Manipulation of Extended Regular Expressions with Derivatives

Rafaela Bastos  Nelma Moreira  Rogério Reis
CMUP & DCC, Faculdade de Ciências da Universidade do Porto
Rua do Campo Alegre, 4169-007 Porto, Portugal

Version 1.0 September 2013
Manipulation of Extended Regular Expressions with Derivatives*

Rafaela Bastos, Nelma Moreira, Rogério Reis
up200801458@alunos.dcc.fc.up.pt ,{nam,rvr}@dcc.fc.up.pt
CMUP & DCC-FC, Universidade do Porto
Rua do Campo Alegre 687, 4169-007 Porto, Portugal
DCC-FC, Universidade do Porto
Rua do Campo Alegre 1021/1055, 4169-007 Porto, Portugal

Abstract

The use of derivatives for efficiently deciding equivalence and membership in regular languages has been a major topic of recent research. To ensure termination, regular expressions must be considered modulo some algebraic properties such as associativity, commutativity, and idempotence of union (ACI). In this paper we describe an implementation of regular expressions modulo ACI and several derivative based methods within the FAdo system. Although regular languages are trivially closed for boolean operations, the manipulation of intersection and complementation with regular expressions or non-deterministic finite automata is non trivial and leads to an exponential blow up. However, due to several applications where extended regular expressions (XRE) are used to represent information, it is important the extension of derivative based methods to those operations. Continuing work of Caron et al., we present new algorithms for computing the (extended) equation automaton and deciding membership and equivalence of XRE using (partial) derivatives.

1 Extended Regular Expressions and Kleene Algebra

In this section we briefly review some basic definitions about extended regular expression and Kleene Algebra.

Let \( \Sigma = \{\sigma_1, \ldots, \sigma_k\} \) be an alphabet of size \( k \). A word over an alphabet \( \Sigma \) is a finite sequence of symbols of \( \Sigma \). The empty word is denoted by \( \epsilon \). The set \( \Sigma^* \) is the set of all words over \( \Sigma \). A language over \( \Sigma \) is a subset of \( \Sigma^* \). The set \( R \) of extended regular expressions (ERE) over an alphabet \( \Sigma \) is defined by:

\[
\alpha := \emptyset | \epsilon | \Sigma^+ | \Sigma^* | \sigma_1 | \ldots | \sigma_n | (\alpha + \beta) | (\alpha.\beta) | \alpha^* | (\alpha \cap \beta) | \neg \alpha
\]

where \( \Sigma^+ \) and \( \Sigma^* \) are equivalent to \( \neg \epsilon \) and \( \neg \emptyset \), respectively, and the operator . (concatenation) is often omitted.

*This work was partially funded by the European Regional Development Fund through the programme COMPETE and by the Portuguese Government through the FCT under projects PEst-C/MAT/UI0144/2011 and CANTE-PTDC/EIA-CCO/101904/2008.
The language $L(\alpha)$ associated to $\alpha$, is inductively defined as follows:

\[
L(\emptyset) = \emptyset \quad L(\alpha + \beta) = L(\alpha) \cup L(\beta) \\
L(\epsilon) = \{\epsilon\} \quad L(\alpha \cap \beta) = L(\alpha) \cap L(\beta) \\
L(\Sigma^*) = \Sigma^* \setminus L(\emptyset) \quad L(\alpha \beta) = L(\alpha).L(\beta) \\
L(\Sigma^+) = \Sigma^* \setminus L(\epsilon) \quad L(\alpha^*) = L(\alpha)^* \\
L(\sigma) = L(\sigma) \quad L(\neg \alpha) = \Sigma^* \setminus L(\alpha)
\]

where $\sigma \in \Sigma$.

An extended regular expression represents the empty word $\epsilon$ if and only if:

\[
e(\emptyset) = \emptyset \quad e(\epsilon) = \epsilon \quad e(\Sigma^*) = \epsilon \quad e(\Sigma^+) = \emptyset \\
e(\sigma) = \emptyset \quad e(\neg \alpha) = \begin{cases} \epsilon, & \text{if } \epsilon(\alpha) \neq \epsilon \\ \emptyset, & \text{otherwise} \end{cases} \\
e(\alpha + \beta) = e(\alpha) + e(\beta)
\]

We say that two regular expressions $\alpha$ and $\beta$ are equivalent, $\alpha \equiv \beta$, if they represent the same language, i.e., $L(\alpha) = L(\beta)$.

Let $\sigma, \sigma_1, \ldots, \sigma_n$ be symbols of the alphabet $\Sigma$ and $\alpha$, $\beta$ and $\gamma$ be extended regular expressions. From the axiomy sytem AX in [AM94] and the axiom system F in [Sal66], we have the equivalences:

\[
\alpha + (\beta + \gamma) \equiv (\alpha + \beta) + \gamma \quad (A_1) \\
(\alpha.\beta).\gamma \equiv \alpha.(\beta.\gamma) \quad (A_2) \\
\alpha + \beta \equiv \beta + \alpha \quad (A_3) \\
\alpha.(\beta + \gamma) \equiv \alpha.\beta + \alpha.\gamma \quad (A_4) \\
(\alpha + \beta).\gamma \equiv \alpha.\gamma + \beta.\gamma \quad (A_5) \\
\alpha + \alpha \equiv \alpha \quad (A_6) \\
\alpha.\epsilon \equiv \alpha \quad (A_7) \\
\alpha.\emptyset \equiv \emptyset \quad (A_8) \\
\alpha + \emptyset \equiv \alpha \quad (A_9) \\
\epsilon + \alpha.\alpha^* \equiv \alpha^* \quad (A_{10}) \\
(\epsilon + \alpha)^* \equiv \alpha^* \quad (A_{11}) \\
\epsilon \cap (\alpha.\beta) \equiv (\epsilon \cap \alpha) \cap \beta \quad (A_{12}) \\
\epsilon \cap \alpha^* \equiv \epsilon \quad (A_{13}) \\
\epsilon \cap \sigma \equiv \emptyset \quad (A_{14}) \\
\emptyset \cap \alpha \equiv \emptyset \quad (A_{15}) \\
\alpha \cap \alpha \equiv \alpha \quad (A_{16}) \\
\alpha \cap \beta \equiv \beta \cap \alpha \quad (A_{17}) \\
\alpha \cap (\beta \cap \gamma) \equiv (\alpha \cap \beta) \cap \gamma \quad (A_{18}) \\
\alpha \cap (\beta + \gamma) \equiv (\alpha \cap \beta) + (\alpha \cap \gamma) \quad (A_{19}) \\
\alpha + (\alpha \cap \beta) \equiv \alpha \quad (A_{20}) \\
(\sigma_1.\alpha) \cap (\sigma_2.\beta) \equiv (\sigma_1 \cap \sigma_2).(\alpha \cap \beta) \quad (A_{21}) \\
(\alpha.\sigma_1) \cap (\beta.\sigma_2) \equiv (\alpha \cap \beta).(\sigma_1 \cap \sigma_2) \quad (A_{22}) \\
\sigma_i \cap \sigma_j \equiv \emptyset \quad \forall \sigma_i \neq \sigma_j \quad (A_{23})
\]
\[-\alpha \cap -\beta \equiv -\alpha + -\beta \quad (A_{24})
\[-\alpha + -\beta \equiv -\alpha \cap -\beta \quad (A_{25})\]

The derivative of an extended regular expression \( \alpha \) with respect to a symbol \( \sigma \in \Sigma \), written \( d_\sigma(\alpha) \), is a regular expression such that:

\[
L(d_\sigma(\alpha)) = \{ w \mid \sigma w \in L(\alpha) \}
\]

And the inductively definition is the following:

\[
\begin{align*}
    d_\sigma(\emptyset) &= \emptyset \\
    d_\sigma(\epsilon) &= \emptyset \\
    d_\sigma(\sigma) &= \epsilon \\
    d_\sigma(\sigma') &= \emptyset \\
    d_\sigma(\alpha + \beta) &= d_\sigma(\alpha) + d_\sigma(\beta) \\
    d_\sigma(\alpha \cap \beta) &= d_\sigma(\alpha) \cap d_\sigma(\beta) \\
    d_\sigma(\alpha \beta) &= d_\sigma(\alpha)\beta + \epsilon(\alpha)d_\sigma(\beta) \\
    d_\sigma(\alpha^*) &= d_\sigma(\alpha)\alpha^* \\
    d_\sigma(-\alpha) &= -d_\sigma(\alpha)
\end{align*}
\]

Two extended regular expressions are modulo-aci if one can be transformed to the other by using the aci-rules, i.e., applying the associativity, commutativity and idempotence of the intersection \((\cap)\) (Axioms \(A_{18}, A_{17}\) and \(A_{16}\)) and disjunction \((+\)\) (Axioms \(A_1, A_3\) and \(A_6\)) operators, and applying the associativity of the concatenation \((.)\) (Axiom \(A_2\)). Brzozowski proved in [Brz64] that the set of derivatives of a regular expression being a finite set is enough that is modulo-aci.

The set of partial derivative of a non-extended regular expression w.r.t. a symbol \( \sigma \in \Sigma \), denoted by \( \partial_\sigma(\alpha) \), is the set of regular expressions defined as follows:

\[
\begin{align*}
    \partial_\sigma(\emptyset) &= \emptyset \\
    \partial_\sigma(\epsilon) &= \emptyset \\
    \partial_\sigma(\sigma) &= \{ \epsilon \} \\
    \partial_\sigma(\sigma') &= \emptyset \\
    \partial_\sigma(\alpha + \beta) &= \partial_\sigma(\alpha) \cup \partial_\sigma(\beta) \\
    \partial_\sigma(\alpha \beta) &= \partial_\sigma(\alpha)\beta \cup \epsilon(\alpha)\partial_\sigma(\beta) \\
    \partial_\sigma(\alpha^*) &= \partial_\sigma(\alpha)\alpha^*
\end{align*}
\]

2 Derivatives

In [PCM11] Caron et al. describe a definition for the partial derivative of an extended regular expression, though this definition requires that the regular expression must be in disjunction normal form. In this section, a new definition for partial derivative is described that not requires such assumption.

Let \( \Sigma = \{\sigma_1, \ldots, \sigma_k\} \) be an alphabet of size \( k \). Consider the following for represent an extended regular expressions (ERE) over the alphabet \( \Sigma \):
\[ \alpha := \emptyset \mid \epsilon \mid \Sigma^+ \mid \sigma_1 \mid \ldots \mid \sigma_n \mid [\alpha, \ldots, \alpha] \mid \alpha.\alpha \mid \alpha^* \mid \langle \alpha, \ldots, \alpha \rangle \mid \neg \alpha \]

Where \([\alpha_1, \ldots, \alpha_n] \) and \(\langle \alpha_1, \ldots, \alpha_n \rangle \) represent the regular expressions \((\alpha_1 + \ldots + \alpha_n) \) and \((\alpha_1 \cap \ldots \cap \alpha_n) \), respectively.

**Definition 1.** Let \(\alpha\) be an extended regular expression and \(\sigma\) a symbol in \(\Sigma\), the partial derivative of \(\alpha\) w.r.t. \(\sigma\), \(\partial_\sigma(\alpha)\), is defined inductively as follows:

\[
\begin{align*}
\partial_\sigma(\Sigma^*) &= \{ \Sigma^* \} \\
\partial_\sigma(\Sigma^+) &= \{ \Sigma^* \} \\
\partial_\sigma(\emptyset) &= \emptyset \\
\partial_\sigma(\epsilon) &= \emptyset \\
\partial_\sigma(\sigma) &= \{ \epsilon \} \\
\partial_\sigma(\sigma') &= \emptyset
\end{align*}
\]

\[
\begin{align*}
\overline{\partial_\sigma}(\Sigma^*) &= \emptyset \\
\overline{\partial_\sigma}(\Sigma^+) &= \emptyset \\
\overline{\partial_\sigma}(\emptyset) &= \{ \Sigma^* \} \\
\overline{\partial_\sigma}(\epsilon) &= \{ \Sigma^* \} \\
\overline{\partial_\sigma}(\sigma) &= \{ \Sigma^+ \} \\
\overline{\partial_\sigma}(\sigma') &= \{ \Sigma^* \}
\end{align*}
\]

**Lemma 1.** Let \(\alpha\) be an extended regular expression and \(\sigma \in \Sigma\), thus

\[ L(d_\sigma(-\alpha)) = \Sigma^* \setminus d_\sigma(\alpha) \quad (2.1) \]

**Proof.** According to the definition of derivatives, we have:

\[
\begin{align*}
L(d_\sigma(-\alpha)) &= \{ w \mid \sigma w \in L(-\alpha) \} \\
&= \{ w \mid \sigma w \notin L(\alpha) \} \\
&= \Sigma^* \setminus \{ w \mid \sigma w \in L(\alpha) \} \\
&= \Sigma^* \setminus d_\sigma(\alpha)
\end{align*}
\]

\(\square\)

**Proposition 1.** Let \(\alpha\) be an extended regular expression over an alphabet \(\Sigma\) and \(\sigma\) a symbol in \(\Sigma\), then

\[ L(\partial_\sigma(-\alpha)) = \Sigma^* \setminus \partial_\sigma(\alpha) \quad (2.2) \]

**Proof.** By induction on the structure of the extended regular expression \(\alpha\):

For \(\alpha = \Sigma^*:\)

\[
L(\overline{\partial_\sigma}(\Sigma^*)) = L(\overline{\partial_\sigma}(-\emptyset)) = L(\partial_\sigma(\emptyset)) = L(\emptyset) = \emptyset.
\]

If \(\alpha\) is \(\Sigma^+:\)

\[
L(\overline{\partial_\sigma}(\Sigma^+)) = L(\overline{\partial_\sigma}(-\epsilon)) = L(\partial_\sigma(\epsilon)) = L(\emptyset) = \emptyset.
\]
If $\alpha$ is $\emptyset$, $\epsilon$ or $\sigma'$:

$$L(\overline{\partial_\sigma(\alpha)}) = L(\{\Sigma^*\}) = \Sigma^* = \Sigma^* \backslash \emptyset = \Sigma^* \backslash \partial_\sigma(\alpha).$$

If $\alpha$ is $\sigma$:

$$L(\overline{\partial_\sigma(\sigma)}) = L(\{\Sigma^+\}) = \Sigma^+ = \Sigma^+ \backslash \{\epsilon\} = \Sigma^+ \backslash \partial_\sigma(\sigma).$$

Consider that $\alpha$ is $[\alpha_1, \ldots, \alpha_n]$, where $\alpha_i$ is an extended regular expression, for $i = 1, \ldots, n$:

$$L(\overline{\partial_\sigma([\alpha_1, \ldots, \alpha_n])}) = L(\{\neg \alpha' \mid \exists \alpha' \in \partial_\sigma(\alpha_i)\})$$

$$= L(\{\neg \alpha' \mid \exists \alpha' \in \partial_\sigma(\alpha_i)\})$$

$$= \Sigma^* \backslash \{\alpha' \mid \exists \alpha' \in \partial_\sigma(\alpha_i)\}$$

$$= \Sigma^* \backslash \partial_\sigma([\alpha_1, \ldots, \alpha_n]).$$

If $\alpha$ is $\langle \alpha_1, \ldots, \alpha_n \rangle$:

$$L(\overline{\partial_\sigma(\langle \alpha_1, \ldots, \alpha_n \rangle)}) = L(\{\langle \neg \alpha' \mid \alpha' \in \bigcup \partial_\sigma(\alpha_i) \rangle \mid \forall i\})$$

$$= L(\langle \neg \alpha' \mid \alpha' \in \partial_\sigma(\alpha_1) \rangle \cup \ldots \cup L(\langle \neg \alpha' \mid \alpha' \in \partial_\sigma(\alpha_n) \rangle)$$

$$= L(\langle \neg \alpha' \mid \alpha' \in \partial_\sigma(\alpha_1) \rangle \cup \ldots \cup L(\langle \neg \alpha' \mid \alpha' \in \partial_\sigma(\alpha_n) \rangle)$$

$$= \Sigma^* \backslash L(\{\alpha' \mid \alpha' \in \partial_\sigma(\alpha_1)\}) \cup \ldots \cup L(\{\alpha' \mid \alpha' \in \partial_\sigma(\alpha_n)\})$$

$$= \Sigma^* \backslash \{\langle \alpha'_1, \ldots, \alpha'_n \rangle \mid \alpha'_i \in \partial(\alpha_i)\}$$

$$= \Sigma^* \backslash \partial_\sigma(\langle \alpha_1, \ldots, \alpha_n \rangle).$$

If $\alpha$ is $\alpha_1 \alpha_2$:

$$L(\overline{\partial_\sigma(\alpha_1 \alpha_2)}) = L(\{\langle \neg \alpha' \mid \alpha' \in \partial_\sigma(\alpha_1 \alpha_2) \rangle\})$$

$$= L(\langle \neg \alpha' \mid \alpha' \in \partial_\sigma(\alpha_1 \alpha_2) \rangle)$$

$$= \Sigma^* \backslash L(\{\alpha' \mid \alpha' \in \partial_\sigma(\alpha_1 \alpha_2)\})$$

$$= \Sigma^* \backslash L(\partial_\sigma(\alpha_1 \alpha_2)).$$

If $\alpha$ is $\beta^*$, where $\beta$ is an extended regular expression:

$$L(\overline{\partial_\sigma(\beta^*)}) = L(\{\langle \neg \alpha' \mid \alpha' \in \partial_\sigma(\beta^*) \rangle\})$$

$$= L(\langle \neg \alpha' \mid \alpha' \in \partial_\sigma(\beta^*) \rangle)$$

$$= \Sigma^* \backslash L(\{\alpha' \mid \alpha' \in \partial_\sigma(\beta^*)\})$$

$$= \Sigma^* \backslash L(\partial_\sigma(\beta^*)).$$

If $\alpha$ is $\neg \beta$:

$$L(\overline{\partial_\sigma(\neg \beta)}) = L(\partial_\sigma(\neg \neg \beta)) = L(\partial_\sigma(\beta)).$$
Example 1. Let $\alpha = -(a^* \cap -ba)$ be an extended regular expression with $\Sigma = \{a, b\}$, where $\alpha$ is represented in the implementation by $-\langle a^*, -ba \rangle$. The correspondent partial derivative $\partial_a(\alpha)$ is calculate as follows:

$$\partial_a(-\langle a^*, -ba \rangle) = \partial_a(-\langle a^*, -ba \rangle)$$

$$= \{\langle -f \mid f \in \partial_a(a^*) \rangle \} \cup \{\langle -f \mid f \in \partial_a(-ba) \rangle \}$$

Since,

$$\partial_a(a^*) = \{a^*\}$$

and

$$\partial_a(-ba) = \{f_1a \mid f_1 \in \partial_b(-b) \} \cup \partial_a(a)$$

$$= \{f_1a \mid f_1 \in \partial_a(b) \} \cup \partial_a(a)$$

$$= \{\Sigma^*a \} \cup \{\epsilon\}$$

$$= \{\Sigma^*a, \epsilon\},$$

because $\epsilon(-b) = \epsilon$. We have:

$$\partial_a(-\langle a^*, -ba \rangle) = \{\langle -f \mid f \in \{a^*\} \rangle \} \cup \{\langle -f \mid f \in \{\Sigma^*a, \epsilon\} \rangle \}$$

$$= \{a^*, \Sigma^*a, \epsilon\}.$$

Champarnaud and Ziadi [CZ01] showed that partial derivatives and Mirkin’s prebases [Mir66] lead to identical constructions of non-deterministic automata. Here, we give an extended version for intersection of the algorithm.

Let $\alpha_0$ be a regular expression. A set $\pi(\alpha_0) = \{\alpha_1, \ldots, \alpha_n\}$, where $\alpha_1, \ldots, \alpha_n$ are non-empty regular expressions, is called a support of $\alpha_0$ if, for $i = 0, \ldots, n$, there are $\alpha_i \in R$ $(l = 1, \ldots, k)$, linear combinations of the elements in $\pi(\alpha_0)$, such that $\alpha_i = \sigma_1.\alpha_{i1} + \ldots + \sigma_k.\alpha_{ik} + \epsilon(\alpha_i)$, where, as above, $\Sigma = \{\sigma_1, \ldots, \sigma_k\}$ is the considered alphabet. If $\pi(\alpha)$ is a support of $\alpha$, then the set $\pi(\alpha) \cup \{\alpha\}$ is called a prebase of $\alpha$.

Proposition 2. Let $\alpha$ be an extended regular expression and $\sigma$ a symbol of the alphabet, then the set $\pi(\alpha)$, inductively defined by

$$\pi(\Sigma^*) = \{\Sigma^*\}$$

$$\pi(\Sigma^+) = \{\Sigma^*\}$$

$$\pi(\emptyset) = \emptyset$$

$$\pi(\epsilon) = \emptyset$$

$$\pi(\sigma) = \{\epsilon\}$$

$$\pi(\alpha + \beta) = \pi(\alpha) \cup \pi(\beta)$$

$$\pi(\alpha \cap \beta) = \{\alpha' \cap \beta' \mid \alpha' \in \pi(\alpha), \beta' \in \pi(\beta)\}$$

$$\pi(\alpha \beta) = \{\alpha'\beta' \mid \beta' \in \pi(\alpha)\} \cup \pi(\beta)$$

$$\pi(\alpha^*) = \{\alpha'^* \mid \alpha' \in \pi(\alpha)\}$$

is a support of $\alpha$. 
Proof. In [CZ01] is the proof for all definitions except for \( \pi(\beta \cap \gamma) \), which is proved below.

Let \( \pi(\beta_0) = \{\beta_1, \ldots, \beta_n\} \) and \( \pi(\gamma_0) = \{\gamma_1, \ldots, \gamma_m\} \) be a support of \( \beta_0 \) and \( \gamma_0 \), respectively. Thus, for \( i = 0, \ldots, n \) one has:

\[
\begin{align*}
\pi(\beta_0) &= \{\beta_0, \ldots, \beta_n\} \\
\pi(\gamma_0) &= \{\gamma_1, \ldots, \gamma_m\}
\end{align*}
\]

Thus, for \( i = 0, \ldots, n \) one has:

\[
\begin{align*}
\beta &= \sum_{r=1}^{k} \sigma_i \beta_{ri} + \epsilon_i \\
\beta_i &= \sum_{r=1}^{k} \sigma_i \beta_{ri} + \epsilon_i
\end{align*}
\]

where \( \beta_{i1}, \ldots, \beta_{ik}, \) for \( i = 0, \ldots, n \) and \( l = 1, \ldots, k \), is a linear combination of elements in \( \pi(\beta_0) \).

For \( j = 0, \ldots, m \):

\[
\begin{align*}
\gamma &= \gamma_0 \\
\gamma_j &= \sum_{r=1}^{k} \sigma_i \gamma_{rj} + \epsilon_j
\end{align*}
\]

where \( \gamma_{jl}, \ldots, \gamma_{jk}, \) for \( j = 0, \ldots, n \) and \( l = 1, \ldots, k \), is a linear combination of elements in \( \pi(\gamma_0) \).

Consider \( \alpha = \beta_0 \cap \gamma_0 \), so

\[
\pi(\beta_0 \cap \gamma_0) = \{\beta_{0i} \cap \gamma_{0j} \mid \beta_{0i} \in \pi(\beta_0), \gamma_{0j} \in \pi(\gamma_0)\}.
\]

Since,

\[
\begin{align*}
\beta_0 &= \sum_{i=1}^{k} \sigma_i \beta_{0i} + \epsilon(\beta_0) \\
\gamma_0 &= \sum_{j=1}^{k} \sigma_i \gamma_{0j} + \epsilon(\gamma_0)
\end{align*}
\]

Therefore:

\[
\begin{align*}
\beta_0 \cap \gamma_0 &= \sum_{i=1}^{k} \sigma_i \beta_{0i} + \epsilon(\beta_0) \cap \sum_{j=1}^{k} \sigma_i \gamma_{0j} + \epsilon(\gamma_0) \\
&= (\sigma_i \beta_{01} \cap \sigma_i \gamma_{01}) + \ldots + (\sigma_i \beta_{0k} \cap \sigma_i \gamma_{0k}) + (\sigma_i \beta_{01} \cap \epsilon(\gamma_0)) + \ldots + (\sigma_i \beta_{0k} \cap \epsilon(\gamma_0)) + \epsilon(\beta_0 \cap \gamma_0) \\
&= (\sigma_1 \cap \sigma_i \beta_{01} \cap \gamma_{01}) + \ldots + (\sigma_k \cap \sigma_i \beta_{0k} \cap \gamma_{0k}) + (\sigma_i \beta_{01} \cap \gamma_{01}) + \ldots + (\sigma_i \beta_{0k} \cap \gamma_{0k}) + \epsilon(\beta_0 \cap \gamma_0) \\
&= (\sigma_1 \beta_{01} \cap \gamma_{01}) + \ldots + (\sigma_k \beta_{0k} \cap \gamma_{0k}) + \epsilon(\beta_0 \cap \gamma_0) \\
&= \sum_{j=1}^{k} \sigma_i (\beta_{0i} \cap \gamma_{0j}) + \epsilon(\beta_0 \cap \gamma_0).
\end{align*}
\]

Let \( \pi(\beta_0) = \{\beta_1, \ldots, \beta_n\} \) be a support of \( \beta_0 \). Thus, for \( i = 0, \ldots, n \) one has

\[
\begin{align*}
\beta &\equiv \beta_0 \\
\beta_i &= \sum_{r=1}^{k} \sigma_i \beta_{ri} + \epsilon_i
\end{align*}
\]

where \( \beta_{il}, \ldots, \beta_{ik}, \) for \( i = 0, \ldots, n \) and \( l = 1, \ldots, k \), is a linear combination of elements in \( \pi(\beta_0) \).

\[
\square
\]
Example 2. Consider the regular expression $\alpha = aa^* \cap a$ with $\Sigma = \{a\}$. The support for $\alpha$ is

\[ \pi(aa^* \cap aa) = \{\beta \cap \gamma \mid \beta \in \pi(aa^*), \gamma \in \pi(a)\} \]

Since,

\[ \pi(aa^*) = \{\beta^a \mid \beta \in \pi(a)\} \cup \pi(a^*) \]
\[ = \{\beta^a \mid \beta \in \{\epsilon\}\} \cup \{a^*\} \]
\[ = \{\epsilon\} \cup \{a^*\} \]

and

\[ \pi(aa) = \{a, \epsilon\}. \]

Thus,

\[ \pi(a^* \cap aa) = \{\beta \cap \gamma \mid \beta \in \{\epsilon a^*\}, \gamma \in \{a, \epsilon\}\} \]
\[ = \{\epsilon \cap a, \epsilon \cap \epsilon, a^* \cap a, a \cap \epsilon\} \]
\[ = \{\epsilon, a^* \cap a\} \]

For the regular expression $\beta = a^* \cap a$, we have

\[ \pi(a^* \cap a) = \{\beta \cap \gamma \mid \beta \in \pi(a^*), \gamma \in \pi(a)\} \]
\[ = \{\beta \cap \gamma \mid \beta \in \{a^*\}, \gamma \in \{\epsilon\}\} \]
\[ = \{a^* \cap \epsilon\}. \]

Note that in the second example of Example 2 the closure of partial derivatives of the regular expression $a^* \cap a$ is equal to support $\pi(a^* \cap a)$ of it.

3 XRE in FAdo

The XRE is a class for the extended regular expressions in FAdo system that preserves the modulo-aci properties in a way to assure the finitude of some algorithms, such as dfaDerivatives (construction of Derivative DFA [Brz64]), nfaPD (Partial Derivative nfa [Ant96]) and equivP (verifies if two regular expressions are equivalent [Brz64]).

The intersections and disjunctions were implemented in XRE as sets of regular expressions, because it guarantees the modulo-aci properties. Consider as an example the following correspondences:

\[ a + b^*c + \emptyset + a + \epsilon \rightarrow \{a, b^*c, \epsilon\} \]
\[ a \cap a^*a \cap \emptyset \cap a \cap \epsilon \rightarrow \{a, a^*a, \epsilon\} \]

The concatenations were represented as ordered lists, which allows to take advantage of the associative concatenation, for example:

\[ a(a + c)^*a \rightarrow [a, (a + c)^*, a] \]
It was used the object-oriented paradigm of programming for the implementation of regular expressions, it had been used a different class for each of the operators (+, , , ∩, ¬, *). Figure 1 presents the classes for XRE and the principal methods coded. The xre class is the base class for all extended regular expression and the subclasses xsigmaP and xsigmaS represent the regular expressions Σ+ and Σ∗, respectively. The methods derivative, partialDerivative and linearForm are implemented for each subclass. The method support, that is defined and proved below, is not implemented for xnot. The same occurs for the method nfaGlushkov [Glu61] that is only for non-extended regular expressions. The algorithm equivP defined in [Alm11] verifies if two regular expressions are equal by creating both derivative automaton. Since we extended the derivatives in FAdo for intersection and negation, the equivP algorithm was extended too. Likewise, the algorithm PD that creates the closure of the partial derivatives of an extended regular expression in relation to all symbols occurring in it.

![Diagram of classes for XRE](image)

Figure 1: Classes for XRE.

References


