# Conservative Groupoids Recognize Only Regular Languages

Danny Dubé[1], Mario Latendresse[2], and Pascal Tesson[3]

[1] Université Laval, danny.dube@ift.ulaval.ca
[2] SRI International, latendre@iro.umontreal.ca
[3] Université Laval, pascal.tesson@ift.ulaval.ca

**Abstract.** The notion of recognition of a language by a finite semigroup can be generalized to recognition by finite groupoids, i.e. sets equipped with a binary operation '$\cdot$' which is not necessarily associative. It is well known that $L$ can be recognized by a groupoid iff $L$ is context-free. But it is also known that some subclasses of groupoids can only recognize regular languages. For example, *loops* recognize exactly the regular open languages and Beaudry et al. described the largest class of groupoids known to recognize only regular languages.
A groupoid $H$ is said to be *conservative* if $a \cdot b$ is in $\{a, b\}$ for all $a$, $b$ in $H$. The main result of this paper is that conservative groupoids can only recognize regular languages. This class is incomparable with the one of Beaudry et al. so we are exhibiting a new sense in which a groupoid can be too weak to have context-free capabilities.

## 1 Introduction

A semigroup $S$ is a set with a binary associative operation '$\cdot$'. It is a monoid if it also has an identity element. The algebraic point of view on automata, which is central to some of the most important results in the study of regular languages, relies on viewing a finite semigroup as a language recognizer. This enables one to classify a regular language according to the semigroups or monoids able to recognize it. There are various ways in which to formalize this idea but the following one will be useful in our context: a language $L \subseteq \Sigma^*$ is recognized by a finite monoid $M$ if there is a homomorphism $\phi$ from the free monoid $\Sigma^*$ to the free monoid $M^*$ and a set $F \subseteq M$ such that $w \in L$ iff $\phi(w)$ is a sequence of elements whose product lies in $F$. Since the operation of $M$ is associative, this product is well defined. This framework underlies algebraic characterizations of many important classes of regular languages (see [7] for a survey).

These ideas have been extended to non-associative binary algebras, i.e. groupoids. If a groupoid is non-associative, a string of groupoid elements does not have a well-defined product so the above notion of language recognition must be tweaked. A language $L$ is said to be recognized by a finite groupoid $H$ if there is a homomorphism $\phi$ from the free monoid $\Sigma^*$ to the free monoid $H^*$ and a set $F \subseteq H$ such that $w \in L$ iff the sequence of groupoid elements $\phi(w)$

can be bracketed so that the resulting product lies in $F$. It can be shown that a language can be recognized by a finite groupoid iff it is context-free.

However, certain groupoids are too weak to recognize non-regular languages. The first non-trivial example was provided by Caussinus and Lemieux who showed that loops (groupoids with an identity element and left/right inverses) can only recognize regular languages. Beaudry et al. later showed that the languages recognized by loops are precisely the regular open languages [4]. Along the same lines, Beaudry showed in [3] that if $H$ is a groupoid whose multiplication monoid $\mathcal{M}(H)$ is in the variety **DA** then it can only recognize regular languages. ($\mathcal{M}(H)$ is the transformation monoid generated by the rows of the multiplication table of $H$.) Finally, Beaudry et al. proved that this still holds if the multiplication monoid lies in the larger variety **DO** [5].

A groupoid $H$ is said to be *conservative* if for all $x, y \in H$ we have $x \cdot y = x$ or $x \cdot y = y$. The simplest example of a non-associative, conservative groupoid is the one defined by the Rock-Paper-Scissors game. In this game, two players simultaneously make a sign with their fingers chosen among Rock, Paper, and Scissors. Rock beats Scissors, Scissors beats Paper, Paper beats Rock, and identical signs result in a tie. The associated groupoid has three elements $R, P, S$ and the multiplication is given by $R{\cdot}R = R{\cdot}S = S{\cdot}R = R$; $P{\cdot}P = P{\cdot}R = R{\cdot}P = P$; and $S{\cdot}S = S{\cdot}P = P{\cdot}S = S$. Note that $H$ is indeed conservative and also non-associative since $(R \cdot P) \cdot S = S \neq R = R \cdot (P \cdot S)$. The main result of our paper is that conservative groupoids recognize only regular languages. Our results are incomparable to those of Beaudry et al. Indeed a straightforward calculation shows that the multiplication monoid of the Rock-Paper-Scissors groupoid does not belong to the variety **DO** nor to the larger variety **DS**.

### 1.1 Conservative Groupoids and Tournaments

It is convenient to think of conservative groupoids as defining a generalization of the Rock-Paper-Scissors game. For any conservative groupoid $H$, we define the game in which players 1 and 2 each choose an element of $H$ (say $a$ and $b$ respectively) and player 1 wins iff $a \cdot b = a$. In fact, it is helpful to think of this game as a competition between elements of $H$.

Consider now a sequence $w \in H^*$ of elements of the groupoid. A bracketing of this sequence can be viewed as specifying a tournament structure involving the symbols of $w$, i.e. a specific way to determine a winner among the elements of $w$. For instance, if $w = abcd$, then $(a \cdot b) \cdot (c \cdot d)$ is the tournament that first pits $a$ against $b$ and $c$ again $d$ and then has the two winners of that first round competing. Similarly in the tournament $((a \cdot b) \cdot c) \cdot d$ we first have $a$ facing $b$ with the winner then facing $c$ and the winner of that facing $d$. Note that this analogy makes sense because $H$ is conservative and the "winner" of any such tournament (i.e. the value of the product given this bracketing) is indeed one of the participants (in the above example, one of $a$, $b$, $c$, or $d$). We intend to study languages $\Lambda(x) = \{w \in H^* \mid w \text{ can be bracketed to give } x\}$ and we accordingly think of them as $\Lambda(x) = \{w \in H^* \mid \text{ an organizer can rig a tournament structure for } w \text{ to ensure that } x \text{ wins }\}$.

Let us define the *contest trees*. We denote the set of all contest trees by $\mathcal{T}$. It is the smallest set such that the leaf tree $a$ is in $\mathcal{T}$, for any $a$ in the alphabet $\Sigma$, and the tree $t_1 \otimes t_2$ is also in $\mathcal{T}$, for any two trees $t_1$ and $t_2$ in $\mathcal{T}$. Let $T : \Sigma^+ \to 2^{\mathcal{T}}$ be the function that computes the set of possible contest trees in a given contest.

$$\begin{aligned} T(a) &= \{a\} \\ T(w) &= \{t_1 \otimes t_2 \mid u, v \in \Sigma^+,\ u\,v = w,\ t_1 \in T(u),\ t_2 \in T(v)\}, \qquad \text{if } |w| > 1 \end{aligned}$$

Note that, when performing the left-to-right traversal of a tree in $T(w)$, the leaves that we successively reach are the symbols that form $w$. Next, function $W : \mathcal{T} \to \Sigma$ computes the *winner* of a contest tree.

$$\begin{aligned} W(a) &= a \\ W(t_1 \otimes t_2) &= W(t_1) \cdot W(t_2) \end{aligned}$$

Note that the winner of a contest tree is unique. Next, we define the set of possible *winners* in a given contest $w$ by overloading function $W$ with an additional definition of type $\Sigma^+ \to 2^{\Sigma}$. We define $W(w)$ as $\{W(t) \mid t \in T(w)\}$. Finally, for $a \in \Sigma$, we denote by $\Lambda(a)$ the *language* of the words for which we can arrange a contest in which $a$ is the winner. We can give a more formal, alternative definition of $\Lambda(a)$ as $\{w \in \Sigma^+ \mid a \in W(w)\}$.

Variable $t$ denotes contest trees. Variables $A$ and $B$ are used to denote the non-terminals of context-free grammars. Variable $r$ denotes regular expressions. The language generated by $A$ (resp. $r$) is denoted by $L(A)$ (resp. $L(r)$). We denote the empty string by $\epsilon$.

The paper is organized as follows. In Section 2, we show that *commutative* conservative groupoids recognize only regular languages. This is extended to the general case in Section 3. In Section 4, we give a partial algebraic characterization of the languages recognized by conservative groupoids.

## 2 Commutative Case

Let us consider an RPS-like game in which the operator is defined everywhere, conservative, and commutative.

**Theorem 1.** *Given an arbitrary symbol $a$ in $\Sigma$, the language $\Lambda(a)$ is regular.*

The demonstration proceeds in two steps. In the first step, we build a context-free grammar $G$ that generates $\Lambda(a)$. In the second step, we show that $G$ can be rewritten into a regular expression.

### 2.1 Building the Grammar

Let us first define the auxiliary function $f : \Sigma \to 2^{\Sigma}$ that, given a symbol $b$, returns the symbols that are *favorable* to $b$; i.e. the symbols that $b$ defeats. Formally: $f(b) = \{c \in \Sigma \mid b \cdot c = b\}$.

We build the grammar $G = (N, \Sigma, A_a, R)$ where $N$ is the set of non-terminals, $\Sigma$ is the set of terminals, $A_a$ is the start non-terminal, and $R$ is the set of productions, where:

$$\begin{aligned}
N &= \{A_a\} \cup \{B_\sigma \mid \emptyset \neq \sigma \subseteq \Sigma\} \quad \text{and} \\
R &= \{A_a \to B_\sigma \, a \, B_\sigma \mid \sigma = f(a)\} \\
&\quad \cup \{B_\sigma \to B_{\sigma'} \, b \, B_{\sigma'} \mid \sigma \subseteq \Sigma,\ b \in \sigma,\ \sigma' = \sigma \cup f(b)\} \\
&\quad \cup \{B_\sigma \to \epsilon \mid \emptyset \neq \sigma \subseteq \Sigma\}.
\end{aligned}$$

We claim that $G$ is built in such a way that all the words generated by $A_a$ allow $a$ to be the winner if we arrange the contest properly. We also claim that those generated by $B_\sigma$ are either empty or allow the contest to be arranged so that the winner is in $\sigma$. We intend to demonstrate that the non-terminals generate words with such properties but, also, that they generate *all* such words.

## 2.2 Correctness of the Grammar

**Lemma 1.** *For all non-empty subsets $\sigma$ of symbols, $L(B_\sigma) \subseteq \bigcup_{b \in \sigma} \Lambda(b) \cup \{\epsilon\}$.*

*Proof.* We proceed by induction on the length of the words generated by the family of the $B$ non-terminals. *Base case.* Let us consider a word $w$ of length 0. This means that $w$ must be $\epsilon$. Then the inclusion is trivially respected, for every $\emptyset \neq \sigma \subseteq \Sigma$. *Induction hypothesis (IH).* Let us suppose that, for all $\emptyset \neq \sigma \subseteq \Sigma$ and for all $w \in \Sigma^*$ such that $|w| \leq n$, we have that, whenever $w \in L(B_\sigma)$, then $w \in \bigcup_{b \in \sigma} \Lambda(b) \cup \{\epsilon\}$. *Induction step.* Let us consider a word $w$ of length $n+1$ and $\emptyset \neq \sigma \subseteq \Sigma$ such that $w \in L(B_\sigma)$. Since $w$ is non-empty, we must show that it is in $\bigcup_{b \in \sigma} \Lambda(b)$. We will do so by constructing a contest tree for $w$ whose winner is in $\sigma$. Let us choose a derivation tree for $w$. Given that $w \neq \epsilon$, let $B_\sigma \to B_{\sigma'} \, b \, B_{\sigma'}$ be the production that is used at the root of the derivation tree, where $\sigma' = \sigma \cup f(b)$. This implies that there exist $u$, $v \in \Sigma^*$ such that $w = u \, b \, v$, where $u$, $v \in L(B_{\sigma'})$. Since both $u$ and $v$ are of length at most $n$, then the IH applies to each. So, if $u \neq \epsilon$, then there exists a contest tree $t_u \in T(u)$ such that $W(t_u) \in \sigma'$. Similarly, if $v \neq \epsilon$, then there exists $t_v \in T(v)$ such that $W(t_v) \in \sigma'$. There are three cases to consider for $u$ (and similarly for $v$): $u = \epsilon$, $W(t_u) \in \sigma' - \sigma$, and $W(t_u) \in \sigma$. Hence, we would have a total of nine cases to analyze. In each case, we would have to show that we can build a contest tree $t_w \in T(w)$ such that $W(t_w) \in \sigma$. For the sake of conciseness, we only examine the case where $u \neq \epsilon \neq v$, $W(t_u) \in \sigma' - \sigma$, and $W(t_v) \in \sigma$. In this case, we select $t_w = (t_u \otimes b) \otimes t_v$ and we have that:

$$\begin{aligned}
&W(t_w) \\
&= W((t_u \otimes b) \otimes t_v) \\
&= W(t_u \otimes b) \cdot W(t_v) && \text{by def. of } W \\
&= (W(t_u) \cdot W(b)) \cdot W(t_v) \\
&= (W(t_u) \cdot b) \cdot W(t_v) \\
&= b \cdot W(t_v) && \text{because } W(t_u) \in \sigma' - \sigma \subseteq f(b) \\
&\in \sigma && \text{because } b,\ W(t_v) \in \sigma \text{ and `}\cdot\text{' is conserv.}
\end{aligned}$$

|  | $v = \epsilon$ | $W(t_v) \in \sigma' - \sigma$ | $W(t_v) \in \sigma$ |
|---|---|---|---|
| $u = \epsilon$ | $b$ | $b \otimes t_v$ | $b \otimes t_v$ |
| $W(t_u) \in \sigma' - \sigma$ | $t_u \otimes b$ | $(t_u \otimes b) \otimes t_v$ | $(t_u \otimes b) \otimes t_v$ |
| $W(t_u) \in \sigma$ | $t_u \otimes b$ | $t_u \otimes (b \otimes t_v)$ | $t_u \otimes (b \otimes t_v)$ |

**Fig. 1.** Key step in the inductive cases in the proof of correctness.

Figure 1 presents a $t_w$ that should be selected in each of the nine cases.

We now have established that $L(B_\sigma) \subseteq \bigcup_{b \in \sigma} \Lambda(b) \cup \{\epsilon\}$. What remains to show is Lemma 2.

**Lemma 2.** *For any $a \in \Sigma$, $L(A_a) \subseteq \Lambda(a)$.*

*Proof.* This is easy as the only $A_a$-production is $A_a \to B_{f(a)}\, a\, B_{f(a)}$, where $B_{f(a)}$ generates either the empty word or a word for which we can arrange a contest such that the winner is favorable to $a$. So, by analyzing four cases, we can show that we can arrange for all words in $L(A_a)$ to have $a$ as a winner.

## 2.3 Completeness of the Grammar

**Lemma 3.** *For all non-empty subsets $\sigma$ of symbols, $\bigcup_{b \in \sigma} \Lambda(b) \cup \{\epsilon\} \subseteq L(B_\sigma)$.*

*Proof.* We do so by induction on the length of the words. *Base case.* Let us consider a word $w$ of length 0. We have that $w = \epsilon$. Because of the productions of the form $B_\sigma \to \epsilon$, we have that $\epsilon \in L(B_\sigma)$, for all $\sigma \subseteq \Sigma$. *Induction hypothesis.* We suppose that the inclusion holds for all $\emptyset \neq \sigma \subseteq \Sigma$ and all words of length at most $n$. *Induction step.* Let us consider $\emptyset \neq \sigma \subseteq \Sigma$ and $w$ of length $n+1$ such that $w \in \bigcup_{b \in \sigma} \Lambda(b) \cup \{\epsilon\}$. Since $w \neq \epsilon$, let $b \in \sigma$ such that $w \in \Lambda(b)$. Let $\sigma' = \sigma \cup f(b)$. There exists a contest tree $t_w \in T(w)$ such that $W(t_w) = b$. There are two possible shapes for $t_w$: it is a leaf or it is a larger tree. If $t_w$ is a leaf, then $w = b$ and the derivation $B_\sigma \Rightarrow B_{\sigma'}\, b\, B_{\sigma'} \Rightarrow b\, B_{\sigma'} \Rightarrow b$ shows that $w \in L(B_\sigma)$. Otherwise, $t_w = t_u \otimes t_v$ where $b$ is the winner of at least one of $t_u$ and $t_v$ (by conservativeness of '$\cdot$'). Without loss of generality, let us suppose that $W(t_v) = b$. We know that $W(t_u) \in f(b)$, since $W(t_u)$ is defeated by $b$. Let $u$, $v \in \Sigma^+$ such that $t_u \in T(u)$ and $t_v \in T(v)$. Note that $1 \leq |v| \leq n$. Since $W(t_v) = b$, then $v \in \Lambda(b)$, so $v \in \Lambda(b) \cup \{\epsilon\}$, and (by IH) $v \in L(B_{\{b\}})$. Since $v \neq \epsilon$, there exists a derivation $B_{\{b\}} \Rightarrow^+ v$ that first uses the production $B_{\{b\}} \to B_{\sigma''}\, b\, B_{\sigma''}$, where $\sigma'' = \{b\} \cup f(b)$. Let $v'$, $v'' \in \Sigma^*$ such that $v = v'\, b\, v''$ and $v'$, $v'' \in L(B_{\sigma''})$. At this point, we have that $w = u\, v'\, b\, v''$, that there exists a contest tree $t_u \in T(u)$ such that $W(t_u) \in f(b)$, either that $v' = \epsilon$ or that there exists a contest tree $t_{v'} \in T(v')$ such that $W(t_{v'}) \in \sigma''$, and either that $v'' = \epsilon$ or that there exists a contest tree $t_{v''} \in T(v'')$ such that $W(t_{v''}) \in \sigma''$. Given the possible emptiness of $v'$ and that of $v''$, we would have four cases to analyze. We only examine the case where both $v'$ and $v''$ are non-empty, as the other cases are simpler. Let $t_{u\,v'} = t_u \otimes t_{v'}$. By conservativeness, we have that $W(t_{u\,v'}) \in \sigma''$ and, so,

$W(t_{u\,v'}) \in \sigma'$. Also, we have that $W(t_{v''}) \in \sigma''$ and, so, $W(t_{v''}) \in \sigma'$. Since $|u\,v'| \leq n \geq |v''|$, we use the IH and obtain that $u\,v'$, $v'' \in L(B_{\sigma'})$. Thus the derivation $B_\sigma \Rightarrow B_{\sigma'}\,b\,B_{\sigma'} \Rightarrow^* u\,v'\,b\,B_{\sigma'} \Rightarrow^* u\,v'\,b\,v'' = w$ shows that $w \in L(B_\sigma)$.

At this point, we have established that $\bigcup_{b\in\sigma} \Lambda(b) \cup \{\epsilon\} \subseteq L(B_\sigma)$. What remains to show is Lemma 4.

**Lemma 4.** *For any $a \in \Sigma$, $\Lambda(a) \subseteq L(A_a)$.*

*Proof.* This is shown by noting that, for any word $w$ in $\Lambda(a)$, there is a contest tree $t_w \in T(w)$ such that $W(t_w) = a$ and analyzing the three following cases: $t_w = a$, $t_w = t_u \otimes t_v$ where $W(t_u) = a$, and $t_w = t_u \otimes t_v$ where $W(t_u) \neq a$. In each case, it is simple to exhibit a derivation $A_a \Rightarrow^* w$, using arguments similar to those used in the demonstration of completeness for the $B_\sigma$'s.

## 2.4 Regularity of the Language Generated by the Grammar

Before we demonstrate that $L(G)$ is regular, we present a couple of lemmas. In the first lemma, we make the rather intuitive observation that, the larger the set $\sigma$ in $B_\sigma$, the larger the generated language.

**Lemma 5.** *For any $\emptyset \neq \sigma \subseteq \sigma' \subseteq \Sigma$, we have that $L(B_\sigma) \subseteq L(B_{\sigma'})$.*

*Proof.*
$$L(B_\sigma) = \bigcup_{b\in\sigma} \Lambda(b) \cup \{\epsilon\} \subseteq \bigcup_{b\in\sigma'} \Lambda(b) \cup \{\epsilon\} = L(B_{\sigma'}).$$

**Lemma 6.** *For any $\emptyset \neq \sigma \subseteq \sigma' \subseteq \Sigma$, we have that $L(B_{\sigma'}) = L(B_{\sigma'}\,B_\sigma)$.*

*Proof.* Showing that $L(B_{\sigma'}) \subseteq L(B_{\sigma'}\,B_\sigma)$ is trivial as it is sufficient to systematically use production $B_\sigma \to \epsilon$. Let us show $L(B_{\sigma'}\,B_\sigma) \subseteq L(B_{\sigma'})$ by induction on the length of the words. *Induction basis.* Let $w \in \Sigma^*$ of length 0. Then, $w$ has to be $\epsilon$ and $w \in L(B_{\sigma'})$. *Induction step.* Let $w \in L(B_{\sigma'}\,B_\sigma)$ of length $n+1$. There exist $u \in L(B_{\sigma'})$ and $v \in L(B_\sigma)$ such that $w = u\,v$. If $u = \epsilon$, then $w = v \in L(B_\sigma) \subseteq L(B_{\sigma'})$ and we are done. Otherwise, there exist $b \in \sigma'$, $\sigma'' = \sigma' \cup f(b)$, and $u'$, $u'' \in L(B_{\sigma''})$ such that $u = u'\,b\,u''$. Note that $|u''\,v| \leq n$ and, since $u''\,v \in L(B_{\sigma''}\,B_\sigma)$ with $\sigma \subseteq \sigma''$, the IH can be used to get that $u''\,v \in L(B_{\sigma''})$. Thus $w = u'\,b\,u''\,v \in L(B_{\sigma''}\,\{b\}\,B_{\sigma''}) \subseteq L(B_{\sigma'})$ and we are done.

Now we can turn to the main task of showing Lemma 7.

**Lemma 7.** *$G$ generates a regular language.*

*Proof.* Let us examine $G$'s productions. Note that each time there is a production of the form $B_\sigma \to B_{\sigma'}\,b\,B_{\sigma'}$, then we have that $\sigma \subseteq \sigma'$. The productions can be classified into two kinds: those for which $\sigma = \sigma'$ and those for which $\sigma \subset \sigma'$. The second kind of productions introduces no recursion among the non-terminals. The first kind does but only via self-recursion. We show that this does not lead to an non-regular language.

$$\begin{array}{llll|lllllll}
C_i \to & C_i & b_1 & C_i & r_i & = & ( & & b_1 & & | \\
\ldots & & & & & & & & \ldots & & | \\
C_i \to & C_i & b_{k'} & C_i & & & & & b_{k'} & & | \\
C_i \to & C_{l_{k'+1}} & b_{k'+1} & C_{l_{k'+1}} & & & & r_{l_{k'+1}} & b_{k'+1} & r_{l_{k'+1}} & | \\
\ldots & & & & & & & & \ldots & & | \\
C_i \to & C_{l_{k-1}} & b_{k-1} & C_{l_{k-1}} & & & & r_{l_{k-1}} & b_{k-1} & r_{l_{k-1}} & )^* \\
C_i \to & \epsilon & & & & & & & & &
\end{array}$$

$$\text{where } \max(l_{k'+1}, \ldots, l_{k-1}) < i$$

**Fig. 2.** Converting productions (possibly) with self-recursion into a regular expression.

The non-empty subsets of $\Sigma$ form a partially ordered set, with respect to inclusion ($\supseteq$). Let $\sigma_1, \ldots, \sigma_{2^{|\Sigma|}-1}$ be a topological ordering of the non-empty subsets of $\Sigma$ such that if $\sigma_j \supset \sigma_i$, then $j < i$. As a consequence, $\sigma_1$ has to be $\Sigma$ and $\sigma_{2^{|\Sigma|}-1}$ has to be one of the singletons. Let us use the alias non-terminals $C_1, \ldots, C_{2^{|\Sigma|}-1}$ for the permutation of the $B_\sigma$'s according to this ordering; i.e. $C_i = B_{\sigma_i}$, for $1 \leq i \leq 2^{|\Sigma|} - 1$. Consequently, we now view production $B_{\sigma_i} \to \epsilon$ as $C_i \to \epsilon$ and production $B_{\sigma_i} \to B_{\sigma_j}\, b\, B_{\sigma_j}$ as $C_i \to C_j\, b\, C_j$, where $i = j$ if and only if $\sigma_i = \sigma_j$.

In order to show that $A_a$ generates a regular language, we successively show, by induction on $i$, that each non-terminal $C_i$ generates the same language as a regular expression $r_i$. We can then conclude that $A_a$ also generates a regular language. We do an inductive reasoning on the $C_i$'s but, as will be apparent, there is no need to provide a special case for the basis.

Let $1 \leq i \leq 2^{|\Sigma|} - 1$. By IH, we know that each $C_j$ generates the same language as some regular expression $r_j$, for $j < i$. Let us consider all the $C_i$-productions $P_1, \ldots, P_k$. Without loss of generality, let us suppose that $P_1, \ldots, P_{k'}$ are the productions that involve self-recursion, that $P_{k'+1}, \ldots, P_{k-1}$ are those that involve non-self-recursion, and that $P_k$ is $C_i \to \epsilon$. Note that we have $0 \leq k' < k$. Value $k'$ might be as low as 0 because there might exist non-terminals for which all productions are non-self-recursive. Value $k'$ cannot get higher than $k - 1$ because there is production $P_k$.[4] For the sake of illustration, we list the productions on the left-hand side of Figure 2. On the right-hand side of Figure 2, there is a single replacement equation whose right member is regular expression $r_i$. The regular expression is a Kleene iteration over a union that contains an alternative for each non-$\epsilon$-production. Each self-recursive production is converted into its middle symbol. Each non-self-recursive production is trivially converted into a regular expression, as the concerned non-terminals already denote regular languages, by IH. We must show that $r_i$ generates $L(C_i)$.

We start by showing that $L(r_i) \subseteq L(C_i)$. We do so by induction on the size of the words. *Induction basis.* Let $w \in \Sigma^*$ of length 0. Then, $w$ must be $\epsilon$ and, clearly, $w \in L(C_i)$. *Induction step.* Let $w \in L(r_i)$ of length $n+1$. By construction,

[4] For instance, in the case of $C_1 = B_{\sigma_1} = B_\Sigma$, $k' = k - 1$ since all the productions are self-recursive except $C_1 \to \epsilon$.

we have that $r_i = (r'_i)^*$, where $r'_i$ is a union. Since $w \neq \epsilon$, then $w \in L(r'_i\, r_i)$, which means that there exist $u \in \Sigma^+$ and $v \in \Sigma^*$ such that $w = u\, v$, $u \in L(r'_i)$, and $v \in L(r_i)$. Since $|v| \leq n$, we know that $v \in L(C_i)$, by IH. There are two cases for $u$: either $u = b_m$, for $1 \leq m \leq k'$, or $u \in L(r_{l_m}\, b_m\, r_{l_m})$, for $k'+1 \leq m \leq k-1$. In the first case, the following derivation shows that $w \in L(C_i)$: $C_i \Rightarrow C_i\, b_m\, C_i \Rightarrow b_m\, C_i \Rightarrow^* b_m\, v = u\, v = w$. In the second case, we have that there exist $u'$, $u'' \in L(r_{l_m})$ such that $u = u'\, b_m\, u''$. By IH, we have that $u'$, $u'' \in L(C_{l_m})$. By construction of $G$, we know that $C_i = B_\sigma$ and $C_{l_m} = B_{\sigma'}$ such that $\sigma \subset \sigma'$. Consequently, $L(C_i) \subseteq L(C_{l_m})$. By the second lemma, $L(C_{l_m}) = L(C_{l_m}\, C_i)$. Consequently, $w = u\, v = u'\, b_m\, u''\, v$, where $u' \in L(C_{l_m})$ and $u''\, v \in L(C_{l_m}\, C_i) = L(C_{l_m})$, which guarantees that $C_i \Rightarrow C_{l_m}\, b_m\, C_{l_m} \Rightarrow^* w$.

We continue by showing that $L(C_i) \subseteq L(r_i)$. Once again, we show this result by induction on the length of the words. *Induction basis.* Let $w \in \Sigma^*$ of length 0. Then, $w$ must be $\epsilon$ and, clearly, $w \in L(r_i)$. *Induction step.* Let $w \in L(C_i)$ of length $n + 1$. Since $w \neq \epsilon$, the derivation $C_i \Rightarrow^* w$ must start with the use of a non-$\epsilon$-production. Then, two cases are possible. If the production is one of the first $k'$ ones, then it is $C_i \rightarrow C_i\, b_m\, C_i$, for $m \leq k'$, and there exist $u$, $v \in L(C_i)$ such that $w = u\, b_m\, v$. Since $|u| \leq n \geq |v|$, we use the IH and have that $u$, $v \in L(r_i)$. Consequently, $w = u\, b_m\, v \in L(r_i)\, L(r_i)\, L(r_i) \subseteq L(r_i)$, since $r_i$ is a Kleene iteration. In the other case, the first production used is $C_i \rightarrow C_{l_m}\, b_m\, C_{l_m}$, for $k' < m \leq k - 1$, and there exist $u$, $v \in L(C_{l_m})$ such that $w = u\, b_m\, v$. By construction of $G$ and the ordering of the $C$ non-terminals, we know that $C_{l_m}$ appears *before* $C_i$ (i.e. $l_m < i$) in the ordering and so $C_{l_m}$ generates the same language as $r_{l_m}$, by IH. Consequently, $w = u\, b_m\, v \in L(r_{l_m})\, \{b_m\}\, L(r_{l_m}) \subseteq L(r_i)$.

## 3 Non-commutative Case

We now study the case where the operator is non-commutative but still conservative and defined everywhere. That is, there exist $a$, $b \in \Sigma$ such that $a \cdot b \neq b \cdot a$. We show Theorem 2. We show it using an adaptation of the grammar-based method of Section 2.

**Theorem 2.** *Given an arbitrary symbol $a$ in $\Sigma$, the language $\Lambda(a)$ is regular.*

As in the commutative case, we start by giving the construction of a context-free grammar that generates $\Lambda(a)$ and then show that its language is regular.

### 3.1 Building the Grammar

Due to the loss of the commutativity property, we now need *two* auxiliary functions that return, for a given symbol $b$, the symbols that are *favorable* to $b$: functions $f_L, f_R : \Sigma \rightarrow 2^\Sigma$ for the symbols that are defeated when they appear on the left-hand side of the operator and those that are defeated when they appear on the right-hand side of the operator, respectively. Formally: $f_L(b) = \{c \in \Sigma \mid c \cdot b = b\}$, and $f_R(b) = \{c \in \Sigma \mid b \cdot c = b\}$.

We define the context-free grammar $G = (N, \Sigma, A_a, R)$ where all the components are the same as in the commutative case except for the productions:

$$\begin{aligned} R = &\{A_a \to B_{\sigma'}\, a\, B_{\sigma''} \mid \sigma' = f_L(a),\ \sigma'' = f_R(a)\} \\ &\cup \{B_\sigma \to B_{\sigma'}\, b\, B_{\sigma''} \mid \sigma \subseteq \Sigma,\ b \in \sigma,\ \sigma' = \sigma \cup f_L(b),\ \sigma'' = \sigma \cup f_R(b)\} \\ &\cup \{B_\sigma \to \epsilon \mid \emptyset \neq \sigma \subseteq \Sigma\}. \end{aligned}$$

The main difference is that we take care of handling the sets of defeated symbols independently on the left- and on the right-hand sides. We do not show the following lemmas as the proofs are similar to those of Section 2.

**Lemma 8.** $L(B_\sigma) = \bigcup_{b \in \sigma} \Lambda(b) \cup \{\epsilon\}$.

**Lemma 9.** $L(A_a) = \Lambda(a)$.

### 3.2 Regularity of the Language Generated by the Grammar

**Lemma 10.** *$G$ generates a regular language.*

*Proof.* In order to show Lemma 10, we use the alias variables $C_1, \ldots, C_{2^{|\Sigma|}-1}$ once again. We remind the reader that $C_i = B_{\sigma_i}$, for $1 \leq i \leq 2^{|\Sigma|} - 1$, that, if $\sigma_j \supseteq \sigma_i$, we have $j \leq i$, and finally that, for any production $C_i \to \alpha$, any $C_j$ that appears in $\alpha$ is such that $j \leq i$. Again, we show by induction on $i$ that each $C_i$ generates a regular language, which is the same as the one generated by the regular expression $r_i$. At step $i$, we suppose that $C_j$ is equivalent to $r_j$, for all $j < i$. The changes that must be made to the proof in the non-commutative case relate to the construction of the regular expression $r_i$ and the demonstrations that $L(r_i) \subseteq L(C_i)$ and $L(C_i) \subseteq L(r_i)$.

Let us consider all the $C_i$-productions $P_1, \ldots, P_k$. Without loss of generality, let us suppose that the productions are grouped by kind of recursion. Note that the non-$\epsilon$-productions are of the form $C_i \to C_j\, b\, C_h$, with $j \leq i \geq h$, and hence cannot be merely categorized as being self-recursive or not. Let $0 \leq k' \leq k'' \leq k''' < k$ such that $P_1, \ldots, P_{k'}$ are completely self-recursive, $P_{k'+1}, \ldots, P_{k''}$ are self-recursive on the left only, $P_{k''+1}, \ldots, P_{k'''}$ are self-recursive on the right only, and $P_{k'''+1}, \ldots, P_{k-1}$ are not self-recursive at all. Figure 3 presents the original $C_i$-productions and the regular expression $r_i$ in which they are transformed.

As in the commutative case, there remains to show that $L(r_i) = L(C_i)$. Due to lack of space, we omit the proof that each language is contained into the other. However, the arguments are similar to those used in the commutative case, except that there are a few extra sub-cases to analyze; i.e. those for the productions that are self-recursive on the left only and for the productions that are self-recursive on the right only.

## 4 Languages Recognized by Conservative Groupoids

We now know that languages recognized by conservative groupoids are regular and it is natural to seek a more precise characterization. This seems challenging.

$$
\begin{array}{llll|lllllll}
C_i \to & C_i & b_1 & C_i & r_i & = & ( & & b_1 & & | \\
\dots & & & & & & & & \dots & & | \\
C_i \to & C_i & b_{k'} & C_i & & & & & b_{k'} & & | \\
C_i \to & C_i & b_{k'+1} & C_{l_{k'+1}} & & & & & b_{k'+1} & r_{l_{k'+1}} & | \\
\dots & & & & & & & & \dots & & | \\
C_i \to & C_i & b_{k''} & C_{l_{k''}} & & & & & b_{k''} & r_{l_{k''}} & | \\
C_i \to & C_{l_{k''+1}} & b_{k''+1} & C_i & & & & r_{l_{k''+1}} & b_{k''+1} & & | \\
\dots & & & & & & & & \dots & & | \\
C_i \to & C_{l_{k'''}} & b_{k'''} & C_i & & & & r_{l_{k'''}} & b_{k'''} & & | \\
C_i \to & C_{l_{k'''+1}} & b_{k'''+1} & C_{l'_{k'''+1}} & & & & r_{l_{k'''+1}} & b_{k'''+1} & r_{l'_{k'''+1}} & | \\
\dots & & & & & & & & \dots & & | \\
C_i \to & C_{l_{k-1}} & b_{k-1} & C_{l'_{k-1}} & & & & r_{l_{k-1}} & b_{k-1} & r_{l'_{k-1}} & )^* \\
C_i \to & \epsilon & & & & & & & & &
\end{array}
$$

$$\text{where } \max(l_{k'+1}, \dots, l_{k-1}, l'_{k'''+1}, \dots, l'_{k-1}) < i$$

**Fig. 3.** Converting productions into a regular expression in the non-commutative case.

One starting point is to consider conservative groupoids which are also associative, i.e. semigroups for which $x \cdot y \in \{x, y\}$. In particular, these satisfy $x^2 = x$ but we can give an exact characterization.

**Lemma 11.** *A semigroup $S$ is conservative iff its set of elements can be partitioned into $k$ classes $C_1, \dots C_k$ such that $x \cdot y = y \cdot x = x$ whenever $x \in C_i$ and $y \in C_j$ for $i > j$ and for any $j$ either $x \cdot y = x$ for all $x, y \in C_j$ (left-zero) or $x \cdot y = y$ for all $x, y \in C_j$ (right-zero).*

*Proof.* By definition, such a semigroup is conservative. Also, the operation defined above is associative. Indeed if $x, y, z$ are three elements lying in the same class $C_i$ then $(x \cdot y) \cdot z = x \cdot (y \cdot z) = x$ if $C_i$ is left-zero and $(x \cdot y) \cdot z = x \cdot (y \cdot z) = z$ if it is right-zero. If $x, y, z$ are not in the same class then associativity follows because the elements in the most absorbing class are the only ones that matter. Suppose for instance that $x$ and $z$ lie in the same class $C_i$ while $y$ lies in some $C_j$ with $i > j$. Since $x \cdot y = x$ and $y \cdot z = z$ we clearly have $(x \cdot y) \cdot z = x \cdot (y \cdot z) = xz$.

Conversely, suppose $S$ is a conservative semigroup. For any $x, y$, one of three cases must hold: (1) $x \cdot y = y \cdot x = x$ (or $\dots = y$), (2) $x \cdot y = x$ and $y \cdot x = y$, or (3) $x \cdot y = y$ and $y \cdot x = x$ and we say that the pair $x, y$ is of type 1, 2, or 3.

First note that cases (2) and (3) define equivalence relations on $S$. Moreover, if $x \neq y$ is a pair of type (2) then there cannot exist a $z \neq x$ such that $x, z$ is a pair of type (3). Indeed, we would then have $z \cdot y = (x \cdot z) \cdot (y \cdot x) = x \cdot z \cdot y \cdot x$. Because $S$ is conservative we must have $z \cdot y \in \{y, z\}$ but this either leads to $x \cdot z \cdot y \cdot x = x \cdot z \cdot x = x$ or $x \cdot z \cdot y \cdot x = x \cdot y \cdot x = x$. Both cases form a contradiction.

These facts allow us to partition $S$ into classes such that within each either $x \cdot y = x$ for all $x, y \in C_j$ or $x \cdot y = y$ for all $x, y \in C_j$. (These $C_j$ correspond to the $\mathcal{J}$-classes of the semigroup.) It remains to show that we can impose a total order on these classes. We simply choose to place class $A$ below class $B$ if there is some $x$ in $A$ and some $y$ in $B$ such that $x \cdot y = y \cdot x = x$. This is well defined:

if, e.g., we choose $z$ another representative of $B$ with $y \cdot z = y$ and $z \cdot y = z$, then $x \cdot z = x \cdot y \cdot z = x \cdot y = x$. Moreover, this forms a total order since for any pair $x, y$ not of type (2) or (3), we must have $x \cdot y = y \cdot x = x$ or $x \cdot y = y \cdot x = y$. It is now straightforward to check that $S$ has the structure described in the statement.

This characterization can be translated into a description of the languages recognizable by conservative semigroups. For an alphabet $\Sigma$, consider a partition $C_1, \ldots, C_k$ each with an associated direction $d_1, \ldots, d_k$ with $d_i \in \{\mathsf{L}, \mathsf{R}\}$. For $a \in C_j$ with $d_j = \mathsf{L}$ (resp. $d_j = \mathsf{R}$), define the language $L_a$ (resp. $R_a$) of words that contain no occurrence of letters in classes $C_i$ with $i < j$ and where the first (resp. last) occurrence of a letter in $C_j$ is an $a$. A language can be recognized by a conservative semigroup iff it is the disjoint union of some $L_a$ and $R_a$.

Note that the class of languages recognized by conservative semigroups does not have many closure properties. For instance, it is not closed under union or intersection: each of the languages $\Sigma^* a \Sigma^*$ and $\Sigma^* b \Sigma^*$ can be recognized but their union (or intersection) has a syntactic semigroup which is not conservative.

The apparent absence of closure properties makes it difficult to provide a complete characterization of languages recognized by *non-associative*, conservative groupoids. We believe that this is nevertheless an interesting challenge and we end this section with a discussion of basic examples and avenues for research.

First, while any language $L$ recognized by a conservative semigroup must be idempotent, this need not be the case for one recognized by a conservative groupoid. A language is idempotent if, for any $x, y, z \in \Sigma^*$, we have $xyz \in L \Leftrightarrow xy^2z \in L$. A language recognized by a conservative groupoid $G$ need not be idempotent despite the fact that $g^2 = g$ for any $g \in G$. For instance, the language $\{a,b\}^* a \{a,b\}^* a \{a,b\}^*$ consisting of words with at least two $a$s is not idempotent. It can however be recognized by the Rock-Paper-Scissors groupoid by setting $\phi(a) = RPS$ and $\phi(b) = \epsilon$ and choosing $\{P\}$ as the accepting set. If $w$ contains no $a$ then $\phi(w) = \epsilon$ cannot produce $P$. If $w$ contains a single $a$ then $\phi(w) = RPS$ and one readily checks that $R \cdot (P \cdot S) = R$ and $(R \cdot P) \cdot S = S$. However, $\phi(aa) = RPSRPS$ which can be bracketed as $((R \cdot P) \cdot (S \cdot (R \cdot (P \cdot S)))) = P$. More generally, if $k \geq 3$ the $k$ copies of $RPS$ can be bracketed as follows. First bracket each of the $k-2$ last copies of $RPS$ individually to obtain $S$, absorb these $S$s to obtain $RPSRPS$, and use the bracketing above.

Similarly, $\{a,b\}^* a \{a,b\}^* a \{a,b\}^* a \{a,b\}^*$ can be recognized by the four-element groupoid with the multiplication table besides. One can see that both $\phi(a) = 1234$ and $\phi(aa) = 12341234$ can never be bracketed to obtain 3. In both cases, the rightmost 3 cannot win against 4 and the leftmost 3 cannot win against 1, possibly after having defeated 2. On the other hand, 3 can be the winner of $\phi(aaa) = 123412341234$, as shown by the bracketing: $((((1 \cdot (2 \cdot 3)) \cdot (4 \cdot 1)) \cdot 2) \cdot (3 \cdot (((4 \cdot 1) \cdot 2) \cdot (3 \cdot 4))))$.

|   | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 4 |
| 2 | 1 | 2 | 3 | 2 |
| 3 | 1 | 3 | 3 | 4 |
| 4 | 4 | 2 | 4 | 4 |

We think that there are similar conservative groupoids that can "count" up to $k$ for any $k$ and have verified this up to $k = 6$. In fact, distinguishing between five and six occurrences of $a$ can be achieved, somewhat counter-intuitively, with a five-element groupoid.

On the other hand, we believe it is impossible to count the occurrences of a letter modulo some $p$ using a conservative groupoid and more generally that every language $L$ recognized by a conservative groupoid $G$ is star-free, i.e. that for some $k$ and any $x, y, z \in \Sigma^*$ we have $xy^kz \in L \Leftrightarrow xy^{k+1}z \in L$.

For any $g$ in a conservative groupoid $G$, we have $g^2 = g$. So whenever $g \in W(u)$, we must also have $g = g^2 \in W(u^2)$. As a referee kindly pointed out, this suffices to show a partial result along those lines, which is that the language $(aa)^*a$ of words of odd length cannot be recognized by $G$.

Due to space restrictions, further partial results are omitted here. An up-to-date extended version of the paper is available from the authors' websites.

## 5 Conclusion and Future Work

We have shown that conservative groupoids can only recognize regular languages. Beaudry, Lemieux, and Thérien had previously exhibited a large class of groupoids with the same limitations but our work is incomparable to theirs and our methods are, accordingly, quite different. It is natural to ask whether our approach can be generalized to find a wider class of "weak" groupoids and an obvious target are the 0-conservative groupoids, i.e. groupoids $H$ with a 0 element such that $0 \cdot x = x \cdot 0 = 0$ for all $x \in H$ and $x \cdot y \in \{x, y, 0\}$ for all $x, y \in H$; i.e. all non-conservative products are 0.

It is well known that left-linear and right-linear context-free grammars generate regular languages [6]. The work presented in this paper can be used to recognize a larger family of context-free grammars that generate regular languages. The work of [1, 2] is similar in that regard: context-free grammars with productions of the form $A \rightarrow A\alpha A$ (called self-embedded), and other simpler forms of recursion, are shown to generate regular languages.

## References

1. Andrei, S., Cavadini, S., Chin, W.N.: Transforming self-embedded context-free grammars into regular expressions. Tech. Rep. TR 02-06, University "A.I.Cuza" of Iaşi, Faculty of Computer Science (2002)
2. Andrei, S., Chin, W.N., Cavadini, S.V.: Self-embedded context-free grammars with regular counterparts. Acta Inf. 40(5), 349–365 (March 2004)
3. Beaudry, M.: Languages recognized by finite aperiodic groupoids. Theor. Comput. Sci. 209(1-2), 299–317 (1998)
4. Beaudry, M., Lemieux, F., Thérien, D.: Finite loops recognize exactly the regular open languages. In: Proc. 24th Int. Conf. Automata, Languages and Programming (ICALP'97). pp. 110–120 (1997)
5. Beaudry, M., Lemieux, F., Thérien, D.: Groupoids that recognize only regular languages. In: Proc. 32nd Int. Conf. Automata, Languages and Programming (ICALP'05). pp. 421–433 (2005)
6. Linz, P.: An Introduction To Formal Languages And Automata. Jones and Bartlett Publishers, Inc., USA, 2nd edn. (1997)
7. Pin, J.E.: Syntactic semigroups. In: Handbook of language theory, vol. 1, chap. 10, pp. 679–746. Springer Verlag (1997)