REPORTS ON MATHEMATICAL LOGIC
44 (2009), 167–200

Stefano CAVAGNETTO

STRING REWRITING AND PROOF COMPLEXITY: AN INTERPRETATION OF RESOLUTION

A b s t r a c t. We interpret the well-known propositional proof system Resolution using string rewriting systems Σ_n^* and Σ_n corresponding to tree-like proofs[a] and sequence-like proofs, respectively. We give a representation of Σ_n^* using planar diagrams.

1. Introduction

Rewriting is a technique for defining complex objects by successively replacing parts of a simple initial object using a set of rewriting rules. The object can be a finite string of characters or more complicated, like for example a polygon.[1] The most extensively studied and best understood

Received 5 June 2007
Supported by Grants #A1019401, AVOZ10190503.

[a]The current notation in the literature for the proof system tree-like resolution is R^*; so that we denote its corresponding rewriting interpretation as Σ_n^*.

[1]For instance the continuos snowflakes curve not differentiable anywhere proposed by von Koch in [16], can be defined using this approach.

rewriting systems deal with character strings. The main reason for that is Chomsky's work on formal grammars, in late 1950s, in which he applied the concept of rewriting in order to describe the syntactic features of natural languages, [8]. Thus a string rewriting system can be interpreted as a device for generating and recognizing formal languages; sometimes in the literature they are also called combinatorial systems, [11].

In this rewriting context we are considering transformations of some object by step by step activity. Given a finite alphabet and a definition of word over it, we allow a set of rewriting rules in order to transform words from the set of all words. The sequence of application of rules can be seen as a proof in the classical sense. We show how this idea can be implemented and used to interpret the well known propositional calculus Resolution. We give a characterization of the rewriting system which represents tree-like resolution using diagrams in van Kampen style [6]; the dag-like case is less natural, but by allowing more complex rules it can be interpreted in the same manner. Formerly van Kampen diagrams have been discussed by Krajíček, in [20]; he gave a link between the Dehn function of finitely presented groups and the length of proofs function in propositional proof complexity. On the one hand a motivation for this work is the hope to get a new perspective on proof systems using a different approach and exploiting also geometric interpretations to charcterise several proof complexity measures. On the other hand, another motivation for studying proof systems in terms of rewriting systems is to obtain, using also a diagrammatic interpretation, some intuitions for proof search heuristic.

The paper is organized as follows: section 2 deals with Resolution and we recall some basics about it. More can be found in [18], [19] and [24]. In section 3 we introduce string rewriting systems, known also as semi-Thue systems.[2] In section 4 we interpret tree-like resolution as semi-Thue system. We show that the new rewriting system is complete and sound with respect to Resolution, see Theorem 4.3 and 4.5 respectively. In this section we also discuss more in detail the motivation for considering rewriting systems in the context of proof complexity. Then section 5 introduces a representation by planar diagrams of the proofs in the semi-Thue system interpreting tree-like resolution. Section 6 is devoted to the dag-like case: we interpret dag-

[2] Axel Thue (1863-1922) introduced the first systematic treatement of string rewriting systems in the early 20th century, see [21], [22].

like resolution as string rewriting system. Section 7 has some concluding remarks.

2. Resolution

The logical calculus Resolution R is a refutation system for formulas in conjunctive normal form, [4]. A literal ℓ is either a variable p or its negation $\bar{p}$. The basic object is a clause, that is a finite or empty set of literals, $C = \{\ell_1, \ldots, \ell_n\}$ and is interpreted as the disjunction $\bigvee_{i=1}^{n} \ell_i$. A truth assignment $\alpha : \{p_1, p_2, \ldots\} \to \{0, 1\}$ satisfies a clause C if and only if it satisfies at least one literal l_i in C. It follows that no assignment satisfies the empty clause, which we denote by $\{\}$. A formula ϕ in conjunctive normal form is written as the collection $\mathcal{C} = \{C_1, \ldots, C_m\}$ of clauses, where each C_i corresponds to a conjunct of ϕ. The only inference rule is the resolution rule, which allows us to derive a new clause $C \cup D$ from two clauses $C \cup \{p\}$ and $D \cup \{\bar{p}\}$

$$\frac{C \cup \{p\} \;\; D \cup \{\bar{p}\}}{C \cup D}$$

where p is a propositional variable. C does not contain p (it may contain $\bar{p}$) and D does not contain $\bar{p}$ (it may contain p). The resolution rule is sound: if a truth assignment $\alpha : \{p_1, p_2, \ldots\} \to \{0, 1\}$ satisfies both upper clauses of the rule then it also satisfies the lower clause.

A resolution refutation of ϕ is a sequence of clauses $\pi = \{D_1, \ldots, D_k\}$ where each D_i is either a clause from ϕ or is inferred from earlier clauses D_u, D_v, $u, v < i$ by the resolution rule and the last clause $D_k = \{\}$. Resolution is sound and complete refutation system; this means that a refutation does exist if and only if the formula ϕ is unsatisfiable.

A resolution refutation $\pi = \{D_1, \ldots, D_k\}$ can be represented as a directed acyclic graph (dag-like) in which the clauses are the vertices, and if two clauses $C \cup \{p\}$ and $D \cup \{\bar{p}\}$ are resolved by the resolution rule, then there exists a direct edge going from each of the two clauses to the resolvent $C \cup D$.

A resolution refutation $\pi = \{D_1, \ldots, D_k\}$ is tree-like if and only if each D_i is used at most once as hypothesis of an inference in the proof. The

underlying graph of π is a tree. The proof system allowing exactly tree-like proofs is called tree-like resolution and denoted by R^*.

In proof complexity, perhaps the most important relation between dag-like refutations and refutations in R^* is that the former can produce exponentially shorter refutations then the latter. A simple remark on this is that in a tree-like proof anything which is nedeed more than once in the refutation must be derived again each time from the initial clauses. A superpolynomial separation between R^* and R was given by Urquhart in [24], and later by others in [9] and [15].

For the benefit of the reader we give an example from [5] of a family of clauses for which R^* suffers an exponential blow-up with respect to R. Let the monotone function GEN_n of n^3 inputs $t_{a,b,c}$, $1 \leq a, b, c \leq n$ be defined as follows: for $c \leq n$, we define the relation $\vdash c$ (c is generated) recursively by $\vdash c$ if and only if $c = 1$ or there are $a, b \leq n$ with $\vdash a$, $\vdash b$ and $t_{a,b,c} = 1$. Finally, $GEN_n(\vec{t}) = 1$ if and only if $\vdash n$.

Raz and McKenzie introduced in [23] a special kind of communication games, called $DART$ games, and a special class of communication protocols for solving them. The communication game $PyrGEN(m, d)$ is a $DART$ game related to GEN_n. It is defined as follows: let $Pyr_d := \{(i, j); 1 \leq j \leq i \leq d\}$. We consider the indices as elements of Pyr_d; then the inputs for the two players I and II are respectively sequences of elements $x_{i,j} \in [m]$ and $y_{i,j} \in \{0, 1\}^m$ such that $(i, j) \in Pyr_d$. We picture these as laid out in a pyramidal form with (i, j) at the top and (d, j), $1 \leq j \leq d$ at the bottom. The goal of the game is to find either an element colored 0 at the top of the pyramid, or an element colored 1 at the bottom of the pyramid, or an element colored 1 with the two elements below colored 0. This means to find indices (i, j) such that one of the following holds: (1) $i = j = 1$ and $y_{1,1}(x_{1,1}) = 0$, or (2) $y_{i,j}(x_{i,j}) = 1$ and $y_{i+1,j}(x_{x+1}, j) = 0$ and $y_{i+1,j+1}(x_{i+1,j+1}) = 0$, or (3) $i = d$ and $y_{d,j}(x_{d,j}) = 1$. Then we define an unsatisfiable set of clauses ψ related to GEN_n. The variables $p_{a,b,c}$ for $a, b, c \in [n]$ represent the input to GEN_n. Variables $q_{i,j,a}$ for (i, j) by a certain mapping $m : Pyr_d \to [n]$, see Corollary 7 in [5]. The variables r_a for $a \in [n]$ represent a coloring of the elements by 0, 1 such that 1 is colored 0, n is colored 1 and the elements colored are closed under generation.

The set $Gen(\vec{p}, \vec{q})$ is given by: $\bigvee_{1 \leq a \leq n} q_{i,j,a}$ for $(i, j) \in Pyr_d$, $\bar{q}_{d,j,a} \vee p_{1,1,a}$, for $1 \leq j \leq d$ and $a \in [n]$, $\bar{q}_{1,1,a} \vee p_{a,a,n}$ for $a \in [n]$, $\bar{q}_{i+1,j,a} \vee \bar{q}_{i+1,j+1,b} \vee$

$\bar{q}_{i,j,c} \vee p_{a,b,c}$ for $(i,j) \in Pyr_{d-1}$ and $a,b,c \in [n]$. The set $Col(\overrightarrow{p}, \overrightarrow{q})$ is given by: $\bar{r}_1$, r_n and $r_a \vee r_b \vee \bar{p}_{a,b,c} \vee \bar{r}_c$ for $a,b,c \in [n]$.

The set of clauses $\psi = Gen(\overrightarrow{p}, \overrightarrow{q}) \cup Col(\overrightarrow{p}, \overrightarrow{q})$ exponentially separates R^* from R, see [5]. For an improvement of the gap the reader can see [2].

3. The String rewriting system Σ_n^*

In general, a string rewriting system is a substitution system used to transform a given string according to specified rewriting rules. A semi-Thue system Σ is a string rewriting system. Throughout this paper, "semi-Thue system" and "string rewriting system" are used meaning the same mathematical concept. It is a tuple (A, Δ) where A is a finite alphabet and Δ is a set of ordered pairs $\Delta \subseteq A^* \times A^*$, where A^* is the set of all words over A. The elements of Δ, (q, z), are referred as string rewriting rules and denoted by $q \to z$. If a semi-Thue system Σ is symmetric, $\Delta = \Delta^{-1}$, then Σ is called a Thue system.

Definition 3.1 (Cook and Reckhow, see [10])**.** A proof system for a language $L \subseteq A^*$ is a binary relation $P \subseteq A^* \times A^*$ (a word x such that $P(x,y)$ is called a P-proof of $y \in L$) satisfying the following conditions:

1. Completeness: If $y \in L$ then $y \in L$ has a P-proof;
2. Soundness: If $y \in L$ has a P-proof then $y \in L$;
3. Polynomial verifiability: There exists a polynomial-time algorithm in $|x| + |y|$ deciding on inputs (x,y) whether the relation $P(x,y)$ holds.

In order to obtain the semi-Thue system, that we call Σ_n^*, we define its alphabet A_n.

Definition 3.2. Let A_n be a finite alphabet containing two uppercase letters L, R, and two types of lowercase letters x_i and $\bar{x}_i$, for $i = 1, \ldots, n$. A word w over A_n is a finite string consisting of zero or more elements of A_n. The set of all words over A_n is denoted by A_n^*. The empty word is denoted by Λ.

In Σ_n^* a special kind of words over A_n, called **regular-words**, will play an important role. By $u \subseteq w$ we mean that u is a subword of w.

Definition 3.3. A word $w \in A_n^*$ is regular if and only if $w = \Lambda$ or it has the following form:

$$w = Lu_1RLu_2R \dots Lu_rR$$

where each $u_i \in (A_n \setminus \{L, R\})^*$, for $1 \leq i \leq r$. Any of u_i can be empty. A word w_i that is regular and $w_i \subseteq w_j$ is called a regular subword of w_j.

Of course, by the definition given above the following strings are regular words: LR, $Lx_1\bar{x_3}R$, and $Lx_1RL\bar{x_2}R$; by definition Lx_1R is a regular subword of $Lx_1RL\bar{x_2}R$, and $Lx_1\bar{x_3}R$ is a regular subword of $Lx_1\bar{x_3}R$. It is implicit in the definition above that L and R must alternate, i.e. we cannot have two consecutive Ls or Rs in the word if the word is regular. Moreover LR is also a regular subword of LR.

The size of a word w is the number of symbols contained in it[3]; excluding Λ which has size 0, the smallest size of a regular word is 2, namely is the first word given in the example above, LR. Then we define a special type of words which is minimal with respect to that of regular word. They are called **clause-words**.

Definition 3.4. A clause-word is a regular word of A_n^* that starts with L and ends with R and it does not include any other occurrences of L and R in it.

Thus given $Lx_1\bar{x_2}RLx_3\bar{x_4}\bar{x_5}RLx_5x_7R$, the words $Lx_1\bar{x_2}R$, $Lx_3\bar{x_4}\bar{x_5}R$ and Lx_5x_7R are clause-words. It follows easily that all clause-words are regular words; the viceversa does not hold. We distinguish between these two notions when needed, otherwise we will use the general notion of regular word introduced before. Then the semi-Thue system Σ_n^* can be defined as follows:

Definition 3.5. Let Σ_n^* be a semi-Thue system in the alphabet A_n with rewriting rules, Σ_n^*-rules:

1. **Elimination rules**:
   (a) $x_iRL\bar{x_i} \to \Lambda$;

[3]Note that the L,R letters in the regular words are just delimiters, but they contribute to the size.

(b) $\bar{x_i}RLx_i \to \Lambda$;

for any i.

2. **Exchanging rules**:

(a) $x_ix_j \to x_jx_i$;

(b) $\bar{x_i}\bar{x_j} \to \bar{x_j}\bar{x_i}$;

(c) $x_i\bar{x_j} \to \bar{x_j}x_i$;

(d) $\bar{x_i}x_j \to x_j\bar{x_i}$;

for any i, j.

Some comments on the rewriting rules that we have choosen. Rules (1a) and (1b) are string rewriting rules simulating the resolution rule given in Section 2. The rules (2a)-(2d) can be useful because sometime we have regular words in which rules (1a), (1b) cannot be directly applied and we need to permute lower case letters first. Then by the exchanging rules we can move lower case letters in order to obtain suitable strings such that the elimination rules can be applied. Notice that from the previous definition follows easily that for any given finite set of lowercase characters $x_1 \dots x_n$ determining A_n we obtain a specific semi-Thue system Σ_n^*.

To be able to define properly the notion of proof in Σ_n^* we introduce a special set of regular words called **initial-words**. We denoted this set by $\mathcal{I}$. Moreover, we introduce in the definition below the rules (i) and (ii) that allow us to manipulate the elements of $\mathcal{I}$. These rules are called **introduction-rules** ($\mathcal{I}$-rules).

Definition 3.6. Let $\mathcal{I} = \{w_1, \dots, w_t\}$ be a non empty set of regular words. $\mathcal{I}$-rules are:
(i) $L \to uL$, where $u \in \mathcal{I}$;
(ii)$R \to Ru$, where $u \in \mathcal{I}$.

The notion of Σ_n^*-proof is defined as follows

Definition 3.7. A proof π' in Σ_n^* of the regular word w from a non empty set of regular words $\mathcal{I}$, denoted by

$$\pi' : \mathcal{I} \vdash_{\Sigma_n^*} w$$

is a finite sequence of regular words $w_1, \ldots, w_k$ such that $w_1 \in \mathcal{I}$, $w_k = w$ and each w_i, for $1 < i \leq k$, w_i is obtained from w_{i-1} by an $\mathcal{I}$-rule or a Σ_n^*-rule. w_1 is called the **source-word** of π'.

Example. We conclude this section giving an example of proof of LR in Σ_n^*. Let $\mathcal{I} = \{Lx_1x_2R,\ Lx\bar{_1}R,\ Lx\bar{_2}R\}$ be a set of initial words. Let Lx_1x_2R be the source-word. The words $Lx\bar{_1}R$, $Lx\bar{_2}R$ are entered using $\mathcal{I}$-rules (i) and (ii):

By the introduction rule (ii), we have

$$Lx_1x_2R \to Lx_1x_2RLx\bar{_2}R$$

then by rule (1a):

$$Lx_1x_2RLx\bar{_2}R \to Lx_1R$$

by rule (ii) (or we can use also (i)):

$$Lx_1R \to Lx_1RLx\bar{_1}R$$

Finally, by rule (1a) if we have used in the previous step the rule (ii) (otherwise, if rule (i) has been applied we use (1b)), we obtain:

$$Lx_1RLx\bar{_1}R \to LR.$$

4. Σ_n^* and R^*: the tree-like case

In order to interpret R^* as the semi-Thue system Σ_n^* we need to set some correspondences.

Definition 4.1. If $C = \{\ell_1, \ldots, \ell_l\}$ is a clause, then $w_C = L\ell_1 \ldots \ell_l R$ is a clause-word in the language of Σ_n^*. For definiteness we assume the ordering on variables given by their indices. If $\mathcal{C} = \{C_1, \ldots, C_t\}$ is a set of clauses, then $w_{\mathcal{C}} = w_{C_1} \ldots w_{C_t}$ is a regular word in the language of Σ_n^*. We assume clauses are canonically ordered in a fixed way.

Using Definition 4.1 we take a clause C_i in the language of Resolution and we obtain a regular word w_{C_i} in the language of Σ_n^*. For example, take the clause $\{x_1x_2\}$; by definition we have the regular word Lx_1x_2R. Notice

that by our definition a set of clauses $\mathcal{C}$ corresponds to a regular word $w_{\mathcal{C}}$. In fact, let $\mathcal{C} = \{C_1, C_2\}$=$\{\{x_1\}, \{x_2x_3x_6\}\}$ be a set of clauses. Then by the definition above $w_{\mathcal{C}} = Lx_1RLx_2x_3x_6R$.

The clause-words defined in the Definition 3.4 give the correspondence with the basic object of the resolution calculus, clauses. Indeed, if w_i is a clause-word then it has the form $L\ell_1 \dots \ell_n R$ that corresponds to a clause $\{\ell_1, \dots, \ell_n\}$ where ℓ_i are literals (with $1 \leq i \leq n$). Notice that while the ordering of variables in the language of Resolution is not important, in the case of the language of Σ_n^* this has some relevance during the manipulation.

Definition 4.2. Let $\mathcal{C} = \{C_1, \dots, C_t\}$ be a set of clauses. Then

$$\mathcal{I}_{\mathcal{C}} = \{w_{C_1}, \dots, w_{C_t}\}$$

where w_{C_1},..., w_{C_t} are clause-words corresponding to the clauses C_1,..., C_t.

Thus a resolution refutation π starting from a set of clauses $\mathcal{C}$ and ending with $\{\}$ in R^* can be conceivably interpreted as a Σ_n^*-proof of LR from $\mathcal{I}_{\mathcal{C}}$. This is what we do next. Notice that the string rewriting system allowing $\mathcal{I}$-rules gives the opportunity to use words from $\mathcal{I}_{\mathcal{C}}$ when needed.

Remark. In classical Resolution we can always back-out from a dead-end. In this new approach the situation is analogous, in fact using $\mathcal{I}$-rules we can always reintroduce an element from $\mathcal{I}_{\mathcal{C}}$, the set of initial clause-words corresponding to the set of initial clauses, a clause-word many times anywhere in the proof.

Theorem 4.3. [4] *Let π be a resolution refutation in R^* of a set of clauses $\mathcal{C}$ in variables x_1,..., x_n. Assume that π has k clauses. Then there exists a Σ_n^*-proof π' of LR from $\mathcal{I}_{\mathcal{C}}$ such that the number of steps k' in π' satisfies:*

$$k' < 2(kn) \ .$$

Proof. Let a tree-like refutation π of $\mathcal{C}$ be fixed. For a clause D in π let $k(D)$ denotes the number of clauses in the subproof of π ending with

[4] Theorem 4.11 about width gives a more effective version of the simulation estimating also the sizes of the lines of the rewriting proof.

D[5]. Then $k(D) = 1$ for initial clauses and $k(\{\}) = k$ for the end clause $\{\}$. If

$$\frac{D_1 \cup \{\ell\}\ D_2 \cup \{\bar{\ell}\}}{D_1 \cup D_2}$$

is a inference in π then, as π is tree-like,

$$k(D_1 \cup D_2) = k(D_1 \cup \{\ell\}) + k(D_2 \cup \{\bar{\ell}\}) + 1.$$

By induction on $k(D)$ we show that for any clause $D \in \pi$ there is π'_D, a derivation in Σ^*_n of w_D from $\mathcal{I}_\mathcal{C}$, such that the number of steps k'_D of π'_D satisfies:

$$k'_D < 2(k(D)n).$$

Taking for D the end-clause of π gives the theorem.

Basis Case: D is an initial clause in π. Then w_D is derived from $\mathcal{I}$ in one step using the $\mathcal{I}$-rules; w_D is the source-word of the derivation π'_D consisting of one step ($k'_D = 1$).

Induction Step: Assume D is in π derived from D_1 (containing ℓ) and D_2 (containing $\bar{\ell}$) by resolving ℓ. By induction assumption applied to D_1 there are a derivation π'_{D_1} of w_{D_1} and a derivation π'_{D_2} of w_{D_2} in Σ^*_n from $\mathcal{I}$ with

$$k'_{D_1} < 2(k(D_1)n)$$

and

$$k'_{D_2} < 2(k(D_2)n)$$

steps respectively. Construct a derivation π'_D in Σ^*_n from $\mathcal{I}$ as follows:

1. Initial part of π'_D is π'_{D_1}.
2. Then continue with derivation carrying w_{D_1} as the left-most clause-word of every step. This subderivation uses the same inferences as π'_{D_2} except for introducing the first line: instead of using a clause C

[5] $k(D)$ is the size of the tree rooted at D.

for a source-word w_C as in π'_{D_2}, use $\mathcal{I}$-rule (ii) to infer from w_{D_1} the word $w_{D_1}w_C$.

3. After the steps (1) and (2) we have a derivation in Σ_n^* of $w_{D_1}w_{D_2}$. Now, use exchanging rules to move ℓ in w_{D_1} towards R and $\bar{\ell}$ in w_{D_2} towards L such that
$$\ell RL\bar{\ell}$$
becomes a subword. This proocess needs at most $2(n-1)$ application of exchanging rules.

4. Finally apply the elimination rules to delete the subword $\ell RL\bar{\ell}$, getting w_D.

The number of steps k'_D in this derivation is bounded above by:

$$k'_D \leq k'_{D_1} + k'_{D_2} + 2(n-1) + 1$$

As $k'_{D_1} < 2k(D_1)n$, $k'_{D_2} < 2k(D_2)n$ and $k(D) = 1 + k(D_1) + k(D_2)$, we also have $k' < 2k(D)n$. □

Now, we want to establish that the simulation of R^* by Σ_n^* is sound, namely that any derivation of LR from $\mathcal{I}_\mathcal{C}$ in Σ_n^* can be transformed into a refutation of $\mathcal{C}$ in R^*. A small obvious technical lemma is the following:

Lemma 4.4. *Let $\mathcal{C}$ be a set of clauses and $\mathcal{I}_\mathcal{C}$ the corresponding set of clause-words. Then any derivation in Σ_n^* from $\mathcal{I}_\mathcal{C}$ contains only regular words.*

Theorem 4.5. *Let $\mathcal{C}$ be a set of clauses and $\mathcal{I}_\mathcal{C} = \{w_C : C \in \mathcal{C}\}$ be the corresponding set of clause-words. Assume that there is a derivation π' in Σ_n^* of LR from $\mathcal{I}_\mathcal{C}$. Then the set of clauses $\mathcal{C}$ is refutable in R^*. In fact, if π' has k' steps then there is a refutation π in R^* of $\mathcal{C}$ with $k \leq k'$ steps.*

Proof. Let π' be $w_1, \ldots, w_{k'}$. By induction on i, we prove that if $w_i = w_{D_1} \ldots w_{D_t}$, with D_j clauses (we know by Lemma 4.4 that w_i is a regular word), then there are derivations π_j, $j = 1, \ldots, t$, in R^* of D_j from $\mathcal{C}$ with $k(\pi_j)$ clauses each such that

$$\sum_{j=1}^{t} k(\pi_j) \leq i$$

For $i = k'$ this gives the theorem.

Basis Case: By definition the source-word w_1 is w_C for some $C \in \mathcal{C}$. Thus the claim holds for $i = 1$.

Induction Step: If w_{i+1} has been obtained by other than the elimination rules or the $\mathcal{I}$-rules it corresponds to the same set of clauses of w_i and there is nothing to prove. The elimination rule is simulated by the resolution rule and the $\mathcal{I}$-rules are initial clauses rule of R^*. □

We can describe a direct way how to extract the derivation in R^* from the derivation in Σ_n^*, using the following procedure. The procedure has a derivation in Σ_n^* from $\mathcal{I}_\mathcal{C}$ as input and marks by all occurrences of a clause-word that:

1. is not the source-word;
2. it was not derived by an $\mathcal{I}$-rule;
3. it was not derived by an elimination rule.

Then delete all clause-words marked by , and replace each regular word $w_{D_1},\ldots,\ w_{D_t}$ that remains by a set of clauses $\{D_1,\ldots,D_t\}$.

The proof of the preceding theorem shows that the sequence of these sets of clauses contains not only the refutation in R^* but, in fact, also information what an algorithm needs to keep in memory in order to check the correctness of the refutation. Before we state a formal theorem we recall a relevant concept of space complexity of resolution derivations introduced in [7] and refined in [13]. We take the definition given by Esteban and Torán in [13] for tree-like proofs.

Definition 4.6. Let $k \in \mathbb{N}$, we say that an unsatisfiable set of clauses $\mathcal{C}$ has a tree-like resolution refutation bounded by space k if there exists a sequence of clauses $\mathcal{C}_1,\ldots,\ \mathcal{C}_s$ such that $\mathcal{C}_1 \subseteq \mathcal{C}$, $\{\} \in \mathcal{C}_s$, in any $\mathcal{C}_i$ there are at most k clauses, and for each $i < s$, $\mathcal{C}_{i+1}$ is obtained from $\mathcal{C}_i$ by one of:

(i) deleting some of its clauses,

(ii) adding the resolvent of two clauses of $\mathcal{C}_i$ and deleting the parent clauses,

(iii) adding some of the clauses of $\mathcal{C}$ (initial clauses).

The definition of space in tree-like resolution expresses the idea of considering list of clauses kept in memory during the refutation, with the particularity that when a clause is used to derive other clauses, it is removed from the memory.

Now, we state a theorem that follows immediately from the proofs of Theorems 4.3 and 4.5.

Theorem 4.7. *Let $\mathcal{C}$ be a set of clauses in variables $x_1,\ldots, x_n$. Assume that there is a refutation in R^* of $\mathcal{C}$ of space t. Then there is a derivation in Σ_n^* of LR from $\mathcal{I}_\mathcal{C}$ such that every line in it contains at most t clause-words.*

Assume, on the other hand, that there is a derivation in Σ_n^* of LR from $\mathcal{I}_\mathcal{C}$ in which all lines contain at most t clause-words. Then $\mathcal{C}$ has a refutation in R^* of space at most t.

In an effort to better understand Resolution another important measure has been introduced: resolution width. The notion of resolution width was made explicit by Galil in [14] and the importance of it was pointed out by Ben-Sasson and Wigderson in [3]. Roughly speaking, the width of a resolution is the largest number of literals in a clause used in the refutation to obtain $\{\}$. We recall the formal definition given in [14].

Definition 4.8. Let $\mathcal{C}$ be a set of clauses, over variables $x_1,\ldots, x_n$. The width($\mathcal{C}$) is the number of literals in the largest clause in $\mathcal{C}$. If π is a resolution refutation of $\mathcal{C}$, width(π) is the number of literals in the largest clause in π. Let proofwidth($\mathcal{C}$) denote the minimum of width(π) over all refutations π of $\mathcal{C}$.

Similar complexity characterizations can be given for Σ_n^*.

Definition 4.9. Let $\mathcal{I}_\mathcal{C}$ be a set of regular word representing a set of clauses $\mathcal{C} = \{C_1,\ldots,C_m\}$. $w_\mathcal{C} = w_{C_1}\ldots w_{C_m}$. Then width($w_\mathcal{C}$) is the number of symbols in the largest clause-word $w_{C_j} \subseteq w_\mathcal{C}$, with $1 \leq j \leq m$. Thus, if π' is a proof in Σ_n^*, then width(π') is the number of symbols in the largest clause-word in π'. At the same manner can be defined the proofwidth(π'), namely the minimum of width(π') over all proofs π' of LR from $\mathcal{I}_\mathcal{C}$

Thus, if $w_\mathcal{C} = Lx_1RLx_3x_5x_2\bar{x_2}x_4RLx_8\bar{x_1}\bar{x_5}R$, representing the set of clauses $\mathcal{C} = \{\{x_1\},\{x_3,x_5,x_2,\bar{x_2},x_4\},\{x_8,\bar{x_1},\bar{x_5}\}\}$, then width($w_\mathcal{C}$) = 7.

Notice that it is easy to obtain from width($w_{\mathcal{C}}$) the width of the corresponding set of clauses $\mathcal{C}$. It is enough eliminate the upper case letters L, R from the designated subword with biggest size and then find the width of corresponding set clauses.

The following theorem is due to Ben-Sasson and Wigderson [3] and it relates size lower bounds on tree like resolution refutations to lower bounds on the width of resolution proofs:

Theorem 4.10. *Any tree-like resolution proof π of $\mathcal{C}$ of size k can be converted to one of width $\lceil log_2 k \rceil +$ width $\mathcal{C}$.*

Combining Theorem 4.10 with Theorems 4.3 and 4.5 yields a non-trivial estimate of the width of derivations in Σ_n^*.

Theorem 4.11. *Let $\mathcal{C}$ be a set of clauses and assume $w_0 =$width$(\mathcal{C})$. Assume there is a derivation in Σ_n^* of LR from $\mathcal{I}_{\mathcal{C}}$ with k' lines. Then there is a derivation in Σ_n^* of LR from $\mathcal{I}_{\mathcal{C}}$ of width bounded above by $log(k') + w_0$.*

The previous theorem allow us to derive weak forms of automatizability as introduced in [1]. Recall that given a proof system P for a language L and a function $f : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$, we say that P is $f(n,k)$-automatizable if and only if there is an algorithm Π_P such that given any input x with $|x| = n$, if $x \in L$, then Π_P outputs a proof π in P of this fact in at most $f(n,k)$ steps, where k is the size of the shortest proof in P of the fact that $x \in L$.

Definition 4.12. Σ_n^* is automatizable if and only if it is $f(n,k)$-automatizable for some function f that is $(n+k)^{O(1)}$.

In this sense automatizable means that for Σ_n^* is possible to find a proof in polynomial time in the size of the smallest one. In fact, it follows by [3] that:

Theorem 4.13. *There is an algorithm Ω having the following properties:*

1. *On input $\mathcal{C}$ (an unsatisfiable set of clauses) it constructs a derivation in Σ_n^* of LR from $\mathcal{I}_{\mathcal{C}}$;*
2. *Ω runs in time $k^{O(logn)}$, where n is the number of variables and k is the number of clause-words in π'.*

The motivation of this restating of Resolution has several sources. First we want understand how the formalism of rewriting systems allows us to formulate basic proof systems. Here we study resolution and its tree-like version. The simulation by rewriting system is fairly straightforward but as a by-product we obtain an interpretation of several proof complexity measures such as the space or the width in essentially geometric terms when using diagrams we associate to rewriting proofs (see section 5 below). This extends to Resolution some geometric interpretations that were known only for the so called group-based proof systems considered in [20].

A second motivation for studying proof systems in terms of rewriting systems is the hope to gain, using also the diagrammatic interpretation mentioned earlier, some intuition for proof search heuristic. One may expect that a heuristic formulated in terms of strategies for rewriting systems could apply also to more complex rewriting systems that would simulate stronger proof systems than Resolution. Virtually no heuristic for proof search in strong proof systems is known.

In particular, we also consider our present paper as a first step towards using in proof complexity rewriting systems that operate in parallel on all symbols of a string (or an array) as for example, cellular automata do. These discrete dynamical systems and models of massively parallel computation [12] are away from the contemporary research in proof complexity. They consist of very large numbers of simple elements that operate in parallel and interact only locally. Thus, it is a fundamental step to build a suitable framework in order to investigate properly their capability in the study of the complexity of proofs. The rewriting approach can give us this unified framework, since one of the basic way to formalize them is to use tables of local rewriting rules [17]. Moreover, we recall that from the computability point of view Turing Machines and cellular automata, the latter ones considered on finite configurations, are equivalent, but from the complexity point of view, cellular automata are much more efficient, the reader can see on this the Part 3 of [12] and [25]. This fact can have some consequences also in proof complexity on the way that we formulate proof systems. At the moment, our work does not bring out new lower bounds but it does, we hope, open a new perspective for looking at proof systems that may be fruitful.

5. Planar Diagrams representing proofs in Σ_n^*

One motivation in [20] was the study of connections between the Dehn function[6], the word problem, various geometric constructions and propositional proof systems. In particular, with the exception of Cutting plane system and their generalizations to discretely ordered modules, these geometric situations are totally absents in proof complexity. Propositional proof complexity relies mostly on finite combinatorics, with connections to bounded arithmetic and computational complexity; to find geometric characterizations would be useful, as suggested in [20], because it could enlarge the set of methods that we have at our disposal. It should be mentioned that in the few cases in which this enlargement was achieved, using for example algebraic or model theoretic methods, not only lower bounds were obtained but also results of a structural type[7]. We propose a characterization using planar diagrams of the proofs in the semi-Thue system Σ_n^*. Finally, at the end of this section, we give an example of geometric construction (three dimensions) of these proofs in Σ_n^*. In this case rules of construction are easily derived from the two-dimensional case.

A diagram representing a proof in Σ_n^* is a directed planar labelled graph, where every edge xy is labelled by a letter from A_n. The contour of every cell (face) is labelled by a regular word w. We describe how to construct the diagram starting with a proof π' in Σ_n^*. Let w_1 be the source-word and $w_2, \ldots, w_t$ be all the words introduced by $\mathcal{I}$-rules in this order. Note that the same clause-words may have several occurrences in the sequence. Each word w_i is a clause-word from $\mathcal{I}$. Let δ be a disc. We fix an origin (usually on the bottom of the disc), denoted by O. Let σ be the concatenation of all the words $w_1, w_2, \ldots, w_t$. Let q be the number of characters in σ; then we mark the disc δ with $q-1$ nodes (other than O). Thus we obtain a disc divided in q edges, since the origin gives the starting node. Then going counterclockwise from O we write for each edge e a character from σ. This

[6]A finitely presented group is given by a finite number of generators and a finite number of basic equations between words defined by the generators (the so called relators). Two words from generators (and their inverses) are equal in the group if their equivalence can be deduced using only the basic relators. The Dehn function measures the minimal number of deduction steps need. For details see [6] and [20].

[7]For example, lower bounds for algebraic systems like Nullstellensatz or Polynomial Calculus also explicitely described the set of all "short provable"(i.e. in their degree) formulas by giving its basis as a linear space.

means that if we had an original proof π' in which have been introduced the regular words $L\bar{x_2}R$, $L\bar{x_1}R$ and the source-word is Lx_1x_2R then δ is divided in 10 labelled edges by 9 nodes plus the origin O.

Consider the following three rules of composition (Figure 1, Figure 2, Figure 3):

1. (*Join*) If there are four consecutives edges, xy, yz, zu and uv, labelled by x_i,R,L, and $\bar{x}_i$ (or labelled by $\bar{x}_i$,R,L, and x_i) then we can join the nodes x,v by a dashed edge.[8]

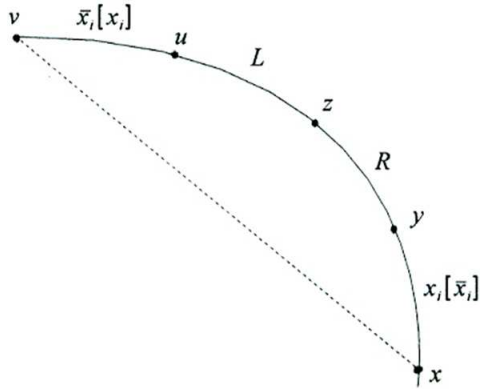

Figure 1: The rule Join.

2. (*Projection*) Let xv be two nodes connected by a dashed edge and let ve be the next edge, going counterclockwise. Then a new edge can be drawn connecting the nodes e, x. The new edge ex will be labelled by the same symbol labelling ve.[9]

3. (*Swap*) This rule allows to swap two edges which are consecutives. Let xy and yz be two consecutives edges, going counterclockwise, labelled by l_i and l_j. Then two new edges connecting x and z can be drawn such that the order of the labelling letters is reversed. This rule can be applied only to lowercase letters.[10]

Definition 5.1. A cell β is a region delimitated by solid edges contained

[8]This is a simulation of Rule 1(a)[Rule 1(b)].
[9]This is a simulation of the contraction rule.
[10]This is a simulation of the exchanging rules.

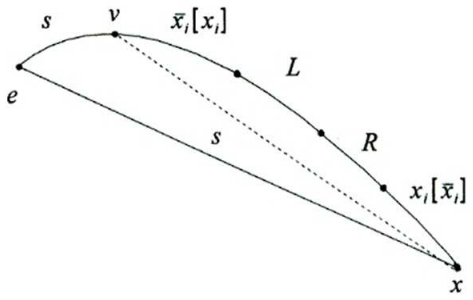

Figure 2: The rule Projection.

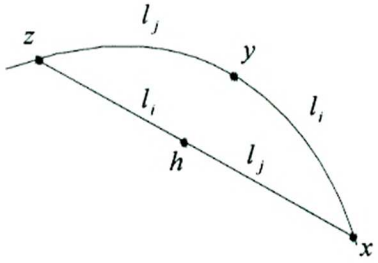

Figure 3: The rule Swap.

in δ; we denote it by $\beta \subseteq \delta$.[11]

Definition 5.2. The perimeter p of a given labelled disc δ is the number of edges on its border. The perimeter p of given cell β is the number of edges of β.

Then in the example given in Figure 4, where $L\bar{x_1}R$, $L\bar{x_2}R$ and Lx_1x_2R are the initial clause-words, the perimeter of the disc is 10.

In case of application of the *Swap* rule the new cell contained in the disc has still perimeter 10. When we apply the *Join* rule by definition the cell has still perimeter 10; then the *Projection* rule gives a new cell with perimeter 6 (labelled by $Lx_1RL\bar{x_1}R$).

We search for configurations suitable for the application of the rule *Join*. We try to reduce the complexity of δ and the *Join* rule gives us the chance to use the *Projection* rule; this last one is the only rule (by creating a new

[11]Note that by Definition 5.1 a cell can contain properly another cell. As a limit case the disc itself is a cell.

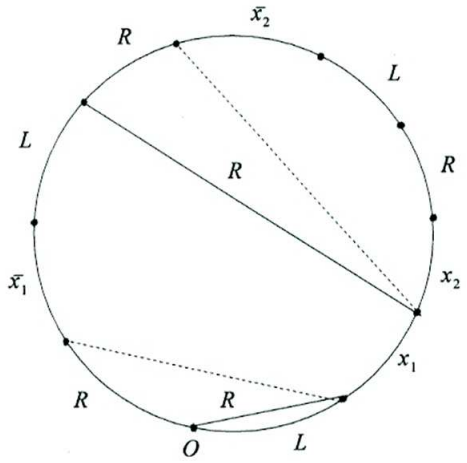

Figure 4: The proof given in section 2.

edge with a new label) which reduces the perimeter of our disc. During search there are two possibilities:

- The sequence of edges represented in Figure 1 exists.
- The sequence of edges represented in Figure 1 does not exist.

If the sequence exists we apply the rules *Join*; next we apply *Projection* and we consider the new cell which has perimeter $(p-4)$; in the second case we search, going counterclockwise, for edges labelled by letters x_i and $\bar{x}_i$; then we apply the *Swap* rule many times is necessary in order to have suitable sequences of edges on which we can apply *Join*. Then we consider the resulting cell with the labels attached and we write down, going counterclockwise from the origin O, the remaining letters. This is the last step of the proof π' in Σ_n^*. In order to discuss more extensively this construction, we need to introduce some additional definitions.

Definition 5.3. A disc δ is regular if and only if the word attached to it is regular. A cell $\beta \subseteq \delta$ is regular if and only if the attached word to it is regular (going counterclockwise from the origin O).

Lemma 5.4. *Let δ (β) be a regular disc (a regular cell). If one of the composition rules Join, Projection and Swap can be applied to δ (β), then the application creates a regular cell $\alpha \in \delta$ ($\alpha \in \beta$).*

Proof. For *Join* it is easy since no new edges are added to the disc; thus, after their application the disc (that is regular by hypothesis) it remains regular. The application of the *Swap* rule add edges connecting only edges labelled by lowercase letters (this is the restriction on the rule) and do not involve movement of uppercase letters; then the new cell is still labelled by the same letters. So if by hypothesis the disc (or the previuos cell) is regular then the cell β obtained by *Swap* is regular. When is applied the *Projection* rule has two cases.

1. The label of the edge is a lowercase letter.

2. The label of the edge is R.

In the first case, by hypothesis the disc (or the previous cell) is regular then is labelled by a regular word and by soundness the new cell is regular. In the second case a similar argument gives the claim. □

Definition 5.5. If a regular disc δ after finitely many steps ends with a cell labelled by LR (having $p = 2$) then the disc δ is called regular and complete.

Theorem 5.6. *If there exists in* Σ_n^* *a proof*

$$\pi' : \mathcal{I} \vdash_{\Sigma_n^*} LR$$

such that $\mathcal{I} = \{w_{C_1}, w_{C_2}, \ldots, w_{C_t}\}$. *Then there exists a regular disc* δ *labelled by words from* $\mathcal{I}$, *where for* $1 \leq i \leq t$ *each* w_{C_i} *may occur more than once, that is also complete.*

Proof. Consider π'. By definition of proof in Σ_n^*, π' is a sequence of regular words $w_1, w_2, \ldots, w_k$. Start with a disc δ labelled by clause-words from $\mathcal{I}$ in the order they are introduced in π'. It is regular and will be regular at any stage, by Lemma 5.4. In order to show that the disc is also complete we must prove that ater finitely many steps the disc ends with LR. It suffice to notice that the diagrams rules are used to simulate how the rewriting rules of Σ_n^* are applied in π' in an obvious way: *Swap* simulates the exchanging rules, *Join* and *Projection* the elimination rules. □

Further, we have that

Theorem 5.7. *If there exists a regular and complete disc δ labelled by words from a non empty set $\mathcal{I}$ of clause-words, then there exists a proof*

$$\pi' : \mathcal{I} \vdash_{\Sigma_n^*} LR.$$

Proof. Let δ be a regular and complete disc whose border is labelled by clause-words from $\mathcal{I}$. We shall construct the Σ_n^*-proof of LR from $\mathcal{I}$ backwards. Let δ_1 be the cell labelled by LR with perimeter 2. Associate with it the word $w_1 = LR$. At any given stage we will have a subdisc δ_i of δ, a set $\mathcal{I}_i$ of clause-words and a regular word w_i such that:

1. $\mathcal{I}_i$ are the words occurring on the perimeter of δ_i;
2. clause subwords of w_i are in $\mathcal{I}_i$;
3. w_i,w_{i-1},...,w_1 is a valid Σ_n^*-derivation.

We write down every border going counterclockwise of each cell that we meet in the process. Every single line will be a regular word and to check if rules are applied correctly and they correspond to the rules in Σ_n^* is easy. The only point in this construction where we must be very careful is when we get the border of the disc. Recall that it collects all the application of the $\mathcal{I}$-rules. In order to get the original proof we must operate as follows; first we write down the complete border; then we consider the labels and we define properly the corresponding clause-words. Then we introduce step by step all the clause-words using the following procedure; if the number of clause-words is n then we obtain a sequence of n lines such that each $n-1$ line does not contain the last clause-word contained in the line n; at the end of this process we have w_1, the source-word of π'. This conclude our construction in the proof. □

Combining the results obtained so far we can prove the following statement which give us the link between tree-like refutation proofs and regular and complete discs.

Theorem 5.8. *A set of clauses $\mathcal{C} = \{C_1, \ldots, C_m\}$ is unsatisfiable if and only if there exists a regular and complete disc δ such that its border is labelled by words from $\mathcal{I}_\mathcal{C}$.*

Definition 5.9. Let δ be a complete and regular disc. The number of cells contained in δ is the size.

Then the following theorem can be proved by inspection on π and δ.

Theorem 5.10. *Let k be the size of tree-like resolution refutation proof π. Let k' be the size of the corresponding regular and complete disc δ. Then $k' < k$.*

We may conceive the construction of our diagrams in a three-dimensional space. A nice representation in Euclidean solid geometry can be obtained. In this context, a proof is represented by a cylinder sectioned by polygons labelled with symbols from A_n. Thus, to represent a given proof[12] means to represent how the volume of the starting cylinder can be reduced. One example is given in Figure 5, where we consider the proof represented before in Figure 4. In this case the initial clause-words were Lx_1x_2R, $L\bar{x_1}R$ and $L\bar{x_2}R$.

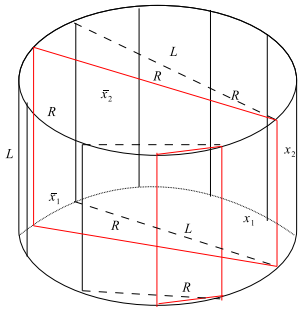

Figure 5: The same proof as in Figure 4 in three dimensions.

We conclude the section considering a more complicated proof with respect to that in Figure 4. Let $Lx_1x_2x_3R$ be the source-word. Let $L\bar{x_2}R$, $L\bar{x_1}R$, $L\bar{x_3}R$ be the words given by $\mathcal{I}$-rules. Having these informations we can construct the disc δ representing the proof in Σ_3^*, see Figure 6. Analyzing the resulting diagram we can obtain all the informations about the original proof. The number of steps in the proof is the number of cells labelled by regular words reading the diagram going counterclockwise with respect to O. The number of variables is given by the number of dashed

[12]It easy to see how to translate the *Swap*, *Join* and *Projection* rules into the three-dimensional case.

lines contained in the disc. To find the size of the proof it is enough look at all the cells (starting from the one with smallest perimeter) and obtain all the regular words used in the proof and then find the clause-words. Thus we obtain the proof starting from the bottom, namely from the regular word LR. Then space and width can be defined similarly from the diagram in a geometric way. By the previous results linking Σ_n^* and R^*, we may find the estimate of complexity of resolution refutation proofs.

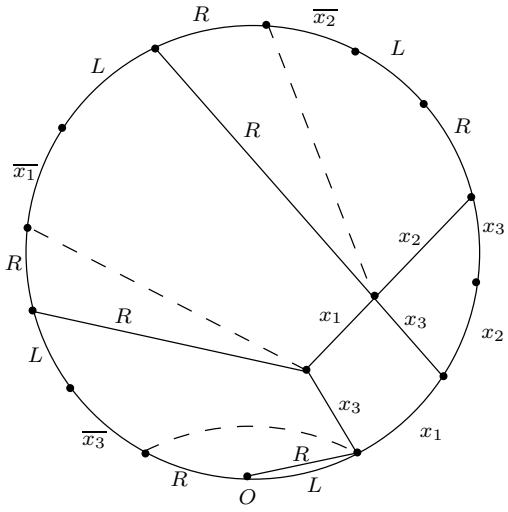

Figure 6: Proof of LR from $Lx_1x_2x_3R$, $L\bar{x_2}R$, $L\bar{x_1}R$ and $L\bar{x_3}R$.

6. Resolution and Σ_n: the dag-like case

We are going to outline in this section a construction of a rewriting system Σ_n that is equivalent to a general resolution R (i.e. no restriction to tree-like proofs) in the same way as Σ_n^* is to R^*.

Tree-like proofs have a very transparent structure. Once a clause is used in an inference it disappears; any other occurence of the same clause has its own subproof and can be treated completely independently from other occurences of the clause. In general, in dag-like proofs this is different. A clause can be used as a hypothesis in one inference but does not necessarily disappears: it can be reused as many times as needed.

Of course, we could simulate such proofs by transforming them first into tree-like proofs and then using the simulation outlined in the preceeding section. However, this would blow-up the size of the proofs, sometimes exponentially. If we want to keep the simulation polynomial (computed by a p-time algorithm) we shall have to replace Σ_n^* by a more complicated rewriting system Σ_n.

The rewriting system Σ_n those definition we outline will be able to perform two tasks. The first is the **DOUBLING** process: having a regular word:

$$uLvRw$$

with a clause-word LvR as a subword, rewrite it into

$$uLvRLvRw\ .$$

This will be used for saving an occurrence of the clause-word for possible future inferences.

The second process is **SHIFTING**: having a regular word:

$$uLvRLwRe$$

with the two clause-words LvR and LwR as subwords rewrite it into

$$uLwRLvRe\ .$$

This procedure will enable us to move a clause-word which is a subword inside the regular word to a position where an elimination rule can be applied.

It is fairly obvious that if we augment Σ_n^* somehow to Σ_n such that the stronger rewriting system can perform the two procedures, the simulation from Theorem 4.3 can be extended to non-treelike proofs too. However, we want to construct such a simulation so that it is sound analogously to Theorem 4.5 about Σ_n^*. For this reason we have defined the system Σ_n not in as minimal way, using the smallest possible number alphabet and rewriting rules, but in a way in which the soundness is easy to verify. The drawback of this is that the systems has a bigger alphabet and the number of rewriting rules considerably increase.

Thus we explain the ideas how **DOUBLING** and **SHIFTING** are implemented and how the soundness is proved; we do not do it formally

since full details are too tedious to follow. Now, we describe informally but with some precision how are the **DOUBLING** and **SHIFTING** procedures simulated so that the resulting system Σ_n properly simulates R (analogously with Theorem 4.5).

Let us consider the **DOUBLING**. The idea is to introduce in the alphabet colored versions of letters from A_n (four copies of different colors suffice). A clause-subword LvR of a regular word whose occurence is to be doubled is first colored green; the rules are formulated in a way that allows to color only one such subword (this use extra "super-script"symbol). Then green L is replaced by uncolored L and blue L. The rules allow to move blue letter right over all green and blue letters. Next the leftmost green letter (a literal unless v is the empty word) is replaced by its uncolored version followed by its blue version. The blue version is again moved as far right as possible over all green and blue letters, etc... At the end of this process the occurence of LvR colored at the start green is now uncolored and it is followed by its blue copy. Finally, the blue color is erased. In a very similar way can be treated the **SHIFTING** procedure.

We give below two examples to illustrate the procedures. In the first we formulate a possible set of rules that govern the **SHIFTING** procedure. In the second example we consider the **DOUBLING** procedure and we define a corresponding set of rewriting rules for it. In both examples we show how a step in a dag-like proof can be simulated by a string rewriting system.

Example 1. Let A'_n be the alphabet containing L, R, h, h^m, h^e and $x_1,\ldots,\ x_n$. We call h the 'head' symbol. The symbols h^m and h^e are called the 'head-moving' and the 'head-erasing' symbols respectively. If $a \in (A'_n \setminus h^e, h^m)$, then each a may have the following form: a, $\hat{a}$ and a^*. [13] Now, consider the rewriting system Σ^*_n extended by the following four sets of rewriting rules:

1. **Structural rules**

   (Sa) $L \to hL$;

   (Sb) $Lh \to hL$;

[13] We can interpret hat and star as green and blue.

(Sc) $Rh \to hR$;

(Sd) $hR \to Rh$;

(Se) $hL \to Lh$;

(Sf) $hx_i \to x_i h$;

(Sg) $h\bar{x_i} \to \bar{x_i} h$;

(Sh) $x_i h \to h x_i$;

(Si) $\bar{x_i} h \to h \bar{x_i}$;

(Sj) $h \to \Lambda$;

These rules allow to introduce, erase and move through subwords the 'head' symbol.

2. **Coloring rules**:

(Ca) $hL \to \hat{L}\hat{h}$;

(Cb) $\hat{h}x_i \to \hat{x_i}\hat{h}$;

(Cc) $\hat{h}\bar{x_i} \to \hat{\bar{x_i}}\hat{h}$;

(Cd) $\hat{h}R \to \hat{R}h^*$;

(Ce) $h^*L \to L^*h^*$;

(Cf) $h^*x_i \to x_i^*h^*$;

(Cg) $h^*\bar{x_i} \to \bar{x_i}^*h^*$;

These rules are used to "color" (by hat and star) subwords.

3. **Moving rules**:

(Ma) $h^*R \to R^*h^m$;

(Mb) $\hat{a}b^* \to b^*\hat{a}$, where $a, b \in \{L, R, x_i, x_j\}$ and $i, j = 1, \dots n$;

(Mc) $\hat{a}h^m \to h^m a$ where $a \neq L$;

These rules allow to move colored symbols.

4. **Erasing rules**:

(Ea) $\hat{R}h^m \to h^e R$;

(Eb) $a^*h^e \to h^e a$, where $a \neq L$;

(Ec) $L^*h^e \to hL$;

(Ed) $\hat{a}h^e \to h^e a$;

These rules erase colors.

Now, consider the following step (i) of a sequence-like proof where clauses C_1 and C_3 are resolved to obtain the clause C_4 at the next step (i+1).

$$(i) \qquad \underbrace{\{x_1x_2x_3\}}_{C_1}\underbrace{\{x_4x_5\}}_{C_2}\underbrace{\{\bar{x_1}x_6\}}_{C_3}$$

$$(i+1) \qquad \underbrace{\{x_2x_3x_6\}}_{C_4}\underbrace{\{x_4x_5\}}_{C_2}$$

Let $w_{C_1C_2C_3}$ be the regular word constituted of the clause-words w_{C_1}, w_{C_2} and w_{C_3} (apply Definition 4.1). Then, $w_{C_1C_2C_3}$ is

$$Lx_1x_2x_3RLx_4x_5RL\bar{x_1}x_6R$$

Now, we show how the step from (i) to (i+1) can be simulated using the rewriting rules of Σ_n^* extended by **Structural**, **Coloring**, **Moving** and **Erasing** rules.

We start our process by introducing the symbol h, thus:

$$hLx_1x_2x_3RLx_4x_5RL\bar{x_1}x_6R$$

by (1Sa);

$$\hat{L}\hat{h}x_1x_2x_3RLx_4x_5RL\bar{x_1}x_6R$$

by (2Ca);

$$\hat{L}\hat{x_1}\hat{h}x_2x_3RLx_4x_5RL\bar{x_1}x_6R$$

by (2Cb);

$$\hat{L}\hat{x_1}\hat{x_2}\hat{h}x_3RLx_4x_5RL\bar{x_1}x_6R$$

by (2Cb);

$$\hat{L}\hat{x_1}\hat{x_2}\hat{x_3}\hat{h}RLx_4x_5RL\bar{x_1}x_6R$$

by (2Cb);

$$\hat{L}\hat{x_1}\hat{x_2}\hat{x_3}\hat{R}h^*Lx_4x_5RL\bar{x_1}x_6R$$

by (2Cd);

$$\hat{L}\hat{x_1}\hat{x_2}\hat{x_3}\hat{R}L^*h^*x_4x_5RL\bar{x_1}x_6R$$

by (2Ce);

$$\hat{L}\hat{x_1}\hat{x_2}\hat{x_3}\hat{R}L^*x_4^*h^*x_5RL\bar{x_1}x_6R$$

by (2Cf);

$$\hat{L}\hat{x_1}\hat{x_2}\hat{x_3}\hat{R}L^*x_4^*x_5^*h^*RL\bar{x_1}x_6R$$

by (2Cf);

$$\hat{L}\hat{x_1}\hat{x_2}\hat{x_3}\hat{R}L^*x_4^*x_5^*R^*h^mL\bar{x_1}x_6R$$

by (3Ma);

Then apply the rule (3Mb) as many times we needed in order to obtain the following regular word:

$$L^*x_4^*x_5^*R^*\hat{L}\hat{x_1}\hat{x_2}\hat{x_3}\hat{R}h^mL\bar{x_1}x_6R.$$

Thus by application of (4Ea):

$$L^*x_4^*x_5^*R^*\hat{L}\hat{x_1}\hat{x_2}\hat{x_3}h^eRL\bar{x_1}x_6R$$

Now, rule (4Ed) can be applied until we obtain the following regular word:

$$L^*x_4^*x_5^*R^*h^eLx_1x_2x_3RL\bar{x_1}x_6R$$

Then we can apply (4Eb):

$$L^*x_4^*x_5^*h^eRLx_1x_2x_3RL\bar{x_1}x_6R$$

Rule (4Eb) can be applied until the symbol on the left hand side h^e is L;

$$L^*h^ex_4x_5RLx_1x_2x_3RL\bar{x_1}x_6R$$

Then by (4Ec) we obtain the regular word:

$$hLx_4x_5RLx_1x_2x_3RL\bar{x_1}x_6R$$

Then using the structural rule (1Sj) we obtain the regular regular word in which the first two clause-words are exchanged:

$$Lx_4x_5RLx_1x_2x_3RL\bar{x_1}x_6R.$$

In order to complete our simulation we apply the exchanging rules from Σ_n^* two times:

$$Lx_4x_5RLx_2x_3x_1RL\bar{x_1}x_6R$$

and then the elimination rule from Σ_n^* can be applied:

$$Lx_4x_5RLx_2x_3x_6R$$

Now, it is easy to check that the resulting regular word is composed by two clause-words w_{C_2} and w_{C_4}; these two words are by Definition 4.1 the clauses C_2 and C_4.

Example 2. Let A_n'' be the alphabet containing L, R, h, $h^\ddagger$ and $x_1,\ldots,\ x_n$. h is the 'head' symbol and $h^\ddagger$ is the 'head-stop' symbol. If $a \in (A_n' \setminus h, h^\ddagger)$, then each a may have the following form: a, $\check{a}$ and $\tilde{a}$. [14] Now, consider the rewriting system Σ_n^* extended by the following two set of rewriting rules:

1. **Structural rules**

   (Sa) $L \to hL$;

   (Sb) $Lh \to hL$;

   (Sc) $Rh \to hR$;

   (Sd) $hR \to Rh$;

   (Se)) $hL \to Lh$;

   (Sf) $hx_i \to x_ih$;

   (Sg) $h\bar{x_i} \to \bar{x_i}h$;

   (Sh) $x_ih \to hx_i$;

   (Si) $\bar{x_i}h \to h\bar{x_i}$;

   (Sj) $h \to \Lambda$;

   These rules allow to introduce, erase and move through subwords the 'head' symbol.

[14] We can interpret 'check' and 'tilde' as red and yellow.

2. **Doubling rules**

   (Da) $hL \to \check{L}\tilde{L}h$;

   (Db) $ha \to \check{a}\tilde{a}h$, where $a \in (A_n \setminus \{L, R\})^*$;

   (Dc) $hR \to \check{R}\tilde{R}h^{\ddagger}$;

   (Dd) $\check{a}\tilde{b} \to \tilde{b}\check{a}$, where $a, b \in A_n$;

   (De) $\tilde{a}h^{\ddagger} \to h^{\ddagger}a$, where $a \in A_n$;

   (Df) $\check{a}h^{\ddagger} \to h^{\ddagger}a$, where $a \in A_n$;

   (Dg) $h^{\ddagger}a \to ha$, where $a = L$.

   These rules allow to make a copy of a subword.

Now, consider the following step (i) in a dag-like proof where the clause C_2 is used two times: first, to derive in the step (i+1) the clause C_4 and then in the step (i+2) to obtain C_5.

$$\begin{array}{ll} (i) & \underbrace{\{x_1x_2\}}_{C_1}\underbrace{\{\bar{x_2}\}}_{C_2}\underbrace{\{\bar{x_2}x_3\}}_{C_3} \\ (i+1) & \underbrace{\{x_1\}}_{C_4}\underbrace{\{\bar{x_2}\}}_{C_2}\underbrace{\{x_2x_3\}}_{C_3} \\ (i+2) & \underbrace{\{x_1\}}_{C_4}\underbrace{\{x_3\}}_{C_5} \end{array}$$

Let $w_{C_1C_2C_3}$ be the regular word constituted of the clause-words w_{C_1}, w_{C_2} and w_{C_3} (apply Definition 4.1). Then, $w_{C_1C_2C_3}$ is

$$Lx_1x_2RL\bar{x_2}RLx_2x_3R.$$

We show how to simulate the previous derivation from (i) to (i+2) using the rewriting rules of Σ_n^* extended by **Structural** and **Doubling** rules. We introduce the symbol h by the rule (1Sa), then:

$$Lx_1x_2RhL\bar{x_2}RLx_2x_3R.$$

Then by (2Da):

$$Lx_1x_2R\check{L}\tilde{L}h\bar{x_2}RLx_2x_3R.$$

By rule (2Db) we obtain:

$$Lx_1x_2R\check{L}\tilde{L}\check{x}_2\tilde{\bar{x}}_2hRLx_2x_3R.$$

Then we apply the rule (2Dc):

$$Lx_1x_2R\check{L}\tilde{L}\check{x}_2\tilde{\bar{x}}_2\check{R}\tilde{R}h^{\ddagger}Lx_2x_3R.$$

After some applications of (2Dd), we obtain the following regular word:

$$Lx_1x_2R\tilde{L}\tilde{\bar{x}}_2\tilde{R}\check{L}\check{x}_2\check{R}h^{\ddagger}Lx_2x_3R$$

then by (2Df) three times:

$$Lx_1x_2R\tilde{L}\tilde{\bar{x}}_2\tilde{R}h^{\ddagger}L\bar{x}_2RLx_2x_3R$$

and by (2De) three times we have:

$$Lx_1x_2Rh^{\ddagger}L\bar{x}_2RL\bar{x}_2RLx_2x_3R.$$

Then by (2Dg),

$$Lx_1x_2RhL\bar{x}_2RL\bar{x}_2RLx_2x_3R.$$

Thus by the structural rule (1Sj) we obtain:

$$Lx_1x_2RL\bar{x}_2RL\bar{x}_2RLx_2x_3R.$$

Then we can apply elimination rules from Σ_n^*

$$Lx_1RL\bar{x}_2RLx_2x_3R$$

and

$$Lx_1RLx_3R.$$

It is easy to verify that the simulation is correct.

Analyzing the previous examples it is fairly clear that it is possible to write down rules allowing the rewriting procedure described above. However, it cannot be enforced that the rules can be applied in a unique way. For example, anytime during the procedures we can insert few applications of the other rules from Σ_n^*. Or we can color the word and then uncolor a part and color again, etc ...

Thus instead of formulating particular rules it seems to us more convenient to formulate a general property of rules to be used in Σ_n that will guarantee the soundness, i.e. whenever there is a derivation in Σ_n of LR from $\mathcal{I}_\mathcal{C}$ then indeed $\mathcal{C}$ is unsatisfiable.

The property of Σ_n we want is:

There is a map associating to every word w that can occur in a Σ_n derivation from $\mathcal{I}_\mathcal{C}$ a set of clauses $\mathcal{D}_w$ such that

- Any truth assignment satisfying all clauses in $\mathcal{C}$ satisfies also all clauses in $\mathcal{D}_{w_\mathcal{C}}$.
- If a word v is derived in one step from a word u then any truth assignment satisfying all clauses in $\mathcal{D}_u$ satisfies also all clauses in $\mathcal{D}_v$.
- $\mathcal{D}_{LR}$ is unsatisfiable.

Such a map is constructed similarly as in the proof of Theorem 4.5. Notice that the colors allow us to reconstruct which literals, which may be being moved around, belong to the same clause-word. We skip the tedious details.

7. Conclusions

In this paper we have shown that a propositional proof system such as Resolution can be interpreted as a string rewriting system; in particular as a Semi-Thue system. The interpretation Σ_n^* of the tree-like case has an interesting representation based on planar diagrams and they are similar to Van Kampen diagrams. This representation can be exploited more also to give a concrete geometrical representation, in Euclidean space. The system Σ_n^* is very elegant and his formal representation does not require too many rules and the proofs in Σ_n^* have a structure which is really transparent as the structure of proofs in R^*. Indeed, all the complexity measures study for R^* can be characterized in a very clear way also for the system Σ_n^*.

In the case of general resolution R things are less smooth, and this is because the proofs in R may have very complicated structure. Thus, in the simulation of them, using a string rewriting approach the number of rules substantially blow up and the resulting translation in Σ_n is much more

tangled. Of course, in principle this is possible as we have outlined. We can simulate using string rewriting systems also general resolution and the gap between the proofs in R and in Σ_n is still polynomial, but it is not satisfactory in terms of its formal representation.

Acknowledgements. I thank Jan Krajíček for his patience, for helpful discussions and for his assistence during the final review of the paper. I also thank Emil Jeřábek, Alan Skelley and Neil Thapen for comments on section 5. Finally, I thank the anonymous referee for numerous suggestions and comments that helped to improve substantially the presentation of the paper.

References

[1] A. Atserias, M. L. Bonet, *On the automatizability of resolution and related Propositional Proof Systems*, Information and Computation **189**, 2 (2004), pp. 182–201.

[2] E. Ben-Sasson, R. Impagliazzo, and A. Wigderson, Near-optimal separation of tree-like and general resolution, ECCC, Report TR02-005 (2000).

[3] E. Ben-Sasson, and A. Wigderson, *Short proofs are Narrow–Resolution made Simple*, Journal of the ACM **48**, 2 (2001), pp. 149–169.

[4] A. Blake, *Canonical expression in boolean Algebra*, Ph.D Thesis (1937), University of Chicago.

[5] M.L. Bonet, J.L. Esteban, N. Galesi and J. Johannsen, *Exponential separations between Restricted Resolution and Cutting Planes Proof Systems*, in 39th Symposium on Foundations of Computer Science (FOCS 1998), pp. 638–647.

[6] M. Bridson, *The geometry of the word problem*, in: Invitations to Geometry and Topology, Oxford University Press, (2002).

[7] K. Büning, T. Lettman, Aussangenlogik: Deduktion und Algorithmen (1994), B.G Teubner Stuttgart.

[8] N. Chomsky, *Three models for the Description of Language*, IRE Transactions on Information Theory **2**, 2 (1956), pp. 113–123.

[9] P. Clote and A. Setzer, *On PHP st-connectivity and odd charged graphs—*, in P. Beame and S. Buss, editors, Proof Complexity and Feasible Arithmetics, AMS DIMACS Series **39** (1998), pp. 93–117.

[10] S.A. Cook, and A.R. Reckhow, *The relative efficiency of propositional proof systems*, Journal of Symbolic Logic **44**, 1 (1977), pp. 36–50.

[11] M. Davis, Computability and Unsolvability, Dover Pubblications, Inc, New York, (1958).

[12] M. Delorme and J. Mazoyer, editors, *Cellular Automata: a parallel model*, Mathematics and its Application, Springer, (1998).

[13] J.L. Esteban, J. Torán, *Space bounds for resolution*, Information and Computation, **171**, 1 (2001), pp. 84–97.

[14] Z. Galil, *On resolution with clauses of bounded size*, SIAM Journal of Computing **6** (1977), pp. 444–459.

[15] K. Iwama and S. Miyazaki, *Tree-like Resolution is superpolinomially slower then dag-like resolution for the Pigeonhole Principle*, in A. Aggarwal and C.P. Rangan, editors, Proceedings: Algorithms and Computation, 10th International Symposium, ISAAC'99, Vol. 1741 (1999), pp. 133–143.

[16] H. von Koch, *Sur une courbe continue sans tangente obtenue par une construction géométrique élémentaire*, Archiv. für Matem. Fys., **1** (1904), pp. 681–702.

[17] J. Kari, *Reversible Cellular Automata*, Proceedings of DLT 2005, Developments in Language Theory, Lecture Notes in Computer Science **3572**, Springer-Verlag, 2005, pp. 57–68.

[18] J. Krajíček, *Bounded arithmetic, propositional logic, and complexity theory*, Encyclopedia of Mathematics and Its Applications **60**, Cambridge University Press, 1995.

[19] J. Krajíček, *Propositional proof complexity I.*, Lecture notes, available at http://www.math.cas.cz/krajicek/biblio.html.

[20] J. Krajíček, *Dehn function and length of proofs*, International Journal of Algebra and Computation, **13**, 5 (2003), pp. 527–542.

[21] A. Thue, *Die Lösung eines Spezialfalles eines gnerellen Logischen Problems*, Kra. Videnskabs-Selskabets Skriften I. Mat. Nat. Kl., **8** (1910).

[22] A. Thue, *Probleme über Veranderungen von Zeichenreihen nach gegeben Regeln*, Skr. Vid. Kristianaia I. Mat. Natarv. Klasse, **10/13** (1914).

[23] R. Raz and P. McKenzie, *Separation of the monotone NC hierarchy*, in Proc. 38th Symposium on Foundations of Computer Science (1997), pp. 234–243.

[24] A. Urquhart, *The Complexity of Propositional Proofs*, Bulletin of Symbolic Logic, **1**, 4 (1996), pp. 425–467.

[25] K. Wagner and G. Wechsung, Computational Complexity, Riedel, 1986.

Institute of Mathematics
Academy of Sciences of Czech Republic
Žitná 25, 11567 Prague 1
Czech Republic

stefanoc@math.cas.cz