Theoretical Computer Science 123 (1994) 273–289
Elsevier

The word problem for inverse monoids presented by one idempotent relator*

Jean-Camille Birget and Stuart W. Margolis
Department of Computer Science and Engineering, University of Nebraska, Lincoln, NE 68588, USA

John C. Meakin
Department of Mathematics and Statistics, University of Nebraska, Lincoln, 68588, USA

Communicated by D. Perrin
Received September 1991
Revised June 1992

Abstract

Birget, J.-C., S.W. Margolis and J.C. Meakin, The word problem for inverse monoids presented by one idempotent relator, Theoretical Computer Science 123 (1994) 273–289.

We study inverse monoids presented by a finite set of generators and one relation $e = 1$, where e is a word representing an idempotent in the free inverse monoid, and 1 is the empty word. We show that (1) the word problem is solvable by a polynomial-time algorithm; (2) every congruence class (in the free monoid) with respect to such a presentation is a deterministic context-free language. Such congruence classes can be viewed as generalizations of parenthesis languages; and (3) the word problem is solvable by a linear-time algorithm in the more special case where e is a "positively labeled" idempotent.

1. Introduction

The word problem for inverse monoids is undecidable in general (since it is even undecidable for groups). In this paper we continue the study of [8] of a class of word problems of inverse monoids which are decidable: we give polynomial-time algorithms for a more restrictive subclass of word problems.

Correspondence to: J.-C. Birget, Department of Computer Science and Engineering, University of Nebraska, Lincoln, NE 68588, USA

* Research supported by N.S.F. Grant No. DMS 8702019 and by the Center for Communication and Information Sciences, University of Nebraska, Lincoln.

SSDI 0304-3975(92)00063-G

We refer the reader to Lallement [6] for basic information about monoids and their relation to automata and formal languages. A more detailed reference on inverse monoids is [10].

An *inverse monoid* is a monoid M such that for every $x \in M$ there exists one and only one $x^{-1} \in M$ satisfying $xx^{-1}x = x$ and $x^{-1}xx^{-1} = x^{-1}$. Thus, $x \to x^{-1}$ is a well-defined function in M; one can check (see [10]) that it satisfies $(x^{-1})^{-1} = x$, $(xy)^{-1} = y^{-1}x^{-1}$, $xx^{-1}yy^{-1} = yy^{-1}xx^{-1}$ for all $x, y \in M$ (the last equality expresses the fact that in an inverse monoid idempotents commute). Every inverse monoid is isomorphic to a monoid of partial one-to-one functions on a set (together with their inverses) under composition (Vagner–Preston theorem, see [10]); this is analogous to the representation of a semigroup by functions, or of a group by permutations.

When M is an inverse monoid and X is a subset of M we say that M *is generated by* X *as an inverse monoid* iff every element of $M - \{1\}$ can be written as a product of elements of $X \cup X^{-1}$; here 1 is the identity of M and $X^{-1} = \{x^{-1} : x \in X\}$. If $w = x_1 \dots x_n$ is a word in $(X \cup X^{-1})^*$, then w^{-1} will denote the word $x_n^{-1} \dots x_1^{-1}$; here, as usual, we identify $(x^{-1})^{-1}$ with x, for $x \in X$. From now on we will always mean "as an inverse monoid", when we say "generated" (unless the contrary is explicitly stated).

For any given set X there exists a *free inverse monoid generated by* X (see [10]), which we will denote by $\mathrm{FIM}(X)$. As a monoid, $\mathrm{FIM}(X)$ can be presented by the generators $X \cup X^{-1}$ and the set of relations $\{ww^{-1}w = w : w \in (X \cup X^{-1})^*\} \cup \{(uv)^{-1} = v^{-1}u^{-1} : u, v \in (X \cup X^{-1})^*\} \cup \{uu^{-1}vv^{-1} = vv^{-1}uu^{-1} : u, v \in (X \cup X^{-1})^*\} \cup \{(u^{-1})^{-1} = u : u \in (X \cup X^{-1})^*\}$ (Vagner relations); here $(X \cup X^{-1})^*$ is the free monoid generated by $X \cup X^{-1}$.

Let X be a set and $R = \{(u_i, v_i\} : i \in I\}$ be a set of pairs of words of $(X \cup X^{-1})^*$; usually a relation $\{u_i, v_i\}$ will be written as $u_i = v_i$. We define the *inverse monoid presented by the generators* X *and relations* R to be the monoid presented by the generators $X \cup X^{-1}$ and the relations R together with the Vagner relations (above). We denote this monoid by $\mathrm{INV}\langle X : R\rangle$ or by $\mathrm{INV}\langle X : \{u_i = v_i : i \in I\}\rangle$. Equivalently, this monoid is the quotient of $\mathrm{FIM}(X)$ under the congruence induced by the relation R.

The *word problem* for a presentation (X, R) of an inverse monoid is the following problem: given two words $u, v \in (X \cup X^{-1})^*$ as an input, do u and v represent the same element of $\mathrm{INV}\langle X : R\rangle$?

In [8], Margolis and Meakin study inverse monoid presentations $(X, R = \{e_i = f_i : i \in I\})$, where each word e_i and f_i represents an idempotent when viewed as an element of $\mathrm{FIM}(X)$. They show that in that case the word problem for (X, R) is decidable (assuming that X and R are finite sets). In fact, they give an algorithm for the following more general problem: Given (as an input) $2n$ words $e_1, f_1, \dots, e_n, f_n \in (X \cup X^{-1})^*$ representing idempotents of $\mathrm{FIM}(X)$, and two words $u, v \in (X \cup X^{-1})^*$, do the words u and v represent the same element of $\mathrm{INV}\langle X : \{e_1 = f_1, \dots, e_n = f_n\}\rangle$? This problem is somewhat more general (and often harder) than the word problem because in the word problem only u and v are inputs, with the presentation kept fixed.

The algorithm of [8] uses Rabin's tree theorem (see e.g. [1, p. 621]); the time complexity of this algorithm is enormous (a linear stack of exponentials in n, where n is the input length $|u|+|v|$). Although in [8] the full generality of Rabin's theorem is not needed, no faster algorithm is known; in any case, it seems unlikely that the general problem of [8] has an algorithm with less than exponential time complexity.

In this paper we consider inverse monoid presentations of the form $(X, e=1)$, where $e \in (X \cup X^{-1})^*$ is a word representing an idempotent of $\mathrm{FIM}(X)$ and 1 is the empty word. Actually, any finite presentation of the form $(X, \{e_1=1, \ldots, e_n=1\})$ where each e_i represents an idempotent of $\mathrm{FIM}(X)$, is equivalent (i.e., induces the same congruence on $(X \cup X^{-1})^*$) to the presentation $(X, e=1)$, where $e=e_1 \ldots e_n$.

We will show that the word problem for the inverse presentation $(X, e=1)$ has polynomial time complexity; in fact, the more general problem (namely: Given $e, u, v \in (X \cup X^{-1})^*$ with e representing an idempotent of $\mathrm{FIM}(X)$, is u equal to v in $\mathrm{INV}\langle X : e=1\rangle$?) has polynomial time complexity. This is proved in Section 2. Needless to say, the algorithm for "$e=1$" avoids Rabin's theorem; instead, finite automata on finite words (rather than on infinite trees) are used.

In [8], and in this paper, the *Cayley graph* $\Gamma(X)$ *of the free group* $\mathrm{FG}(X)$ *generated by* X plays an important role. This is a directed graph whose vertices are all the elements of $\mathrm{FG}(X)$; the edges are all pairs of the form (g, gz), where $g \in \mathrm{FG}(X)$ and $z \in X \cup X^{-1}$; such an edge carries the label z; also for every edge (g, gz) there is an opposite edge (gz, g), with label z^{-1}. The underlying undirected graph of $\Gamma(X)$ (obtained by ignoring directions and labels, and identifying edges that have the same endpoints) is a tree.

The easiest word problem for inverse monoid presentations is the word problem for the free inverse monoid $\mathrm{FIM}(X)$. This was solved by Munn [9], using a tree representation of the elements of $\mathrm{FIM}(X)$. We briefly review some of his results. Let u be any word in $(X \cup X^{-1})^*$. Denote by $\mathrm{MT}(u)$ the labeled subtree of $\Gamma(X)$, traversed by reading the walk in $\Gamma(X)$ labeled by the word u, starting at the vertex 1 (the identity of $\mathrm{FG}(X)$) and ending at the vertex $r(u)$ (the reduced form of the word u with respect to $\mathrm{FG}(X)$). The tree $\mathrm{MT}(u)$ is referred to as the *Munn tree* of u; we may view $\mathrm{MT}(u)$ as a *birooted* tree, the roots being 1 (the initial root) and $r(u)$ (the terminal or final root); if we need to emphasize the birooted nature of $\mathrm{MT}(u)$ we shall use the notation $(1, \mathrm{MT}(u), r(u))$. One sees easily that u is an idempotent of $\mathrm{FIM}(X)$ iff $1=r(u)$ (i.e. the two roots of the birooted tree are the same). Munn's solution to the word problem for $\mathrm{FIM}(X)$ is contained in the following result: *Two words* u, v *are equal in* $\mathrm{FIM}(X)$ *if and only if* $\mathrm{MT}(u)=\mathrm{MT}(v)$ *and* $r(u)=r(v)$. *Equivalently*, $(1, \mathrm{MT}(u), r(u))=(1, \mathrm{MT}(v), r(v))$.

The main technique used in [8], and in this paper, to attack word problems of an inverse monoid presentation (X, R) was developed by Stephen [11]. In that technique, an automaton (usually infinite) is associated to every element of $\mathrm{INV}\langle X : R\rangle$. In the following definition, $\mathscr{R}$ denotes the Green relation related to right ideals (see [6]).

Definition 1.1. Let M be an inverse monoid, let X be a set of generators of M (in the inverse monoid sense), and let m be an element of M. The $\mathscr{R}$-class automaton of m is

defined by the set of states $\mathscr{R}_m$ (the $\mathscr{R}$-class of m *in* M), the input alphabet $X \cup X^{-1}$, the start state mm^{-1}, the single accept state m, and the next-state function $(q, z) \in \mathscr{R}_m \times (X \cup X^{-1}) \to q \cdot z$ (product in M) if $q \cdot z \in \mathscr{R}_m$; the next state is undefined if $q \cdot z \notin \mathscr{R}_m$. We denote the $\mathscr{R}$-class automaton of m by $A(m)$ (or, if ambiguities are possible, $A_{X,M}(m)$).

If M is given by an inverse presentation (X, R) and $m \in M$ is represented by a word $u \in (X \cup X^{-1})^*$ we will also write $A(u)$ (or $A_{X,M}(u)$) instead of $A(m)$ (or $A_{X,M}(m)$).

The language in $(X \cup X^{-1})^*$ recognized by $A(u)$ is denoted $L(A(u))$.

It is straightforward to verify (see [11]) that $A(u)$ is a *minimum automaton* (i.e., there are no useless states, and no two states are equivalent); however, the number of states of $A(u)$ is not necessarily finite. Stephen also proved that $L(A(u)) = \{w \in (X \cup X^{-1})^* \mid w\tau \geqslant u\tau\}$ (the latter set is also denoted $u\uparrow$); here $w\tau$ or $u\tau$ is the element of M represented by w or u, respectively; $\geqslant$ is the natural partial order on M (defined by $m_1 \geqslant m_2$ iff $m_2 = dm_1$ for some idempotent d of M).

This automaton is related to the Schützenberger representation of M relative to $\mathscr{R}_m$; therefore it is also called "the Schützenberger automaton".

To attack word problems (and, hopefully, solve them when they are solvable) we use the following theorem.

Theorem 1.2. (1) *Stephen's criterion: Two words* $u, v \in (X \cup X^{-1})^*$ *are equal in* $\mathrm{INV}\langle X:R\rangle$ *iff* $L(A(u)) = L(A(v))$.

(2) *Stephen's construction process – as used in this paper: The* $\mathscr{R}$*-class automaton* $A(u)$ *is obtained "in the limit" (see* [11] *for a definition and existence proof) by the following process: Start with* $(1, \mathrm{MT}(u), r(u))$, *the Munn tree of* u, *viewed as an automaton. Inductively, when we have a finite automaton* $A_n(u)$, *we get a next finite automaton* $A_{n+1}(u)$ *by applying the following two operations to* $A_n(u)$.

Sewing: *If* $(u_i = v_i) \in R$ *and* u_i *occurs as a label of a walk in the automaton* $A_n(u)$ *(beginning at a state* p *and ending at a state* q*), then a new path going from* p *to* q *and labeled by* v_i *is attached to the automaton* ($|v_i| - 1$ *new states are introduced along this path, where* $|v_i|$ *is the length of the word* v_i*). Similarly, if* v_i *occurs in* $A_n(u)$ *then* u_i *is "sewed on". This is continued until no further sewing can be applied to the original finite automaton* $A_n(u)$.

Folding: *If from a state* p *two states* q_1 *and* q_2 *can be reached using the same input letter* $z \in X \cup X^{-1}$ *then* q_1 *and* q_2 *are made the same state. This is continued until no further folding can be applied.*

Remarks. (1) A process is an "algorithm" (in which every step is constructive) whose execution does not necessarily terminate. A nonterminating process cannot usually be used to compute a result, but can be very useful to give an inductive description of an infinite object.

(2) In [11] $A(u)$ is described in a much more general form. Depending on how sewings and foldings are alternated, many different processes can be devised (and

many nonconstructive descriptions of $A(u)$ are obtained as well). Not all of them will converge, and not all of the convergent ones limit to $A(u)$. In [8] a slightly different construction process than the one of Theorem 1.2 is used (which also limits to $A(u)$).

For the special case of a presentation $(X, \{e_i=f_i: i\in I\})$ where each e_i and f_i is a word which represents an idempotent of FIM(X), Stephen's construction of $A(u)$ (for any word $u\in(X\cup X^{-1})^*$) can actually be carried out (slightly modified as said in the remark), because of the following theorem.

Theorem 1.3 (Margolis and Meakin [8]). *Let* $M=\mathrm{INV}\langle X:\{e_i=f_i: i\in I\}\rangle$, *where each* e_i, f_i, *represents an idempotent of* FIM(X). *Then for any word* u, $A(u)$ *can be embedded into* $\Gamma(X)$ (*the Cayley graph of the free group* FG(X)); *one identifies* 1 *of* $\Gamma(X)$ *with the start state* $(uu^{-1})\tau$ *of* $A(u)$, *and one identifies* $r(u)$ (*the free-group reduction of* u) *in* $\Gamma(X)$ *with the accept state* $(u)\tau$ *of* $A(u)$. *Here* τ *is the morphism* $(X\cup X^{-1})^*\to M$ *determined by the presentation.*

As a consequence (see [8]) the *Stephen construction process can be performed inside* $\Gamma(X)$ in this case. The word problem for $(X, e=1)$, studied in this paper, is a special case of this, so we can use Theorem 1.3.

In [8], it is shown that $A(u)$ and $L(A(u))$ can be described by sentences in the second order monadic theory of the free group FG(X) with the "successor operations" $\cdot z$ (as z ranges over $X\cup X^{-1}$). Rabin's theorem (see Rabin's chapter in [1]) applies to this situation (see [8] and references therein); thus the word problem is decidable.

In Section 2 of this paper we give a simpler and much more efficient (in fact, polynomial-time) algorithm for the word problem of an inverse monoid presentation $(X, e=1)$, when e represents an idempotent of FIM(X).

2. The word problem for the presentation $(X, e=1)$, where e represents an idempotent of FIM(X)

Let $(X, e=1)$ be an inverse monoid presentation, where X is a finite set and $e\in(X\cup X^{-1})^*$ represents an idempotent of FIM(X). The following theorem uses Stephen's Theorem 1.2 to reduce the word problem to a question about finite-state languages in $(X\cup X^{-1})^*$.

Theorem 2.1. *Two words* $u, v\in(X\cup X^{-1})^*$ *are equal in* $\mathrm{INV}\langle X:e=1\rangle$ *iff*

(1) $r(u)=r(v)$ *and*

(2) $r(\mathrm{pref}(u)\cdot(\mathrm{pref}(e))^*)=r(\mathrm{pref}(v)\cdot(\mathrm{pref}(e))^*)$.

Condition (2) *is equivalent to the following condition*:

(2′) *For every prefix* u' *of* u: $r(u')\in r(\mathrm{pref}(v)\cdot(\mathrm{pref}(e))^*)$ *and for every prefix* v' *of* v: $r(v')\in r(\mathrm{pref}(u)\cdot(\mathrm{pref}(e))^*)$.

Notation. $r(.)$ is the reduction operation of the free group; pref(w) is the set of prefixes (initial segments) of the word w (including the empty word 1 and w itself); $(.)^*$ is the Kleene star operation (see e.g. [5, 6]); "." denotes concatenation.

Proof. By Stephen's criterion (Theorem 1.2(1)) we only need to show that $L(A(u))=L(A(v))$ iff the above two conditions hold. Since $A(u), A(v)$ are minimum automata, this is equivalent to saying that the two automata $A(u)$ and $A(v)$ are isomorphic (as automata). By Theorem 1.3, $A(u)$ and $A(v)$ can both be considered as embedded in $\Gamma(X)$; so, replacing the automata by their embeddings in $\Gamma(X)$, $A(u)$ and $A(v)$ are isomorphic iff they are equal. Moreover, since the underlying undirected graph of $\Gamma(X)$ is a tree, the underlying undirected graphs of $A(u)$ and $A(v)$ are also trees; for a subtree of $\Gamma(X)$, the set of vertices (which is a subset of $\mathrm{FG}(X)$) determines the tree. Therefore, $A(v)$ and $A(v)$ are equal iff $A(u)$ and $A(v)$ (viewed as embedded into $\Gamma(X)$) have the same set of vertices ($\subseteq \mathrm{FG}(X)$), and the same accept state ($\in \mathrm{FG}(X)$). (In their embedding into $\Gamma(X)$, $A(u)$ and $A(v)$ necessarily have the same start state, namely 1.) Theorem 2.1 now follows from the next claim. We will represent elements of $\mathrm{FG}(X)$ by reduced words of the form $r(w)$. Recall also that $r(uv)=r(r(u)\cdot r(v))$.

Claim. *The set of states of $A(u)$, embedded into $\Gamma(X)$, is $r(\mathrm{pref}(u)\cdot(\mathrm{pref}(e))^*)$, and the accept state is $r(u)$.*

Proof. We apply the Stephen construction process (Theorem 1.2(2)). We start with the Munn tree $A_1(u)=(1, \mathrm{MT}(u), r(u))$ of u, embedded into $\Gamma(X)$; the set of vertices of $\mathrm{MT}(u)$ is $r(\mathrm{pref}(u))$. Inductively, suppose we have constructed an automaton $A_n(u)$, to which no folding operation can be applied anymore, which is embedded in $\Gamma(X)$, whose vertices are a subset of $r(\mathrm{pref}(u)\cdot(\mathrm{pref}(e))^*)$ and whose accept state is $r(u)$. Now we obtain a new automaton $A_{n+1}(u)$ by sewing on the word e as a loop, on every vertex of $A_n(u)$; after folding inside $\mathrm{MT}(e)$, this is equivalent to attaching the Munn tree of e (whose two roots are equal) at every vertex, and folding $\mathrm{MT}(e)$ into $A_n(u)$ as much as possible. Then $A_{n+1}(u)$ is also a subtree of $\Gamma(X)$; $A_{n+1}(u)$ has $A_n(u)$ as a subtree; the vertices of $A_{n+1}(u)$ are still a subset of $r(\mathrm{pref}(u)\cdot(\mathrm{pref}(e))^*)$; the accept state is still $r(u)$.

The process goes on forever but the three properties mentioned are preserved. Moreover, every element of $r(\mathrm{pref}(u)\cdot(\mathrm{pref}(e))^*)$ will become a vertex of $A_n(u)$ for n large enough. □

Clearly, $\mathrm{pref}(u)\cdot(\mathrm{pref}(e))^*$ is a finite-state language. By Theorem 2.2 (Benois 1969 (see [3]) we conclude that $r(\mathrm{pref}(u)\cdot(\mathrm{pref}(e))^*)$ is also a finite-state language. We give a proof of Benois' theorem, which yields an efficient algorithm. The algorithm below is essentially the same as a slightly more general algorithm of Book and Otto [4, pp. 7–9]. We have simplified the presentation by using finite automata with ε-transitions.

Theorem 2.2 (Benois). *If $L \subseteq (X \cup X^{-1})^*$ is a rational language then $r(L)$ is also a rational language.*

Moreover, there is an algorithm which takes as input a non-deterministic finite automaton N with n states recognizing L, and which outputs a nondeterministic finite automaton N', with $n \cdot (1+2|X|)$ states, recognizing $r(L)$. The time complexity of the algorithm is bounded above by a polynomial in n.

Proof (*algorithm*). We present the algorithm in two parts. Let N be a nondeterministic finite automaton with n states, recognizing a language $L \subseteq (X \cup X^{-1})^*$.

Part 1: We construct a nondeterministic finite automaton N_1 which recognizes the closure of L under the "free-group reductions"; we denote this language $L\uparrow$. Formally $L\uparrow$ is the smallest language $K \subseteq (X \cup X^{-1})^*$ such that (1) $L \subseteq K$, and (2) for all $u, v \in (X \cup X^{-1})^*$, $z \in X \cup X^{-1}$: if $uzz^{-1}v \in K$ then $uv \in K$.

The automaton N_1 is obtained from N by the following procedure:

begin start with the state-transition table of N;
repeat scan the current state-transition table, and for all pairs of states p, q such that $p \xrightarrow{zz^{-1}} q$ (with $z \in X \cup X^{-1}$), add the ε-transition $p \xrightarrow{\varepsilon} q$; "eliminate" all ε-transitions (as, e.g., in [5], p. 24]), but also keep the ε-transitions in place;
until no *new* transitions are introduced into the table, during the last pass;
now drop all ε-transitions **end**.

Let N_1 be defined by the state-transition table obtained this way. No new states are introduced (N and N_1 have the same states).

This is a polynomial-time algorithm: scanning the transition table once takes time $O(n^2 \cdot |X \cup X^{-1}|)$; "elimination" of ε-transitions takes polynomial time (see e.g. [5], p. 24]); there are at most $n^2 \cdot |X \cup X^{-1}|$ iterations of the *repeat* loop, since an automaton with n states has at most $n^2 \cdot |X \cup X^{-1}|$ state transitions.

Part 2: To obtain $r(L)$, we intersect $L\uparrow$ with the language $r((X \cup X^{-1})^*)$; observe that $r((X \cup X^{-1})^*$ is simply the complement of $(X \cup X^{-1})^* \cdot \{zz^{-1}: z \in X \cup X^{-1}\} \cdot (X \cup X^{-1})^*$. Thus, $r((X \cup X^{-1})^*$ is recognized by a (deterministic) finite automaton with $1 + |X \cup X^{-1}|$ states (a start state, and states to remember the last letter read).

One obtains a nondeterministic automaton N' for $r(L) = L\uparrow \cap r((X \cup X^{-1})^*)$ by using the cartesian product of N_1 and the above automaton (see [5, pp. 59–60] and observe that the construction there also works in the nondeterministic case). This is done in polynomial time; the new nondeterministic finite automaton has $n \cdot (|X \cup X^{-1}| + 1)$ states, and recognizes $r(L)$.

The algorithm is quite intuitive, and we omit its correctness proof (see [4]). □

Remark. The proof of Benois' theorem as it appears in [3] (see also [8]) also leads to a polynomial-time algorithm. The algorithm given here seems to be simpler and faster;

a more detailed analysis (carried out in [4]) shows that the running time is $O(n^4)$. Benois and Sakarovitch [2] improved the algorithm to obtain a time complexity of $O(n^3)$.

The decision criterion of Theorem 1.1 (in combination with an algorithmic version of Benois' theorem) can be turned into a polynomial-time algorithm to solve the word problem (and, in fact, the more general problem, where e is also an input).

Theorem 2.3. *The algorithm below, on input* $u, v, e \in (X \cup X^{-1})^*$ *(where* e *represents an idempotent of* $\mathrm{FIM}(X)$*), decides whether* u *is equal to* v *in* $\mathrm{INV}\langle X : e = 1\rangle$*. The time complexity is bounded above by a polynomial (of degree* 3*) in the input length* $|u| + |v| + |e|$.

Algorithm. (1) We first check, given $u, v \in (X \cup X^{-1})^*$, whether $r(u) = r(v)$. For this we compute $r(u)$ (and similarly $r(v)$) as follows: A deterministic push-down automaton reads u from left to right; initially the stack is empty. The machine works as follows: (1) whenever the stack is empty, the next letter of u is pushed on the stack; (2) if the top letter of the stack is $y \in X \cup X^{-1}$ and the next letter of u is y^{-1}, then y is popped off the stack (when y^{-1} is read); (3) if the top letter of the stack is $y \in X \cup X^{-1}$ and the next letter of u is z, with $z \neq y^{-1}$, then (when z is read) z is pushed on the stack. One can see that after u has been entirely read, the stack content is the string $r(u)$. This machine makes $|u|$ steps to compute $r(u)$. Once we have $r(u)$ and $r(v)$, one can check in linear time whether they are equal.

(2) The second (and main) part of the algorithm checks condition (2′) of Theorem 1.1.

Step 1: Build a finite automaton recognizing $r(\mathrm{pref}(e))$. This is done by constructing the Munn tree of e viewed as a finite automaton (with alphabet $X \cup X^{-1}$) but leaving out those edges that point in the direction of the root (the two roots are equal since in $\mathrm{FIM}(X)$, e is an idempotent). The root is the start state; all vertices are made accept states. This automaton has $\leqslant |e| + 1$ states; it can be constructed from e in time $O(|e|)$.

From this automaton one easily obtains a nondeterministic finite automaton (also with $\leqslant |e| + 1$ states) recognizing $(r(\mathrm{pref}(e)))^*$. This is done by connecting every accept state (i.e. every state, in this case) to the start state via an ε-transition (see e.g. [5, p. 24]); the ε-transitions can be eliminated without increasing the number of states (see [5, pp. 26–27]). All this will take time $O(|e|)$.

In a similar way, one constructs a finite automaton recognizing $r(\mathrm{pref}(u))$, with $\leqslant |u| + 1$ states: one first constructs the Munn tree of u, viewed as a finite automaton, but one leaves out the edges that point in the direction of the initial root. The initial root is the start state, and all vertices are accept states (the final root of the Munn tree plays no special role here). This takes time $O(|u|)$.

Finally, from the automata constructed for $(r(\mathrm{pref}(e)))^*$, resp. $r(\mathrm{pref}(u))$, one obtains a nondeterministic finite automaton for the concatenation $r(\mathrm{pref}(u)) \cdot (r(\mathrm{pref}(e)))^*$, with $\leqslant |u| + |e| + 2$ states; the classical constructions will work (see [5, pp. 26–27, 31). This takes time $O(|u| + |e|)$.

At the end of step 1 we have a nondeterministic finite automaton with $\leqslant |e|+|u|+2$ states, which recognizes $r(\text{pref}(u))\cdot(r(\text{pref}(e)))^*$. It took time $O(|u|+|e|)$ to construct this automaton.

In the same way one deals with $r(\text{pref}(v))\cdot(r(\text{pref}(e)))^*$.

Step 2: We apply our algorithmic proof of Benois' theorem, which yields a nondeterministic finite automaton with $(|u|+|e|+2)(|X\cup X^{-1}|+1)$ states, recognizing $r(r(\text{pref}(u))\cdot(r(\text{pref}(e)))^*)=r(\text{pref}(u)\cdot(\text{pref}(e))^*)$; the time complexity is a polynomial in $|u|+|e|$ (more precisely, the time is $O((|e|+|u|)^3)$, by [2]). For v one proceeds similarly (and the time is $O((|e|+|v|)^3)$).

Step 3: Since we have a nondeterministic finite automaton N_u (with $O(|u|+|e|)$ states) for $r(\text{pref}(u)\cdot(\text{pref}(e))^*)$, we can check in polynomial time (as a function of $|u|+|e|+|v|$) whether N_u accepts $r(v')$, for any prefix v' of v (where $r(v')$ is computed as in part 1). This is done as follows: start with the start state of N_u, read the first letter of $r(v')$, and remember the set of states reached; in general, remember a set of states of N_u, read the next letter of $r(v')$, compute the set of states reached, and replace the old set by the new one. (Note that this is not the classical subset construction, which would take exponential time in general, but a "lazy" form of it: only one set of states is computed and remembered at every step, and one uses only those sets that appear as $r(v')$ is processed.)

All this takes polynomial time (each set has size $\leqslant |u|+|e|+2$, and $r(v')$ has length $\leqslant |v'|$; so, step 3 takes time $O(|v'|(|u|+|e|))$. Since v has $|v|+1$ prefixes, checking this for every prefix of v the time is bounded by $O(|v|^2(|u|+|e|))$.

One proceeds similarly when u, v are switched; the time complexity then is $O(|u|^2(|v|+|e|))$.

Finally, adding up all the running times gives $O((|u|+|e|+|v|)^3)$. □

3. Relation with context-free languages

For a presentation $(X, e=1)$, where e represents an idempotent of $\text{FIM}(X)$, we can characterize the congruence class $u\tau\subseteq(X\cup X^{-1})^*$ for every word u, and we shall prove that $u\tau$ is a deterministic context-free language. Here and in the sequel we identify $\text{FG}(X)$ and $r(X\cup X^{-1})^*$ (the set of reduced words); we also identify the element $u\tau$ of $\text{INV}\langle X:e=1\rangle$ and the congruence class of u (which is a subset of $(X\cup X^{-1})^*$). By $V(\text{MT}(u))$ and $V(A(u))$ we denote the set of vertices of $\text{MT}(u)$ and $A(u)$ respectively.

Definition 3.1. Let $P=r(\text{pref}(e))$ be the set of vertices of $\text{MT}(e)$, and let $\langle P\rangle$ be the submonoid of $\text{FG}(X)$ generated by P. Let u be a word $\in(X\cup X^{-1})^*$. For two vertices g, h of $A(u)$, we write $g\to h$ iff $h\in g\cdot\langle P\rangle$; here $\cdot$ denotes the multiplication in $\text{FG}(X)$. We write $g\leftrightarrow h$ iff $g\to h$ and $h\to g$; it is clear that $\leftrightarrow$ is an equivalence relation on the set of vertices of $A(u)$. The equivalence class of g is denoted $[g]_u$.

An equivalence class $[g]_u$ is *essential* for u iff
(1) $[g]_u \cap V(\mathrm{MT}(u)) \neq \emptyset$, and
(2) $\forall g_1 \in V(\mathrm{MT}(u))$: if $g_1 \to g$ then $g \to g_1$.

Lemma 3.2. *For each vertex g of $\mathrm{MT}(u)$ there exists at least one vertex h of $\mathrm{MT}(u)$ such that $[h]_u$ is essential for u and $h \to g$. In particular, $A(u)$ contains at least one essential equivalence class.*

Proof. Let g be any vertex in $\mathrm{MT}(u)$. If $[g]_u$ is essential, there is nothing to prove. If not, there exists $g_1 \in V(\mathrm{MT}(u))$ such that $g_1 \not\to g$ and $g_1 \to g$. If $[g_1]_u$ is essential, we are done. If not, there exists $g_2 \in V(\mathrm{MT}(u))$ such that $g_2 \not\to g_1$ and $g_2 \to g_1$. Then $[g_2]_u \neq [g]_u$ or else $g_2 \to g_1 \to g \to g_2$, whence $[g_2]_u = [g_1]_u$. Continuing in this manner, we obtain a sequence $g_n \to g_{n-1} \to \cdots \to g_2 \to g_1 \to g_0 = g$ such that no $[g_i]_u$ is essential and $[g_i]_u \neq [g_j]_u$ if $i \neq j$. In particular, the elements $g_0, g_1, \ldots, g_n$ are all distinct and all in $V(\mathrm{MT}(u))$, which is finite, so the length of any such sequence is bounded. Hence, we must eventually reach some element $g_m \in V(\mathrm{MT}(u))$ such that $[g_m]_u$ is essential and $g_m \to g_{m-1} \to \cdots \to g_0 = g$, and so $g_m \to g$. □

Theorem 3.3. *Let $M = \mathrm{INV}\langle X : e = 1\rangle$ and let $u \in (X \cup X^{-1})^*$. For $w \in (X \cup X^{-1})^*$, we have $w \in u\tau$ if and only if $\mathrm{MT}(w)$ embeds into $A(u)$ (both viewed within $\Gamma(X)$), $r(w) = r(u)$ and $\mathrm{MT}(w) \cap [g]_u \neq \emptyset$ whenever $[g]_u$ is essential for u. (Equivalently, $w \in u\tau$ if and only if w labels a walk π from 1 to $r(u)$ in $A(u)$ such that π contains at least one vertex from each essential equivalence class $[g]_u$ contained in $A(u)$.)*

Proof. Suppose that $\mathrm{MT}(w) \subseteq A(u)$, $r(w) = r(u)$ and $\mathrm{MT}(w)$ intersects each essential equivalence class of $A(u)$. Clearly, $A(w) \subseteq A(u)$, by the Stephen construction process, since $\mathrm{MT}(w) \subseteq A(u)$. We need to show conversely that $A(u) \subseteq A(w)$, in order to obtain $w \in u\tau$. Again (by Theorem 1.2(2)) it suffices to show that $\mathrm{MT}(u) \subseteq A(w)$. Let $g \in V(\mathrm{MT}(u))$. If $[g]_u$ is essential for u then there exists $g_1 \in [g]_u \cap \mathrm{MT}(w)$ and since $g_1 \to g$, this forces $g \in V(A(w))$. If $[g]_u$ is not essential for u then, by Lemma 3.2, there exists $h \in V(\mathrm{MT}(u))$ such that $[h]_u$ is essential for u and $h \to g$. By the above argument this forces $h \in V(A(w))$ and so $g \in V(A(w))$ since $h \to g$. Hence, $\mathrm{MT}(u) \subseteq A(w)$ and so $w \in u\tau$.

Conversely, suppose that $w \in u\tau$. Then $A(u) = A(w)$ by Theorem 1.2 (1), so $r(u) = r(w)$ and $\mathrm{MT}(w) \subseteq A(u)$. Let $g \in V(\mathrm{MT}(u))$ such that $[g]_u$ is essential for u. Now $g \in V(A(u)) = V(A(w))$ so there exists $g_1 \in V(\mathrm{MT}(w))$ such that $g_1 \to g$. By Lemma 3.2, there exists $h \in V(\mathrm{MT}(w))$ such that $[h]_w$ is essential for w and $h \to g_1$, from which it follows that $h \to g$. By the same argument, since $[h]_w$ is essential for w and $h \in V(\mathrm{MT}(w))$ and $A(w) = A(u)$, we see that there exists $h_1 \in V(\mathrm{MT}(u))$ such that $[h_1]_u$ is essential for u and $h_1 \to h$. Then $h_1 \to h \to g$ and both $[h_1]_u$ and $[g]_u$ are essential for u; so, this forces $[h_1]_u = [g]_u$. But $[h_1]_u = [h_1]_w$ since $A(u) = A(w)$ so we must have $[h_1]_u = [h]_u = [g]_u$. Then it follows that $[g]_u \cap V(\mathrm{MT}(w)) \neq \emptyset$ since $h \in [g]_u \cap V(\mathrm{MT}(w))$. This completes the proof of the theorem. □

Theorem 3.3 may be viewed as an extension of the result of Munn [9] characterizing the equivalence class of a word $u \in (X \cup X^{-1})^*$ with respect to $\mathrm{FIM}(X)$ as the set of words in $(X \cup X^{-1})^*$ labeling walks from 1 to $r(u)$ that traverse all vertices of $\mathrm{MT}(u)$. We will now prove that every congruence class relative to a presentation of the form $M = \mathrm{INV}\langle X : e = 1\rangle$ is a deterministic context-free language. We first need a preliminary result about context-free languages.

Theorem 3.4. *Let $\boldsymbol{M}$ be a push-down automaton with input alphabet Σ and stack alphabet Γ, let $\{L_1, \ldots, L_k\}$ be a finite set of rational languages over the alphabet Γ; let L_∞ also be a rational language over Γ, and let f be a word in L_∞. We assume that L_∞ is closed under prefix (i.e., when a word is in L_∞, all its prefixes are also in L_∞). Then the set L' defined below is context-free:*

$L' = \{w \in \Sigma^*$: *there exists an accepting computation of $\boldsymbol{M}$ on input w such that*

(1) *for each i $(1 \leqslant i \leqslant k)$ the stack content of $\boldsymbol{M}$ is a word in L_i at least once during the computation;*

(2) *the stack content of $\boldsymbol{M}$ always belongs to L_∞ during the computation;*

(3) *the stack content at the end of the computation is $f\}$.*

Moreover, if $\boldsymbol{M}$ is a deterministic push-down automaton then L' is a deterministic context-free language.

Proof. See [5] for terminology not defined here. Let $\boldsymbol{M}$ be described by Q (states), Γ (stack alphabet), Σ (input alphabet), δ (transition relation), q_0 (start state), F (accept states), z_0 (initial stack symbol); $\boldsymbol{M}$ "accepts by final state". Let L_i be accepted by the deterministic complete finite automaton A_i (for $1 \leqslant i \leqslant k$ or $i = \infty$), described by Q_i (states), Γ (input alphabet, which is also the stack alphabet of $\boldsymbol{M}$), δ_i (transition function), q_{0i} (start state), F_i (accept states). Since L_∞ is closed under prefix we can also assume that $Q_\infty = F_\infty \cup \{s\}$, where s is a sink state.

We shall construct a pda $\boldsymbol{M}'$ accepting L'. If $\boldsymbol{M}$ is deterministic, then $\boldsymbol{M}'$ will also be deterministic. At first we will ignore the condition that the final stack content must be the word f; we will handle this at the end of the proof.

$\boldsymbol{M}'$ is described as follows: $\boldsymbol{M}' = (Q', \Sigma', \Gamma', \delta', q_0', Z_0', F')$, where the set of states is $Q' = (Q \cup Q \times \{c\}) \times \{0,1\}^k$ (here $\{0,1\}^k$ is the set of all strings of 0's and 1's of length k); the stack alphabet is $\Gamma' = \Gamma \times Q_1 \times \cdots \times Q_k \times F_\infty$; $\Sigma' = \Sigma$ (the same as for the original pda $\boldsymbol{M}$) is the input alphabet; $Z_0' = (z_0, q_1, \ldots, q_k, q_\infty)$ is the initial stack symbol, where $q_i = \delta_i(q_{0i}, z_0)$ for every i; the start state is $q_0' = (q_0, v_1, \ldots, v_k)$, where $v_i = 1$ if $z_0 \in L_i$, and $v_i = 0$ if $z_0 \notin L_i$: the set of accept states is $F' = F \times \{1^k\}$ (where 1^k is the all-1 string of length k). Note that Z_0' does not exist if $z_0 \notin L_\infty$ (and L' is the empty language in that case).

Before describing rigorously the transition relation of $\boldsymbol{M}'$ let us briefly say how $\boldsymbol{M}'$ is intended to work: $\boldsymbol{M}'$ imitates $\boldsymbol{M}$ (in the first coordinate of both the state and the stack symbols). But at the same time, if $\boldsymbol{M}$ pushes stack symbols on its stack, $\boldsymbol{M}'$ applies these symbols to A_i (for $1 \leqslant i \leqslant k$) and pushes the resulting states on the stack

(side by side with what M does). When M pops symbols off the stack, M' also pops those symbols off its stack, including the accompanying state symbols of A_i; that way M' uncovers the earlier states of A_i (that had previously been pushed on the stack). M' accepts iff M accepts and if in addition for every i (with $1 \leqslant i \leqslant k$), the top of the stack was ever of the form $(z, q_1, \ldots, q_k, q_\infty)$, where $q_i \in F_i$ (the ith position of the string in $\{0,1\}^k$ that is part of the state of M' serves to remember this). Moreover, for M' to accept, the top symbol of the stack must always be defined (i.e., $q_\infty \in F_\infty$); this enforces the condition that the stack content of M must always be in L_∞ during the accepting computation.

We will use the short-hand notation $\delta_i(q, z_1 z_2 \ldots z_n)$ to denote $\delta_i(\ldots(\delta_i(\delta_i(q, z_1), z_2), \ldots), z_n)$.

Let us now define δ'. Suppose $(q, z, a) \to (q', \gamma)$ is a transition of M (where $a \in \Sigma \cup \{\varepsilon\}$, q and q' belong to Q, $z \in \Gamma$ is the symbol on top of the stack; in the transition z is replaced by $\gamma = z_1 \ldots z_n \in \Gamma^*$, where z_n is now the top of the stack).

Case 1: If $\gamma \neq \varepsilon$ (here ε denotes the empty word) then into δ' we put the set of transitions

$$\{((q, u_1, \ldots, u_k), (z, q_1, \ldots, q_k, q_\infty), a) \to ((q', u_1', \ldots, u_k'), \gamma'):$$

$$(u_1, \ldots, u_k) \in \{0,1\}^k, (q_1, \ldots, q_k, q_\infty) \in Q_1 \times \cdots \times Q_k \times F_\infty,$$

$$\gamma' = (z_1, \delta_1(q_1, z_1), \ldots, \delta_k(q_k, z_1), \delta_\infty(q_\infty, z_1))(z_2, \delta_1(q_1 z_1, z_2),$$

$$\ldots, \delta_k(q_k, z_1 z_2), \delta_\infty(q_\infty, z_1 z_2))$$

$$\ldots (z_n, \delta_1(q_1, z_1 z_2 \ldots z_n), \ldots, \delta_k(q_k, z_1 z_2 \ldots z_n),$$

$$\delta_\infty(q_\infty, z_1 z_2 \ldots z_n)),$$

and $u_i' = 1$ if $u_i = 1$ or $\delta_i(q_1, z_1 z_2 \ldots z_n) \in F_i$: $u_i' = 0$ otherwise$\}$.

Note: We use the right-most symbol to denote the top of the stack. Observe that, by the definition of the stack alphabet Γ' of M', γ' is not defined if not all of the $\delta_\infty(q_\infty, z_1 z_2 \ldots z_j)$ belong to F_∞.

Case 2: If $\gamma = \varepsilon$ (i.e., M pops the top symbol off its stack) then into δ' we put the set of transitions

$$\{((q, u_1, \ldots, u_k), (z, q_1, \ldots, q_k, q_\infty), a) \to (((q', c), u_1, \ldots, u_k), \varepsilon):$$

$$(u_1, \ldots, u_k) \in \{0,1\}^k, (q_1, \ldots, q_k, q_\infty) \in Q_1 \times \cdots \times Q_k \times F_\infty\}$$

$$\cup \{(((q', c), v_1, \ldots, v_k), (y, p_1, \ldots, p_k, p_\infty), \varepsilon) \to ((q', u_1', \ldots, u_k'), (y, p_1, \ldots, p_k, p_\infty)):$$

$$(v_1, \ldots, v_k) \in \{0,1\}^k, (y, p_1, \ldots, p_k, p_\infty) \in \Gamma \times Q_1 \times \cdots \times Q_k \times F_\infty,$$

and $u_i' = 1$ if $v_i = 1$ or $p_i \in F_i, u_i' = 0$ otherwise$\}$.

Comment. The first transition pops off the top of the stack, goes to the next state q', but marks that state with a "c" to indicate that the new top of the stack has to be examined. The second set of transitions only applies when the state is marked by a "c";

it removes the "c", and modifies the $\{0,1\}$-string (in the state) if the new top of the stack indicates acceptance by the A_i automata (i.e., $p_i \in F_i$).

It is straightforward, although tedious, to verify that $\boldsymbol{M}'$ recognizes L' (except that the condition that the final stack content should be f has been ignored). To enforce this condition, we consider the pda $\boldsymbol{M}'$ (constructed so far) to be a pda that can read the top $|f|$ symbols of the stack (rather than just the top-most symbol); when the stack contains less than $|f|$ symbols the pda can see the entire stack content. Such a pda can be simulated by an ordinary pda, which only sees the top-most stack symbol, but which remembers the top $|f|$ symbols of the stack in its finite control. Determinism is preserved. The details of this construction are a classical exercise. When $\boldsymbol{M}'$ is modified according to this construction we obtain a pda (let us still call it $\boldsymbol{M}'$) which recognizes the language L'. □

As a consequence of Theorem 3.4, we obtain the following extension of Theorem 4.4 of [8] in the case of a presentation $(X, e=1)$. Theorem 4.4 of [8] considers the more general case of a presentation $(X, \{e_i = f_i : i \in I\})$ where each e_i and f_i represents an idempotent of $\mathrm{FIM}(X)$; it states that $u\uparrow$ is a deterministic context-free language, for every $u \in (X \cup X^{-1})^*$ (see Section 1 for a definition of $u\uparrow$). It is still an open problem whether $u\tau$ itself is context-free in this general case.

Theorem 3.5. *Let* $\boldsymbol{M} = \mathrm{INV}\langle X : e=1\rangle = (X \cup X^{-1})^*/\tau$, *where* e *represents an idempotent in* $\mathrm{FIM}(X)$. *Then for each* $u \in (X \cup X^{-1})^*$, $u\tau$ *is a deterministic context-free language.*

Proof. Let u be a fixed word (as in the theorem). Let $\boldsymbol{M}$ be the dpda which computes the reduction function r (i.e., after $\boldsymbol{M}$ has read an input $v \in (X \cup X^{-1})^*$ the stack contains $r(v)$). Let us apply Theorem 3.4 to the dpda $\boldsymbol{M}$, with $k = |u| + 1$, $f = r(u)$, and with the following rational languages: $L_\infty = r(\mathrm{pref}(u) \cdot (\mathrm{pref}(e))^*)$, and $L_i = r(u_{i-1} \cdot (\mathrm{pref}(e))^*)$, where u_{i+1} is the prefix of u of length i $(1 \leqslant i \leqslant k)$. We claim that the language L' recognized by the dpda $\boldsymbol{M}'$ is precisely $u\tau$. Indeed, by Theorem 2.1, $v \in r(u)$ iff (1) $r(v) = r(u)$, and (2′) every prefix v' of v belongs to L_∞, and every prefix u' of u belongs to $r(\mathrm{pref}(v) \cdot (\mathrm{pref}(e))^*)$. The first condition is equivalent to the condition that the final stack content of $\boldsymbol{M}'$ on input v must be $f = r(u)$; the first part of condition (2′) means that the stack content of $\boldsymbol{M}'$ must always be in L_∞; finally, it is not difficult to see that the second condition of (2′) means exactly that for every i $(0 \leqslant i \leqslant |u|)$, the stack content belongs to L_{i+1} at least once during the computation of $\boldsymbol{M}'$ on input v. □

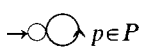
4. The presentation $(X: e=1)$ when e represents a positively labeled idempotent of the free inverse monoid

In Section 2 we gave a fast algorithm for solving the word problem of the presentation $(X, e=1)$ when e represents any idempotent of $\mathrm{FIM}(X)$. This was much simpler

than the more general case considered in [8]. Further simplifications in the solution to the word problem occur for $M = \mathrm{INV}\langle X: e = 1\rangle$ when e is "positively labeled".

Definition 4.1. An idempotent of $\mathrm{FIM}(X)$ represented by a word e is *positively labeled* iff every vertex of the Munn tree $\mathrm{MT}(e)$ is in X^*.

In this case it is clear that the set of vertices of $A(1)$ is just P^*, the submonoid of $(X \cup X^{-1})^*$ generated by the set $P = r(\mathrm{pref}(e)) = V(\mathrm{MT}(e))$; this follows immediately in the positively labeled case since all words in $P^*(\subseteq X^*)$ are already reduced. The *flower automaton*



can be used to recognize P^* and the complications inherent in constructing the Benois automaton of $r(P^*) = P^*$ do not arise in this case; so, the solution to the word problem is particularly simple here. The flower automaton is not deterministic in general but standard automata-theoretic methods can be used to easily construct the minimal automaton of P^* from the flower automaton.

An alternative solution, involving the technique of "*pruning*" *Munn trees* may also be employed here. This section gives a brief description of this technique since it yields a particularly pleasant canonical form for words in this case. For the remainder of the section, e is a positively labeled idempotent of $\mathrm{FIM}(X)$, $M = \mathrm{INV}\langle X: e = 1\rangle$, and $P = r(\mathrm{pref}(e)) = V(\mathrm{MT}(e)) \subseteq X^*$ is the set of vertices of $\mathrm{MT}(e)$.

Definition 4.2. (1) Let $(1, T, \omega)$ be a finite birooted labeled subtree of $\Gamma(X)$ (with initial label 1 and terminal label ω). An *extremal* vertex γ of T is called *prunable* if $\gamma \neq 1, \gamma \neq \omega$ and there is some vertex $\delta \neq \gamma$ in T such that the word w labeling the geodesic from δ to γ is in P.

(2) If γ is a prunable vertex of T then we can form a new birooted labeled subtree $(1, T', \omega)$ of $\Gamma(X)$ by removing from T the vertex γ and the edge that connects γ to the remainder of T: we say that $(1, T', \omega)$ is obtained from $(1, T, \omega)$ by *pruning* the vertex γ and write $(1, T, \omega) \Rightarrow (1, T', \omega)$. We write $(1, T, \omega) \overset{*}{\to} (1, T', \omega)$ if there is a finite sequence of prunings $(1, T, \omega) \Rightarrow (1, T_1, \omega) \Rightarrow \cdots \Rightarrow (1, T_n, \omega) = (1, T', \omega)$.

Lemma 4.3 (Pruning is confluent). *If* $(1, T_1, \omega)$ *and* $(1, T_2, \omega)$ *are each obtained by pruning one vertex from* $(1, T, \omega)$, *then there is another birooted labeled tree* $(1, T', \omega)$ *such that* $(1, T_1, \omega) \Rightarrow (1, T', \omega)$ *and* $(1, T_2, \omega) \Rightarrow (1, T', \omega)$. *As a consequence, for each finite birooted labeled subtree* $(1, T, \omega)$ *of* $\Gamma(X)$, *there is a unique birooted labeled subtree* $(1, T', \omega)$ *of* $\Gamma(X)$ *such that* T' *has no prunable vertices and* $(1, T, \omega) \overset{*}{\Rightarrow} (1, T', \omega)$.

We call T' *the pruned tree* obtained from T.

Proof. Suppose that $\gamma_1 \neq \gamma_2$ are prunable vertices of T, and that $(1, T, \omega) \Rightarrow (1, T_1, \omega)$ and $(1, T, \omega) \Rightarrow (1, T_2, \omega)$ by pruning γ_1 and γ_2, respectively. Then there exist vertices

δ_1, δ_2 of T such that the geodesic from δ_1 (δ_2) to γ_1 (γ_2) is labeled by a word in P. Let μ_1 (μ_2) be the vertex of T adjacent to γ_1 (γ_2). Then the edge form μ_1 to γ_1 (μ_2 to γ_2) is labeled by a letter in X and, so, it follows that $\gamma_1 \neq \delta_2$ (or else the geodesic from δ_2 to γ_2 would not be labeled by a positive word); similarly $\gamma_2 \neq \delta_1$. Thus δ_2 is a vertex in T_1 and δ_1 is a vertex in T_2; so, γ_2 is a prunable vertex in T_1 and γ_1 is a prunable vertex in T_2. Pruning γ_2 from T_1 yields the same tree T' as pruning γ_1 from T_2; hence pruning is confluent. □

With these notations and results in hand we are able to provide an alternate solution to the word problem for $M = \mathrm{INV}\langle X : e = 1\rangle$ in the positively labeled case.

Theorem 4.4. *Let* $M = \mathrm{INV}\langle X : e = 1\rangle = (X \cup X^{-1})^*/\tau$, *where* e *represents a positively labeled idempotent of* $\mathrm{FIM}(X)$. *If* $u, v \in (X \cup X^{-1})^*$ *then* $u = v$ *in* M *if and only if* $\mathrm{MT}'(u) = \mathrm{MT}'(v)$, *where* $\mathrm{MT}'(u)$, *and* $\mathrm{MT}'(v)$ *are the pruned trees obtained from* $\mathrm{MT}(u)$, *and* $\mathrm{MT}(v)$, *respectively.*

Proof. Suppose first that $u_1, v_1 \in (X \cup X^{-1})^*$ with $(1, \mathrm{MT}(u_1), r(u_1)) \Rightarrow (1, \mathrm{MT}(v_1), r(v_1))$; so, $r(u_1) = r(v_1)$ and $\mathrm{MT}(v_1)$ is obtained from $\mathrm{MT}(u_1)$ by pruning one extremal vertex γ from $\mathrm{MT}(u_1)$. There is a vertex δ in $\mathrm{MT}(u_1)$ such that the geodesic from δ to γ is labeled by an element of P. Hence $\mathrm{MT}(v_1) \cup \delta \cdot \mathrm{MT}(e)$ contains the vertex γ and so $\mathrm{MT}(v_1) \cup \delta \cdot \mathrm{MT}(e) = \mathrm{MT}(u_1) \cup \delta \cdot \mathrm{MT}(e)$. It follows that $A(u_1) = A(v_1)$ and hence $u_1 = v_1$ in M. By induction, if u_2 is as word in $(X \cup X^{-1})^*$ whose Munn tree is $\mathrm{MT}'(u)$, then $u = u_2$ in M, and it follows that $u = v$ in M if $\mathrm{MT}'(u) = \mathrm{MT}'(v)$.

To prove the converse, suppose first that $u_1, v_1 \in (X \cup X^{-1})^*$ with $u_1 = w_1 w_2$ and $v_1 = w_1 e w_2$ in $(X \cup X^{-1})^*$. Then there is some vertex δ of $\mathrm{MT}(u_1)$ such that $\mathrm{MT}(v_1) = \mathrm{MT}(u_1) \cup \delta \cdot \mathrm{MT}(e)$. If γ is an extremal vertex of $\mathrm{MT}(v_1)$ that is not in $\mathrm{MT}(u_1)$, then γ is prunable (since the geodesic from δ to γ is labeled by an element of P). By induction, all extremal vertices of $\mathrm{MT}(v_1)$ that are not in $\mathrm{MT}(u_1)$ may be pruned away and so $\mathrm{MT}(v_1) \overset{*}{\Rightarrow} \mathrm{MT}(u_1)$. It follows by induction that if $u = v$ in M, then there is a sequence of words $u = u_0, u_1, u_2, \ldots, u_n = v$ such that, for each $i = 0, 1, \ldots, n$, either $\mathrm{MT}(u_i) \overset{*}{\Rightarrow} \mathrm{MT}(u_{i+1})$ or $\mathrm{MT}(u_{i+1}) \overset{*}{\Rightarrow} \mathrm{MT}(u_i)$. The confluence Lemma 4.3 then implies that $\mathrm{MT}'(u) = \mathrm{MT}'(v)$. □

Corollary 4.5. *Let* $M = \mathrm{INV}\langle X : e = 1\rangle$, *where* e *represents a positively labeled idempotent of* $\mathrm{FIM}(X)$. *Then each congruence class in* $\mathrm{FIM}(X)$ *with respect to this presentation contains a maximum element with respect to the natural partial order on* $\mathrm{FIM}(X)$.

Proof. Let $u \in (X \cup X^{-1})^*$ and let v be any word in $(X \cup X^{-1})^*$ such that $\mathrm{MT}(v) = \mathrm{MT}'(u)$. Since $\mathrm{MT}(v) \subseteq \mathrm{MT}(u)$ and $r(u) = r(v)$ it follows from the results of Munn [9] that $u \leqslant v$ in $\mathrm{FIM}(X)$. Also $u = v$ in M from the proof of Theorem 3.3, so u and v are congruent as elements of $\mathrm{FIM}(X)$. It follows that v is a maximum element in its congruence class in $\mathrm{FIM}(X)$. □

We can also obtain a simpler characterization of the τ-class of a word $u \in (X \cup X^{-1})^*$ in this case.

Theorem 4.6. *Let* $M = \mathrm{INV}\langle X : e = 1\rangle = (X \cup X^{-1})^*/\tau$ *where* e *represents a positively labeled idempotent of* $\mathrm{FIM}(X)$. *Then for each* $w \in (X \cup X^{-1})^*$ *we have* $w \in u\tau$ *if and only if* $r(u) = r(w)$ *and* $\mathrm{MT}'(u) \subseteq \mathrm{MT}(w) \subseteq A(u)$.

Proof. Note that the concept of an "essential" equivalence class $[g]_u$ of $A(u)$ as introduced in Definition 3.1 simplifies considerably in this case. Since $P = V(\mathrm{MT}(e)) \subseteq X^*$, we now have $g \leftrightarrow h$ if and only if $g = h$. Hence, $[g]_u = \{g\}$ for all $g \in V(A(u))$. In analogy with Definition 3.1, we call a vertex $g \in V(\mathrm{MT}(u))$ *essential* iff there is no vertex $g_1 \neq g$ in $\mathrm{MT}(u)$ such that $g_1 \to g$. Theorem 3.3 then tells us that $w \in u\tau$ if and only if $r(u) = r(w)$, $\mathrm{MT}(w) \subseteq A(u)$, and $\mathrm{MT}(w)$ contains every essential vertex of $\mathrm{MT}(u)$. It is clear that a prunable vertex of $\mathrm{MT}(u)$ is not essential and that all extremal vertices of $\mathrm{MT}'(u)$ are essential; so the result follows since $\mathrm{MT}(w)$ contains all vertices of $\mathrm{MT}'(u)$ (and hence all essential vertices of $\mathrm{MT}(u)$) if and only if $\mathrm{MT}(w)$ contains all extremal vertices of $\mathrm{MT}'(u)$). □

Note that Theorem 4.6 implies the well-known result of Munn [9] that for any $u, w \in (X \cup X^{-1})^*$: u and w are equal in $\mathrm{FIM}(X)$ if and only if w labels a walk from 1 to $r(u)$ in $\mathrm{MT}(u)$ that includes all vertices of $\mathrm{MT}(u)$.

Let us now put these results into algorithmic form, in order to decide the word problem. We use Theorem 4.6: Given two words $u, v \in (X \cup X^{-1})^*$ and a positively labeled idempotent e of $\mathrm{FIM}(X)$, we first prune the trees $\mathrm{MT}(u), \mathrm{MT}(v)$. Then we check if $r(u) = r(v)$ and if $\mathrm{MT}'(u) = \mathrm{MT}'(v)$. The more detailed algorithms follow.

Part 1: The *pruning algorithm.* On input e and u, it outputs the pruned tree $\mathrm{MT}'(u)$. The algorithm does the following:

(1) Construct the Munn trees of e and u. This takes time $O(|e| + |u|)$.

(2) Compute $r(\mathrm{pref}(e)) = \mathrm{pref}(e) = P$. To do this one traverses the Munn tree of e in breath-first order and outputs any word read so far; this continues until all search paths have ended at a leaf. This takes time $O(|e|)$. Note that P has $\leqslant |e|$ elements.

(3) For every leaf l of the Munn tree $\mathrm{MT}(u)$, compare every element of P with the positively labeled paths in $\mathrm{MT}(u)$ ending at that leaf l. If no such path is equal to an element of $P = r(\mathrm{pref}(e))$, mark this leaf "in red"; otherwise, remove l and the edge incident with l from the tree. This is then continued until all leaves of the tree are marked "red", or until the tree has been entirely removed. The time complexity of part (3) is $O(|e|^2 \cdot |u|)$: Every edge of u is removed at most once; to compare an element of $r(\mathrm{pref}(e))$ with a path in the tree of u takes $\leqslant |e|$ steps, and there are $\leqslant |e|$ elements in $r(\mathrm{pref}(e)) = \mathrm{pref}(e) = P$.

This shows that the pruning algorithm has time complexity $O(|e|^2 \cdot |u|)$; this is linear as a function of $|u|$.

Part 2: Solving the problem whether $u = v$ in $\mathrm{INV}\langle X : e = 1\rangle$. One first computes $r(u)$ and $r(v)$ and checks whether $r(u) = r(v)$; this takes time $O(|u| + |v|)$. Next one

applies the above pruning algorithm to u and v (in time $O(|e|^2 \cdot (|u|+|v|))$, and checks whether the resulting trees are equal (time $O(|u|+|v|)$). Overall, it takes time $O(|e|^2 \cdot (|u|+|v|))$ to solve the problem; this is linear in $|u|+|v|$ (=length of the input for the word problem).

We summarize the algorithmic result as follows.

Theorem 4.7. *When e is a positively labeled idempotent of* $\mathrm{FIM}(X)$ *then the problem* "$u=v$ *in* $\mathrm{INV}\langle X:e=1\rangle$?" *can be solved by an algorithm which has time complexity* $O(|e|^2 \cdot (|u|+|v|))$. *In particular, the word problem (when u, v are inputs, and e is fixed) has linear time complexity.*

Concluding remarks

The result of Theorem 4.4 was announced at a conference at Louisiana State University in honor of R.J. Koch [7] (where the hypothesis that e is positively labeled was omitted). In fact, the pruning technique works in some other cases as well but not in general; it is not too difficult to give an example of an idempotent $e \in \mathrm{FIM}(X)$ for which the pruning technique is not confluent.

Acknowledgments

The authors wish to thank J.B. Stephen for many fruitful conversations concerning parts of the paper. The authors owe special thanks to the referee who suggested a nicer and simpler proof of Theorem 3.5 and who pointed out references [2, 4].

References

[1] J. Barwise, ed., *Handbook of Mathematical Logic* (North-Holland, Amsterdam, 1977).
[2] M. Benois and J. Sakarovitch, On the complexity of some extended word problems defined by cancellation rules, *Inform. Process. Lett.* **23** (1986) 281–287.
[3] J. Berstel, *Transductions and Context-free Languages* (Teubner Studienbücher, Stuttgart, 1979).
[4] R. Book and F. Otto, Cancellation rules and extended word problems, *Inform. Process. Lett.* **20** (1985) 5–11.
[5] J.E. Hopcroft and J.D. Ullman, *Introduction to Automata Theory, Languages, and Computation* (Addison-Wesley, Reading, MA, 1979).
[6] G. Lallement, *Semigroups and Combinatorial Applications* (Wiley, New York, 1979).
[7] S.W. Margolis and J. Meakin, On a class of one-relator inverse monoid presentations, in: *Proc. LSU Semigroup Conf.*, Louisiana State University (1986) 37–41.
[8] S.W. Margolis and J.C. Meakin, Inverse monoids, trees and context-free languages, *Trans. AMS*, to appear.
[9] W.D. Munn, Free inverse semigroups, *Proc. London Math. Soc.* **30** (1974) 385–404.
[10] M. Petrich, *Inverse Semigroups* (Wiley, New York, 1984).
[11] J.B. Stephen, Presentations of inverse monoids, *J. Pure Appl. Algebra* **63** (1990) 81–112.