The factorization process

Dakotah Lambert

The Local Language Toolkit includes an executable `./factorize` which, given one or more automata (in Jeff format) and a possibly empty set of non-strict constraints will produce a factorization in terms of SL, co-SL, SP, co-SP, and a minimal (in number of constraints) set of the non-strict constraints. If the stringset cannot be fully defined in these terms, an approximation

The first step in factoring a stringset is to obtain a set of Strictly Local (SL) and Strictly Piecewise (SP) constraints (forbidden factors), the process of which is described in [RL18] and [RL17]. The redundancy between Piecewise and Local constraints is resolved by omitting the Piecewise constraints that are implied by the local ones, and then removing the Local constraints that are implied by the remaining Piecewise ones. If the stringset is SL + SP (strict), then this suffices. Otherwise, the conjunction of these strict constraints is a strict approximation of the stringset.

If the stringset is not strict, then the next step is to find a set of costrict constraints: a set of factors where a string is accepted if and only if it contains at least one of these factors. As the complements of strict constraints, these costrict constraints can be found from the complement of the given automaton — but not without some work.

If there is any set of factors F such that any string whose set of factors is a superset of F is rejected, then there cannot possibly be any forbidden factors in the complement of the stringset. For this reason, we do not extract costrict constraints as strict constraints of the complement of the given automaton, but instead as strict constraints of the difference between the strict approximation and the given automaton (the residue). This removes the effect of the strict constraints. If there are any strict constraints in the residue, then these will include the costrict constraints, but in general these will also contain factors that are not actually sufficient to satisfy the constraints of the intended stringset.

Because of this, we must find some (preferably minimal) subset of factors where at least one must occur. We use a search tree of subsets of the set of discovered factors, where the root is the entire set and the children are those subsets containing one fewer element. If the intersection of the given automaton with the complement of all factors in a set is empty, then some subset of these factors is required. Otherwise, no subset of this subset will contain the full set of the stringset's costrict constraints, and so we can prune its elements from every subset in our frontier. The smallest subset that was found to be sufficient is the set of costrict constraints.

The conjunction of these costrict constraints with the earlier strict ones is a strict+costrict approximation of the given stringset. This is the extent to which `factorize` is able to extract fully automatically. For constraints of higher complexity, it needs help from the user. A (possibly empty) set of non-strict constraints must be provided, which will be tested in turn.

If the conjunction of the strict+costrict approximation with one of these constraints is equal to the given automaton, this is reported. Note that the conjunctions must be explicitly provided; `factorize` does not perform this on its own because one may wish to test only specific combinations. If none of these constraints is able to complete the factorization, then this is also reported. Note also that costrict constraints may need to be included, as they may not always be extractable (as explained previously).

This is the tool we have used to produce the factorizations of the lects of the StressTyp2 database reported in [RL17, RL18]. In order to run `factorize` one needs two resources. The first is a collection of automata files, in Jeff format, that recognize the stringsets to be factored. Jeff format is a (deprecated) format of the automata in StressTyp2. It was the only format provided by the Stress Pattern Database which is one of the progenitors of StressTyp2. It is supported because this line of research started before the evolution of StressTyp2. It is the only supported format because it suffices.

This version of `factorize` expects (read requires) automata filenames to be in the StressTyp2 format '`<nnn>_<lect name>`' where '`<nnn>`' is the lect number in the database and '`<lect name>`' is a string naming the lect. The results are written to a collection of files '`./Results/<nnn>`'. Then contents of these files should be self-explanatory, except that if the non-strict constraints section includes the line '`complete="no"`', then the file contains only the strict+costrict approximation of the stringset.

The second is a set of prioritized constraints that comprise one's current hypothesis about non-strict constraints that may be necessary to fully define the stringsets. These are contained in a directory '`./Compiled/`' which contains a collection of Jeff-format FSA files that embody the constraints, along with a file '`./Compiled/constraints`' which is a text file listing the filenames, relative to '`./`', of the constraints (i.e., '`Compiled/<basename>`') in order of preference, usually lowest complexity first.

Note that `factorize` does not automatically check the conjunctions of each subset of these constraints; the user must explicitly provide each desired combination.

Important: The file '`./Compiled/constraints`' must exist, although it may be empty.

Warning: Every line in '`./Compiled/constraints`' will be interpreted as the name of a compiled constraint. Any stray lines, including empty lines, will cause `factorize` to fail with an `openFile` error. This 'empty' means 0 lines and 0 characters — no spaces or newlines at all.

Jeff-format files for the lects in StressTyp2 are available in the `st2-v1-archive-0415/transducers/fsasJeff/` directory of its archive (available from `http://st2.ullet.net/files/files/st2-v1-archive-0415.tar.gz`).

A set of non-strict constraints adequate for all of the StressTyp2 lects can be generated generated by the executable '`make-non-strict-constraints`' which must be run in the '`./Compiled`' directory (as `../make-non-strict-constraints`). This will generate all subsets of the constraints we refer to as c89, c9x, c91, c145, and c146 in the pleb file '`stressConstraints.pleb`'.

A summary of the factorization process described above:

1. Extract SL and SP constraints from the given automaton.

2. Delete redundant constraints.

3. Build a strict approximation from the conjunction of these constraints.
4. Find the difference between this approximation and the given automaton, and call this difference the residue.
5. Repeat steps 1 and 2 on the residue to find the potential costrict constraints.
6. In order to find a minimal sufficient costrict constraint:[1]
   (a) Create a search tree of subsets of these potential costrict constraints, where the root is the entire set, and the children of a node are its subsets that are smaller by exactly one element.
   (b) For each node in the frontier, construct the automaton that represents the forbidding of all factors in the set. If the intersection of this automaton with the given one is empty, then at least one of its subsets is a sufficient costrict constraint. This set is then a viable candidate. Replace it in the frontier by its children.
   (c) On the other hand, if this intersection is nonempty, then no subset of this set is a sufficient costrict constraint. Prune its elements from every element of the frontier.
   (d) Repeat steps 6b–6d with a largest element of the frontier. Eventually the frontier will be exhausted.
   (e) Then the minimal sufficient costrict constraint is the smallest viable candidate discovered.
7. Create a strict+costrict approximation by intersection the strict approximation with the automaton representing this minimal sufficient costrict constraint.
8. If the stringset is strict+costrict, this is sufficient.
9. Else test each hypothesized constraint from "`./Compiled/constraints`" in order by intersection with the strict+costrict approximation. The first constraint such that this intersection is equal to the given automaton is the desired non-strict constraint. If no such constraint exists, the factorization is incomplete—merely a strict+costrict approximation of the given automaton.

References

[RL17] James Rogers and Dakotah Lambert, *Extracting forbidden factors from regular stringsets*, Proceedings of the 15th Meeting on the Mathematics of Language, Association for Computational Linguistics, 2017, pp. 36–46.

[RL18] ———, *Extracting subregular constraints from regular stringsets*, Under review, 2018.

[1] A costrict constraint is a set of factors where a string satisfies the constraint if and only if it contains at least one of these factors. A sufficient costrict constraint is one where a string is necessarily rejected if it does not contain a factor from the set. If there is more than one sufficient costrict constraint, a minimal one is one with the fewest elements.