FLClassify

Classify formal languages

0.1.0

10 November 2022

Dakotah Lambert

Dakotah Lambert
Email: dakotahlambert@acm.org
Homepage: https://vvulpes0.github.io/

Abstract

FLClassify is a GAP package whose purpose is to analyze and classify formal languages.

Copyright

Contents

Chapter 1

Introduction

FLClassify is a package dedicated to the algebraic theory of formal languages, and specifically to the problem of classifying these languages.

1.1 Installation

Simply unpack the source archive in your GAP pkg directory. Use of this package also requires that the Semigroups and Automata packages are installed, or other packages that offer a compatible interface. These packages are part of the base installation.

1.2 Preliminaries

A semigroup is a set S augmented with some operation, denoted by adjacency, which is associative. That is, for any a, b, and c in S, it holds that $(ab)c = a(bc)$, and thus abc is without ambiguity. A semigroup is a monoid iff it contains some element 1 such that for all x, $1x = x = x1$. For every semigroup S, there is an associated monoid S^1 which is equal to S if it is already a monoid, else it is equal to S with an adjoined element that acts as an identity. Semigroups are not groups: they need not have inverses. Given some element x, there may or may not exist some y such that $xyx = x$. If this exists, x is called a regular element and y a pseudoinverse of x. Note that if y is a psudoinverse of x it necessarily follows that $xyx = x$ and yxy are pseudoinverses of one another.

A formal language is any subset of the free monoid Σ^*, although in many cases the empty word is irrelevant to the membership problem, and it is easier to work in terms of the free semigroup Σ^+. As there is a standard construction for this structure from a finite automaton, this package provides a mechanism by which to import finite automata in a standard format.

When presented with a language as a finite automaton, rather than as a semigroup, additional operations are unlocked. These are discussed in the final chapter.

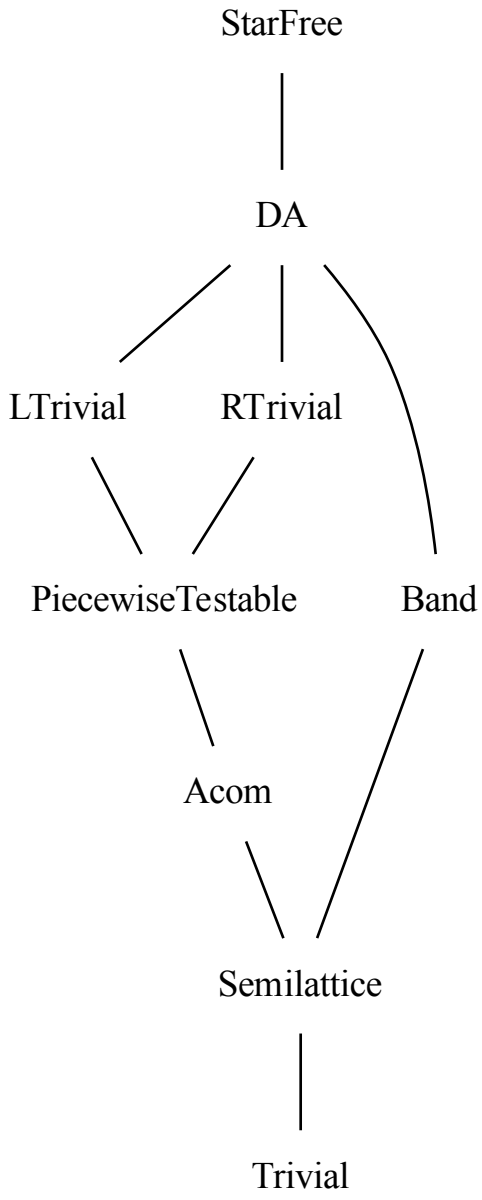

Figure 1.1: A hierarchy of classes which triplicates.

Chapter 2

Green's Relations

The following relations, introduced by Green [Gre51], have been fundamental in studying the structure of semigroups:

- $a \mathscr{L} b$ iff $S^1a = S^1b$
- $a \mathscr{R} b$ iff $aS^1 = bS^1$
- $a \mathscr{J} b$ iff $S^1aS^1 = S^1bS^1$
- $a \mathscr{H} b$ iff $a \mathscr{L} b$ and $a \mathscr{R} b$
- $a \mathscr{D} b$ iff there is a c where $a \mathscr{L} c$ and $c \mathscr{R} b$

A language is regular iff its syntactic semigroup is finite, so for this work we can operate in a space where $\mathscr{J} = \mathscr{D}$.

Regular $\mathscr{D}$-classes are sometimes relevant. A $\mathscr{D}$-class is regular iff it contains a regular element.

2.1 Triviality Under Certain Relations

Many important classes of languages are defined by properties related to these relations. The simplest are those corresponding to first-order and propositional logic over general precedence.

2.1.1 IsStarFree

▷ `IsStarFree(`*semigroup*`)` (property)

Returns: true or false

Determine if a semigroup recognizes only star-free languages. This is implemented as a synonym, so there is no corresponding `HasIsStarFree`.

A language is star-free iff it is definable by a generalized regular expression without the use of the star. That is, it is the closure under concatenation, union, intersection, and complement of singleton sets of words. As demonstrated by Schützenberger [Sch65], a language is star-free iff its syntactic semigroup is finite and aperiodic. Aperiodic simply means that every $\mathscr{H}$-class is singleton, that the semigroup is $\mathscr{H}$-trivial. McNaughton and Papert show that a language is first-order definable with less-than iff it is star-free [MP71].

2.1.2 IsPiecewiseTestable

▷ `IsPiecewiseTestable(`*semigroup*`)` (property)

Returns: true or false

Determine if a semigroup recognizes only piecewise testable languages. This is implemented as a synonym, so there is no corresponding `HasIsPiecewiseTestable`.

A language is piecewise testable iff it is definable by its allowed sets of subsequences, where a subsequence is a (finite) sequence of symbols that occur in order but not necessarily adjacent to one another. As demonstrated by Simon [Sim75], a language is piecewise testable iff its syntactic semigroup is finite and $\mathscr{J}$-trivial. This definition corresponds to a propositional logic whose propositions are the occurrence of subsequences, defined by less-than alone.

The $\mathscr{L}$- and $\mathscr{R}$-trivial languages are useful in some sense, but have not been given other names in the literature. Thus no synonyms are provided. We shall see them later when discussing definability by affixes. First, though, let us consider two special subclasses of piecewise testable semigroups: Acom and semilattices.

2.1.3 IsAcom

▷ `IsAcom(`*semigroup*`)` (property)

Returns: true or false

Determine if a semigroup is aperiodic and commutative. This is implemented as a synonym, so there is no corresponding `HasIsAcom`.

The corresponding languages are based on restrictions on their multisets of symbols, where counts saturate at some threshold t. The special case where $t = 1$ is a restriction on which sets of symbols appear; this smaller subclass corresponds to `IsSemilattice`.

Finally, there is a class which contains all of the $\mathscr{L}$- and $\mathscr{R}$-trivial langugaes without moving all the way up to star-free:

2.1.4 IsDA (for IsSemigroup)

▷ `IsDA(`*semigroup*`)` (property)

Returns: true or false

Decide if a semigroup is in DA, that is, whether all of its regular $\mathscr{D}$-classes are aperiodic semigroups. This is equivalent to asking whether all regular elements are idempotent. Thérien and Wilke demonstrate that this class corresponds precisely to first-order definability with less-than but only two formal variables [TW98].

2.2 Affixes

This section details languages defined by sets of permitted affixes. They might just as well be in the later chapter on local properties, but in fact they are decidable without invoking the mechanisms of that section.

2.2.1 IsGeneralizedDefinite (for IsSemigroup)

▷ `IsGeneralizedDefinite(`*semigroup*`)` (property)

▷ `IsTierGeneralizedDefinite(`*semigroup*`)` (property)

Returns: true or false

A language is generalized definite iff it is defined by a Boolean combination of permitted suffixes. This corresponds to having all idempotents be $\mathscr{D}$-related, which trivially makes this a subclass of DA. A language is tier-based generalized definite iff its projected subsemigroup is generalized definite. The projected subsemigroup is the semigroup generated by $S^1 \setminus \{1\}$.

2.2.2 Is(Tier)Definite

▷ `Is(Tier)Definite(`*`semigroup`*`)` (property)

Returns: true or false

Determine if a semigroup is (tier-based) definite. These are implemented as synonyms, so there is no corresponding `HasIsDefinite` or `HasIsTierDefinite`.

A language is (tier-based) definite iff it is defined by a Boolean combination of suffixes. These are all and only those languages that are both (tier-based) generalized definite and $\mathscr{L}$-trivial.

2.2.3 Is(Tier)ReverseDefinite

▷ `Is(Tier)ReverseDefinite(`*`semigroup`*`)` (property)

Returns: true or false

Determine if a semigroup is (tier-based) reverse definite. These are implemented as synonyms, so there is no corresponding `HasIsDefinite` or `HasIsTierDefinite`.

A language is (tier-based) reverse definite iff it is defined by a Boolean combination of prefixes. These are all and only those languages that are both (tier-based) generalized definite and $\mathscr{R}$-trivial.

Chapter 3

Local Properties

For all elements x of a semigroup S, the set $xSx = \{xsx : s \in S\}$ is a subsemigroup of S. If further it holds that $x = xx$ then it is a submonoid with identity e. A submonoid generated by an idempotent in this way is called a local submonoid.

3.1 Generating Local Submonoids

3.1.1 LocalSubsemigroup (for IsSemigroup, IsMultiplicativeElement)

▷ `LocalSubsemigroup(`*`semigroup, element`*`)` (operation)

Returns: a subsemigroup

Compute the local subsemigroup of the *`semigroup`* induced by the *`element`*.

3.1.2 LocalSubmonoids (for IsSemigroup)

▷ `LocalSubmonoids(`*`semigroup`*`)` (attribute)

Returns: a list of submonoids

Determine all local submonoids of the *`semigroup`*.

3.1.3 IteratorOfLocalSubmonoids (for IsSemigroup)

▷ `IteratorOfLocalSubmonoids(`*`semigroup`*`)` (operation)

Returns: an iterator of submonoids

Determine all local submonoids of the *`semigroup`*. These are given as an external iterator rather than being stored as an attribute of the *`semigroup`*.

3.1.4 Locally (for IsFunction, IsSemigroup)

▷ `Locally(`*`proposition, semigroup`*`)` (operation)

Returns: true or false

Determine if all local submonoids of the *`semigroup`* satisfy the *`proposition`*.

As convenience, we declare some specific named attributes regarding local properties. As these are stored in the object, they can be used implicationally to speed up classification by omitting redundant tests.

3.1.5 IsLocallyX

▷ `IsLocallyDA(`*semigroup*`)` (property)
▷ `IsLocallyLTrivial(`*semigroup*`)` (property)
▷ `IsLocallyRTrivial(`*semigroup*`)` (property)
▷ `IsLocallyJTrivial(`*semigroup*`)` (property)
▷ `IsLocallyThresholdTestable(`*semigroup*`)` (property)
▷ `IsLocallyTestable(`*semigroup*`)` (property)

Returns: true or false

A semigroup satisfies a property locally iff all of its local submonoids satisfy the property.

A language is locally threshold testable iff its syntactic semigroup is locally Acom, that is, locally aperiodic and commutative. This result derives from applying the work of Almeida [Alm89] to the results of Beauquier and Pin [BP89]. A language is locally testable iff its syntactic semigroup is locally a semilattice, that is, locally idempotent and commutative [BS73].

3.1.6 ProjectedSubsemigroup (for IsSemigroup)

▷ `ProjectedSubsemigroup(`*semigroup*`)` (attribute)

Returns: a subsemigroup

The projected subsemigroup of a *semigroup* S is that subsemigroup generated by $S^1 \setminus \{1\}$. This corresponds to the syntactic semigroup of the projection of a language to all and only its salient symbols. Thus this structure is in a class iff *semigroup* recognizes only languages of the corresponding tier-based class such as those discussed by Lambert [Lam22]. See also `SalienceProjection` (4.2.3).

3.1.7 TierLocally (for IsFunction, IsSemigroup)

▷ `TierLocally(`*proposition, semigroup*`)` (operation)

Returns: true or false

A semigroup satisfies a property tier-locally iff its projected subsemigroup satisifies it locally.

3.1.8 IsTierLocallyX

▷ `IsTierLocallyDA(`*semigroup*`)` (property)
▷ `IsTierLocallyLTrivial(`*semigroup*`)` (property)
▷ `IsTierLocallyRTrivial(`*semigroup*`)` (property)
▷ `IsTierLocallyJTrivial(`*semigroup*`)` (property)
▷ `IsTierLocallyThresholdTestable(`*semigroup*`)` (property)
▷ `IsTierLocallyTestable(`*semigroup*`)` (property)

Returns: true or false

Determine if the property is satisfied tier-locally. These are stored on the semigroup to speed up classification through implications.

Chapter 4

Operations on Automata

While the majority of the functionality of this package comes from algebraic manipulation, some problems are just not amenable to that setting. For those times, one might wish to work directly with finite automata. Here we offer one mechanism for importing machines and some decision problems that cannot be handled with semigroups alone.

4.1 Importing Automata

4.1.1 AutomatonFromATT (for IsInputStream)

▷ `AutomatonFromATT(`*`stream`*`)` (operation)

Returns: an automaton

Import an automaton from AT&T-style tabular text read from *stream*. Each row contains no more than five whitespace-separated fields. An empty row is ignored. A row with one or two fields represents an accepting state, perhaps with a weight which is ignored here. A row with three, four, or five fields represents a transition. The first, second, and third fields always represent the source, destination, and input symbol, respectively, and subsequent fields are again ignored.

For example the following text represents Figure 4.1, which itself represents strings over $\Sigma = \{a,b,c\}$ not containing an "ab"-substring.

Example

```
1 2 a
1 1 b
1 1 c
1
2 2 a
2 3 b
2 1 c
2
3 3 a
3 3 b
3 3 c
```

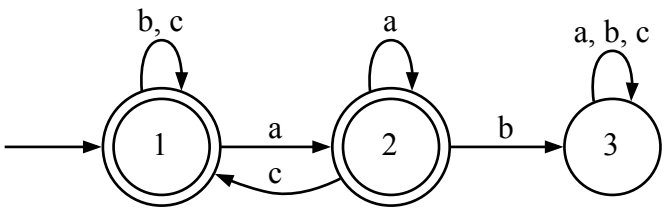

Figure 4.1: A complete minimal automaton accepting all and only those words that do not contain "*ab*".

4.2 Strict Locality and Relativizations

A language is locally testable in the strict sense (strictly local) iff there is some value k and some set S of anchored words of length at most k such that the words in the language are all and only those that do not contain any element of S as a substring when anchored. As this class is not closed under complement, there can be no characterization for the class based on the syntactic semigroup alone without further metadata. But all is not lost.

Characterized by closure under substitution of suffixes following shared length-k substrings, the strictly local languages are associated with automata that synchronize after no more than k steps [ELM+08]. In other words, from any state p it is necessarily the case that following a length-k word w will lead to the same state as it would when following w from any other state q.

There exists an exponential-time algorithm for deciding membership in this class, which admits simultaneously extracting the grammar of forbidden substrings [RL19], but the polynomial-time process of [ELM+08] is used here. If the trim, minimal automaton has states Q, then class membership is decidable using the subgraph of the product construction containing only nodes of the form $Q \times Q$ whose two components are inequal. The language is strictly local iff this graph is acyclic.

4.2.1 Is(Tier)StrictlyLocal

▷ `IsStrictlyLocal(aut)` (property)
▷ `IsTierStrictlyLocal(aut)` (property)

Returns: true or false

Determine if the language represented by `aut` is strictly k-local for some k that is, if it is definable as a conjunction of forbidden substrings. A language is *tier-based* strictly local iff there is some subset T of its alphabet (a tier of salient symbols) such that symbols not in T may be freely inserted and deleted and the projection to T is strictly local [Lam22].

The constructions used in deciding (tier-based) strict locality are exposed in case they might be useful for other purposes. Recall that salience has an algebraic characterization as well, which should be preferred when possible.

4.2.2 DoubletonGraph (for IsAutomatonObj)

▷ `DoubletonGraph(`*aut*`)` (operation)

Returns: an automaton

Construct an automaton whose underlying graph is the pair-graph used in deciding strict locality. The initial state is the pair containing two copies of the initial state of *aut*, and the final states are all pairs in which both elements are accepting.

4.2.3 SalienceProjection (for IsAutomatonObj)

▷ `SalienceProjection(`*aut*`)` (operation)

Returns: an automaton

Remove nonsalient symbols, those that are freely insertable and deletable. For a language in a tier-based class, this provides the projection under which the grammar operates. See also `ProjectedSubsemigroup` (3.1.6).

References

[Alm89] Jorge Almeida. Semidirect products of pseudovarieties from the universal algebraist's point of view. *Journal of Pure and Applied Algebra*, 60(2):113–128, October 1989. 10

[BP89] Danièle Beauquier and Jean-Éric Pin. Factors of words. In Giorgio Ausiello, Mariangiola Dezani-Ciancaglini, and Simonetta Ronchi Della Rocca, editors, *Automata, Languages and Programming: 16th International Colloquium*, volume 372 of *Lecture Notes in Computer Science*, pages 63–79. Springer Berlin / Heidelberg, 1989. 10

[BS73] Janusz Antoni Brzozowski and Imre Simon. Characterizations of locally testable events. *Discrete Mathematics*, 4(3):243–271, March 1973. 10

[ELM+08] Matt Edlefsen, Dylan Leeman, Nathan Myers, Nathaniel Smith, Molly Visscher, and David Wellcome. Deciding strictly local (SL) languages. In Jon Breitenbucher, editor, *Proceedings of the 2008 Midstates Conference for Undergraduate Research in Computer Science and Mathematics*, pages 66–73, 2008. 12

[Gre51] James Alexander Green. On the structure of semigroups. *Annals of Mathematics*, 54(1):163–172, July 1951. 6

[Lam22] Dakotah Lambert. *Unifying Classification Schemes for Languages and Processes with Attention to Locality and Relativizations Thereof*. PhD thesis, Stony Brook University, May 2022. 10, 12

[MP71] Robert McNaughton and Seymour Aubrey Papert. *Counter-Free Automata*. MIT Press, 1971. 6

[RL19] James Rogers and Dakotah Lambert. Extracting Subregular constraints from Regular stringsets. *Journal of Language Modelling*, 7(2):143–176, September 2019. 12

[Sch65] Marcel-Paul Schützenberger. On finite monoids having only trivial subgroups. *Information and Control*, 8(2):190–194, April 1965. 6

[Sim75] Imre Simon. Piecewise testable events. In Helmut Brakhage, editor, *Automata Theory and Formal Languages*, volume 33 of *Lecture Notes in Computer Science*, pages 214–222. Springer-Verlag, Berlin, 1975. 7

[TW98] Denis Thérien and Thomas Wilke. Over words, two variables are as powerful as powerful as one quantifier alternation: $\mathrm{FO}^2 = \Sigma_2 \cap \Pi_2$. In *STOC '98: Proceedings of the Thirtieth Annual ACM Symbosium on Theory of Computing*, pages 234–240, New York, NY, 1998. Association for Computing Machinery. 7

Index