# Basic format for describing operations

### Abstract

This document describes the syntax for three different judges. The first one is a judge that allows to construct regular languages by means of regularity-preserving operations applied on automata. The second one extends the previous judge by allowing also the introduction of context-free grammars, thus being able to define context-free languages. The third one is focused on reducing undecidable problems on context-free grammars.

## 1 Regular languages

Before formally detailing the format we introduce an illustrative example. The program

defines a variable a whose language is  $\{a\}$ , a variable b whose language is  $\{b\}$ , a variable ab whose language is the union of the two previous variables, i.e.,  $\{a,b\}$ , and a variable even\_a that is assigned an automaton recognizing the language  $\{w \in \{a,b\}^* \mid |w|_a \in \dot{2}\}$ . Finally, it outputs the language obtained by: the Kleene's closure of the variable ab, concatenated with the variable a, concatenated twice with the variable ab, and everything intersected with the variable even\_a, i.e., the output is  $\{w_1aw_2 \mid w_1, w_2 \in \{a,b\}^* \land |w_2| = 2 \land |w_1aw_2|_a \in \dot{2}\}$ .

Now we precise the syntax of the programs. In order to describe a regular language it suffices to write a program of the following form:

```
main {
     <ident1> = <expr1>;
          ...
     <identN> = <exprN>;
     output <expr>;
}
```

where the <ident1>, ..., <identN> are variable identifiers, and the <expr1>, ..., <exprN>, <expr> are expressions over regular languages using operators that preserve the regularity. More concretely, a basic expression is one of the following:

- A variable identifier (see **IDENTIFIER** below).
- A word (representing the language that contains exactly that word; see **WORD** below).
- A deterministic finite automaton (representing its recognized language; see **DFA** below).

Expressions can be combined using the following regularity-preserving operators:

- Language intersection, denoted by the binary operator &.
- $\bullet$  Language union, denoted by the binary operator  $\mid$  .
- Language subtraction, denoted by the binary operator -.

- Language concatenation, a binary operation denoted without any symbol.
- Kleene's closure of the language, denoted by the unary postfix operator \*.
- Language reverse, denoted by the function reverse (...).
- Language substitution, denoted by the function **substitution(...)**. This function has as a first parameter the expression defining the language where the substitution must be performed. The actual substitution is defined by the following parameters, each being of the form **word** -> expr, denoting that the image of the terminal symbol **word** is the language defined by the expression expr (the **word** must have one single symbol).

As another example, the following two programs define the language of words over the alphabet  $\{a,b\}$  having the subword abbba:

As a final remark, we provide the grammar describing the syntax of the programs:

where:

- IDENTIFIER is a string over alphanumeric characters and underscore,
- WORD is a string delimited by quotation marks " and composed over lower-case letters, digits, and the special characters +-\*/()[]><, and
- **DFA** is a string delimited by quotation marks " that describes a deterministic finite automaton.

## 2 Context-free languages

This format extends the previous one by adding the option to have context-free grammars as basic literals. More precisely, **basic** can also be a token **CFG**, that is, a string delimited by quotation marks " that describes a context-free grammar. For example, the program:

```
main {
   g = "S -> aSa | bSb | a | b |";
   output g - ("a"|"b") * "abbba" ("a"|"b")*;
}
```

outputs the language of palindromes over  $\{a, b\}$  that do not contain the subword abbba.

Recall that context-free languages are not closed under intersection or complementation. Thus, an expression like op1 & op2 is invalid when op1 and op2 are both context-free languages, and an expression like op1 - op2 is invalid when op2 is a context-free language. Due to technical reasons, we do not allow the empty word in the image of a symbol under a substitution when context-free languages are involved.

## 3 Reductions of grammars

We modify the previous programming language in order to allow reductions from undecidable problems on CFG, such as universality or non-empty intersection. To this end, we have the following redefinitions of program and instruction:

Note that the program receives input in the form of input variables, multiple output is permitted, and conditional execution is allowed by checking word membership to a language.

For example, to prove undecidability of the equivalence problem between two CFGs over  $\{a,b\}$  we can perform a reduction from the undecidable problem of universality on CFGs, i.e.,  $\{G \mid \mathcal{L}(G) = \{a,b\}^*\} \leq \{\langle G_1,G_2\rangle \mid \mathcal{L}(G_1) = \mathcal{L}(G_2)\}$ . The program to compute such reduction receives as input a single grammar G, and outputs two grammars  $G_1$  and  $G_2$  satisfying that  $\mathcal{L}(G) = \{a,b\}^*$  if and only if  $\mathcal{L}(G_1) = \mathcal{L}(G_2)$ . The following program performs such reduction:

```
input g {
  output ("a"|"b")* , g;
}
```

Note that the program outputs two grammars, the first one generates  $\{a,b\}^*$  and the second one is precisely the input grammar g.