lemma 10. *For every* DR*-acyclic asl, the least preorder $\preccurlyeq$ induced by the binary relation "u_1 is immediately below u_0" is a partial order.*

$$(\overset{0}{\triangleright}) \qquad G\,;\,\beta \overset{0}{\triangleright} \alpha \;\to_c\; G\,;\,\beta \overset{*}{\triangleright} \alpha$$

$$(\overset{1}{\triangleright}) \quad G\,;\,\beta \overset{*}{\triangleright} \alpha, u_1, u_2\,;\,u_1, u_2 \overset{1}{\triangleright} u_0 \;\to_c\; G\,;\,\beta \overset{*}{\triangleright} u_0, \alpha$$

$$(\overset{2}{\triangleright}) \qquad G\,;\,u_1, u_2 \overset{2}{\triangleright} u_0 \;\to_c\; G\,;\,u_1, u_2 \overset{*}{\triangleright} u_0$$

$$(\overset{*}{\triangleright}) \qquad G\,;\,\beta_1 \overset{*}{\triangleright} \alpha_1, u\,;\,u, \beta_2 \overset{*}{\triangleright} \alpha_2 \;\to_c\; G\,;\,\beta_1, \beta_2 \overset{*}{\triangleright} \alpha_1, \alpha_2$$

Fig. 3. Contraction rules (c-rules).

Proof. Let $\varphi = u_0 \preccurlyeq u_1 \preccurlyeq \cdots \preccurlyeq u_k$ be a cyclic chain s.t. $u_0 = u_k$ is the only vertex that appears twice in the sequence, and u_i is immediately below u_{i+1} for $i = 0, \ldots, k-1$. It is readily seen that there is $S \in \mathsf{Sw}(G)$ s.t. φ is a path in S. But, since S is acyclic by hypothesis, $k = 0$. □

As an immediate consequence of the previous lemma, in a DR-acyclic asl, the maximal elements of the vertical preorder are either conclusions of a link without premises or premises of the asl. In particular, in any DR-acyclic aps G all the maximal elements are conclusions of axiom links, and in G there is at least one axiom link.

3.4. Contractibility

If we know that a sub-asl G_0 of an aps G is DR-correct, checking the correctness of $G = G_1\,;\,G_0$ can be reduced to checking the correctness of G_1. In fact, in every switch of $G_0 : \beta \vdash \alpha$ there is a path between every pair of vertices $u, v \in \alpha \cup \beta$. Therefore, let G' be the structure obtained from G by replacing G_0 with a new link $\beta \overset{*}{\triangleright} \alpha$; any switch of G is topologically equivalent to a corresponding switch of G', provided that the new link $\beta \overset{*}{\triangleright} \alpha$ is interpreted as a tree connecting the vertices $\alpha \cup \beta$.

Definition 11 (*Star Link, Contracting Structures*)**.** A *star link* is a link $\beta \overset{*}{\triangleright} \alpha$ with any number of premises and conclusions (but, by the definition of link, $\alpha \cup \beta \neq \emptyset$). A *contracting asl* (or a contracting aps) is an asl (an aps) that may contain star links also. A switch $S = (|G|, \frown)$ of the contracting asl G is a graph satisfying the constraints in Definition 2 plus:

(5) $\beta \overset{*}{\triangleright} \alpha \in G$ implies that there is a permutation $u_0, u_1, \ldots, u_k$ of α, β s.t. $u_i \frown u_{i+1}$, for $i = 0, 1, \ldots, k-1$.

A contracting asl G is DR-correct if every switch of G is connected and acyclic.

Lemma 12. *Let $G = G_1\,;\,G_0$ be a* DR*-correct contracting asl s.t. the contracting asl $G_0 : \beta \vdash \alpha$ is* DR*-correct. G is* DR*-correct iff the contracting asl $G' = G_1\,;\,\beta \overset{*}{\triangleright} \alpha$ is* DR*-correct.*

Proof. Every $S \in \mathsf{Sw}(G)$ can be transformed into an $S' \in \mathsf{Sw}(G')$ by replacing the $S_0 \in \mathsf{Sw}(G_0)$ contained in S with an $S_* \in \mathsf{Sw}(\beta \overset{*}{\triangleright} \alpha)$, and vice versa. By hypothesis, both S_* and S_0 are trees. Therefore, there exists $S \in \mathsf{Sw}(G)$ that is not a tree iff there exists $S' \in \mathsf{Sw}(G')$ that is not a tree. □

The previous lemma suggests the contraction rules, or c-rules, in Fig. 3. The application of a c-rule cannot introduce an invalid link, i.e. a star link without premises and conclusions, or with a conclusion that is also a premise of the link. Because of this, the $\overset{1}{\triangleright}$-rule and the $\overset{*}{\triangleright}$-rule can be applied only if

$(\overset{1}{\triangleright})$ in the $\overset{1}{\triangleright}$-rule, $u_0 \notin \beta$;
$(\overset{*}{\triangleright})$ in the $\overset{*}{\triangleright}$-rule, $\alpha_1 \cap \beta_2 = \alpha_2 \cap \beta_1 = \emptyset$ and at least one of $\alpha_1, \alpha_2, \beta_1, \beta_2$ is not empty.

It is readily seen that the above provisos always hold in a DR-correct aps.

The key point of the c-rules is that every star link introduced along the c-reduction of an asl G corresponds to a sub-asl of G.

Lemma 13. *Let G be a contracting asl s.t. $G \to_c^* G'$. For every star link $\beta \overset{*}{\triangleright} \alpha$ in G', there exists a unique* DR*-correct sub-asl $G_0 : \beta \vdash \alpha$ of G s.t. $G_0 \to_c^* \beta \overset{*}{\triangleright} \alpha$ and, if $G = G_1\,;\,G_0$ and $G' = G'_1\,;\,\beta \overset{*}{\triangleright} \alpha$, then $G_1 \to_c^* G'_1$.*

Proof. By induction on the length of the derivation $G \to_c^* G'$ we prove the existence of G_0. Then, let us assume that there is another DR-correct sub-asl $G'_0 : \beta \vdash \alpha$. Let $G'_0 = G''_0\,;\,G_x$, with $G''_0 = G_0 \cap G'_0$. Since every vertex is a premise/conclusion of at most one link, it is readily seen that $G''_0 : \beta \vdash \alpha$ and, as a consequence, $G_x : \vdash$. But this implies $G_x = \emptyset$ also, otherwise in any switch S of G'_0 the vertices of G''_0 and the vertices of G_x would be in distinct connected components, contradicting the hypothesis that G'_0 is DR-correct. Thus, $G'_0 \subseteq G_0$. But, since by the previous reasoning we can get $G_0 \subseteq G'_0$ also, we conclude that $G_0 = G'_0$. □

Therefore, DR-correctness is an invariant of c-reduction.

Lemma 14. *Let G be a contracting asl s.t. $G \to_c^* G'$. The contracting asl G' is* DR*-correct iff G is* DR*-correct.*

Proof. By induction on the number of star links in G' and by Lemmas 12 and 13. □

$$(\overset{0}{\triangleright}) \qquad G;\ \overset{0}{\triangleright}\ u_1, u_2 \ \to_p\ G;\ \overset{*}{\triangleright}\ u_1, u_2$$

$$(\overset{*}{\triangleright}) \qquad G;\ \overset{*}{\triangleright}\ \alpha, u\ ;\ u\ \overset{0}{\triangleright} \ \to_p\ G;\ \overset{*}{\triangleright}\ \alpha$$

$$(\overset{1}{\triangleright}) \qquad G;\ \overset{*}{\triangleright}\ \alpha, u_1, u_2\ ;\ u_1, u_2\ \overset{1}{\triangleright}\ u_0 \ \to_p\ G;\ \overset{*}{\triangleright}\ \alpha, u_0$$

$$(\overset{2}{\triangleright}) \quad G;\ \overset{*}{\triangleright}\ \alpha_1, u_1;\ \overset{*}{\triangleright}\ \alpha_2, u_2\ ;\ u_1, u_2\ \overset{2}{\triangleright}\ u_0 \ \to_p\ G;\ \overset{*}{\triangleright}\ \alpha_1, \alpha_2, u_0$$

Fig. 4. Parsing rules.

The rewriting system defined by the c-rules is terminating. Moreover, every DR-correct contracting asl has only one normal form.

Lemma 15. *Let $G : \beta \vdash \alpha$ be a contracting asl.*

(1) *There is no infinite* c*-reduction of G.*
(2) *If G is* DR*-correct, $\beta\ \overset{*}{\triangleright}\ \alpha$ is the only normal form of G.*

Proof.

(1) Every application of a c-rule decreases the number of root links, or the number of binary links, or the number of links in the contracting asl.
(2) Let us start with the case $\beta = \emptyset$. Let $G \to_c^* G'$ with G' in normal form. G' does not contain any binary or root link (otherwise, it would not be a normal form). By Lemma 14, G' is DR-correct and contains at least a link $\overset{*}{\triangleright}\ \gamma$ (the link above any vertex of G' that is maximal w.r.t. the vertical partial order extended to the case of contracting structures, see Lemma 10). The set γ contains some vertices $u_1, \ldots, u_k$ that are not conclusions of G' and some conclusions of G', that is $\gamma = u_1, \ldots, u_k, \alpha'$ with $\alpha' \subseteq \alpha$. For $i = 1, \ldots, k$, the vertex u_i is premise of a unary link $u_i, v_i\ \overset{1}{\triangleright}\ w_i$ s.t. $v_i \notin \gamma$ (by the hypothesis that G' is in n.f.). As a consequence, every switch of G' s.t. $v_i \frown w_i$ contains the switch of $\overset{*}{\triangleright}\ \gamma$ as an isolated connected component; which does not lead to a contradiction (by hypothesis G is DR-correct and then, by Lemma 14, G' is DR-correct too) only if $k = 0$ and $\gamma = \alpha' = \alpha$ and $G' = \overset{*}{\triangleright}\ \alpha$.

When $\beta = u_1, \ldots, u_k \neq \emptyset$, let us take the DR-correct aps G_β obtained by adding a root link $\overset{0}{\triangleright}\ u_i$ for every $u_i \in \beta$, that is $G_\beta = G;\ \overset{0}{\triangleright}\ u_1; \ldots;\ \overset{0}{\triangleright}\ u_k$. We have already proved that $\overset{*}{\triangleright}\ \alpha$ is the only normal form of G_β. Now, let us take any reduction $G \to_c^* G'$; there is a corresponding reduction of G_β s.t.

$$G_\beta \to_c^* G';\ \overset{0}{\triangleright}\ u_1\ ;\ \ldots;\ \overset{0}{\triangleright}\ u_k \to_c^* G';\ \overset{*}{\triangleright}\ u_1\ ;\ \ldots;\ \overset{*}{\triangleright}\ u_k \to_c^*\ \overset{*}{\triangleright}\ \alpha.$$

In particular, when G' is in normal form, $G' = \beta\ \overset{*}{\triangleright}\ \alpha$. □

We can then define the following *contractibility criterion*: an asl $G : \beta \vdash \alpha$ is correct when $\to_c$ reduces G to a structure formed of a star link only, i.e. G is correct iff $\beta\ \overset{*}{\triangleright}\ \alpha$ is its normal form (the only one) w.r.t. $\to_c$. This criterion yields to another characterization of proof nets [2].

Proposition 16 (c*-Correctness*)**.** *Let us say that the contracting aps $G \vdash \alpha$ is* c*-correct when $G \to_c^*\ \overset{*}{\triangleright}\ \alpha$. Then, G is* c*-correct iff G is* DR*-correct.*

Proof. It is an immediate consequence of Lemmas 14 and 15. □

3.5. Parsing

The proof of Proposition 16 [2,9,7] suggests a particular strategy for the application of the contractibility criterion. In fact, in any c-correct contracting asl G, there is at least a contraction $G \to_c G'$ such that G' is obtained by applying one of the rules in Fig. 3 at a source link. In other words, the contraction rules can be applied following an uppermost strategy. We can then define the parsing reduction, or p-reduction, as the transitive and reflexive closure of the parsing rules, or p-rules, in Fig. 4.

As for the c-rules, we must ensure that the p-rules do not introduce invalid links. Because of this, we have to add the proviso that the $\overset{*}{\triangleright}$-rule can be applied only if $\alpha \neq \emptyset$ (that always hold in a DR-correct asl). The other rules do not require any particular proviso instead. In fact, the assumption that in an asl a vertex is premise/conclusion of at most one link implies the correctness on the right-hand sides of these rules; in particular, the assumption that the left-hand side is an asl, in the $\overset{1}{\triangleright}$-rule implies that $u_0 \notin \alpha$, while in the $\overset{2}{\triangleright}$-rule implies that $\alpha_1 \cap \alpha_2 = \emptyset$ and $u_o \notin \alpha_1 \cup \alpha_2$.

The p-rules define a *parsing algorithm* for proof nets. In fact, any reduction $G \to_c^*\ \overset{*}{\triangleright}\ \alpha$ yields a sequentialization of $G \vdash \alpha$.

Definition 17 (*Parsing Aps*)**.** A *parsing aps* is a contracting aps that contains star links without premises only.

The p-rules define a subsystem of the c-rules.

Lemma 18. *If $G \to_p^* G'$, then $G \to_c^* G'$.*

Proof. By inspection of the contraction and parsing rules. □

As a consequence, we have that the p-reduction is terminating and that DR-correctness is an invariant of p-reduction. Therefore, in order to show that, as the c-rules, the p-rules define a correctness criterion, we have to prove that $\overset{*}{\triangleright}\ \alpha$ is the only normal form of a DR-correct aps $G \vdash \alpha$. For that purpose, let us first extend Lemma 8 to parsing aps (let us note that the star links without premises are a sort of axiom links).

Lemma 19. *Let G be a* DR*-correct parsing aps. If n_* is the number of star links, n_a the number of axiom links and n_2 the number of binary links in G, then*

$$n_* + n_a - n_2 = 1.$$

Proof. The only difference w.r.t. Lemma 7 is that we have to add h vertices and $h-1$ edges for every star link in G. Thus, if c_* is the number of conclusions of star links, we have to add c_* vertices and $c_* - n_*$ edges. Summing up, $n_v - n_e = (2n_a + n_1 + n_2 + c_*) - (n_a + n_1 + 2n_2 + c_* - n_*) = n_* + n_a - n_2$. □

Lemma 20. *The only* p*-normal form of any* DR*-correct parsing aps $G \vdash \alpha$ is the parsing aps $\overset{*}{\triangleright}\ \alpha$.*

Proof. Let $G \to_p^* G'$ with G' in normal form. By Lemmas 14 and 18, G' is DR-correct. Let $G' = G_0;\ \overset{*}{\triangleright}\ \alpha_1\ ;\ \ldots\ ;\ \overset{*}{\triangleright}\ \alpha_l$, where G_0 is an aps that does not contain star links. By hypothesis, G' does not contain p-redexes, more precisely: (i) G_0 does not contain axiom links; (ii) no $u \in \alpha_i$ can be premise of a dummy link; (iii) for every $u \in \alpha_i$ s.t. $u, v \overset{1}{\triangleright}\ w \in G_0$, $v \notin \alpha_i$. Since $G_0 = \emptyset$ implies $l = 1$ and $\alpha_1 = \alpha$ (by the assumption that G' is DR-correct), we shall show that $G_0 \neq \emptyset$ contradicts the hypotheses on G'.

Let $\alpha_i^x = \{u \in \alpha_i : u, v \overset{x}{\triangleright}\ w \in G_0\}$, with $x = 1, 2$ (by (ii), $\alpha_i = \alpha_i^1 \cup \alpha_i^2$). $G_0 \neq \emptyset$ implies that $\alpha_i^2 \neq \emptyset$ for every i, otherwise, by (iii), we could construct a switch S with an isolated connected component (by taking S s.t., for every $u \in \alpha_i^1$, if $u, v \overset{1}{\triangleright}\ w \in G_0$, then $u \smile w$). If $G_0 \neq \emptyset$, there are at least $n_* = l$ conclusions of star links that are premises of binary links. But, since $n_* = n_2 + 1$ (by (i) and Lemma 19), if $G_0 \neq \emptyset$, there is at least a link $u, v \overset{2}{\triangleright}\ w$ s.t. u and v are both conclusions of star links, for instance, $u \in \alpha_i$ and $v \in \alpha_j$. The case $i = j$ contradicts the DR-correctness of G', while the case $i \neq j$ contradicts that G' is a p-normal form. □

Proposition 21 (p*-Correctness*)**.** *Let us say that the proof structure $G \vdash \alpha$ is* p*-correct when $G \to_p^* \overset{*}{\triangleright}\ \alpha$. Then, G is* p*-correct iff G is* DR*-correct.*

Proof. It follows from Lemma 18 (only if) and Lemma 20 (if). □

4. Sequentialization

The parsing rules simulate the inference rules in Fig. 2. In fact, $G \in$ Seq iff G is p-correct. Therefore, any proof of Proposition 21 proves the sequentialization theorem too, see [9,7] also.

Theorem 22 (*Sequentialization*)**.** *An aps G is an apn iff $G \in$* Seq*.*

Proof. By Proposition 21, it suffices to prove, by induction on the size of G, that G is p-correct iff $G \in$ Seq. The base case, an aps formed of an axiom link only, is trivial.

The inductive case for the if direction follows by the induction hypothesis and by the observation that: if the last rule in the derivation of $G \in$ Seq is a par inference, then the last rule of the corresponding p-reduction is a $\overset{1}{\triangleright}$-rule; if it is a tensor inference, then the corresponding rule is a $\overset{2}{\triangleright}$-rule; if it is a cut inference, then the corresponding p-reduction ends with a $\overset{2}{\triangleright}$-rule followed by a $\overset{*}{\triangleright}$-rule.

We are left to prove the inductive case for the only if direction. Let us assume w.l.o.g. that in the reduction $\pi : G \to_p^* \overset{*}{\triangleright}\ \alpha$ every $\overset{2}{\triangleright}$-rule that eliminates a $\overset{2}{\triangleright}$-link whose conclusion is the premise of a dummy link is immediately followed by the $\overset{*}{\triangleright}$-rule that eliminates the dummy link too. Thus, if the last rule of π is a $\overset{*}{\triangleright}$-rule, we have that $G \to_p^* \overset{*}{\triangleright}\ \alpha_1, u_1;\ \overset{*}{\triangleright}\ \alpha_2, u_2\ ;\ u_1, u_2 \overset{2}{\triangleright}\ u_o\ ;\ u_o \overset{0}{\triangleright}$, with $\alpha = \alpha_1, \alpha_2$, for some binary link $u_1, u_2 \overset{2}{\triangleright}\ u_0$ above the dummy link $u_0 \overset{0}{\triangleright}$. By Lemma 13 (let us mention that, by Lemma 18, we can assume that the c-reductions in the statement of Lemma 13 are p-reductions), for $i = 1, 2$, there are $G_i \vdash \alpha_i, u_i$ s.t. $G_i \to_p^* \overset{*}{\triangleright}\ \alpha_i, u_i$. Then, since by the induction hypothesis $G_i \in$ Seq, by a cut inference we conclude that $G \in$ Seq. The cases in which the last rule of π is a $\overset{2}{\triangleright}$-rule or a $\overset{1}{\triangleright}$-rule are proved in the same way: the $\overset{2}{\triangleright}$-rule becomes a tensor inference, while the $\overset{1}{\triangleright}$-rule becomes a par inference. □

5. Computational complexity

Let us define the *size* of a proof structure G as the number of registers $size(G)$ required for the memorization of G on some random access machine (RAM). Although the definition of $size(G)$ depend on the representation of G, in any non-redundant encoding, $size(G)$ is linear in the number of vertices of G.

Remark 23 (*Asymptotic Notation*)**.** Let us recall the definition of the following classes of positive functions:

$$\mathcal{O}(f(n)) = \{g(n) \mid \exists c, k > 0 : n > k \Rightarrow cf(n) \geq g(n)\} \text{ (asymptotic upper-bound);}$$
$$\Omega(f(n)) = \{g(n) \mid \exists c, k \geq 0 : n > k \Rightarrow cf(n) \leq g(n)\} \text{ (asymptotic lower-bound);}$$
$$\Theta(f(n)) = \mathcal{O}(f(n)) \cap \Omega(f(n)).$$

In the following, we shall also use $\mathcal{O}(f(n))$, $\Theta(f(n))$ and $\Omega(f(n))$ to denote a generic element of the corresponding classes. We shall also use the $=$ symbol to denote that a function is in a given class, for instance, $g(n) = \mathcal{O}(f(n))$.

Using the asymptotic notation, $size(G) = \Theta(|\mathcal{V}(G)|)$. Moreover, since the number of links in G is linear in the number of vertices of G (i.e. $|G| = \Theta(|\mathcal{V}(G)|)$), $size(G) = \Theta(|G|)$ also. In the following, we shall analyze the worst case asymptotic complexity of DR-correctness in terms of $size(G)$.

D(anos)R(egnier)-correctness is the simplest and most appealing characterization of the proof structures that can be inductively constructed according to the rules of multiplicative linear logic. However, a straight application of the Danos–Regnier criterion requires $\Theta(size(G)\, 2^{size(G)})$ time: an aps G with $n = \Theta(|G|)$ unary links has 2^n switches and checking that a switch is a tree requires $\Theta(|G|)$ time. Instead, the criteria that we have presented in Sections 3.4 and 3.5 are quadratic. We could easily give a parsing strategy checking correctness in $\mathcal{O}(n \log n)$ time, but we shall do much better in the following.

6. Unification

It is a trivial programming exercise to give an algorithm equivalent to parsing that, instead of removing the vertices contracted along the reduction, marks them with a token. Moreover, since the key properties of the parsing algorithm are that every $\overset{*}{\triangleright}$-link corresponds to a sub-apn and that the parsing of a new sub-apn can be started at any point by replacing an axiom link with a $\overset{*}{\triangleright}$-link, we might also assume that more then one sub-apn is marked/parsed in parallel by distinct marking/parsing threads. This parallel marking approach will be presented as a sort of unification algorithm in the rest of this section.

However, in order to get a (sequential) linear algorithm, we shall see that we have to resort to a more sequential marking/parsing strategy. In particular, we shall see that the marking of a new sub-apn may be started only when the current marking thread reaches one of the premises of a $\overset{2}{\triangleright}$-link l whose second premise has not been marked yet—in terms of the parsing rules, one of the conclusions of the $\overset{*}{\triangleright}$-link corresponding to the current parsing thread is one of the premises of a $\overset{2}{\triangleright}$-link whose second premise is not the conclusion of a $\overset{*}{\triangleright}$-link. When this happens, the current marking thread must be suspended and a new thread must be started picking an axiom above the $\overset{2}{\triangleright}$-link l. The details of the sequential unification algorithm deriving from these considerations will be given in Section 7; where we shall also see that whenever we reach a situation such as the one described above, if the asl is correct, then all the links (and in particular the axioms) above the second premise of the $\overset{2}{\triangleright}$-link l have not been marked yet.

6.1. Parallel unification

The rules of the parallel algorithm for unification are:

(start) Assign a *fresh* token to the conclusions of an unmarked axiom (axiom $\overset{0}{\triangleright}$-rule).

(forward) Assign the token t to the conclusion of a unary link whose premises contain t ($\overset{1}{\triangleright}$-rule).

(unify) When the premises of a binary link contain two distinct tokens s and t, *equate* s and t and assign s or t to the conclusion of the link ($\overset{2}{\triangleright}$-rule).

Each of the previous rules corresponds to a virtual application of a parsing rule (the rule written in parentheses). The only parsing rule without correspondence in the unification approach is the $\overset{*}{\triangleright}$-rule (see Remark 24).

In order to give a more formal account of the rules described above, let us assume that

(1) a token is a set of integer indexes and distinct tokens correspond to disjoint sets;
(2) "assign the token t to the vertex v" means "mark the vertex v with any index in t";
(3) a vertex contains the token t when it has been marked by one of the indexes in t;
(4) in the start rule, "fresh" means "the least index in the interval $[0, k]$ that has not been used yet", where $k + 1$ is the number of axioms in the aps (for $h \geq 0$, $[0, h]$ denotes the closed interval of $\mathbb{N}$ from 0 to h; $[0, h[$ denotes instead the corresponding right open interval, i.e. $[0, h[= [0, h] \setminus \{h\}$);
(5) in the unify rule, "equate" means "unite the sets of".

If the aps G has $k + 1$ axioms, the application of a sequence of marking rules leads to the construction of a pair $\mu :: \pi$, where $\mu : \mathcal{V}(G) \rightharpoonup [0, k]$ is a (partial) *marking function* and π is a partition of the range of μ s.t. every element of π is a token and

(1) the asl $G[\mu, t]$ formed of the set of links whose premises and conclusions contain the token $t \in \pi$ is an aps;
(2) if $G[\mu, t] \vdash \alpha$, then there exists a parsing reduction $G[\mu, t] \rightarrow_p^* \overset{*}{\triangleright}\, \alpha$ (and then $G[\mu, t]$ is DR-correct);
(3) there is $G \rightarrow_p^* G'$ s.t. G' is obtained from G by replacing each $G[\mu, t] \vdash \alpha_t$ with $\overset{*}{\triangleright}\, \alpha_t$, for every t in π.

A formal proof of the above statements will be given in Proposition 27.

$$(\mathsf{start}) \quad \mu :: \pi \xrightarrow[u]{G} \mu[u_1, u_2 \mapsto next(\mu)] :: (\pi\,;\, next(\mu)) \quad \text{when } \overset{0}{\triangleright}\, u_1, u_2 \in G \text{ and } \mu(u_1) = \mu(u_2) = \bot$$

$$(\mathsf{forward}) \quad \mu :: \pi \xrightarrow[u]{G} \mu[u_0 \mapsto i] :: \pi \quad \text{when } u_1, u_2 \overset{1}{\triangleright}\, u_0 \in G \text{ and } \pi(\mu(u_1)) = \pi(\mu(u_2)) = i$$

$$(\mathsf{unify}) \quad \mu :: \pi \xrightarrow[u]{G} \mu[u_0 \mapsto j] :: \pi[i = j] \quad \text{when } u_1, u_2 \overset{2}{\triangleright}\, u_0 \in G \text{ and } j = \pi(\mu(u_1)) < \pi(\mu(u_2)) = i$$

Fig. 5. Unification.

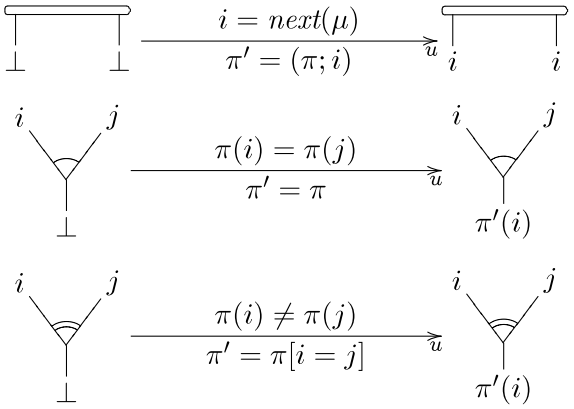


Fig. 6. Unification: a pictorial account.

Remark 24. In the marking approach, and then in the unification rules of Fig. 5, a marking rule has the purpose of extending to the conclusions of a link the marking of the premises of the link—with the relevant exception of the start-rule, which has the purpose to mark the conclusions of an axiom. Therefore, we do not need any particular treatment for dummy links (this is not the case for the parsing rules, where we need the $\overset{*}{\triangleright}$-rule to parse dummy links, see Fig. 4). Moreover, a dummy link is included into the asl $G[\mu, t]$ as soon as its premise contains the token t, that is as soon as the $\overset{2}{\triangleright}$-link above it propagates the token t to its conclusion. In other words, the rules in Fig. 5 suffice to ensure that a dummy link is parsed together with the $\overset{2}{\triangleright}$-link above it.

Every parsing reduction induces a corresponding unification reduction: for any $G \to^*_p G'$, there is a pair $\mu :: \pi$ of G s.t. there is a one-to-one correspondence between the star links in G' and the set of the asl's $G[\mu, t]$ s.t. $t \in \pi$ (let us recall that $G[\mu, t]$ is the asl formed of the set of links whose premises and conclusions contain the token $t \in \pi$).

Fig. 5 defines the rewriting system (parametric on G) that, when the reduction starts with the empty marking, derives the valid markings of G. Let us recall that a marking function is a partial function $\mu : \mathcal{V}(G) \rightharpoonup [0, h[$, where $h \leq k$ and $k + 1$ is the number of axioms in G. In the rules of Fig. 5 we use the following notations:

(1) the marking function μ with range $[0, h[$ is represented by means of the list of the pairs $\alpha_i \mapsto i$, where α_i is a set of vertices and $0 \leq i < h$, with $\alpha_i \cap \alpha_j = \emptyset$ if $i \neq j$. The vertices in α_i are the vertices of G marked by i; then $\mu(u) = i$ for every $u \in \alpha_i$;
(2) a partition is represented by means of a list of disjoint sets t_i, namely $\pi = (t_1\,;\, \ldots\,;\, t_l)$;
(3) if μ has range $[0, h[$, then $next(\mu) = h$;
(4) $\mu(u) = \bot$ when $\mu(u)$ is undefined;
(5) $\mu[\alpha \mapsto i]$ is the marking μ' s.t. $\mu'(u) = i$, when $u \in \alpha$, and $\mu'(u) = \mu(u)$, otherwise;
(6) $\pi(i)$ is the canonical representative of the set containing i, for instance, its least element;
(7) if $\pi(i) \neq \pi(j)$, then $\pi[i = j]$ is the partition obtained from π by merging the set containing i and the set containing j.

Any application of the rules in Fig. 5 starts with the empty marking $() :: ()$ and, when $() :: () \xrightarrow[u]{G}{}^* \mu :: \pi$, we shall write $\downarrow^G (\mu :: \pi)$.

Fig. 6 gives a pictorial account of the unification rules. In Fig. 6, π and π' are the partitions before and after the rewriting, respectively, and the relevant values of the marking functions are drawn in the place of the corresponding vertices.

Remark 25 (*Threads*)**.** Let ρ be a unification reduction s.t. $\downarrow^G (\mu :: \pi)$, and t be some token in π. The *thread* corresponding to t is the subset of the rewriting rules in ρ that assign an index $i \in t$ to some vertex of G. Each thread of ρ spans the sub-aps $G[\mu, t]$. Interpreting each thread as an independent unification process running in parallel with the other threads, we see that each start rule creates a new thread, and that the unify and forward rules are thread synchronization directives: forward asks for the synchronization of the threads of its premises; unify signals that the threads of its premises have synchronized and unites them into a unique thread.

When G has $k+1$ axioms and $\downarrow_G (\mu :: \pi)$, the range of μ is an interval $[0, h[$ with $h \leq k+1$, and π is a partition of $[0, h[$ with a class for each thread of the reduction. Then, G is an apn iff there is a marking pair $\mu :: \pi$ of G s.t. μ marks all the vertices of G and π equates all the tokens in the range of μ, that is G is an apn iff the unification of G ends up with a unique thread that spans all G.

Definition 26 (*Unifier,* u*-Correctness*)**.** Let G be an aps with $k+1$ axiom links. A marking function μ is a *unifier* of G when μ is a total function onto $[0, k]$ and $\downarrow^G (\mu :: 0, \ldots, k)$. The aps G is u*-correct* when it has a unifier.

Proposition 27. *A proof structure is* u*-correct iff it is* DR*-correct.*

Proof. By Proposition 21, it suffices to prove that an aps G is u-correct iff it is p-correct.
Only if direction. Given the aps $G \vdash \alpha$, let $\downarrow^G (\mu :: \pi)$ with $\pi = (t_1 ; \ldots ; t_l)$. Let $G[\mu :: \pi]$ be the parsing aps obtained by replacing $G[\mu, t_i] \vdash \alpha_i$ with $\overset{*}{\triangleright} \alpha_i$, for $i = 1, \ldots, l$ (the fact that $G[\mu, t_i]$ is an aps follows from the observation that $\mu(v) \in t_i$ implies $\mu(w) \in t_i$ for every pair of vertices $v \prec w$). By induction on the length of the unification reduction ρ that leads to $\mu :: \pi$, we see that $G[\mu, t_i] \to_p^* \overset{*}{\triangleright} \alpha_i$. When π is a unifier, $l = 1$ and $G[\mu, t_1] = G$. Thus, $G \to_p^* \overset{*}{\triangleright} \alpha$.
If direction. Let $G \to_p^* G'$ with $G' = G_0; \overset{*}{\triangleright} \alpha_1 ; \ldots ; \overset{*}{\triangleright} \alpha_l$, where G_0 does not contain any star link. By Lemma 13, $G = G_0; G_1; \ldots; G_l$ with $G_i \to_p^* \overset{*}{\triangleright} \alpha_i$ for $i = 1, \ldots, l$. By induction on the length of the parsing reduction, there exists $\downarrow^G (\mu :: \pi)$ with $\pi = (t_1 ; \ldots ; t_l)$ s.t. $G[\mu, t_i] = G_i$. In particular, when $G \to_p^* \overset{*}{\triangleright} \alpha$, the marking function μ is a unifier. □

Remark 28. Let ρ be a unification reduction. After m starts and n unifys, ρ contains $m - n$ threads—a thread for each token. Since a successful unification ends with only one thread, when G has a unifier, G contains $k+1$ axioms iff G contains k binary links.

6.2. Ready and waiting links

During unification, a link is *armed* when both its premises contain a token, and is *idle* otherwise. Each forward or unify rule *fires* an armed link and, because no rule can erase or change the token assigned to a vertex, no link can be fired twice. However, not every armed link can be fired. For instance, an armed binary link whose premises contain the same token is a *deadlock*. Neither can an armed unary link whose premises contain distinct tokens can be fired; since a following unify might unite the tokens of its premises, such a unary link is in a *waiting* state. An armed link that is not waiting and is not a deadlock is *ready* to be fired. A vertex is in the same state as the link above it: a vertex is ready when it is the conclusion of a ready link.

The unification algorithm consists of a main loop that picks a ready vertex (link) and fires one of the rules in Fig. 5. During this process, unification queries and updates the set of vertices that are ready and the set of vertices that are waiting, say R and W respectively. In particular, after removing and marking a ready vertex v from R, one of the following two cases applies:

(1) if the marking of v arms a unary link with conclusion w, then insert w in the set of the ready vertices R or in the set of the waiting vertices W, according to its state;
(2) if the marking of v arms a binary link with conclusion w and w is not a deadlock, then apply a unify rule, put w in R, and move the waiting vertices that become ready from W to R.

6.3. Implementation of unification

The following two considerations must be taken into account, if we want to implement unification efficiently.

A. First of all, let us observe the basic operations that we need on the sets of indexes in the partition π:
   (a) The side conditions of the forward and of the unify rules require one to know if the indexes assigned to the premises of an armed link belong to the same set.
   (b) The application of a unify rule requires one to merge two disjoint sets of indexes into a unique set.
   The above operations correspond to basic operations on the abstract data structure of a disjoint-set, and can be efficiently implemented by the so-called union-find algorithm.
B. If the set of waiting vertices W has no structure, finding the waiting vertices that have become ready after a unify rule requires scanning all W. As a consequence, a flat implementation of W causes a linear cost of unify, and an overall quadratic cost for unification, at least. The solution to this problem consists in choosing a particular strategy for the application of unification, which we shall call sequential unification, and which will be presented in Section 7.

6.3.1. Disjoint set union-find

As already remarked, the data structure implementing the partition π must be optimized for the so-called *disjoint-set union-find* operations [1, Chapter 22]:

FindSet(i): It computes the least element in the token of i (i.e. it computes $\pi(i)$).
Union(i, j): It merges the tokens of i and j and leaves the other tokens unchanged (i.e. it computes $\pi[i = j]$, provided that $\pi(i) \neq \pi(j)$).
MakeSet(i): It adds the token $\{i\}$ to π (i.e. it computes $(\pi; i)$, provided that $\pi(i) = \bot$).

Efficient data structures for (disjoint-set) union-find have been widely studied and used (e.g., in term unification [12]). The main property of the corresponding algorithms, also known as α-algorithms, is that the overall cost of m FindSet, Union and MakeSet operations is $\mathsf{UF}(m, n) = \mathcal{O}(m\,\alpha(m, n))$, where n is the number of MakeSet operations and α is a very slowly increasing function—in terms of growth slope, α is the inverse of the Ackermann function.

Remark 29. From a theoretical point of view, an α-algorithm is not linear, for $\alpha(h, k)$ is not a constant. Nevertheless, in any conceivable application, $\alpha(h, k) < 4$. In fact, $\alpha(h, k) = \min\{i \geq 1 : \mathsf{Ack}(i, \lfloor h/k \rfloor) > \lg k\}$, where Ack (the Ackermann function) is defined by: $\mathsf{Ack}(1, j) = 2^j$; $\mathsf{Ack}(i, 1) = \mathsf{Ack}(i-1, 2)$; $\mathsf{Ack}(i, j) = \mathsf{Ack}(i-1, \mathsf{Ack}(i, j-1))$; in particular, $\mathsf{Ack}(2, n)$ is a tower of exponentials of length n. Then, for every $h \geq k$, $\mathsf{Ack}(4, \lfloor h/k \rfloor) \geq \mathsf{Ack}(4, 1) = \mathsf{Ack}(2, 16)$, that is far greater than the estimated number of atoms in the observable universe (roughly 10^{80}).

However, even using an efficient solution for union-find with a (pseudo-)linear cost, without an efficient data structure for the representation of the waiting vertices W, the algorithm that we get is (pseudo-)quadratic.

7. Sequential unification

The worst case for the unification algorithm is when the proof structure G is correct—in that case, every unification reduction of G requires $|G|$ steps. As already noticed, this does not immediately imply linearity. In fact, for a correct evaluation of the computational cost of unification, we must take into account the cost of choosing the rule that applies and the cost of the operations on the partition π—the marking function μ is not a problem, it is an index field in the records used to represent links.

The rules in Fig. 5 define a parallel unification algorithm: there is a thread for each token and the threads run in parallel (see Remark 25). Unfortunately, this parallel point of view does not give any help in implementing the data structures of the ready vertices R and the waiting vertices W, as vertices are inserted and moved from R and W in no particular order. Instead, an efficient implementation of R and W requires finding a good unification strategy; in particular, it requires controlling thread creation.

During unification, the start rules number axioms (the index of an axiom is that of its conclusions) according to the order in which they are visited. Therefore, the token of i is older than the token of j when $\pi(i) < \pi(j)$, and similarly for the corresponding threads. Now, let us say that a ready link belongs to a thread if the rule that can fire it belongs to that thread (see Remark 25), e.g., a ready unary link belongs to the thread of its premises token. We want to control the order in which the links are fired by giving priority to the youngest thread in a unification reduction.

The youngest thread (token) of a reduction is its *active thread* (token). An *active link* is a ready link that belongs to the active thread, while an *active vertex* is the conclusion of an active link.

Definition 30 (*Sequential Strategy*)**.** After an initialization step that fires an axiom (any one), the *sequential strategy* for the application of the unification rules in Fig. 5 is formed of the following two steps:

(1) Repeat firing an active link, if any, as long as you do not mark a vertex v that is the premise of a binary link whose other premise w is not marked.
(2) When step 1 ends marking the vertex v, fire an axiom (i.e. start a new thread) above the vertex w found in step 1 and return to step 1.

The sequential strategy ensures that threads are united according to their age. In fact, when the thread of i creates a new thread assigning the index j to some axiom, by construction, $j > x$ for every x in the token of i. Moreover, there is a binary link whose commitment is to unite the thread of i and the thread of j. Then, let us assume that j creates a new thread assigning the index $j+1$ to some axiom; there is a binary link whose commitment is to unite the threads of j and $j+1$. After some steps, let the thread of $j+1$ unite with the thread of i. If j and $j+1$ are not in the same thread, there are two binary links committed to unite the same pair of threads—the thread of j and the thread of i and $j+1$. Therefore, unification will eventually reach a deadlock.

Let $\downarrow^G (\mu :: \pi)$ be obtained according to the sequential strategy. If $[0, h[$ is the range of μ, there is a sequence $i_0 < \cdots < i_n < \cdots < i_{l+1}$, with $i_0 = 0$ and $i_{l+1} = h$, s.t. π splits $[0, h[$ in the subintervals $[i_0, i_1[, [i_1, i_2[, \ldots, [i_l, i_{l+1}[$. Now, let us represent π by means of the stack $\sigma = i_0 : \cdots : i_n : \cdots : i_l$ and write $\sigma(i) = i_n$ when $i_n \leq i < i_{n+1}$. The ready vertices can be arranged into a stack of sets $R = \rho_0 : \cdots : \rho_n : \cdots : \rho_l$, s.t. $v \in \rho_n$ iff v is the conclusion of a unary link belonging to the thread of i_n, and then ρ_l is the set of the active vertices. Each unify merges the intervals $[i_{l-1}, i_l[$ and $[i_l, h[$, and adds the vertices in ρ_{l-1} to the set of the active links, namely a unify pops i_l from σ and merges the two sets on the top of R.

$$
\begin{array}{ll}
(\mathsf{init}) & \bot :: \emptyset :: \bot :: \emptyset \\
& \qquad \xrightarrow{G}_s \bot :: 0 :: \bot :: (u_1, u_2) \\
& \text{when } \overset{0}{\triangleright}\, u_1, u_2 \in G \\
\\
(\mathsf{concl}) & \mu :: (\sigma : i) :: W :: (R : \rho, u_0) \\
& \qquad \xrightarrow{G}_s \mu[u_0 \mapsto i] :: (\sigma : i) :: W :: (R : \rho) \\
& \text{when } G \vdash \alpha, u_0 \\
& \quad \text{or } u_0 \overset{0}{\triangleright} \in G \\
\\
(\mathsf{nop}) & \mu :: (\sigma : i) :: W :: (R : \rho, u_1) \\
& \qquad \xrightarrow{G}_s \mu[u_1 \mapsto i] :: (\sigma : i) :: W :: (R : \rho) \\
& \text{when } u_1, u_2 \overset{1}{\triangleright} u_0 \in G \text{ with } \mu(u_2) = \bot \\
\\
(\mathsf{wait}) & \mu :: (\sigma : i) :: W :: (R : \rho, u_1) \\
& \qquad \xrightarrow{G}_s \mu[u_1 \mapsto i] :: (\sigma : i) :: W[\sigma(\mu(u_2)) \leftarrow\!\shortmid u_0] :: (R : \rho) \\
& \text{when } u_1, u_2 \overset{1}{\triangleright} u_0 \in G \text{ with } \mu(u_2) < i \\
\\
(\mathsf{forward}) & \mu :: (\sigma : i) :: W :: (R : \rho, u_1) \\
& \qquad \xrightarrow{G}_s \mu[u_1 \mapsto i] :: (\sigma : i) :: W :: (R : \rho, u_0) \\
& \text{when } u_1, u_2 \overset{1}{\triangleright} u_0 \in G \text{ with } \mu(u_0) = \bot \text{ and } \mu(u_2) \geq i \\
\\
(\mathsf{new}) & \mu :: (\sigma : i) :: W :: (R : \rho, u_1) \\
& \qquad \xrightarrow{G}_s \mu[u_1 \mapsto i] :: (\sigma : i : j) :: W[i \mapsto \emptyset] :: (R : \rho : v_1, v_2) \\
& \qquad\qquad \text{where } j = \mathit{next}(\mu) \\
& \text{when } u_1, u_2 \overset{2}{\triangleright} u_0 \in G \text{ and } \overset{0}{\triangleright}\, v_1, v_2 \in G \text{ with} \\
& \qquad u_2 \preccurlyeq v_1 \text{ and } \mu(u_2) = \mu[u_1 \mapsto i](v_1) = \mu[u_1 \mapsto i](v_2) = \bot \\
\\
(\mathsf{unify}) & \mu :: (\sigma : j : i) :: W :: (R : \rho' : \rho, u_1) \\
& \qquad \xrightarrow{G}_s \mu[u_1 \mapsto i] :: (\sigma : j) :: W[j \mapsto \bot] :: (R : u_0, W(j), \rho', \rho) \\
& \text{when } u_1, u_2 \overset{2}{\triangleright} u_0 \in G \text{ with } j \leq \mu(u_2) < i
\end{array}
$$

Fig. 7. Sequential unification: main rules.

Using the sequential strategy has a consequence on the structure of the waiting vertices too. Let us say that v_0 is *waiting for* i when it is the conclusion of a unary link $v_1, v_2 \overset{1}{\triangleright} v_0$ s.t. $i = \pi(\mu(v_1))$ and $\mu(v_1) < \mu(v_2)$ (note that this means $\pi(\mu(v_1)) < \pi(\mu(v_2))$ also); we can assume that W is a function (or an array) s.t.: for $j = i_0, \ldots, i_{l-1}$ (we recall that $\sigma = i_0 : \cdots : i_l$), $W(j)$ is the set of vertices that are waiting for j; otherwise, $W(j) = \bot$. Since unify unites the tokens of i_l and i_{l-1}, after a unify, we have that:

(1) the vertices in $W(i_{l-1})$ become ready and active, i.e. they must be added to the set on the top of R;
(2) for $n < l - 1$, the vertices in $W(i_n)$ keep waiting for i_n.

The latter considerations lead to the *sequential unification* algorithm, whose main rules are given in Fig. 7. The following notations are new:

(1) $W' = W[i \leftarrow\!\shortmid u]$, with the proviso $W(i) \neq \bot$, means that $W'(i) = W(i), u$, while $W'(j) = W(j)$, if $j \neq i$;
(2) $W' = W[i \mapsto \bot]$ and $W' = W[i \mapsto \emptyset]$ set the value of $W'(i)$ to $\bot$ and to $\emptyset$, respectively, leaving $W'(j) = W(j)$, for $j \neq i$ (we stress that $W(j) = \emptyset$ means that the jth stack has been initialized and is empty, while $W(j) = \bot$ means that the jth stack is undefined, therefore no operation can be done on it).

$$(\mathsf{true}) \quad \mu :: 0 :: \bot :: \emptyset \xrightarrow{G}_s \mathsf{true} \quad \text{when } \mu(u) \neq \bot \text{ for every } u \in \mathcal{V}(G)$$

$$(\mathsf{false}_1) \quad \mu :: \sigma :: W :: (R : \emptyset) \xrightarrow{G}_s \mathsf{false} \quad \text{when } \mu(u) = \bot \text{ for some } u \in \mathcal{V}(G)$$

$$(\mathsf{false}_2) \quad \mu :: (\sigma : j : i) :: W :: (R : \rho' : \rho, u_1) \xrightarrow{G}_s \mathsf{false} \quad \text{when } u_1, u_2 \overset{2}{\rhd} u_0 \in G \text{ with } \mu(u_2) < j \text{ or } \mu(u_2) \geq i$$

$$(\mathsf{false}_3) \quad \mu :: (\sigma : i) :: W :: (R : \rho, u_1) \xrightarrow{G}_s \mathsf{false}$$
when $u_1, u_2 \overset{2}{\rhd} u_0 \in G$ with $\mu(u_2) = \bot$
and one of the following cases holds:
1. $u_2 \prec u_2$
2. $\mu(v)[u_1 \mapsto i] \neq \bot$
for some v s.t. $v \succcurlyeq u_2$ or s.t. $\overset{0}{\rhd} v, w \in G$ and $w \succcurlyeq u_2$

Fig. 8. Sequential unification: answer rules.

The rules for sequential unification are completed by the answer rules in Fig. 8. An instance of an answer rule can only be the last rule of a sequential unification reduction; moreover, by Proposition 37, we shall see that the last rule of any maximal sequential unification reduction is an instance of an answer rule, that is true and false are the only normal forms of sequential unification.

Fig. 9 gives a pictorial account of sequential unification. We have added the case in which the vertex v popped from R is the premise of a deadlock; by the way, this corresponds to an incorrect aps.

Sequential unification rewrites four-tuples with the shape $\mu :: \sigma :: W :: R$. Let $\xrightarrow{G}_s^*$ be the transitive and reflexive closure of $\xrightarrow{G}_s$; in the following, we shall write $\Downarrow^G (\mu :: \sigma :: W :: R)$, when $((\,) :: (\,) :: (\,) :: (\,)) \xrightarrow{G}_s^* (\mu :: \sigma :: W :: R)$.

For technical reasons, in the rules of sequential unification, we do not mark a vertex immediately after firing the link above it; instead, we put it into R. This simplifies the specification of the algorithm, preserving the property that each set in R contains the vertices that are ready to be marked with the corresponding index in σ.

Apart for init, each rule in Fig. 7 pops a vertex u from R, marks it with the top index of σ, verifies the state of the link below u and, according to this state, performs some operations on $\mu :: \sigma :: W :: R$. For instance, when u is the premise of a unary link with conclusion v; if we can apply a wait, then u is a premise of a waiting link, and wait inserts its conclusion v in the proper set of W; if we can apply a forward, then u is the premise of a ready link, and forward inserts its conclusion v in the top set of R. We remark new: it implements step 2 of the sequential strategy.

Remark 31. We recall that R is a stack of sets, i.e. $R = \alpha_0 : \alpha_1 : \ldots : \alpha_k$. Therefore, the vertex u popped from R is any vertex in α_k. We also remark that, when the top set of R is empty, no rule of sequential unification applies. In particular, the case in which $k > 0$ and $\alpha_k = \emptyset$ corresponds to a deadlock and cannot arise in an apn.

Neglecting the cost of the union-find operations, all the sequential unification rules apart from new can be executed in $\mathcal{O}(1)$ time. This is not true for new, as it requires us to go up along the net until we find an axiom. The function in Fig. 10 implements this search.

During the search, NextAxiom sets a tag associated to each vertex. That tag, initially equal to false, is true when the vertex v has already been visited by some NextAxiom or when v is the conclusion of the axiom fired by init—in practice, after initializing all the tags to false, we can assume that the starting axiom is found calling NextAxiom(v), where v is any vertex. According to this, NextAxiom(v) returns an error whenever v has already been visited, i.e. $\mu(v) \neq \bot$ or $\mathsf{tag}(v) = \mathsf{true}$ (if NextAxiom is properly called by init and by new only; both cases are not possible in an apn). We stress that the use of tags ensures that NextAxiom cannot loop (by the way, this might happen in an incorrect aps) and that, during sequential unification, either NextAxiom does not visit any vertex twice or it stops after finding an already marked vertex.

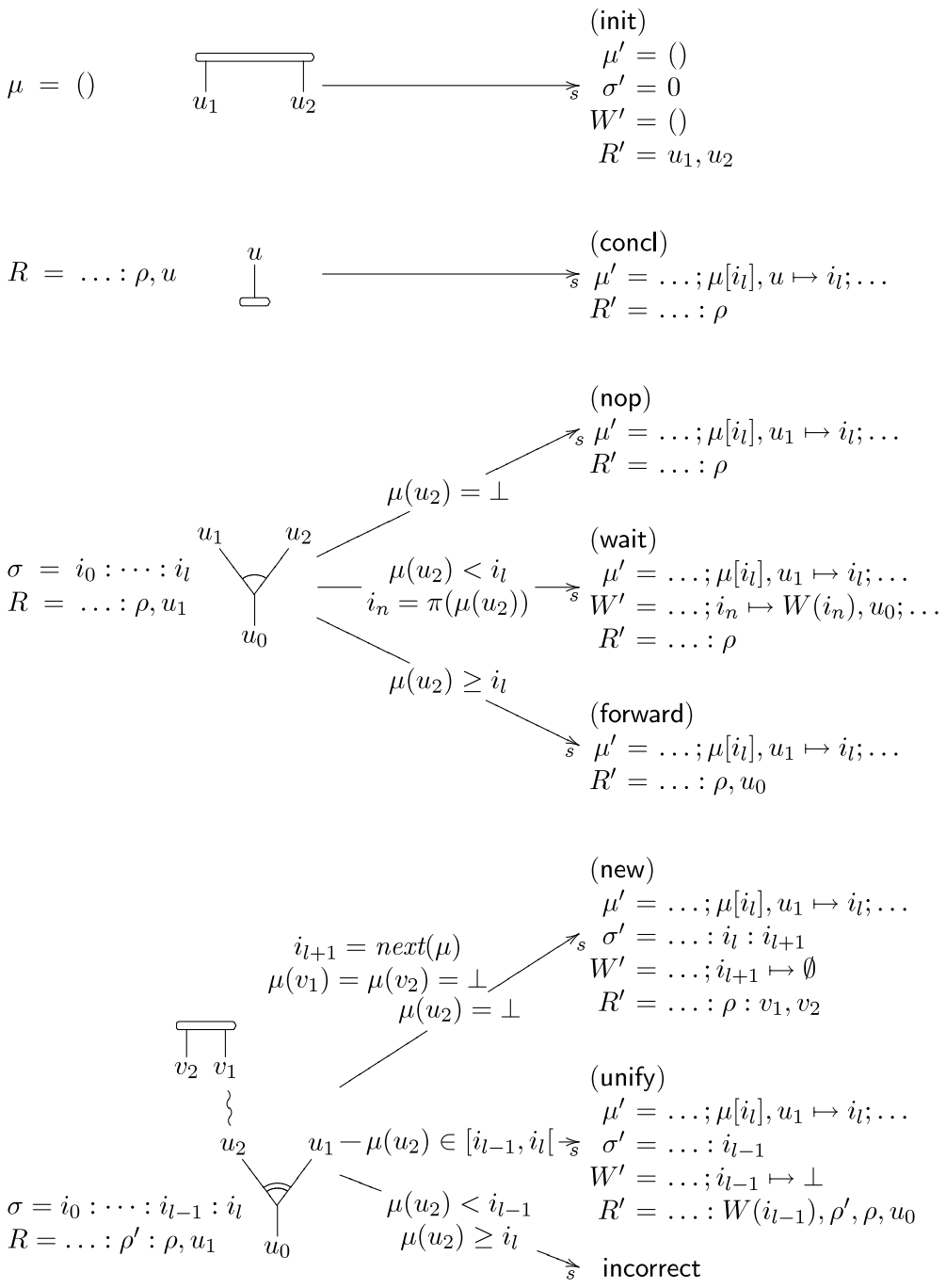


Fig. 9. Sequential unification: a pictorial account.

Every sequential unification reduction corresponds to a (parallel) unification reduction. More precisely, whenever $\Downarrow^G (\mu :: 0 : i_1 \cdots : i_l :: W :: R)$ with $next(\mu) = h$, we have $\downarrow^G (\mu :: [0, i_1[; [i_1, i_2[; \ldots; [i_l, h[)$ also. Moreover, if G is an apn and μ is not a unifier, the top set of R is not empty. Hence, sequential unification of an apn cannot stop before finding a unifier.

7.1. Correctness of sequential unification

In order to prove correctness of sequential unification (Proposition 37), we prove by induction on the length of any non-empty reduction

$$(\bot :: \emptyset :: \bot :: \emptyset) \xrightarrow{G}{}^*_s (\mu' :: \sigma' :: W' :: R') \xrightarrow{G}_s (\mu :: \sigma :: W :: R)$$

```
NextAxiom(v)
  if (tag(v) = true or μ(v) ≠ ⊥)
      return error
  else if (v' <0▷ v, v ∈ G)
      tag(v') ← true; tag(v) ← true;
      return v, v';
  else if (v', v'' <1,2▷ v ∈ G)
      tag(v) ← true;
      return NextAxiom(v');
```

Fig. 10. The NextAxiom function.

some useful invariants of the main rules of sequential unification (Fact 32 and from Lemma 33 to Lemma 36). The fact that true and false are the only normal forms of sequential unification, namely that every maximal sequential unification reduction ends with an answer rule, will be proved in Proposition 37.

Fact 32. *Let $\sigma = i_1 : i_2 \cdots : i_l$ and $\sigma' = i'_1 : i'_2 \cdots : i'_{l'}$.*

(1) *The length of σ (σ') is equal to the number of sets in R (R'). That is*

$$R = \rho_1 : \rho_2 : \cdots : \rho_l \qquad R' = \rho'_1 : \rho'_2 : \cdots : \rho'_{l'}$$

(2) *$W(i) \neq \bot$ iff $i = i_n$, for some $1 \leq n < l$ (and $W'(i) \neq \bot$ iff $i = i'_n$, for some $1 \leq n < l'$). Therefore, if we take*

$$W(i_n) = W_n \qquad W'(i'_n) = W'_n$$

$u \in W_n$ (and $u \in W'_n$) only if u is a conclusion of a unary link and $\mu(u) = \bot$.

(3) *Let us define*

$$V_n = \{u \in \mathcal{V}(G) \mid \sigma(\mu(u)) = i_n\} \qquad V'_n = \{u \in \mathcal{V}(G) \mid \sigma'(\mu'(u)) = i'_n\}$$
$$\overline{V}_n = V_n \cup \rho_n \qquad \overline{V'}_n = V'_n \cup \rho'_n$$
$$V = \bigcup_{n=1}^{l} V_n \qquad V' = \bigcup_{n=1}^{l'} V'_n$$
$$\overline{V} = \bigcup_{n=1}^{l} \overline{V}_n \qquad \overline{V'} = \bigcup_{n=1}^{l'} \overline{V'}_n$$

$V' \subseteq V$ and $\overline{V'} \subseteq \overline{V}$; moreover, if the last rule is not init, *$|V| = |V' + 1|$.*

Proof. By induction on the length of the derivation and by case analysis of the last rule. □

The notations introduced in Fact 32 will be used in all the lemmas of this section.

According to the definition in Fact 32, V_n is the set of the vertices marked by an index in the equivalence class (the token) of i_n. By the next lemmas instead, we shall see (Lemma 34) that ρ_n is a set of vertices ready to be marked by the index i_n, namely that every vertex in ρ_n has not been marked yet and that it is either a conclusion of an axiom whose other conclusion has already been marked by an index in the class of i_n, or a conclusion of a link whose premises have been marked by an index in the class of i_n. According to this, $\overline{V}_n$ is the set of vertices that at a given point are spanned by the indexes in the class of i_n, and that correspond to a sub-aps (see Lemma 35).

The next lemmas will be proved by induction on the length of the derivation and by case analysis of the last rule. We stress that, since the first rule of every non-empty sequential unification reduction is an init, the base case of the induction will be that of a reduction formed of an init rule only.

First of all, we take into account axioms, proving that either none of the conclusions of an axiom has been marked, or that both the conclusions have been or will be marked by indexes in the same class.

Lemma 33. *Let $u' \overset{0}{\triangleright} u''$, $u'' \in G$. One of the following two cases holds:*

A. *$u', u'' \notin \overline{V}$.*

B. *There is a unique n s.t. $u', u'' \in \overline{V}_n$ and*
   (a) *$u' \in \rho_n$ implies $\mu(u') = \bot$;*
   (b) *if $n < l$, then $u' \in \rho_n$ implies $u'' \in V_n$;*
   (c) *if $n = l$, then $u', u'' \in \rho_l$ implies $\overline{V}_l = \rho_l = \{u', u''\}$.*

Proof. The base case is trivial. Therefore, to conclude the proof, it suffices to analyze the following possibilities.

A. $u', u'' \notin \overline{V'}$. If the last rule is not a new, no vertex in $\overline{V} \setminus \overline{V'}$ is conclusion of an axiom link. Therefore, $u', u'' \notin \overline{V}$. Now, let us assume that the last rule is a new that involves the axiom $\overset{0}{\triangleright}\, v_1, v_2$. By the side condition of new, $\mu(v_1) = \mu(v_2) = \bot$ and $\rho'_{l'} \neq \{v_1, v_2\}$; that is, by the induction hypothesis, $\overline{V} \setminus \overline{V'} = \{v_1, v_2\}$. Then, when $\{u', u''\} = \{v_1, v_2\}$, B holds with $n = l$; otherwise, A holds.

B. $u', u'' \in \overline{V'}_m$ for some $m \leq l'$. If the last rule is a

concl, nop, wait, forward. We have that: (i) for $n < l' = l$, $V'_n = V_n$ and $\rho_n = \rho'_n$; (ii) $V_l = V'_l \cup \{v\}$ for some v s.t. $\rho'_l = \rho, v$ and $\rho_l = \rho, \alpha$, for some ρ, where α is empty or equal to the conclusion of a unary link. This suffices to conclude that $u', u'' \in \overline{V}_n$ iff $n = m$ and that items 1, 2 hold. Now, in order to conclude item 3 too, let us notice that $m = l$ and $u', u'' \in \rho_l$ implies $u', u'' \in \rho'_l$; but, by the induction hypothesis, $u', u'' \in \rho'_l$ implies $v \notin \rho'_l = \{u', u''\}$. Therefore, $\{u', u''\} \not\subseteq \rho_l$.

new. Let us start by proving $u', u'' \notin \overline{[V_l]}$. In particular, let us prove $\{u', u''\} \neq \{v_1, v_2\}$. This is definitely the case when $\{u', u''\} \not\subseteq \rho'_n$. In fact, w.l.o.g., let $u' \notin \rho'_n$; we have $u' \in V_n$ and, by the side condition of the rule, $\mu(v_1) = \mu(v_2) = \bot$. Now, let $\{u', u''\} \subseteq \rho'_n$. By the induction hypothesis, that implies $n = l'$ and $\rho'_{l'} = \{u', u''\}$. W.l.o.g., let us assume that u' is the premise of the binary link in the rule. The side condition of the rule implies $\mu(u') \neq \bot$ and $\mu(v_1) = \mu(v_2) = \bot$. Therefore, as $V_l = \emptyset$ (by $i_l = next(\mu')$), this complete the proof of $u', u'' \notin \overline{V}_l$. Now, after noticing that $\overline{V}_n = \overline{V'}_n$ and $\rho_n = \rho'_n$ for $n < l' = l - 1$, that $V_{l'} = V_{l'} \cup \{v\}$, and that $\rho'_l = \rho_l, v$, we can proceed as in the cases seen above.

unify. We have: $V_n = V'_n$ and $\rho_n = \rho'_n$ for $n < l = l' - 1$; $V_l = V'_l \cup V'_{l+1} \cup u_1$ and $\rho'_l = \rho'_{l+1}, u_1$ for some ρ and $\rho_l = \rho'_l, \rho'_{l+1}, W_l, u_0$, where $u_1, u_2 \overset{2}{\triangleright} u_0$ is the binary link in the redex of the rule. Therefore, $u', u'' \in V_n$ iff $n = \min\{m, l\}$. Moreover, for $m < l$, items 1 and 2 hold immediately by the induction hypothesis. For $m = l$, $\{u', u''\} \not\subseteq \rho'_l$ by the induction hypothesis. For $m = l + 1$ instead, we can have $u', u'' \in \rho_l$ only if $u', u'' \in \rho'_{l+1}$; but, $u', u'' \in \rho'_{l+1}$ implies $u_1 \in \{u', u''\}$ (by the induction hypothesis) and $u_1 \notin \rho_l$. As a consequence, for $m = l, l + 1$, item 3 holds because of the fact that $\{u', u''\} \not\subseteq \rho_l$. □

We consider now the case of a $\overset{1}{\triangleright}$ and $\overset{2}{\triangleright}$-links, namely of some $u', u'' \triangleright u \in G$. If the conclusion u has been marked by an index i_n or it is ready to be marked by i_n (namely $u \in \rho_n$), then all the vertices above it (every $v \succ u$) have been marked by an index in the class of i_n (and in the case $u \in \rho_n$, the vertex u has not been marked yet). Instead, u is waiting for i_n, that is $u \in W_n$, iff u is the conclusion of a $\overset{1}{\triangleright}$-link, and one of its premises has been marked with an index in the class of i_n, while the other premise has been marked with an index in the class of i_m, for some $m > n$. Finally, it is not the case that the two premises of a $\overset{2}{\triangleright}$-link might be marked with indexes in distinct classes.

Lemma 34. *Let $u', u'' \triangleright u \in G$ and $n = 1, \ldots, l$.*

A. *$u \in V_n$ implies $v \in V_n$ for every $v \succ u$;*
B. *$u \in \rho_n$ iff $\mu(u) = \bot$ and $v \in V_n$ for every $v \succ u$;*
C. *$u \in W_n$, with $n < l$, iff $u'' \in V_m$ for some $m > n$, $u' \in V_n$ and $u', u'' \overset{1}{\triangleright} u \in G$;*
D. *$u' \in V_n$ and $u'' \in V_m$ with $m \neq n$ only if $u', u'' \overset{1}{\triangleright} u \in G$.*

Proof. As a preliminary step, let us remark that, by the induction hypothesis and Lemma 33, $u \in \rho'_{l'}$ only if $\mu(u) = \bot$ and that $V = V' \cup \{v\}$ for some $v \in \rho'_{l'}$. Then, just for the proof of the lemma, let us define

$$\widehat{\rho}_n = \{u \in \mathcal{V}(G) \mid \mu(u) = \bot \wedge \exists v \succ u \wedge \forall v \succ u : v \in V_n\}$$
$$\widehat{W}_n = \{u \in \mathcal{V}(G) \mid \exists u', u'' \overset{1}{\triangleright} u \in G \text{ s.t. } u' \in V_n \wedge \exists m > n : u'' \in V_m\}.$$

With these notations, items B and C become $\rho_n = \widehat{\rho}_n$ and $W_n = \widehat{W}_n$, respectively.

We can now prove each item of lemmas by induction on the length of the derivation (the base case is immediate).

A. Whichever is the last rule, $V_n = V'_n$ for $n < \min\{l, l'\}$. Then, for $n \geq \min\{l, l'\}$, let us proceed by cases w.r.t. the last rule of the reduction.

concl, nop, wait, forward: As $l = l'$, we have $n = l$ only and $V_l = V'_l \cup \{u'\}$ for some $u' \in \rho'_l$.
new: As $l = l' + 1$, we have the cases $n = l - 1, l$. But, $V_l = \emptyset$ and $V_{l-1} = V'_{l-1} \cup \{u'\}$ for some $u' \in \rho'_{l-1}$.
unify: As $l = l' - 1$, we have $n = l$ only and $V_l = V'_l \cup V'_{l+1} \cup \{u'\}$ for some $u' \in \rho'_{l+1}$.
In each of the previous cases, A follows by the induction hypothesis (A and B).

B. Whichever is the last rule, $\rho_n = \rho'_n$ and $\widehat{\rho}_n = \widehat{\rho}'_n$ for $n < \min\{l, l'\}$. For the other values of n, let us proceed by cases w.r.t. the last rule.

concl, nop, wait, new: $l' \leq l$ and $\widehat{\rho}_{l'} = \widehat{\rho}'_{l'} \cup \{u'\}$ for some u' s.t. $\rho_{l'} = \rho'_{l'} \cup \{u'\}$. Moreover, new is the only case in which $l' < l = l' + 1$; but, after a new, $\widehat{\rho}_l = \widehat{\rho}'_l = \emptyset$. By the induction hypothesis, we conclude.

forward: $l' = l$ and there is $u_1, u_2 \overset{1}{\triangleright} u_0 \in G$ s.t. $u_2 \in V_n$ and, for some ρ, $\rho'_l = \rho \cup \{u_1\}$ and $\rho_l = \rho \cup \{u_0\}$. By the induction hypothesis, $\widehat{\rho}'_l = \rho \cup \{u_1\}$ (by B) and $\mu(u_0) = \bot$ (by A). Therefore, $\widehat{\rho}_l = \rho \cup \{u_0\}$.

unify: As $l = l' - 1$, we have to analyze the case $n = l$ only. By inspection of the rule, $\rho_l = \rho'_l \cup \rho \cup W'_l \cup \{u_0\}$ and $\rho_{l+1} = \rho \cup \{u_1\}$ for some ρ and some $u_1, u_2 \overset{2}{\triangleright} u_0 \in G$ s.t. $u_2 \in V_l$. By the induction hypothesis, we see that: $\widehat{\rho}'_{l+1} = \rho \cup \{u_1\}$ (by B); $u_0 \notin V'$ (by A); for $w \in W_l$, $\mu(w) = \bot$ (by C and A) and $v \in V'_l \cup V'_{l+1}$ for every $v \succ w$ (by C). Therefore, $\rho \cup \widehat{\rho}'_l \cup W_l \cup \{u_0\} = \rho_l \subseteq \widehat{\rho}_l$. Now, let $w \in \widehat{\rho}_l$ with $w \neq u_0$ and $w \notin \rho \cup \rho'_{l+1}$; by definition, w is not conclusion of an axiom. If $w' \in V'_n$ with $n = l$ or $n = l + 1$, then $w \in \widehat{\rho}'_n \setminus \{u_1\} = \rho'_n \setminus \{u_1\} \subseteq \rho_l$. Now, let us assume that there exist $w' \in V'_l$ and $w'' \in V'_{l+1}$ with $w', w'' \succ w$. By the induction hypothesis (item A), we can take, w.l.o.g., $w', w'' \triangleright w$. By the induction hypothesis (C and D), $w', w'' \overset{1}{\triangleright} w$ and $w \in W_l \subseteq \rho_l$. Summing up, $\widehat{\rho}_l \subseteq \rho_l$. Since we have already seen the converse, we conclude.

C. Let us proceed by case analysis.

concl, nop, forward, unify: Trivial, as $\widehat{W}_n = \widehat{W}'_n$ for $n < l' = l$.

new: As in the previous case for $n < l' = l - 1$. If $n = l - 1$, $W_{l-1} = \emptyset$ and $V_l = \emptyset$ (by the definition of the rule); so, $\widehat{W}_{l-1} = \emptyset$ also.

forward: Let $u_1, u_2 \overset{1}{\triangleright} u_0$ be the link in the redex, with $u_1 \in V_l$ and $u_2 \in V_m$. It is readily seen that $\widehat{W}_n = \widehat{W}'_n$ for $n \neq m$ and that $\widehat{W}_m = \widehat{W}'_m \cup \{u_0\}$.

D. By inspection of the rules. □

We can now conclude that every $\overline{V}_n$ determines a sub-aps of G defined by

$$G_n = \{\alpha \triangleright u \in G \mid u \in \overline{V}_n\} \cup \{\overset{0}{\triangleright} u_1, u_2 \in G \mid u_1, u_2 \in \overline{V}_n\} \cup \{u \overset{0}{\triangleright} \in G \mid u \in V_n\}.$$

In fact, let $\alpha' \triangleright \alpha \in G_n$. By Lemmas 33 and 34, $u \in \overline{V}_n$ for every $u \in \alpha, \alpha'$. Moreover, every vertex in ρ_n is a conclusion of G_n, that is

$$G_n \vdash \gamma_n, \rho_n \quad \text{where } \gamma_n = \{u \in V_n \mid u \overset{0}{\triangleright} \notin G \wedge \forall v \prec u : v \notin \overline{V}_n\}$$

and G_n is correct.

Lemma 35. *$G_n \to_p^* \overset{*}{\triangleright} \gamma_n, \rho_n$ for $n = 1, \ldots, l$.*

Proof. The case of init is trivial, as $l = 1$ and $G_1 = \overset{0}{\triangleright} u_1, u_2$ (refer to the corresponding rule for the names of the vertices). In the other cases, $G_n = G'_n$ for $n < l$ (by inspection of the rules). While, by Lemmas 33 and 34, when the last rule is a

concl: $G_l = G'_l$ or $G_l = G'_l$; $u \overset{0}{\triangleright}$ if $u \overset{0}{\triangleright} \in G$.

nop, wait: $G_l = G'_l$.

forward: $u_1, u_2 \in \rho'_l, \gamma'_l$ and $G_l = G'_l$; $u_1, u_2 \overset{1}{\triangleright} u_0$.

new: $G_l = \overset{0}{\triangleright} v_1, v_2$.

unify: $u_1 \in \rho'_{l+1}, u_2 \in \gamma'_l$ and $G_l = G'_l$; G'_{l+1} ; $u_1, u_2 \overset{2}{\triangleright} u_0$.

In every case, we conclude by the induction hypothesis. □

The previous lemma shows that every equivalence class of indexes corresponds to a sub-apn. The new-rule is the rule that creates a new class of indexes starting the marking of a new sub-apn from an axiom. The new-rule applies when the sequential unification marks, with the last index i_l in σ, one of the premises u_1 of a $\overset{2}{\triangleright}$-link whose other premise u_2 has not been marked yet. If $[i_l, h[$ is the equivalence class of i_l, the new-rule finds the premise of an axiom above u_2 and starts from such an axiom a new marking with the index $i_{l+1} = h$. It is readily seen that in this case there is a switch with a path that connects a conclusion of G_{l+1} to a conclusion of G_l (the vertex u_1) through the tensor $u_1, u_2 \overset{2}{\triangleright} u_0$. This implies also that, in an apn, the classes of i_l and i_{l+1} must be merged by a unify-rule whose $\overset{2}{\triangleright}$-link is $u_1, u_2 \overset{2}{\triangleright} u_0$. Moreover, in an apn, if marking with i_l we reach the premise u_1 of a binary link whose other premise u_2 has been already marked, we must have $\mu(u_2) = i_{l-1}$, otherwise we would have a switch with two distinct paths between the two correct sub-apn G_{l-1} and G_l. By the same reasoning, in an apn we also have to exclude the case in which u_2 has not been marked but there is $v \succ u_2$ that has been already marked. The next lemma formally states and proves the above facts.

Lemma 36. *Let $v', v'' \overset{2}{\triangleright} v \in G$.*

A. *If $v' \in \gamma_n$, then $n < l$ and*
   (a) *v' is the only premise of a binary link in γ_n;*
   (b) *$v'' \preccurlyeq w$ for some $w \in \overline{V}_{n+1}$;*
   (c) *if G is an apn, for every $w \succcurlyeq v''$ either $w \notin \overline{V}$ or $w \in \overline{V}_m$ for some $m > n$.*
   Therefore, γ_l does not contain premises of binary links and, for $n = 1, \ldots, l - 1$, there is a unique sequence of binary links $v'_n, v''_n \overset{2}{\triangleright} v_n \in G$ s.t. $v'_n \in \gamma_n$ and a sequence of vertices $w_{n+1} \succcurlyeq v''_n$ s.t. $w_{n+1} \in \overline{V}_{n+1}$.

B. *If $v' \in \gamma_l$ and G is an apn, then either $v'' \in \gamma_{l-1}$ or $w \notin \overline{V}$ for every $w \succcurlyeq v''$.*

Proof. Let us start by proving that A implies B. That is, let us assume that A and that $v' \in \rho_l$; we shall prove that, if $w \in \overline{V}$ for some $w \succcurlyeq v''$ and $v'' \notin \gamma_{l-1}$, then G is not an apn. Let us take the maximum $m \leq l-1$ for which there exists $w \succcurlyeq v''$ s.t. $w \in \overline{V}_m$; w.l.o.g., let $w = w_m \in \gamma_m \cup \rho_m$. For $k = m, \ldots, l-1$, let $v'_k, v''_k \overset{2}{\triangleright} v_k$ be the unique binary link s.t. $v'_k \in \gamma_k$ and let $w_{k+1} \in \gamma_{k+1}$ be any vertex s.t. $w_{k+1} \succcurlyeq v''_k$ (the link and the vertex exist by the induction hypothesis). For the sake of the exposition, let $v''_{m-1} = v''$ and $v'_l = v'$. For $k = m, \ldots, l$, let $v''_{k-1} \frown \phi_k \frown w_k$ be the ascending path from v''_{k-1} to w_k (that is, ϕ_k is a path in some switch of G and $v_{k-1} \prec w \prec w_k$ for every $w \in \phi_k$) and let $w_k \frown \overline{\phi}_k \frown v'_k$ be a path in some switch of G_k (this path exists by Lemma 35). Let $\psi_k = \phi_k \frown w_k \frown \overline{\phi}_k$. It is readily seen that $v''_{k-1} \frown \psi_k \frown v'_k$ is a path in some switch of G. Now, let us take the composition of the paths ψ_k; that is, let $\psi = \psi_m \frown v'_m \frown v_m \frown v''_m \frown \cdots \frown \psi_l$. If for no $w \in \mathcal{V}(G)$ there exists a pair $h \neq k$ s.t. $w \in \psi_h$ and $w \in \psi_k$, there is a switch of G that contains the cycle $v \frown v''(= v''_{m-1}) \frown \psi \frown v'(= v'_l) \frown v$; that is, G is not an apn. Otherwise, let $w \in \psi_h \cap \psi_k$ with $h < k$. It is readily seen that $w \in \phi_h \cap \phi_k$. By construction, $v''_{k-1} \prec w \prec w_h$. So, as $k - 1 \geq h$, G is not an apn (by the induction hypothesis).

Now, let us prove A by induction on the length of the reduction. In the base case (an init rule only), $l = 1$ and $\rho_1 = \emptyset$. Therefore, let us analyze the other cases under the assumption $v' \in \gamma_n$ for some n.

concl, nop, wait, forward: Trivial.

new: As $\gamma_l = \emptyset$, $n < l = l' + 1$. If $n < l - 1$, we conclude by the induction hypothesis. So, let $n = l - 1$. We see that $\gamma_{l-1} = \gamma'_{l-1}, u_1$ (refer to the rule for the names of the vertices). By the induction hypothesis, γ'_{l-1} does not contain premises of binary links. So, $v' = u_1$ is the only premise of a binary link in γ_{l-1}. Moreover, $v'' = u_2$ and $u_2 \preccurlyeq v_1 \in \overline{V}_l$. As $w \notin \overline{V}'$ for every $w \succcurlyeq u_2$ and $\overline{V} = \overline{V}' \cup \overline{V}_l$ (by the induction hypothesis), we conclude.

unify: By the induction hypothesis, for $n < l = l' - 1$. When $n = l$, let us notice that u_1 (refer to the rule for the names of the vertices) is the only conclusion of a binary link in $\gamma'_l = \gamma, u_1$ and that γ'_{l+1} does not contain conclusions of binary links (by the induction hypothesis). Therefore, as $\gamma_l = \gamma'_{l+1}, \gamma$ (recall that $u_0 \in \rho_l$), we conclude. □

We are now ready to prove that sequential unification is correct and that, if n is the number of vertices of G, it executes at most $n + 2$ steps: one for each vertex marked, plus one init rule and one true/false rule.

Proposition 37. *Let $n = |\mathcal{V}(G)|$. Either $\Downarrow^G$ true or $\Downarrow^G$ false in at most $n + 2$ reduction steps. Moreover, $\Downarrow^G$ true after $n + 2$ reduction steps iff G is an apn.*

Proof. By Lemmas 33 and 34, when the last rule is not an init, $|V| = |V'| + 1 \leq n$. Therefore, the longest reduction that we can have starts with an init and ends with a true/false after n applications of the other rules. Moreover, as $(\mu :: \sigma :: W :: \rho) \xrightarrow{G}_s$ true only if $V = \mathcal{V}(G)$, every reduction ending with true requires exactly $n + 2$ steps.

In order to prove that true and false are the only normal forms, let us verify that for every $(\mu :: \sigma :: W :: \rho)$ there is a rule that we can apply. If $\rho_l = \emptyset$, we can apply true when $V = \mathcal{V}(G)$ or false_1 otherwise. Therefore, let us assume $\rho_l = \rho, u$. If u is a conclusion of G or the premise of a dummy link, we can apply concl. The rules nop, wait and forward cover every case in which u is premise of a unary link. We leave the case in which $u, u' \overset{2}{\triangleright} v \in G$. If $u' \in V$, by Lemma 36, $u' \in V_{l-1}$; so, we can apply unify. Therefore, let $\mu(u') = \bot$. When $w \in V$ for some $w \succ u'$, we can apply false_2. So, let us assume $\mu(w) = \bot$ for every $w \succ u'$. It is readily seen that either $u' \succ u'$ or there is $\overset{0}{\triangleright} v_1, v_2$ with $v_1 \succcurlyeq u'$. Now, if $\mu(v_2) = \bot$ and $v_2 \neq u_1$, we can apply new; otherwise, we can apply false_2.

Let $(\mu :: \sigma :: W :: \rho \xrightarrow{G}_u \text{true})$. By hypothesis, $l = 1$, $\rho = \rho_1 = \emptyset$ and $V = \mathcal{V}(G)$. Let $G \vdash \alpha$. It is readily seen that $G = G_1$ and $\alpha = \gamma_1$ (by $V = \mathcal{V}(G)$). So, by Lemma 35, $G \to_p \overset{0}{\triangleright} \alpha$; that is, G is an apn (by Lemma 21).

In order to conclude, we have to prove that $\Downarrow^G$ false only if G is not an apn. Let $(\mu :: \sigma :: W :: \rho) \xrightarrow{G}_u$ false. We distinguish the cases of false rule.

1. $\rho_l = \emptyset$ and $\mu(v) = \bot$ for some v. By Lemma 36, γ_l does not contain premises of binary links. By Lemma 34, for every $u' \in \gamma_l$ with $u', u'' \overset{1}{\triangleright} u \in G$, $u'' \notin V_l$ (in fact, $u'' \in V_n$ would imply $u \in \rho_l$; but, as $\rho_l = \emptyset$, this is not the case). Therefore, for every $u' \in \gamma_l$ s.t. $u', u'' \overset{1}{\triangleright} u \in G$, let us take a switch with $u \frown u''$; it is readily seen that there is no connection between v and any $w \in V_l$.
2. There is $u_1, u_2 \overset{2}{\triangleright} u_0 \in G$ with $u_1 \in \rho_l$ and $\mu(u_2) = \bot$, and one of the two cases of the side condition holds. When $u_2 \prec u_2$, G cannot be DR-correct (by Lemma 10). In the other case, by item C of Lemma 36, G is not an apn. □

8. The cost of sequential unification

Sequential unification allows one to efficiently implement the data structures for the waiting and ready vertices (see Section 6.2), which we denoted W and R in the quadruples of sequential unification.

- R is a stack of sets of vertices. The only operations that we have to perform on R are: to get a vertex from or to put a vertex into the set on the top of the stack; to insert a new set with the conclusions of an axiom; to merge the two sets on the top of the stack with a set of waiting vertex obtained from W.

- W is an array of sets. The set $W[i]$ is defined only if i is an index in σ (the minimal element in some class of indexes). The only operations that we have to perform on a set in W is the insertion of an element. All the waiting vertices in a set $W[j]$ become ready at the same time, when j becomes the last index in σ. This happens when i is the second-top index in σ and a unify rule removes the top index j. This operation forces the moving of the set $W[i]$ from W and its merging with the two top sets in R.

Summing up, since we do not need to find elements in the sets in W or R – almost all the rules analyze and mark a ready vertex from R, but such a vertex can be any one, and not a particular one, in the top set of R – we can use any standard data structure for sets that implements insertion, deletion and union in constant time.

For the partition σ instead, we have to use a disjoint set union-find implementation (see Section 6.3). In this way, since any sequential unification terminates in at most $n + 2$ steps, where $n = |\mathcal{V}(G)|$, and since, apart for the union-find operations, the cost of the application of a rule of sequential unification is constant, we get a pseudo-linear algorithm for sequential unification, namely an algorithm with a cost $\mathcal{O}(n\,\alpha(n, m))$, for some $m < n$, where α is the very slow increasing function described in Section 6.3.

Such pseudo-linear α-algorithms are practically linear (see Remark 29) and, since the constants in the upper-bound for a union-find α-algorithm are small, a pseudo-linear implementation using an α-algorithm is frequently preferred to a linear solution whose upper-bound requires bigger constants (e.g., this is the case for term unification).

However, there is a special case – and the disjoint-set union-find used by the sequential unification is an instance of it– where the amortized cost of union-find becomes linear without any particular increasing of the upper bound constants [4]. The linear algorithm for that special case uses an α-algorithm, but, exploiting the order in which sets are united, optimizes the size of the problem on which to apply the α-algorithm. Let us see how that technique applies in the case of sequential unification.

8.1. A special case of union-find

The natural data structure for the implementation of $\sigma = i_0 : \cdots : i_n : \cdots : i_l$ is an array of bits b s.t. $b[i] = 0$, for $i = i_0, \ldots, i_l$, and $b[i] = 1$, otherwise. In this way, setting $b[i_n]$ to 1 unites the intervals $[i_{n-1}, i_n[$ and $[i_n, i_{n+1}[$, while $\sigma(i)$ is the greatest $j \leq i$ s.t. $b[j] = 0$. In order to optimize the use of memory registers, we represent b by means of an array B of words of length w and define $b[i] = \mathsf{bit}(i \bmod w, B[i \div w])$,[2] where $\mathsf{bit}(n, x)$ is the nth bit of x (start counting bits from 0). Computing $\sigma(i)$ decomposes in the following steps: (i) check if the bit of $\sigma(i)$ is in the word $i \div w$, e.g., by means of the function $\mathsf{firstz}(n, x)$ that returns the greatest $m \leq n$ s.t. $\mathsf{bit}(m, x) = 0$, if any, and -1 otherwise; (ii) if $\mathsf{firstz}(i \bmod w, i \div w) = -1$, find the greatest $n < i \div w$ s.t. $B[n]$ contains a bit equal to 0; (iii) return $\mathsf{firstz}(i \bmod w, B[i \div w])$, if this is not equal to -1, or $\mathsf{firstz}(w - 1, B[n])$, otherwise.

Step (ii) is the critical operation. Before analyzing it, let us remark that $\mathsf{firstz}(n, x)$ is $\mathcal{O}(1)$. For instance, let us assume that the processor computes $\mathsf{bor}(x, y)$ (the bitwise-or of the words x and y) in $\mathcal{O}(1)$ time[3]; let $x = \mathsf{mask}(i)$ be the word s.t. $\mathsf{bit}(j, x) = 0$, for $j \leq i$, and $\mathsf{bit}(j, x) = 1$, for $j > i$; let $\mathsf{lastz}(x)$ be equal to the greatest j s.t. $\mathsf{bit}(j, x) = 0$, if any, and equal to -1, otherwise. Then, $\mathsf{firstz}(i, x) = \mathsf{lastz}(\mathsf{bor}(\mathsf{mask}(i), x))$. Now, mask and lastz can be implemented by means of two tables of length $\mathcal{O}(w)$ and $\mathcal{O}(2^w)$, respectively. That is, at the cost of a $\mathcal{O}(2^w)$ initialization, $\mathsf{firstz}(n, x)$ is $\mathcal{O}(1)$.

The efficient algorithm for $\sigma(i)$ uses a disjoint-set data structure for the implementation of step (ii). In fact, let $j_0 < \cdots < j_n < \cdots < j_k$ be the sequence of indexes s.t.: if $j = j_n$, then $B[j]$ contains at least a 0; otherwise, $B[j]$ does not contain any 0 (by hypothesis, $j_0 = 0$, for in any case $b[0] = 0$). If $j_{k+1} = L$ is the length of B, then B splits into the family of disjoint intervals $[j_0, j_1[, [j_1, j_2[, \ldots, [j_k, j_{k+1}[$, and step (ii) is a FindSet of the corresponding disjoint-set problem. Hence, using an α-algorithm, the cost of n find/set-bit operations on b is $\mathcal{O}(n\ \alpha(n + L, L))$, plus $\mathcal{O}(2^w)$ for the initialization of $\mathsf{firstz}(n, x)$. Then, if N is the length of b and $w = \Omega(\log\log N)$ (but even smaller values of w suffice [16]), $\alpha(n + L, L) = \alpha(n + \mathcal{O}(N/\log\log N), \mathcal{O}(N/\log\log N)) = \mathcal{O}(1)$; that is, n union-find operations on b cost $\mathsf{UF}(n, N) = \mathcal{O}(n) + \mathcal{O}(2^w)$.

8.2. Sequential unification is linear time

Any sequential unification of G requires at most $|\mathcal{V}(G)|$ steps—no vertex can be inserted twice into R and, when G is an apn, every vertex transits into R; moreover, as NextAxiom does not visit the same vertex twice, the amortized cost of its calls is $\mathcal{O}(size(G))$. The array b has a bit for each axiom, i.e. $N = \mathcal{O}(size(G))$. The number of find-bit operations is bound by the number of unary links (wait is the only rule that requires a find-bit); the number of set-bit operations is bound by the number of binary links. Therefore, sequential unification costs $\mathcal{O}(size(G)) + \mathsf{UF}(2\ size(G), size(G))$. Moreover, when $w \leq \log size(G)$ and $w = \Omega(\log\log size(G))$,[4] the cost simplifies to $\mathcal{O}(size(G)) + \mathcal{O}(size(G)) + \mathcal{O}(size(G))$.

Theorem 38. *The cost of any sequential unification of an aps G is $\mathcal{O}(size(G))$.*

---

[2] Where $i \div w$ is the integer quotient of i divided by w and $i \bmod w$ is its remainder.

[3] If n is the size of the problem, one of the basic hypotheses of the RAM model is that its word length is $\mathcal{O}(\log n)$. Under that hypothesis, the arithmetic and bitwise operations on words have cost $\mathcal{O}(1)$.

[4] That assumptions on w are compatible with the basic hypothesis $w = \mathcal{O}(size(G))$, see footnote 3.

9. Conclusions

Let us remark that sequential unification is not just a theoretical trick for finding a theoretically linear time algorithm for correctness practically superseded by non-optimal but more effective algorithms. In fact, all the steps leading to the linear algorithm correspond to natural optimizations that do not increase the constants in the upper-bound.

Apart for the result on the complexity of correctness, the unification criterion is interesting from a semantical point of view. The operations that unification performs at each $\parr/\otimes$-link correspond to synchronization directives: a $\parr$-link asks for the synchronization of its premise; a $\otimes$-link (or cut) notifies that its premises have synchronized. This issue has not been investigated yet, but the interpretation of that directives in some kind of process algebra might lead to some kind of interpretation of proof nets in terms of concurrent processes.

Acknowledgements

The author was partially supported by the Italian P.R.I.N. grant CONCERTO and by the grant "Applicazione di Strumenti Logici alla Progettazione e Analisi di Strumenti Software" from Sapienza Università di Roma, Facoltà di Scienze M.F.N.

References

[1] T.H. Cormen, C.H. Leiserson, R.L. Rivest, Introduction to Algorithms, in: The MIT Electrical Engineering and Computer Science Series, The MIT Press, 1989.
[2] V. Danos, Une Application de la Logique Linéaire à l'Ètude des Processus de Normalisation (principalement du λ-calcul), Ph.D. Thesis, Université Paris 7, June 1990.
[3] V. Danos, L. Regnier, The structure of multiplicatives, Archive for Mathematical Logic 28 (1989) 181–203.
[4] H.N. Gabow, R.E. Tarjan, A linear-time algorithm for a special case of disjoint set union, Journal of Computer and System Sciences 30 (2) (1985) 209–221.
[5] J.-Y. Girard, Linear logic, Theoretical Computer Science 50 (1) (1987) 1–102.
[6] S. Guerrini, Correctness of multiplicative proof nets is linear, in: 14th Annual IEEE Symposium on Logic in Computer Science, LICS '99, IEEE Computer Society, Trento, Italy, 1999, pp. 454–463.
[7] S. Guerrini, A. Masini, Parsing MELL Proof Nets, Theoretical Computer Science 254 (1–2) (2001) 317–335.
[8] P. Jacobé de Naurois, V. Mogbil, Correctness of multiplicative (and exponential) proof structures is NL-complete, in: J. Duparc, T.A. Henziger (Eds.), Computer Science Logic, CSL 2007, in: Lecture Notes in Computer Science, vol. 4646, Springer, 2007, pp. 435–450.
[9] Y. Lafont, From proof nets to interaction nets, in: J.-Y. Girard, Y. Lafont, L. Regnier (Eds.), Advances in Linear Logic (Proceedings of the Workshop on Linear Logic, Ithaca, New York, June 1993), in: London Mathematical Society Lecture Note Series, vol. 222, Cambridge University Press, 1995, pp. 225–247.
[10] F. Lamarche, Proof nets for intuitionistic linear logic I: essential nets. Preliminary Report, April 1994.
[11] F. Lamarche, From proof nets to games, in: A.S. Jean-Yves Girard, Mitsuhiro Okada (Eds.), Linear Logic 96 Tokyo Meeting, in: Electronic Notes in Theoretical Computer Science, vol. 3, Elsevier, 1996, pp. 107–119.
[12] A. Martelli, U. Montanari, An efficient unification algorithm, ACM Transactions on Programming Languages and Systems 4 (3) (1982) 258–282.
[13] A.S. Murawski, C.-H.L. Ong, Dominator trees and fast verification of proof nets, in: LICS'00: Proceedings of the 15th Annual IEEE Symposium on Logic in Computer Science, IEEE Computer Society, 2000, pp. 181–191.
[14] A.S. Murawski, C.-H.L. Ong, Fast verification of MLL proof nets via IMLL, ACM Transactions on Computational Logic 7 (3) (2006) 473–498.
[15] M. Nagayama, M. Okada, A new correctness criterion for the proof-nets of non-commutative multiplicative linear logic, The Journal of Symbolic Logic 66 (4) (2001) 1524–1542.
[16] R.E. Tarjan, Efficiency of a good but not linear set union algorithm, Journal of the ACM 22 (2) (1975) 215–225.
[17] H. Vollmer, Introduction to Circuit Complexity: A Uniform Approach, Springer-Verlag, 1999.