Constrained Expressions and their Derivatives

Jean-Marc Champarnaud, Ludovic Mignot and Florent Nicart

LITIS, Université de Rouen, 76801 Saint-Étienne du Rouvray Cedex, France
{jean-marc.champarnaud,ludovic.mignot,florent.nicart}@univ-rouen.fr

Abstract. First order logic without quantifiers, a.k.a. zeroth order logic, is more expressive than propositional logic. In this paper, we introduce two new operators for regular expressions allowing us to handle zeroth order boolean formula in order to increase the expressive power of expressions; We also extend the notion of interpretation and of realization using expression interpretation. We thus define the notion of constrained expression, the language denoted of which is not necessarily regular. Furthermore, we use an extension of Antimirov partial derivatives in order to solve the membership test in the general case. Finally, we show that once the interpretation is fixed, the membership test of a word in the language denoted by a constrained expression is undecidable whereas it is decidable when the interpretation is not fixed.

1 Introduction

Regular expressions are finite objects allowing us to represent potentially infinite languages. Their expressive power make them easy to manipulate and therefore they are widely used in numerous domains, such as pattern matching, specification or scheme validation. However, they are based on the combination of three simple operators (sum, catenation and iteration) that restrict their expressive power to regular languages. Even when any boolean operator is added to the set of operators, the expressive power of the so-called extended regular expressions is still the same.

Several approaches exist in order to make this expressive power larger; e.g. by adding new operators \cite{8} or by modifying the way symbols are combined \cite{4,17}. In these latter cases, the expressive power of this so-called regular-like expressions is increased to the linear languages, that is a strict subclass of context-free languages in the Chomsky hierarchy \cite{10}.

Our approach here is a little bit different; it is also based on the introduction of two new operators; however, their action is to build a link with the first order logic without quantifiers (a.k.a. zeroth order logic), allowing us to easily describe non-regular languages, using both predicates and variables in order to evaluate what we call constrained expressions. Given an expression $E$ and a boolean formula $\phi$, we define the expression $E \mid \phi$ (such that $\phi$) that denotes $L(E)$ when $\phi$ is satisfied, \emptyset otherwise. Given a word $\alpha$, based on symbols and variables, and an expression $E$, we define the expression $\alpha \models E$ denoting $\alpha$ if it is in $L(E)$, \emptyset otherwise. The combination of these two new operators allows us to go beyond regular languages.

The goal of this paper is to define these new operators, and to show how to interpret them. We also show how to solve the membership problem, that is to determine whether a given word belongs to the language denoted by a given regular expression.

The membership test for regular expression can be performed via the computation of a finite state machine, an automaton. However, we cannot apply this technique here, since we deal with non-regular languages and therefore with infinite state machine. Nevertheless, we apply a well-known method, the expression derivation \cite{2,3}, in order to reduce the membership problem of any word to the membership test of the empty word. Once this reduction made, we study the decidability of this problem.

Section 2 is a preliminary section: we recall here some basic definitions of formal language theory, such as languages or expression derivation. We also introduce the notion of zeroth-order logic that we use in the following of the paper. Section 3 defines the constrained expressions and the different languages they may denote. We also define in this section the way we derive them. Section 4 is devoted to illustrate the link between the empty word membership test and a satisfiability problem. In Section 5, we show that the satisfiability problem we use is decidable in the general case and that it is not in a particular subclass of our evaluations.
2 Preliminaries

2.1 Languages and Expressions

Let \( \Sigma \) be an alphabet. We denote by \( \Sigma^* \) the free monoid generated by \( \Sigma \) with \( \cdot \) the catenation product and \( \varepsilon \) its identity. Any element in \( \Sigma^* \) is called a word. A language over \( \Sigma \) is a subset of \( \Sigma^* \).

A language over \( \Sigma \) is regular if and only if it belongs to the family \( \text{Reg}(\Sigma) \) which is the smallest family containing all the subsets of \( \Sigma \) and closed under the three following operations:

- **union**: \( L \cup L' = \{ w \in \Sigma^* \mid w \in L \lor w \in L' \} \),
- **catenation**: \( L \cdot L' = \{ w \cdot w' \in \Sigma^* \mid w \in L \land w' \in L' \} \),
- **Kleene star**: \( L^* = \{ w_1 \cdots w_k \in \Sigma^* \mid k \geq 0 \land 1 \leq j \leq k, \ w_j \in L \} \).

A regular expression \( E \) over \( \Sigma \) is inductively defined as follows:

\[
E = a, \ E = \varepsilon, \ E = \emptyset, \\
E = (E_1) + (E_2), \ E = (E_1) \cdot (E_2), \ E = (E_1)^*,
\]

where \( a \) is any symbol in \( \Sigma \) and \( E_1 \) and \( E_2 \) are any two regular expressions over \( \Sigma \). Parenthesis can be omitted when there is no ambiguity. The language denoted by \( E \) is the language \( L(E) \) inductively defined by:

\[
L(a) = \{ a \}, \ L(\varepsilon) = \{ \varepsilon \}, \ L(\emptyset) = \emptyset, \\
L(E_1 + E_2) = L(E_1) \cup L(E_2), \ L(E_1 \cdot E_2) = L(E_1) \cdot L(E_2), \ L(E_1^*) = (L(E_1))^*,
\]

where \( a \) is any symbol in \( \Sigma \) and \( E_1 \) and \( E_2 \) are any two regular expressions over \( \Sigma \). It is a folk knowledge that the language denoted by a regular expression is regular.

Given a word \( w \) and a language \( L \), the membership problem is the problem defined by "Does \( w \) belong to \( L \)?". Many methods exist in order to solve this problem: as far as regular languages, given by regular expressions, are concerned, a finite state machine, called an automaton, can be constructed with a polynomial time complexity w.r.t. the size of the expression, that can decide with a polynomial time complexity w.r.t. its size if a word \( w \) belongs to the language denoted by the expression \( [11][13][15][19] \). See [12] for an exhaustive study of these constructions and of their descriptional complexities.

As far as regular expressions are concerned, the computation of a whole automaton is not necessary; the very structure of regular expressions is sufficient. Considering the residual \( w^{-1}(L) \) of \( L \) w.r.t. \( w \), that is the set \( \{ w' \in \Sigma^* \mid w w' \in L \} \), the membership test \( w \in L \) is equivalent to the membership test \( \varepsilon \in w^{-1}(L) \). This operation of quotient can be performed directly through regular expressions using the partial derivation \([2]\), which is an extension of the derivation \([3]\).

**Definition 1 ([2]).** Let \( E \) be a regular expression over an alphabet \( \Sigma \). The partial derivative of \( E \) w.r.t. a word \( w \) in \( \Sigma^* \) is the set \( \partial_w(E) \) inductively defined as follows:

\[
\partial_w(E) = \begin{cases} 
\{E\} & \text{if } w = \varepsilon, \\
\{\varepsilon\} & \text{if } E = w = a \in \Sigma, \\
\emptyset & \text{if } w = a \in \Sigma, E \in \Sigma \setminus \{a\} \cup \{\emptyset, \varepsilon\}, \\
\partial_w(F) \cup \partial_w(G) & \text{if } E = F + G, \\
\partial_w(F) \cdot G \cup (\partial_w(G) \mid \varepsilon \in L(F)) & \text{if } E = F \cdot G, \\
\partial_w(F) \cdot F^* & \text{if } E = F^*,
\end{cases}
\]

where \( F \) and \( G \) are any two regular expressions over \( \Sigma \) and where for any set \( \mathcal{E} \) of regular expressions, for any regular expression \( E' \), \( \mathcal{E} \cdot E' = \bigcup_{E \in \mathcal{E}} \{E \cdot E'\} \).

Furthermore, the membership test of \( \varepsilon \) is syntactically computable. Considering the predicate \( \text{Null}(E) = (\varepsilon \in L(E)) \), it can be checked that:

\[
\text{Null}(E) = \begin{cases} 
1 & \text{if } E = \varepsilon, \\
0 & \text{if } E \in \Sigma \cup \{\emptyset\}, \\
\text{Null}(F) \lor \text{Null}(G) & \text{if } E = F + G, \\
\text{Null}(F) \land \text{Null}G & \text{if } E = F \cdot G, \\
1 & \text{if } E = F^*,
\end{cases}
\]

where \( F \) and \( G \) are any two regular expressions over \( \Sigma \).

Consequently, from these definitions, the membership test of \( w \) in \( L(E) \) can be performed as follows:
**Proposition 1** ([2]). Let $E$ be a regular expression over an alphabet $\Sigma$ and $w$ be a word in $\Sigma^*$. The two following conditions are equivalent:

1. $w \in L(E)$
2. $\exists E' \in \partial_w(E), \varepsilon \in L(E')$.

Derivation and partial derivation have already been used in order to perform the membership test over extensions of regular expressions [5][6][7], expressions denoting non-necessarily regular languages [8], guarded strings [9] or even context-free grammars [10]. In the following of this paper, we extend regular expressions by introducing new operators based on boolean formulae in order to increase the expressive power of expressions. Let us first recall some well-known definitions of logic.

### 2.2 Zeroth-Order Logic

The notion of constrained expression that is introduced in this paper is expressed through the formalism of zeroth-order logic, that is first order logic without quantifiers (see primitive recursive arithmetic in [11] for an example of the difference between the expressiveness of propositional logic and zeroth-order logic).

More precisely, we consider two $\mathbb{N}$-indexed families $\mathcal{F} = (\mathcal{F}_k)_{k \in \mathbb{N}}$ and $\mathcal{P} = (\mathcal{P}_k)_{k \in \mathbb{N}}$ of disjoint sets, where for any integer $k$ in $\mathbb{N}$, $\mathcal{F}_k$ is a set of $k$-ary function symbols and $\mathcal{P}_k$ is a set of $k$-ary predicate symbols. The family $\mathcal{F}$ is combined with a set of variables in order to obtain a set of terms. This set of terms is combined with the family $\mathcal{P}$ in order to obtain boolean formulae.

Given a set $X$ of variables, a term $t$ over $(\mathcal{F}, X)$ is inductively defined by:

$$t = x \text{ with } x \in X,$$

$$t = f(t_1, \ldots, t_k),$$

where $k$ is any integer, $f$ is any element in $\mathcal{F}_k$, $t_1, \ldots, t_k$ are any $k$ terms over $(\mathcal{F}, X)$. We denote by $\mathcal{F}_k(X)$ the set of the terms over $(\mathcal{F}, X)$.

A subterm of a term $t$ is a term in the set $\text{Subterm}(t)$ inductively computed as follows:

$$\text{Subterm}(x) = \{x\},$$

$$\text{Subterm}(f(t_1, \ldots, t_k)) = \{f(t_1, \ldots, t_k)\} \cup \bigcup_{j \in \{1, \ldots, k\}} \text{Subterm}(t_j),$$

where $x$ is any element in $X$, $k$ is any integer, $f$ is any function symbol in $\mathcal{F}_k$ and $t_1, \ldots, t_k$ are any $k$ terms in $\mathcal{F}(X)$.

A boolean formula $\phi$ over $(\mathcal{P}, \mathcal{F}(X))$ is inductively defined by:

$$\phi = P(t_1, \ldots, t_k),$$

$$\phi = \phi_1 \ldots \phi_k,$$

where $k$ and $k'$ are any two integers, $P$ is any element in $\mathcal{P}_k$, $t_1, \ldots, t_k$ are any $k$ terms in $\mathcal{F}(X)$, $o$ is any $k'$-ary boolean operator associated with a mapping from $\{0, 1\}^k$ to $\{0, 1\}$ and $\phi_1, \ldots, \phi_k$ are any $k'$ boolean formulae over $(\mathcal{P}, \mathcal{F}(X))$. We denote by $\mathcal{P}(\mathcal{F}(X))$ the set of boolean formulae over $(\mathcal{P}, \mathcal{F}(X))$.

Given a formula $\phi$ in $\mathcal{P}(\mathcal{F}(X))$, a term $t$ in $\mathcal{F}(X)$ and a symbol $x$ in $X$, we denote by $\phi_{x \leftarrow t}$ the substitution of $x$ by $t$ in $\phi$, which is the boolean formula inductively defined by:

$$(o(\phi_1, \ldots, \phi_{k'}))_{x \leftarrow t} = o((\phi_1)_{x \leftarrow t}, \ldots, (\phi_{k'})_{x \leftarrow t}),$$

$$(P(t_1, \ldots, t_{k}))_{x \leftarrow t} = P((t_1)_{x \leftarrow t}, \ldots, (t_{k})_{x \leftarrow t}),$$

where $k$ and $k'$ are any two integers, $P$ is any element in $\mathcal{P}_k$, $t_1, \ldots, t_{k}$ are any $k$ terms in $\mathcal{F}(X)$, $o$ is any $k'$-ary boolean operator associated with a mapping from $\{0, 1\}^{k'}$ to $\{0, 1\}$, $\phi_1, \ldots, \phi_{k'}$ are any $k'$ boolean formulae over $(\mathcal{P}, \mathcal{F}(X))$ and where for any term $t'$ in $\mathcal{F}(X)$, $t'_{x \leftarrow t}$ is the substitution of $x$ by $t$ in $t'$, which is the term inductively defined by:

$$(f(t_1, \ldots, t_k))_{x \leftarrow t} = f((t_1)_{x \leftarrow t}, \ldots, (t_k)_{x \leftarrow t}),$$

$$y_{x \leftarrow t} = \begin{cases} y & \text{if } x \neq y, \\ t & \text{otherwise,} \end{cases}$$

where $y$ is any element in $X$, $k$ is any integer, $f$ is any function symbol in $\mathcal{F}_k$ and $t_1, \ldots, t_k$ are any $k$ terms in $\mathcal{F}(X)$.

**Example 1.** Let us consider the two families $\mathcal{F} = (\mathcal{F}_k)_{k \in \mathbb{N}}$ and $\mathcal{P} = (\mathcal{P}_k)_{k \in \mathbb{N}}$ defined by

$$\mathcal{F}_k = \begin{cases} \{f\} & \text{if } k = 1, \\ \{g, h\} & \text{if } k = 2, \\ \emptyset & \text{otherwise,} \end{cases}$$

$$\mathcal{P}_k = \begin{cases} \{P\} & \text{if } k = 1, \\ \{Q, R\} & \text{if } k = 2, \\ \emptyset & \text{otherwise,} \end{cases}$$

and the set $X = \{x, y, z\}$. As an example, the set of boolean formulae over $(\mathcal{P}, \mathcal{F}(X))$ contains the formulae:
\[ \phi_1 = P(x) \lor Q(x, z), \]
\[ \phi_2 = Q(y, f(x)) \land R(h(z, z), g(f(y), z)), \]
\[ \phi_3 = \neg P(f(g(x, z))), \]
\[ \phi_4 = \mathcal{Z} \phi_1, \phi_2, \phi_3, \]
where \( \mathcal{Z} \) is the ternary boolean operator associated with the If-Then-Else-like conditional expression, generally written \( \phi_1 \ ? \phi_2 : \phi_3 \).

After having defined the syntactic part of the logic formulae we use, let us show how to evaluate these formulae, i.e. how to define the semantics of the logic formulae. The boolean evaluation of a formula is performed in two steps. First, an interpretation defines a domain and associates the function and predicate symbols with function; Then, each variable symbol is associated with a value from the domain by a realization.

**Definition 2 (Interpretation).** Let \( \mathcal{F} = (\mathcal{F}_k)_{k \in \mathbb{N}} \) and \( \mathcal{P} = (\mathcal{P}_k)_{k \in \mathbb{N}} \) be two families of disjoint sets. An interpretation \( I \) over \( \mathcal{P}, \mathcal{F} \) is a couple \((\mathcal{D}, \mathfrak{g})\) where:

- \( \mathcal{D} \) is a set, called the interpretation domain of \( I \),
- \( \mathfrak{g} \) is a function:
  - from \( \mathcal{P}_k \) to \( 2^{\mathcal{D}_k} \),
  - from \( \mathcal{F}_k \) to \( 2^{\mathcal{D}_k+1} \) such that for any function symbol \( f \) in \( \mathcal{F}_k \), for any two elements \((e_1, \ldots, e_k, e_{k+1})\) and \((e'_1, \ldots, e'_k, e'_{k+1})\) in \( \mathfrak{g}(f) \), \((e_1, \ldots, e_k) = (e'_1, \ldots, e'_k) \Rightarrow e_{k+1} = e'_{k+1} \), and such that for any \( k \) elements \((e_1, \ldots, e_k)\) in \( \mathcal{D}_k \), there exists \( e_{k+1} \) in \( \mathcal{D} \) such that \((e_1, \ldots, e_k, e_{k+1}) \in \mathfrak{g}(f) \) called the interpretation function.

**Definition 3 (Realization).** Let \( \mathcal{P} \) and \( \mathcal{F} \) be two families of disjoint sets, and \( I = (\mathcal{D}, \mathfrak{g}) \) an interpretation over \( \mathcal{P}, \mathcal{F} \). Let \( X \) be a set. An \( X \)-realization \( r \) over \( I \) is a function from \( X \) to \( \mathcal{D} \).

Once an interpretation \( I \) and a realization \( r \) given, a term can be evaluated as an element of the domain and a formula as a boolean via the function \( \text{eval}_{(I,r)} \), the \((I,r)\)-evaluation:

**Definition 4 (Term Evaluation).** Let \( \mathcal{P} \) and \( \mathcal{F} \) be two families of disjoint sets and \( I = (\mathcal{D}, \mathfrak{g}) \) an interpretation over \( \mathcal{P}, \mathcal{F} \). Let \( X \) be a set. Let \( r \) be an \( X \)-realization over \( I \). Let \( t \) be a term in \( \mathcal{F}(X) \). The \((I,r)\)-evaluation of \( t \) is the element \( \text{eval}_{I,r}(t) \) in \( \mathcal{D} \) defined by:

\[
\text{eval}_{I,r}(t) = \begin{cases} 
    r(x) & \text{if } t = x \land x \in X, \\
    x_{k+1} & \text{if } t = f(t_1, \ldots, t_k) \land (\text{eval}_{I,r}(t_1), \ldots, \text{eval}_{I,r}(t_k), x_{k+1}) \in \mathfrak{g}(f) 
\end{cases}
\]

where \( k \) is any integer, \( f \) is any function symbol in \( \mathcal{F}_k \), and \( t_1, \ldots, t_k \) are any \( k \) elements in \( \mathcal{F}(X) \).

**Definition 5 (Formula Evaluation).** Let \( \mathcal{P} \) and \( \mathcal{F} \) be two families of disjoint sets and \( I = (\mathcal{D}, \mathfrak{g}) \) an interpretation over \( \mathcal{P}, \mathcal{F} \). Let \( X \) be a set. Let \( r \) be an \( X \)-realization over \( I \). Let \( \phi \) be a boolean formula in \( \mathcal{P}(\mathcal{F}(X)) \). The \((I,r)\)-evaluation of \( \phi \) is the boolean \( \text{eval}_{I,r}(\phi) \) inductively defined by:

\[
\text{eval}_{I,r}(P(t_1, \ldots, t_k)) = \begin{cases} 
    1 & \text{if } (\text{eval}_{I,r}(t_1), \ldots, \text{eval}_{I,r}(t_k)) \in \mathfrak{g}(P), \\
    0 & \text{otherwise,} 
\end{cases}
\]

\[
\text{eval}_{I,r}(o(\phi_1, \ldots, \phi_k)) = o'(\text{eval}_{I,r}(\phi_1), \ldots, \text{eval}_{I,r}(\phi_k)),
\]

where \( k \) is any integer, \( P \) is any predicate symbol in \( \mathcal{P}_k \), \( t_1, \ldots, t_k \) are any \( k \) elements in \( \mathcal{F}(X) \), \( o \) is any \( k \)-ary boolean operator associated with a mapping \( o' \) from \( \{0,1\}^k \) to \( \{0,1\} \) and \( \phi_1, \ldots, \phi_k \) are any \( k \) boolean formulae over \( \mathcal{P}(X) \).

**Example 2.** Let us consider Example 1. Let \( \Sigma = \{a, b\} \) be an alphabet. Let \( I \) be the interpretation \((\mathcal{D}, \mathfrak{g})\) over \((\mathcal{P}, \mathcal{F})\) and \( r \) be the \( X \)-realization over \( I \) defined by:

\[
\mathcal{D} = \Sigma^*, \quad \mathfrak{g}(P) = \{w \in \mathcal{D} \mid w \neq \varepsilon\}, \quad \mathfrak{g}(Q) = \{(w_1, w_2) \in \mathcal{D}^2 \mid w_1 = w_2\}, \quad \mathfrak{g}(R) = \{(w_1, w_2) \in \mathcal{D}^2 \mid w_1 = \text{rev}(w_2)\}, \quad \mathfrak{g}(f) = \{(w_1, \text{rev}(w_1)) \in \mathcal{D}^2\},
\]

where for any word \( w \) in \( \Sigma^* \), \( \text{rev}(w) \) is the word defined by:

\[
\text{rev}(w) = \begin{cases} 
    w & \text{if } w = \varepsilon, \\
    \text{rev}(w')x & \text{if } w = xw' \land w' \in \Sigma^* \land x \in \Sigma.
\end{cases}
\]

\[
\mathfrak{g}(g) = \{(w_1, w_2, w_1 \cdot w_2) \in \mathcal{D}^3\}, \quad \mathfrak{g}(h) = \{(w_1, w_2, w_1w_2) \in \mathcal{D}^3\}, \quad r(x) = aa, \quad r(y) = bb, \quad r(z) = \varepsilon.
\]
Then:
\[
\text{eval}_{I,r}(\phi_1) = \text{eval}_{I,r}(P(x) \lor Q(x, z)) = (aa \neq \varepsilon) \lor (aa == \varepsilon) = 1
\]
\[
\text{eval}_{I,r}(\phi_2) = \text{eval}_{I,r}(Q(y, f(x)) \land R(h(z, z), g(f(y), z)) = (bb == aa) \land (\varepsilon == \text{rev}(bb)) = 0
\]
\[
\text{eval}_{I,r}(\phi_3) = \text{eval}_{I,r}(-P(f(g(x, x)))) = \text{Not}(aaaa \neq \varepsilon) = 1
\]
\[
\text{eval}_{I,r}(\phi_4) = (\text{eval}_{I,r}(\phi_1) \text{Implies} \text{eval}_{I,r}(\phi_2)) \text{And} (\text{Not} (\text{eval}_{I,r}(\phi_1)) \text{Implies} \text{eval}_{I,r}(\phi_3)) = (1 \text{Implies} 0) \text{And} (0 \text{Implies} 1) = 0 \quad \Box
\]

3 Constrained Expressions, Languages and Derivatives

In this section, zeroth-order logic is combined with classical regular expressions in order to define constrained expressions the denoted language of which is not necessarily regular. We extend the membership problem for constrained expressions using partial derivatives and then show that it is equivalent to a satisfiability problem.

3.1 Constrained Expressions and their Languages

Whereas regular expressions are defined over a unique symbol alphabet, constrained expressions deal with zeroth-order logic and therefore include function, predicate and variable symbols. Hence the notion of alphabet is extended to the notion of expression environment in order to take into account all these symbols.

**Definition 6 (Expression Environment).** An expression environment is a 4-tuple \((\Sigma, \Gamma, \mathcal{P}, \mathcal{F})\) where:

- \(\Sigma\) is an alphabet, called the symbol alphabet,
- \(\Gamma\) is an alphabet, called the variable alphabet,
- \(\mathcal{P}\) is a \(\mathbb{N}\)-indexed family of disjoint sets, called the family of predicate symbols,
- \(\mathcal{F}\) is a \(\mathbb{N}\)-indexed family of disjoint sets, called the family of function symbols such that \(\Sigma \cup \{\varepsilon\} \subset \mathcal{F}_0\) and \(\{\cdot\} \in \mathcal{F}_2\).

Once this environment stated, we can syntactically defined the set of constrained expressions, by adding two new operators to regular operators: the first operator, \(|\), is based on the combination of an expression \(E\) and of a boolean formula \(\phi\), producing the expression \(E \mid \phi\); the second operator, \(-\), links a word \(\alpha\) composed of variable and letter symbols to an expression \(E\), producing the expression \(\alpha \vdash E\).

**Definition 7 (Constrained Expression).** Let \(\mathcal{E} = (\Sigma, \Gamma, \mathcal{P}, \mathcal{F})\) be an expression environment. A constrained expression \(E\) over \(\mathcal{E}\) is inductively defined by:

\[
E = \alpha, \quad E = \emptyset,
\]

\[
E = o(E_1, \ldots, E_k), \quad E = (E_1) \cdot (E_2), \quad E = (E_1)^*,
\]

\[
E = (E_1) \mid (\phi), \quad E = (\alpha) \vdash (E_1),
\]

where \(k\) is any integer, \(o\) is any \(k\)-ary boolean operator associated with a mapping from \(\{0,1\}^k\) to \(\{0,1\}\), \(E_1, \ldots, E_k\) are any \(k\) constrained expressions over \(\mathcal{E}\), \(\alpha\) is any word in \((\Sigma \cup \Gamma)^*\) and \(\phi\) is a boolean formula in \(\mathcal{P}(\mathcal{F}(\Gamma))\).

Parenthesis can be omitted when there is no ambiguity.

Any boolean formula that appears in a constrained expression over an environment \(\mathcal{E} = (\Sigma, \Gamma, \mathcal{P}, \mathcal{F})\) is, by definition, a formula in \(\mathcal{P}(\mathcal{F}(\Gamma))\). Furthermore, since we want that a constrained expression denotes a subset of \(\Sigma^*\), variable symbols in \(\Gamma\) have to be evaluated as words in \(\Sigma^*\). Moreover, classical symbols, like \(\varepsilon\) or \(a\) in \(\Sigma\), have to be considered as 0-ary functions in the interpretation. All these considerations imply some specializations of the notions of interpretation and realizations, defined as follows.
Definition 8 (Expression Interpretation). Let $\mathcal{E} = (\Sigma, \Gamma, \mathcal{P}, \mathcal{F})$ be an expression environment. An expression interpretation $I$ over $\mathcal{E}$ is an interpretation $(\mathfrak{D}, \mathfrak{F})$ over $(\mathcal{P}, \mathcal{F})$ satisfying the three following conditions:

1. $\mathfrak{D} = \Sigma^*$,
2. for any symbol $\alpha$ in $\Sigma \cup \{\varepsilon\} \subset \mathcal{F}_0$, $\mathfrak{F}(\alpha) = \{\alpha\}$,
3. $\mathfrak{F}(\cdot) = \{(u, v, w) \in (\Sigma^*)^3 \mid w = uv\}$.

The two new operators appearing in a constrained expression are used to extend the expressive power of regular expressions. The expression $E \phi$ denotes the set of words that $E$ may denote whenever the formula $\phi$ is satisfied. The expression $\alpha \Phi E$ denotes the set of words that $E$ may denote and that can be "matched" by $\alpha$. In order to perform this matching, we extend any $\Gamma$-realization over an expression interpretation as a morphism from $(\Sigma \cup \Gamma)^*$ to $\Sigma^*$ as follows.

Let $\mathcal{E} = (\Sigma, \Gamma, \mathcal{P}, \mathcal{F})$ be an expression environment. Let $I$ be an expression interpretation over $\mathcal{E}$ and $r$ be a $\Gamma$-realization over $I$. The domain of the realization $r$ can be extended to $(\Sigma \cup \Gamma)^*$ as follows:

for any word $\alpha$ in $(\Sigma \cup \Gamma)^*$, $r(\alpha)$ is the word in $\Sigma^*$ inductively computed by:

$$ r(\alpha) = \begin{cases} \varepsilon & \text{if } \alpha = \varepsilon, \\ ar(\alpha') & \text{if } \alpha = aa' \land a \in \Sigma, \\ r(x)r(\alpha') & \text{if } \alpha = xa' \land x \in \Gamma. \end{cases} $$

Using this extension, we can now formally define the different languages that a constrained expression may denote. We consider the three following cases where first the interpretation and the realization are both given, then only the interpretation is fixed and finally nothing is fixed.

Definition 9 ((I,r)-Language). Let $\mathcal{E} = (\Sigma, \Gamma, \mathcal{P}, \mathcal{F})$ be an expression environment. Let $I$ be an expression interpretation over $\mathcal{E}$ and $r$ be a $\Gamma$-realization over $I$. Let $E$ be a constrained expression over $\mathcal{E}$. The $(I,r)$-language denoted by $E$ is the language $L_{I,r}(E)$ inductively defined by:

$\begin{align*}
L_{I,r}(\alpha) &= \{r(\alpha)\}, \quad L_{I,r}(\emptyset) = \emptyset, \\
L_{I,r}(\alpha \mathcal{F} E_1) &= \{r(\alpha) \mid r(\alpha) \in L_{I,r}(E_1)\}, \\
L_{I,r}(\mathcal{I}(E_1, \dotsc, E_k)) &= r'(L_{I,r}(E_1), \dotsc, L_{I,r}(E_k)), \\
L_{I,r}(E_1 \cdot E_2) &= L_{I,r}(E_1) \cdot L_{I,r}(E_2), \\
L_{I,r}(E_1^*) &= (L_{I,r}(E_1))^*, \\
L_{I,r}(E_1 \mid \phi) &= \begin{cases} L_{I,r}(E_1) & \text{if } \text{eval}_{I,r}(\phi), \\ \emptyset & \text{otherwise}, \end{cases}
\end{align*}$

where $k$ is any integer, $\mathcal{I}$ is any $k$-ary boolean operator, $r'$ is the language operator associated with $\mathcal{I}$, $E_1, \dotsc, E_k$ are any $k$ constrained expression over $\mathcal{E}$, $\alpha$ is any word in $(\Sigma \cup \Gamma)^*$ and $\phi$ is any boolean formula in $\mathcal{P}(F(\Gamma))$.

We denote by $\text{Real}_I(I)$ the set of the $\Gamma$-realizations over an interpretation $I$.

Definition 10 (I-Language). Let $\mathcal{E} = (\Sigma, \Gamma, \mathcal{P}, \mathcal{F})$ be an expression environment. Let $E$ be a constrained expression over $\mathcal{E}$. Let $I$ be an expression interpretation over $\mathcal{E}$. The $I$-language denoted by $E$ is the language $L_I(E)$ defined by:

$$ L_I(E) = \bigcup_{I \in \text{Real}_I(I)} L_{I,r}(E). $$

Given an expression environment $\mathcal{E} = (\Sigma, \Gamma, \mathcal{P}, \mathcal{F})$, we denote by $\text{Int}(\mathcal{E})$ the set of the expression interpretations over $\mathcal{E}$.

Definition 11 (Language). Let $\mathcal{E} = (\Sigma, \Gamma, \mathcal{P}, \mathcal{F})$ be an expression environment. Let $E$ be a constrained expression over $\mathcal{E}$. The language denoted by $E$ is the language $L(E)$ defined by:

$$ L(E) = \bigcup_{I \in \text{Int}(\mathcal{E})} L_I(E). $$

Example 3. Let $\mathcal{E} = (\Sigma, \Gamma, \mathcal{P}, \mathcal{F})$ be the expression environment defined by:

- $\Sigma = \{a, b, c\}$,
- $\Gamma = \{x, y, z\}$,
- $\mathcal{P} = \mathcal{P}_2 = \{\lll, \sim\}$,
- $\mathcal{F}_0 = \Sigma \cup \{\varepsilon\}, \mathcal{F}_1 = \{f\}, \mathcal{F}_2 = \{\}. $
Let us consider the constrained expressions \( E_1 = xb^*y \lhd \lhd (f(x), f(y)) \) and \( E_2 = (ab)^*x \rhd (yx \mid \prec(y, x)) \). Let \( I = (\Sigma^*, \mathfrak{F}_1) \) be the expression interpretation defined by:

- \( \mathfrak{F}(\prec) = \{(u, v) \mid |u| \leq |v|\} \),
- \( \mathfrak{F}(\rhd) = \{(u, u)\} \),
- \( \mathfrak{F}(\alpha) = \{\alpha\} \), for any \( \alpha \in \mathcal{F}_0 \),
- \( \mathfrak{F}(f) = \{(u, a^{\mid u\mid})\} \),
- \( \mathfrak{F}(\cdot) = \{(u, v, u \cdot v)\} \).

In other words, the evaluation of an expression w.r.t. \( I_1 \) considers that:

- \( \prec(u, v) \) is true if and only if \( u \) is shorter than \( v \),
- \( \rhd(u, v) \) is true if and only if \( u = v \),
- \( f \) is a function that changes all the symbols in \( u \) different from \( a \) into a symbol \( \varepsilon \).

By abuse of notation, let us syntactically apply the interpretation as follows:

\[
[E_1]_I = xb^*y \mid a^{|x|} = a^{|y|}, [E_2]_I = (ab)^*x \rhd (yx \mid |y| \leq |x|)
\]

Let us consider the \( I \)-languages denoted by these expressions:

- \( L_1(E_1) \) is the set of words \( u^m v \) with \( n \geq 0 \) and \( |u|^m = |v|^m \),
- \( L_1(E_2) \) is the set of words \( (ab)^m u \) with \( n \geq 0 \) and \( 2n \leq |u| \).

As an example, the word \( ababbbbaa \) belongs to:

- \( L_1(E_1) \) since it can be obtained by considering the realization \( r_1 \) associating \( aba \) with \( x \) and \( aa \) with \( y \):
  \[
  [E_1]_{I, r_1} = abab^*aa \mid a^3 = a^3 \text{ that is equivalent to } abab^*aa,
  \]
- \( L_1(E_2) \) since it can be obtained by considering the realization \( r_2 \) associating \( bbbaa \) with \( x \) and \( abab \) with \( y \):
  \[
  [E_2]_{I, r_2} = (ab)^*bbbaa \rhd (ababbaa \mid 4 \leq 4) \text{ that is equivalent to } (ab)^*bbbaa \rhd (ababbaa) \text{ and finally to } ababbaa.
  \]

In other words, the word \( ababbbbaa \) belongs to \( L_{I, r_1}(E_1) \subset L_1(E_1) \) and to \( L_{I, r_2}(E_2) \subset L_1(E_2) \).

Notice that the word \( ababbbbaa \) is not in the \( (I, r_1) \)-language denoted by \( [E_1]_{I, r_1} = bbaab^*aba \mid a^4 = a^4 \) that is equivalent to \( bbaab^*aba \) nor in the \( (I, r_1) \)-language denoted by \( [E_2]_{I, r_1} = (ab)^*aba \rhd (aadbaa \mid 2 \leq 3) \) that is equivalent to \( (ab)^*aba \cap aababa \) and finally to \( \emptyset \).

### 3.2 The \( (I, r) \)-Language of a Constrained Expression is Regular

As far as an interpretation and a realization are given, the language denoted by a constrained expression is a regular one. The proof is based on the computation of an equivalent regular expression.

**Definition 12 (Regularization).** Let \( \mathcal{E} = (\Sigma, \Gamma, P, F) \) be an expression environment and \( E \) be a constrained expression over \( \mathcal{E} \). Let \( I \) be an expression interpretation over \( \mathcal{E} \) and \( r \) be a \( \Gamma \)-realization over \( I \). The \( (I, r) \)-regularization of \( E \) is the regular expression \( \text{reg}_{I,r}(E) \) inductively defined as follows:

\[
\text{reg}_{I,r}(\alpha) = r(\alpha), \quad \text{reg}_{I,r}(\emptyset) = \emptyset,
\]

\[
\text{reg}_{I,r}(o(E_1, \ldots, E_k)) = o(\text{reg}_{I,r}(E_1), \ldots, \text{reg}_{I,r}(E_k)),
\]

\[
\text{reg}_{I,r}(E_1 \cdot E_2) = \text{reg}_{I,r}(E_1) \cdot \text{reg}_{I,r}(E_2),
\]

\[
\text{reg}_{I,r}(E^*_1) = \text{reg}_{I,r}(E_1)^*,
\]

\[
\text{reg}_{I,r}(E_1 | \phi) = \begin{cases} \text{reg}_{I,r}(E_1) & \text{if eval}_{I,r}(\phi), \\ \emptyset & \text{otherwise}, \end{cases}
\]

\[
\text{reg}_{I,r}(\alpha \cdot I_1) = \alpha \cap \text{reg}_{I,r}(E_1),
\]

where \( k \) is any integer, \( o \) is any \( k \)-ary boolean operator associated with a mapping from \( \{0, 1\}^k \) to \( \{0, 1\} \), \( E_1, \ldots, E_k \) are any \( k \) constrained expressions over \( \mathcal{E} \), \( \alpha \) is any word in \( (\Sigma \cup \Gamma)^* \) and \( \phi \) is a boolean formula in \( \mathcal{P}(\mathcal{F}(\Gamma)) \).

**Proposition 2.** Let \( \mathcal{E} = (\Sigma, \Gamma, P, F) \) be an expression environment and \( E \) be a constrained expression over \( \mathcal{E} \). Let \( I \) be an expression interpretation over \( \mathcal{E} \) and \( r \) be a \( \Gamma \)-realization over \( I \). Then:

\[
L_{I,r}(E) = L(\text{reg}_{I,r}(E)).
\]
Proof. By induction over the structure of $E$. According to Definition 12 and to Definition 9:

$$L(\text{reg}_{I,\tau}(\alpha)) = L(\tau(\alpha)) = L_{I,\tau}(\alpha)$$
$$L(\text{reg}_{I,\tau}(\emptyset)) = \emptyset = L_{I,\tau}(\emptyset)$$
$$L(\text{reg}_{I,\tau}(o(E_1, \ldots, E_k))) = L(o(\text{reg}_{I,\tau}(E_1), \ldots, \text{reg}_{I,\tau}(E_k)))$$
$$= o'(L(\text{reg}_{I,\tau}(E_1)), \ldots, L(\text{reg}_{I,\tau}(E_k)))$$
$$= o'(L_{I,\tau}(E_1), \ldots, L_{I,\tau}(E_k)) \quad \text{(Induction hypothesis)}$$

$$L(\text{reg}_{I,\tau}(E_1 \cdot E_2)) = L(\text{reg}_{I,\tau}(E_1) \cdot \text{reg}_{I,\tau}(E_2))$$
$$= L(\text{reg}_{I,\tau}(E_1)) \cdot L(\text{reg}_{I,\tau}(E_2))$$
$$= L_{I,\tau}(E_1) \cdot L_{I,\tau}(E_2) \quad \text{(Induction hypothesis)}$$

$$L(\text{reg}_{I,\tau}(E_1^*)) = L(\text{reg}_{I,\tau}(E_1)^*)$$
$$= L(\text{reg}_{I,\tau}(E_1))^* \quad \text{(Induction hypothesis)}$$

$$L(\text{reg}_{I,\tau}(E_1 | \phi)) = \begin{cases} L(\text{reg}_{I,\tau}(E_1)) & \text{if } \text{eval}_{I,\tau}(\phi), \\ L(\emptyset) & \text{otherwise}, \end{cases}$$

$$= L_{I,\tau}(E_1 | \phi) \quad \text{(Induction hypothesis)}$$

$$L(\text{reg}_{I,\tau}(\alpha \leftarrow E_1)) = L(\tau(\alpha) \cap \text{reg}_{I,\tau}(E_1))$$
$$= L(\tau(\alpha)) \cap L(\text{reg}_{I,\tau}(E_1))$$
$$= \{v(\alpha) \cap L_{I,\tau}(E_1)\}$$
$$= \{v(\alpha) \mid v(\alpha) \in L_{I,\tau}(E_1)\}$$

where $k$ is any integer, $o$ is any $k$-ary boolean operator associated with a mapping from $\{0,1\}^k$ to $\{0,1\}$, $o'$ is the language operator associated with $o$, $E_1, \ldots, E_k$ are any $k$ constrained expressions over $\mathcal{E}$, $\alpha$ is any word in $(\Sigma \cup \Gamma)^*$ and $\phi$ is a boolean formula in $\mathcal{P}(\mathcal{F}(\Gamma))$.

Once this regular expression is computed, any classical membership test can be performed; hence:

**Corollary 1.** Let $\mathcal{E} = (\Sigma, \Gamma, \mathcal{P}, \mathcal{F})$ be an expression environment and $E$ be a constrained expression over $\mathcal{E}$. Let $I$ be an expression interpretation over $\mathcal{E}$ and $\tau$ be a $\Gamma$-realization over $I$. Let $w$ be a word in $\Sigma^*$. Then:

To determine whether or not $w$ belongs to $L_{I,\tau}(E)$ is polynomially decidable.

### 3.3 Derivatives for Constrained Expressions

Unlike the previous case, whenever the realization or the interpretation is not given, the language denoted by a constrained expression is an infinite union of regular languages that is not necessarily regular. In order to perform the membership test, the notion of partial derivatives is extended to the case of constrained expressions. While deriving expressions, choices have to be made in order to fix a realization. As an example, deriving the expression $x \cdot x$, where $x$ is a variable symbol, with respect to the symbol $a$ implies that the variable symbol $x$ is associated with $a$; otherwise, the derivative would be empty. Consequently, such a realization transforms $x$ in $a$ and then associates the expression $x \cdot x$ with the expression $a \cdot a$. Deriving this expression w.r.t. $a$ returns the expression $\varepsilon \cdot a$ that is equivalent to $a$.

As a direct consequence, the partial derivation has to memorize the assumptions made through the computation. And this is the reason why a partial derivative is a set of couples composed of an expression and a set of assumptions, where an assumption is a couple composed of a variable symbol $x$ and a word $\alpha$: the realization associates the variable $x$ with the word $\alpha$. These assumptions are needed to transform subexpressions of the initial expression. As an example, let us consider the expression $E \cdot F$. If assumptions are needed to perform the membership test while deriving $E$, these assumptions have to be applied over $F$ too via a substitution. Let us then extend the notion of substitution to words, to boolean formulae and to constrained expressions.

8
Let \( \Sigma \) be an alphabet and let \( \alpha \) and \( w \) be two words in \( \Sigma^* \). Let \( x \) be a symbol in \( \Sigma \). We denote by \( (\alpha)(x,w) \) the word obtained by substituting any occurrence of \( x \) in \( \alpha \) by \( w \), that is:

\[
(\alpha)(x,w) = \begin{cases} 
\varepsilon & \text{if } \alpha = \varepsilon, \\
\alpha'(x,w) & \text{if } \alpha = a\alpha' \land a \in \Sigma \setminus \{x\}, \\
w' & \text{if } \alpha = xa'.
\end{cases}
\]

For a boolean formula \( \phi \), we denote by \( \phi(x,w) \) the boolean formula defined by:

\[
\phi(x,w) = \phi_{x \leftarrow \text{Term}(w)}.
\]

Finally, for any constrained expression \( E \), we denote by \( E(x,w) \) the expression:

\[
E(x,w) = \emptyset \land (\alpha)(x,w) = (\alpha)(x,w) \land (E_1)(x,w),
\]

\[
(E_1 + E_2)(x,w) = (E_1)(x,w) + (E_2)(x,w), 
\]

\[
(E_1 \cdot E_2)(x,w) = (E_1)(x,w) \cdot (E_2)(x,w), 
\]

\[
(E_1^+)(x,w) = (E_1)(x,w)^+ (E_1)(x,w) \cdot (E_2)(x,w),
\]

\[
(E_1 | \phi)(x,w) = (E_1)(x,w) | \phi(x,w).
\]

We extend this notation for any subset \( X \) of \( \Sigma \times \Sigma^* \) as follows:

\[
(\alpha)X = \begin{cases} 
\alpha & \text{if } X = \emptyset, \\
(\alpha(x,w))X' & \text{if } X = X' \cup \{(x,w)\}
\end{cases}
\]

\[
(\phi)X = \begin{cases} 
\phi & \text{if } X = \emptyset, \\
(\phi(x,w))X' & \text{if } X = X' \cup \{(x,w)\}
\end{cases}
\]

\[
(E)X = \begin{cases} 
E & \text{if } X = \emptyset, \\
(E(x,w))X' & \text{if } X = X' \cup \{(x,w)\}
\end{cases}
\]

Let us continue the previous example, concerning the expression \( x \cdot x \). If we want to check that the word \( aa \) belongs to the language denoted by this expression, \( x \) can be replaced by \( a \), and then the derivation of \( xx \) w.r.t. \( aa \) produces an expression that denotes \( \varepsilon \). However, substituting \( x \) by a symbol is not sufficient in the general case. If we want to perform the membership test of the word \( abab \), the derivation w.r.t. \( a \) has to memorize that the realization associates \( x \) with a word that starts with the symbol \( a \). Then the variable \( x \) can be replaced by the word \( ax \): the expression \( xx \) is transformed into \( axax \) when the derivation is computed, producing the expression \( xxax \). Deriving w.r.t. \( b \), the assumption that \( x \) (the new \( x \), not the old one) is associated with a word that starts with \( b \) has to be made, replacing \( axax \) by \( bxabx \) and producing \( xabx \). Deriving it w.r.t. \( a \), a new assumption can be made: if the new \( x \) is replaced by \( \varepsilon \), then the expression \( xabx \) is replaced by the word \( ab \) and its derivation w.r.t. \( ab \) will produce \( \varepsilon \), proving that the word \( abab \) is denoted by \( xx \).

As a direct consequence, the partial derivation of a constrained expression will compute all the combinations of assumptions that can be made while deriving.

**Remark 1.** From now on, the set of boolean operators is restricted to the sum.

Let \( E = (\Sigma, \Gamma, \mathcal{P}, \mathcal{F}) \) be an expression environment. We denote by \( \text{Exp}(E) \) the set of the constrained expressions over \( E \).

**Definition 13 (Constrained Derivative).** Let \( E = (\Sigma, \Gamma, \mathcal{P}, \mathcal{F}) \) be an expression environment and let \( E \) be a constrained expression over \( E \). Let \( a \) be a symbol in \( \Sigma \). The constrained derivative of \( E \) w.r.t. \( a \) is the subset \( \frac{\partial}{\partial_\alpha} (E) \) of \( \text{Exp}(E) \times 2^{(\Sigma \cup \Gamma)^*} \) inductively computed as follows:

\[
\frac{\partial}{\partial_\alpha}(\emptyset) = \frac{\partial}{\partial_\alpha}(\varepsilon) = \emptyset,
\]

\[
\frac{\partial}{\partial_\alpha}(\alpha \cdot (\alpha')(x,a)) = \begin{cases} 
\{(\alpha', \emptyset)\} & \text{if } \alpha = a\alpha', \\
\{(x \cdot (\alpha')(x,a), \emptyset)\} \cup \cup_{x' \in \text{Exp}(E)} \{(\alpha'', x, \{(x, \varepsilon)\}) \} & \text{if } \alpha = x \cdot x' \wedge x \in \Gamma,
\end{cases}
\]

\[
\frac{\partial}{\partial_\alpha}(\alpha \cdot (E_1 \land E_2)) = \frac{\partial}{\partial_\alpha}(E_1) \cup \frac{\partial}{\partial_\alpha}(E_2),
\]

\[
\frac{\partial}{\partial_\alpha}(\alpha \cdot (E_1 \lor E_2)) = \frac{\partial}{\partial_\alpha}(E_1) \lor \frac{\partial}{\partial_\alpha}(E_2),
\]

\[
\frac{\partial}{\partial_\alpha}(\alpha \cdot (\text{Term}(w))) = \frac{\partial}{\partial_\alpha}(E_1) \circ \text{Term}(w),
\]

\[
\frac{\partial}{\partial_\alpha}(\alpha \cdot (\text{Term}(w))) = \frac{\partial}{\partial_\alpha}(E_1) \circ \text{Term}(w),
\]

where for any subset \( E \) of \( \text{Exp}(E) \times 2^{(\Sigma \cup \Gamma)^*} \), for any expression \( F \) and for any formula \( \phi \),

\[
E \circ F = \bigcup_{(E', X) \in E} \{(E \cdot F, X)\},
\]

\[
F \circ E = \bigcup_{(E, X) \in E} \{(E \cdot F, X)\},
\]

\[
E \parallel \phi = \bigcup_{(E, X) \in E} \{(E \mid \phi, X)\}.
\]
The following of this section is devoted to prove that the derivation can be used to perform the membership test. In fact, we show that to determine whether or not a word \( w \) is denoted by a constrained expression \( E \) is equivalent to determine whether or not \( \varepsilon \) is denoted by one of the derived expressions from \( E \).

We first model the fact that an assumption made through the derivation can be performed through a substitution without modify the membership test: the main idea is that if a realization associates a word \( ax \) with a symbol \( x \), the result is the same as if any occurrence of \( x \) is replaced by \( ax \) and if another realization is considered, where \( u \) is associated with \( x \). Hence, we can transfer a symbol from the realization to the expression.

**Definition 14 (Compatible Realization).** Let \( \mathcal{E} = (\Sigma, \Gamma, \mathcal{P}, \mathcal{F}) \) be an expression environment. Let \( I \) be an expression interpretation over \( \mathcal{E} \) and \( r \) be a \( \Gamma \)-realization over \( I \). Let \( X \) be a subset of \( \{ (x, ax), (x, \varepsilon) \mid a \in \Sigma, x \in \Gamma \} \). The realization \( r \) is said to be compatible with \( X \) if and only if the following conditions hold:

\[- \forall (x, ax) \in X, r(x) = au \text{ for some word } u \text{ in } \Sigma^*, \]
\[- \forall (x, \varepsilon) \in X, r(x) = \varepsilon. \]

**Definition 15 (Associated Realization).** Let \( \mathcal{E} = (\Sigma, \Gamma, \mathcal{P}, \mathcal{F}) \) be an expression environment. Let \( X \) be a subset of \( \{ (x, ax), (x, \varepsilon) \mid a \in \Sigma, x \in \Gamma \} \). Let \( I \) be an expression interpretation over \( \mathcal{E} \) and \( r \) be a \( \Gamma \)-realization over \( I \) compatible with \( X \). The realization \( X \)-associated with \( r \) is defined for any symbol \( x \) in \( \Gamma \) as follows:

\[ r'(x) = \begin{cases} w & \text{if } r(x) = aw \land (x, ax) \in X, \\ \varepsilon & \text{if } r(x) = \varepsilon \land (x, \varepsilon) \in X, \\ r(x) & \text{otherwise.} \end{cases} \]

**Lemma 1.** Let \( \mathcal{E} = (\Sigma, \Gamma, \mathcal{P}, \mathcal{F}) \) be an expression environment and let \( \phi \) be a boolean formula in \( \mathcal{P}(\mathcal{F}(\Gamma)) \). Let \( X \) be a subset of \( \{ (x, ax), (x, \varepsilon) \mid a \in \Sigma, x \in \Gamma \} \). Let \( I \) be an expression interpretation over \( \mathcal{E} \) and \( r \) be a \( \Gamma \)-realization over \( I \) compatible with \( X \). Let \( r' \) be the realization \( X \)-associated with \( r \). Then:

\[ \text{eval}_{I,r}(\phi) = \text{eval}_{I,r'}(\phi_X). \]

**Proof.** We proceed in two steps.

1. Let us first show that for any term \( t \) in \( \mathcal{F}(\Gamma) \), \( \text{eval}_{I,r}(t) = \text{eval}_{I,r'}(t_X) \). By induction over \( t \).
   
   (a) Suppose that \( t = x \in \Gamma \). Then
   
   \[ \text{eval}_{I,r}(x) = r(x) = \begin{cases} aw & \text{if } r(x) = aw \land (x, ax) \in X, \\ \varepsilon & \text{if } r(x) = \varepsilon \land (x, \varepsilon) \in X, \\ r(x) & \text{otherwise.} \end{cases} \]

   Furthermore,

   \[ x_X = \begin{cases} ax & \text{if } r(x) = aw \land (x, ax) \in X, \\ \varepsilon & \text{if } r(x) = \varepsilon \land (x, \varepsilon) \in X, \\ x & \text{otherwise.} \end{cases} \]

   Consequently,

   \[ \text{eval}_{I,r'}(x_X) = r'(x) = \begin{cases} ar'(x) & \text{if } r(x) = aw \land (x, ax) \in X, \\ \varepsilon & \text{if } r(x) = \varepsilon \land (x, \varepsilon) \in X, \\ r'(x) & \text{otherwise.} \end{cases} \]

   (b) Suppose that \( t = f(t_1, \ldots, t_n) \), that \( I = (\Sigma^*, \mathfrak{F}) \) and that \( \text{eval}_{I,r}(t_1), \ldots, \text{eval}_{I,r}(t_n), x_{k+1} \in \mathfrak{F}(f) \).

   By induction hypothesis, it holds that:

   \( \text{eval}_{I,r}(t_1), \ldots, \text{eval}_{I,r}(t_n), x_{k+1} \in \mathfrak{F}(f) \iff \text{eval}_{I,r'}(t_1 X), \ldots, \text{eval}_{I,r'}(t_n X), x_{k+1} \in \mathfrak{F}(f). \)

   Then:

   \[ \text{eval}_{I,r}(f(t_1, \ldots, t_n)) = x_{k+1} = \text{eval}_{I,r'}(f(t_1 X, \ldots, t_n X)) = \text{eval}_{I,r'}(t_X) \]

2. Let us show now by induction over \( \phi \) that \( \text{eval}_{I,r}(\phi) = \text{eval}_{I,r'}(\phi_X) \).

   (a) If \( \phi = P(t_1, \ldots, t_k) \), then \( \phi_X = P(t_1 X, \ldots, t_n X) \).

   Then

   \[ \text{eval}_{I,r}(\phi) = 1 \]
   \[ \Leftrightarrow \text{eval}_{I,r}(P(t_1, \ldots, t_k)) = 1 \]
   \[ \Leftrightarrow (\text{eval}_{I,r}(t_1), \ldots, \text{eval}_{I,r}(t_n)) \in \mathfrak{F}(P) \]
   \[ \Leftrightarrow (\text{eval}_{I,r}(t_1 X), \ldots, \text{eval}_{I,r}(t_n X)) \in \mathfrak{F}(P) \]
   \[ \Leftrightarrow \text{eval}_{I,r'}(P(t_1 X, \ldots, t_n X)) = 1 \]
   \[ \Leftrightarrow \text{eval}_{I,r'}(\phi_X) = 1 \] (Previous item 1)
Proof. By induction over the structure of $E$.

(b) Suppose that $\phi = o(\phi_1, \ldots, \phi_n)$. Then $\phi_X = o(\phi_{1_X}, \ldots, \phi_{n_X})$. Then
\[ \text{eval}_{I,r}(o(\phi_1, \ldots, \phi_n)) = 1 \]
\[ \leftrightarrow o(\text{eval}_{I,r}(\phi_1), \ldots, \text{eval}_{I,r}(\phi_n)) = 1 \]
\[ \leftrightarrow o(\text{eval}_{I,r}(\phi_{1_X}), \ldots, \text{eval}_{I,r}(\phi_{n_X})) = 1 \]
\[ \leftrightarrow \text{eval}_{I,r}(o(\phi_{1_X}, \ldots, \phi_{n_X})) = 1 \]
\[ \leftrightarrow \text{eval}_{I,r}(\phi) = 1 \]

(Induction hypothesis)

Lemma 2. Let $E = (\Sigma, \Gamma, P, F)$ be an expression environment and let $E$ be a constrained expression over $E$. Let $X$ be a subset of $\{((x, ax), (x, \varepsilon) \mid a \in \Sigma, x \in \Gamma\}$. Let $I$ be an expression interpretation over $E$ and $r$ be a $\Gamma$-realization over $I$ compatible with $X$. Let $r'$ be the realization $X$-associated with $r$. Then:
\[ L_{I,r}(E) = L_{I,r'}(E_X) \]

Proof. By induction over the structure of $E$.

1. Let us suppose that $E = \alpha$. By recurrence over the length of $\alpha$.
   (a) If $\alpha = \varepsilon$ or $\alpha = a \in \Sigma$, $E = E_X$ and then $L_{I,r}(E) = \{\alpha\} = L_{I,r'}(E_X)$.
   (b) If $\alpha = x \in \Gamma$, then
   \[ L_{I,r}(x) = \{r(x)\} = \begin{cases} \{aw\} & \text{if } r(x) = aw \land (x, ax) \in X, \\ \{\varepsilon\} & \text{if } r(x) = \varepsilon \land (x, \varepsilon) \in X, \\ \{r(x)\} & \text{otherwise.} \end{cases} \]
   Furthermore, $E_X = \begin{cases} ax \text{ if } r(x) = aw \land (x, ax) \in X, \\ \varepsilon \text{ if } r(x) = \varepsilon \land (x, \varepsilon) \in X, \\ x \text{ otherwise.} \end{cases}$
   and then
   \[ L_{I,r'}(E_X) = \begin{cases} \{aw\} & \text{if } r(x) = aw \land (x, ax) \in X, \\ \{\varepsilon\} & \text{if } r(x) = \varepsilon \land (x, \varepsilon) \in X, \\ \{r(x)\} & \text{otherwise.} \end{cases} \]
   (c) Suppose that $\alpha = \alpha' \beta$ with $\alpha' \in \Sigma \cup \Gamma$. Then
   \[ L_{I,r}(\alpha) = L_{I,r}(\alpha')L_{I,r}(\beta) \]
   \[ = L_{I,r}(\alpha')L_{I,r}(\beta_X) \]
   \[ = L_{I,r}(\alpha_X \beta_X) \]
   \[ = L_{I,r}(\alpha_X) \]
   (Recurrence hypothesis)

2. Let us suppose that $E = \alpha \vdash F$. Then
   \[ L_{I,r}(\alpha \vdash F) = \{r(\alpha) \mid r(\alpha) \in L_{I,r}(F)\} \]
   \[ = \{r(\alpha_X) \mid r(\alpha_X) \in L_{I,r}(F_X)\} \]
   \[ = L_{I,r}(\alpha_X \vdash F_X) \]
   \[ = L_{I,r}(\alpha_X) \]
   (Induction hypothesis)

3. Let us suppose that $E = F \mid \phi$. According to Lemma \[\] eval$(\phi) = \text{eval}_{I,r}(\phi_X)$. Hence if eval$_{I,r}(\phi) = 0$, $L_{I,r}(E) = L_{I,r}(E_X) = \emptyset$. Otherwise,
   \[ L_{I,r}(F \mid \phi) = L_{I,r}(F) \]
   \[ = L_{I,r}(F_X) \]
   \[ = L_{I,r}(F_X \mid \phi_X) \]
   \[ = L_{I,r}(E_X) \]
   (Induction hypothesis)

4. Let us suppose that $E = F + G$. Then:
   \[ L_{I,r}(F + G) = L_{I,r}(F) \cup L_{I,r}(G) \]
   \[ = L_{I,r}(F_X) \cup L_{I,r}(G_X) \]
   \[ = L_{I,r}(F_X + G_X) \]
   \[ = L_{I,r}(E_X) \]
   (Induction hypothesis)

5. Let us suppose that $E = F \cdot G$. Then:
   \[ L_{I,r}(F \cdot G) = L_{I,r}(F) \cdot L_{I,r}(G) \]
   \[ = L_{I,r}(F_X) \cdot L_{I,r}(G_X) \]
   \[ = L_{I,r}(F_X \cdot G_X) \]
   \[ = L_{I,r}(E_X) \]
   (Induction hypothesis)
6. Let us suppose that \( E = F^* \). Then:
\[
L_{I,I}(F^*) = L_{I,r}(F)^* = L_{I,r'}(F_X)^* = L_{I,r'}(F_X^r) = L_{I,r'}(E_X^r)
\]

(Induction hypothesis)

Let us now show that the partial derivation can be used to perform the membership test over constrained expressions whenever the realization is not fixed: if a word \( aw \) belongs to the language of a constrained expression \( E \) whenever a realization \( r \) is considered, the partial derivation w.r.t. \( a \) always produces at least a couple \((E',X)\) where \( E' \) denotes \( w' \) when another realization is considered (the realization \( X \)-associated with \( r' \)). The following lemma is useful for the proof of this fact.

**Lemma 3.** Let \( E = (\Sigma, \Gamma, \mathcal{P}, \mathcal{F}) \) be an expression environment and let \( E \) be a constrained expression over \( \mathcal{E} \). Let \( X \) be a subset of \( \{(x,ax), (x,\varepsilon) \mid a \in \Sigma, x \in \Gamma\} \). Let \( a \) be a symbol in \( \Sigma \). Then:
\[
(E',X') \in \frac{a}{\partial_\alpha}(E_X) \Rightarrow \{x \mid \exists (x,u) \in X \} \cap \{x' \mid \exists (x',u) \in X' \} = \emptyset.
\]

**Proof.** Let us notice that according to Definition\[\text{[3]}\] there exists a symbol \( x' \) in \( \Gamma \) and a word \( u \) in \( (\Sigma \cup \Gamma)^* \) satisfying \((x',u) \in X'\) only if there exists a subexpression \( \alpha \) of \( E_X \) such that \( \alpha = x'\alpha' \) for some \( \alpha' \in (\Sigma \cup \Gamma)^* \). Furthermore, since applying \((x,u) \in X\) for some \( u \in \{(x,\varepsilon)\} \) over \( E \) (producing \( E_{x \leftarrow u} \)) replaces any occurrence of \( x \) either by \( \varepsilon \) or by \( ax \), there exists no subexpression \( \alpha \) of \( E_X \) such that \( \alpha = x'\alpha' \) for some \( \alpha' \in (\Sigma \cup \Gamma)^* \). Consequently, the same symbol \( x' \) cannot be the first component of a couple in \( X \) and of a couple in \( X' \).

**Proposition 3.** Let \( E = (\Sigma, \Gamma, \mathcal{P}, \mathcal{F}) \) be an expression environment and let \( E \) be a constrained expression over \( \mathcal{E} \). Let \( I \) be an expression interpretation over \( \mathcal{E} \) and \( r \) be a \( \Gamma \)-realization over \( I \). Let \( w \) be a word in \( \Sigma^* \) and \( a \) be a symbol in \( \Sigma \). Then the following two conditions are equivalent:

\(- w \in \{a^{-1}(L_{I,I}(E)) \}
\)

there exists a couple \((E',X) \in \frac{\partial_\alpha}{a}(E)\) such that \( w \in L_{I,r'}(E') \), where \( r' \) is the realization \( X \)-associated with \( r \).

**Proof.** By induction over the structure of \( E \). By definition, \( w \in a^{-1}(L_{I,I}(E)) \Leftrightarrow aw \in L_{I,I}(E) \).

1. Whenever \( E \in \{\emptyset, \varepsilon\}, a^{-1}(L_{I,I}(E)) \) and \( \frac{\partial_\alpha}{a}(E) \) are both empty.
2. Let us suppose that \( E = \alpha \). Three cases can occur.
   (a) Let us suppose that \( \alpha = a\beta' \). If \( b \neq a \), \( a^{-1}(L_{I,I}(E)) = \emptyset \).
   Otherwise (if \( b = a \)), \( (\alpha',\emptyset) \notin \frac{\partial_\alpha}{a}(E) \). Furthermore, \( aw \in L_{I,I}(a\alpha') \Leftrightarrow \emptyset \in L_{I,I}(\alpha') \)
   Finally, since \( r' = r \) is the realization \( \emptyset \)-associated with \( r \),
   \( w \in a^{-1}(L_{I,I}(a\alpha')) \Leftrightarrow w \in L_{I,I}(\alpha') \).
   (b) Let us suppose that \( \alpha = xa' \) and that \( r(x) = aw' \) for some \( w' \in \Sigma^* \). Let us denote by \( r' \) the realization \( \{(x,ax)\} \)-associated with \( r \).
   Then \( r(\alpha) = r'(\alpha)_{x \leftarrow ax} = r'(ax(\alpha')_{x \leftarrow ax}) \)
   As a direct consequence,
   \( aw \in L_{I,r}(\alpha) \Leftrightarrow aw \in L_{I,r'}(ax(\alpha')_{x \leftarrow ax}) \Leftrightarrow w \in L_{I,r'}(x(\alpha')_{x \leftarrow ax}) \)
   Finally, it holds by definition that \( (x(\alpha')_{x \leftarrow ax}, \{(x,ax)\}) \in \frac{\partial_\alpha}{a}(E) \).
   (c) Let us suppose that \( \alpha = xa' \) and that \( r(x) = \varepsilon \). Then:
   \( r(\alpha) = r(\alpha') = r((\alpha')_{x \leftarrow \varepsilon}) \).
   As a direct consequence,
   \( w \in a^{-1}(L_{I,I}(\alpha')) \).
   According to induction hypothesis, there exists a couple \( (\alpha'',X) \) belonging to \( \frac{\partial_\alpha}{a}((\alpha')_{x \leftarrow \varepsilon}) \) such that \( w \in L_{I,r'}(\alpha'') \) with \( r'' \) the realization \( X \)-associated with \( r \). Let \( r' \) be the realization \( X \cup \{(x,\varepsilon)\} \)-associated with \( r \). Since there is no occurrence of \( x \) in \( \alpha'' \) (since \( \alpha'' \) is a derivated term of \( \alpha_{x \leftarrow \varepsilon} \)), it holds that \( r''(\alpha'') = r'(\alpha'') \). Furthermore, it holds by definition that \( (\alpha'',X \cup \{(x,\varepsilon)\}) \in \frac{\partial_\alpha}{a}(E) \).
   Finally, \( w \in L_{I,r'}(\alpha'') \)
3. Let us suppose that \( E = \alpha \vdash E_1 \). Consider that there exists a couple \((E', X) \in \frac{\partial}{\partial x}(E)\) such that 
\( w \in L_{1,r}(E') \), where \( r' \) is the realization \( X \)-associated with \( r \).
Equivalently, there exist \((\alpha', X_1) \in \frac{\partial}{\partial x}(\alpha)\) and \((E_2, X_2) \in \frac{\partial}{\partial x}(E_1, X_1)\) such that \((E' = (\alpha')_X \vdash E_2, X_1 \cup X_2) \in \frac{\partial}{\partial x}(E)\) and 
\( w \in L_{1,r}(E') \), where \( r' \) is the realization \( X_1 \cup X_2 \)-associated with \( r \).
By definition, \( L_{1,r}(E') = L_{1,r}((\alpha')_X) \cap L_{1,r}(E_2) \). Consequently \( w \in L_{1,r}((\alpha')_X) \) and \( w \in L_{1,r}(E_2) \).
Let us denote by \( r_1 \) (resp. \( r_2 \)) the realization \( X_1 \)-associated (resp. \( X_2 \)-associated) with \( r \).
Since \( r' \) is the realization \((X_1 \cup X_2)\)-associated with \( r \), and since according to Lemma \( \exists \{ x_1 \mid \exists(x_1, u) \in X_1 \} \cap \{ x_2 \mid \exists(x_2, u) \in X_2 \} = \emptyset \) it satisfies by definition for any symbol \( x \) in \( I' \) the following equality:

\[
r'(x) = \begin{cases} 
    w & \text{if } r(x) = aw \land (x, ax) \in X_1, \\
    w & \text{if } r(x) = aw \land (x, ax) \in X_2, \\
    \varepsilon & \text{if } r(x) = \varepsilon \land (x, \varepsilon) \in X_1, \\
    \varepsilon & \text{if } r(x) = \varepsilon \land (x, \varepsilon) \in X_2, \\
    r(x) & \text{otherwise.}
\end{cases}
\]

By definitions of \( r_1 \) and \( r_2 \), for any symbol \( x \) in \( I' \):

\[
r_1(x) = \begin{cases} 
    w & \text{if } r(x) = aw \land (x, ax) \in X_1, \\
    \varepsilon & \text{if } r(x) = \varepsilon \land (x, \varepsilon) \in X_1, \\
    r(x) & \text{otherwise,}
\end{cases}
\]

\[
r_2(x) = \begin{cases} 
    w & \text{if } r(x) = aw \land (x, ax) \in X_2, \\
    \varepsilon & \text{if } r(x) = \varepsilon \land (x, \varepsilon) \in X_2, \\
    r(x) & \text{otherwise.}
\end{cases}
\]

Hence, since \( X_1 \cap X_2 = \emptyset \),

\[
r'(x) = \begin{cases} 
    w & \text{if } r_1(x) = aw \land (x, ax) \in X_1, \\
    w & \text{if } r(x) = aw = r_1(x) \land (x, ax) \in X_2 \setminus X_1, \\
    \varepsilon & \text{if } r_1(x) = \varepsilon \land (x, \varepsilon) \in X_1, \\
    \varepsilon & \text{if } r(x) = r_1(x) \land (x, \varepsilon) \in X_2 \setminus X_1, \\
    r(x) & \text{otherwise.}
\end{cases}
\]

Consequently, \( r' \) is \( X_2 \)-associated with \( r_1 \). Symetrically, \( r' \) is \( X_1 \)-associated with \( r_2 \).
Since \( w \) belongs to \( L_{1,r}(E_2) \), there exists a couple \((E_2, X_2) \in \frac{\partial}{\partial x}(E_1, X_1)\) such that \( w \in L_{1,r}(E_2) \), where \( r' \) is the realization \( X_2 \)-associated with \( r_1 \). By induction hypothesis \( aw \in L_{1,r}(E_1, X_1) \). According to Lemma \( \exists \{ E_1 \mid \exists(x_1, u) \in X_1 \} \cap \{ x_2 \mid \exists(x_2, u) \in X_2 \} = \emptyset \). Finally, it holds that \( aw \in L_{1,r}(E) \).
4. Let us suppose that \( E = E_1 + E_2 \). Then \( w \in a^{-1}(L_{1,r}(E_1 + E_2)) \Leftrightarrow w \in a^{-1}(L_{1,r}(E_1)) \lor w \in a^{-1}(L_{1,r}(E_2)) \).
By induction, it is equivalent to \( \exists k \in \{1, 2\} | \exists(E', X) \in \frac{\partial}{\partial x}(E_k), w \in L_{1,r}(E') \) with \( r' \) is the realization \( X \)-associated with \( r \). Since \( \frac{\partial}{\partial x}(E_1) \cup \frac{\partial}{\partial x}(E_2) \subseteq \frac{\partial}{\partial x}(E_1 + E_2) \), it is equivalent to \( \exists(E', X) \in \frac{\partial}{\partial x}(E_1 + E_2) \), \( w \in L_{1,r}(E') \) with \( r' \) is the realization \( X \) associated with \( r \).
5. Let us suppose that \( E = E_1 \cdot E_2 \). Then

\[
w \in a^{-1}(L_{1,r}(E_1 \cdot E_2)) \Leftrightarrow \begin{cases} 
    w \in a^{-1}(L_{1,r}(E_1) \cdot L_{1,r}(E_2)) \\
    \lor (\varepsilon \in L_{1,r}(E_1) \land w \in a^{-1}(L_{1,r}(E_2)))
\end{cases}
\]

Moreover

\[
w \in a^{-1}(L_{1,r}(E_1) \cdot L_{1,r}(E_2)) \Leftrightarrow \begin{cases} 
    w = w_1 \cdot w_2 \\
    \land \exists(E', X) \in \frac{\partial}{\partial x}(E_1), \ w_1 \in L_{1,r}(E') \\
    \land w_2 \in L_{1,r}(E_2)
\end{cases}
\]

According to Lemma \( \exists \{ E_2 \mid \exists(x_1, u) \in X_1 \} \cap \{ x_2 \mid \exists(x_2, u) \in X_2 \} = \emptyset \). Hence

\[
w \in a^{-1}(L_{1,r}(E_1) \cdot L_{1,r}(E_2)) \Leftrightarrow \begin{cases} 
    w \in L_{1,r}(E_1 \cdot E_2) \\
    \land r' \text{ is the realization } X \text{-associated with } r
\end{cases}
\]

Finally consider that \( \varepsilon \in L_{1,r}(E_1) \land w \in a^{-1}(L_{1,r}(E_2)) \). By induction \( w \in a^{-1}(L_{1,r}(E_2)) \Leftrightarrow \exists(E', X) \in \frac{\partial}{\partial x}(E_2) \), \( w \in L_{1,r}(E') \) where \( r' \) is the realization \( X \)-associated with \( r \). Moreover \( \varepsilon \in L_{1,r}(E_1) \Leftrightarrow \varepsilon \in L_{1,r}(E_1) \) that is equivalent to \( \varepsilon \in L_{1,r}(\varepsilon \cdot (E_1)) \) according to Lemma \( \exists \). Hence

\[
w \in a^{-1}(L_{1,r}(E_2)) \Leftrightarrow \begin{cases} 
    w \in L_{1,r}(\varepsilon \cdot (E_1)) \cdot E_2) \\
    \land (\varepsilon \cdot (E_1) \cdot E_2) \in (\varepsilon \cdot (E_1) \cdot E_2) \subseteq \frac{\partial}{\partial x}(E_1 \cdot E_2)
\end{cases}
\]
6. Let us suppose that \( E = E_1 \). Then \( w \in a^{-1}(L_{i,r}(E_1)) \Leftrightarrow w = w_1 \cdot w_2 \wedge w_1 \in a^{-1}(L_{i,r}(E_1)) \wedge w_2 \in L_{i,r}(E_1) \). By induction, \( w_1 \in a^{-1}(L_{i,r}(E_1)) \Leftrightarrow \exists (E',X) \in \frac{\partial}{\partial_a}(E_1) w_1 \in L_{i,r}(E') \) where \( r' \) is the realization \( X \)-associated with \( r \). According to Lemma 2, \( w_2 \in L_{i,r}(E_1) \Leftrightarrow w_2 \in L_{i,r}((E_1)_X) \). Hence

\[
\begin{align*}
\text{w} \in a^{-1}(L_{i,r}(E_1)) & \Leftrightarrow \left\{ \begin{array}{l}
w = w_1 \cdot w_2 \\
\wedge w \in L_{i,r}(E' \cdot (E_1)_X) \\
r' \text{ is the realization } X \text{-associated with } r
\end{array} \right.
\end{align*}
\]

7. Let us suppose that \( E = E_1 \mid \phi \). Then \( w \in a^{-1}(L_{i,r}(E_1 \mid \phi)) \Leftrightarrow w \in a^{-1}(L_{i,r}(E_1)) \lor \text{eval}_{i,r}(\phi) \). By induction, \( w_1 \in a^{-1}(L_{i,r}(E_1)) \Leftrightarrow \exists (E',X) \in \frac{\partial}{\partial_a}(E_1), w_1 \in L_{i,r}(E') \) where \( r' \) is the realization \( X \)-associated with \( r \). According to Lemma 4, \( \text{eval}_{i,r}(\phi) = \text{eval}_{i,r}(\phi_X) \). Consequently,

\[
\begin{align*}
\text{w} \in a^{-1}(L_{i,r}(E_1 \mid \phi)) & \Leftrightarrow \left\{ \begin{array}{l}
w \in L_{i,r}(E' \mid \phi_X) \\
\wedge (E' \mid \phi_X, X) \in \frac{\partial}{\partial_a}(E_1) \lor \phi \subset \frac{\partial}{\partial_a}(E_1 \mid \phi)
\end{array} \right.
\end{align*}
\]

As a direct consequence of the previous proposition, the partial derivation of a constrained expression w.r.t. a symbol is valid.

**Theorem 1.** Let \( \mathcal{E} = (\Sigma, \Gamma, \mathcal{P}, \mathcal{F}) \) be an expression environment and let \( E \) be a constrained expression over \( \mathcal{E} \). Let \( I \) be an expression interpretation over \( \mathcal{E} \). Let \( a \) be a symbol in \( \Sigma \). Then the two following conditions hold:

1. \( a^{-1}(L_I(E)) = \bigcup_{(E',X)\in \frac{\partial}{\partial_a}(E)} L_I(E') \),
2. \( a^{-1}(L(E)) = \bigcup_{(E',X)\in \frac{\partial}{\partial_a}(E)} L(E') \).

**Proof.** Let \( w \) be a word in \( \Sigma^* \).

1. By definition of \( L_I(E) \), \( w \in a^{-1}(L_I(E)) \Leftrightarrow \) there exists a realization \( r \) such that \( w \in a^{-1}(L_{i,r}(E)) \). According to Proposition 3, \( w \in a^{-1}(L_{i,r}(E)) \Leftrightarrow \) there exists a couple \( (E',X) \in \frac{\partial}{\partial_a}(E) \) such that \( w \in L_{i,r}(E') \), where \( r' \) is the realization \( X \)-associated with \( r \). As a direct conclusion, \( w \in \bigcup_{(E',X)\in \frac{\partial}{\partial_a}(E)} L_{i,r}(E') \). Suppose that \( w \in \bigcup_{(E',X)\in \frac{\partial}{\partial_a}(E)} L_I(E') \). Hence there exists a couple \( (E',X) \in \frac{\partial}{\partial_a}(E) \) such that \( w \in L_I(E') \). By definition of \( L_I(E') \), there exists a realization \( r' \) such that \( w \in L_{i,r}(E') \). Let \( r \) be the realization defined for any symbol \( x \) in \( \Gamma \) as follows:

\[
r(x) = \begin{cases} 
w & \text{if } r'(x) = w \wedge (x,ax) \in X, \\
\varepsilon & \text{if } r'(x) = \varepsilon \wedge (x,\varepsilon) \in X, \\
r'(x) & \text{otherwise.}
\end{cases}
\]

As a direct consequence, \( r' \) is the realization \( X \)-associated with \( r \), and according to Proposition 3 since there exists a couple \( (E',X) \in \frac{\partial}{\partial_a}(E) \) such that \( w \in L_{i,r}(E') \), where \( r' \) is the realization \( X \)-associated with \( r \), it holds \( w \in a^{-1}(L_{i,r}(E)) \). By definition of \( L_I(E) \), \( w \in a^{-1}(L_I(E)) \).

2. By definition of \( L(E) \), \( w \in a^{-1}(L(E)) \Leftrightarrow \) there exists an interpretation \( I \) such that \( w \in a^{-1}(L_I(E)) \Leftrightarrow \) there exists an interpretation \( I \) such that \( w \in \bigcup_{(E',X)\in \frac{\partial}{\partial_a}(E)} L_I(E') \), which is by definition of \( L(E') \) equivalent to \( w \in \bigcup_{(E',X)\in \frac{\partial}{\partial_a}(E)} L(E') \).

The partial derivation can be extended from symbols to words as follows:

**Definition 16 (Word Derivative).** Let \( \mathcal{E} = (\Sigma, \Gamma, \mathcal{P}, \mathcal{F}) \) be an expression environment and let \( E \) be a constrained expression over \( \mathcal{E} \). Let \( a \) be a symbol in \( \Sigma \) and \( w \) be a word in \( \Sigma^* \). Then:

\[
\frac{\partial}{\partial_a}(E) = \left\{ \begin{array}{l}
\frac{\partial}{\partial_a}(E) \\
\bigcup_{(E',X)\in \frac{\partial}{\partial_a}(E)} \frac{\partial}{\partial_a}(E')
\end{array} \right. \text{ if } w = \varepsilon, \\
onlysource{\text{otherwise.}}}
\]
Theorem 2. Let $E = (\Sigma, \Gamma, \mathcal{P}, \mathcal{F})$ be an expression environment and let $E$ be a constrained expression over $\mathcal{E}$. Let $w$ be a word in $\Sigma^+$. Then the following conditions hold:

1. $w^{-1}(L_I(E)) = \bigcup_{(E', X) \in \frac{\partial}{\partial w}(E)} L^1_I(E')$,
2. $w^{-1}(L(E)) = \bigcup_{(E', X) \in \frac{\partial}{\partial w}(E)} L(E')$.

Proof.

1. By recurrence over the length of $w$. If $w \in \Sigma$, the condition is satisfied according to Theorem 1. Let $w = aw'$ with $a \in \Sigma$ and $w' \in \Sigma^+$. Then $w^{-1}(L_I(E)) = w'^{-1}(a^{-1}(L_I(E)))$. According to Theorem 1, $w'^{-1}(a^{-1}(L_I(E))) = w'^{-1}(\bigcup_{(E', X) \in \frac{\partial}{\partial w'}(E')} L^1_I(E'))$. Hence $\bigcup_{(E', X) \in \frac{\partial}{\partial w'}(E')} w'^{-1}(L_I(E')) = \bigcup_{(E', X) \in \frac{\partial}{\partial w'}(E')} \bigcup_{(E'' \in E') \in \frac{\partial}{\partial w'}(E')} L^1_I(E'')$. Since by definition, $\{(E', X') \in \frac{\partial}{\partial w'}(E') | (E', X) \in \frac{\partial}{\partial w'}(E)\}$ is equal to $\{(E'', X') \in \frac{\partial}{\partial w'}(E') \bigcup_{(E'', X'') \in \frac{\partial}{\partial w'}(E')} L^1_I(E'') = \bigcup_{(E', X) \in \frac{\partial}{\partial w'}(E')} L_I(E')$.

2. Let $u$ be a word in $w^{-1}(L(E))$. By definition, it is equivalent to the fact that there exists an interpretation $I$ such that $u \in w^{-1}(L_I(E))$. From the previous point, there exists an interpretation $I$ such that $u \in w^{-1}(L_I(E))$ if and only if there exists an interpretation $I$ such that $u \in \bigcup_{(E', X) \in \frac{\partial}{\partial w'}(E')} L_I(E')$, which is equivalent by definition of $L(E')$ to $u \in \bigcup_{(E', X) \in \frac{\partial}{\partial w'}(E')} L(E')$.

\[ \square \]

Corollary 2. Let $E = (\Sigma, \Gamma, \mathcal{P}, \mathcal{F})$ be an expression environment and let $E$ be a constrained expression over $\mathcal{E}$. Let $w$ be a word in $\Sigma^+$. Then:

\[ w \in L_I(E) \iff \exists (E', X) \in \frac{\partial}{\partial w}(E), \varepsilon \in L_I(E'), \]

\[ w \in L(E) \iff \exists (E', X) \in \frac{\partial}{\partial w}(E), \varepsilon \in L(E'). \]

Example 4. Let us consider the expression $E_1 = x b^* y | \sim (f(ax), f(y))$ from Example 3. Let us compute the constrained derivative of $E_1$ w.r.t. the symbol $a$. The process can be expressed as follows. Since the expression starts with a variable, an assumption has to be made:

1. Maybe the variable $x$ starts with an $a$. In this case, we replace all the occurrences of $x$ by $ax$ except the first one, where the symbol $a$ is erased by the derivation. Hence we get the couple ($F_1$, $\{(x, ax)\}$) with $F_1 = x b^* y | \sim (f(ax), f(y))$.

2. Otherwise the variable $x$ can be replaced by $\varepsilon$, and we try to derive the obtained expression $eb^* y \sim (f(\varepsilon), f(y))$. The catenation $eb^* y$ implies that we have to make the assumption that $y$ starts with the symbol $a$. In this case, we replace all the occurrences of $y$ by $ay$ except the first one, where the symbol $a$ is erased by the derivation. Hence we get the couple ($F_2$, $\{(x, \varepsilon), (y, ay)\}$) with $F_2 = y \sim (f(\varepsilon), f(ay))$.

Hence:

\[ \frac{\partial}{\partial a}(E_1) = \left\{ \begin{array}{l} (x b^* y \sim (f(ax), f(y)), \{(x, ax)\}), \\ (y \sim (f(\varepsilon), f(ay)), \{(x, \varepsilon), (y, ay)\}) \end{array} \right\}. \]

The constrained derivative of $E_1$ w.r.t. the word $ab$ is obtained by computing the constrained derivative of $F_1$ and $F_2$ w.r.t. $b$:

\[ \frac{\partial}{\partial b}(F_1) = \left\{ \begin{array}{l} (x b^* y \sim (f(axb), f(y)), \{(x, bx)\}), \\ (eb^* y \sim (f(a), f(y)), \{(x, \varepsilon)\}), \\ (y \sim (f(a), f(by)), \{(x, \varepsilon), (y, by)\}) \end{array} \right\} \]

\[ \frac{\partial}{\partial b}(F_2) = \left\{ \begin{array}{l} (y \sim (f(\varepsilon), f(aby)), \{(y, by)\}) \end{array} \right\} \]

Hence

\[ \frac{\partial}{\partial ab}(E_1) = \left\{ \begin{array}{l} (x b^* y \sim (f(axb), f(y)), \{(x, bx)\}), \\ (eb^* y \sim (f(a), f(y)), \{(x, \varepsilon)\}), \\ (y \sim (f(a), f(by)), \{(x, \varepsilon), (y, by)\}), \\ (y \sim (f(\varepsilon), f(aby)), \{(y, by)\}) \end{array} \right\} \]

\[ \square \]
4 \(\varepsilon\) Membership Test for Constrained Expressions

In this section, we consider the membership test for the empty word \(\varepsilon\). We first consider the case where both the interpretation and the realization are fixed, which is a case where this test is decidable. Then we consider the two other cases and show that they are equivalent to a satisfiability problem.

4.1 \(\varepsilon\) and \((I, r)\)-Language

Corollary asserts that the \((I, r)\)-language denoted by a constrained expression is regular. Consequently, any membership test can be performed via the regularization. However, this transformation may be avoided by directly and inductively computing the classical predicate function Null and embedding the regularization in its computation.

Definition 17 (Null\(_{I,r}\) Predicate). Let \(\mathcal{E} = (\Sigma, \Gamma, \mathcal{P}, \mathcal{F})\) be an expression environment and let \(E\) be a constrained expression over \(\mathcal{E}\). Let \(I\) be an expression interpretation over \(\mathcal{E}\) and \(r\) be a \(\Gamma\)-realization over \(I\). The boolean \(\text{Null}_{I,r}(E)\) is defined by:

\[
\text{Null}_{I,r}(E) = (\varepsilon \in L_{I,r}(E)).
\]

Proposition 4. Let \(\mathcal{E} = (\Sigma, \Gamma, \mathcal{P}, \mathcal{F})\) be an expression environment and let \(E\) be a constrained expression over \(\mathcal{E}\). Let \(I\) be an expression interpretation over \(\mathcal{E}\) and \(r\) be a \(\Gamma\)-realization over \(I\). Then the boolean \(\text{Null}_{I,r}(E)\) is inductively computed as follows:

\[
\text{Null}_{I,r}(\alpha) = (r(\alpha) == \varepsilon), \quad \text{Null}_{I,r}(\emptyset) = 0,
\]

\[
\text{Null}_{I,r}(\alpha \cdot E_1) = (r(\alpha) == \varepsilon) \land \text{Null}_{I,r}(E_1),
\]

\[
\text{Null}_{I,r}(F \cdot G) = \text{Null}_{I,r}(F) \land \text{Null}_{I,r}(G),
\]

\[
\text{Null}_{I,r}(F^*) = 1,
\]

\[
\text{Null}_{I,r}(E_1 | \phi) = \text{Null}_{I,r}(E_1) \land \text{eval}_{I,r}(\phi),
\]

where \(k\) is any integer, \(f\) is any \(k\)-ary boolean operator, \(f'\) is the boolean operator associated with \(f\), \(E_1, \ldots, E_k\) are any \(k\) constrained expression over \(\mathcal{E}\), \(\alpha\) is any word in \((\Sigma \cup \Gamma)^*\) and \(\phi\) is any boolean formulae in \(\mathcal{P}(\mathcal{F}(I))\).

Proof. By induction over the structure of constrained expressions.

\[
\text{Null}_{I,r}(\alpha) = (r(\alpha) == \varepsilon) = (\varepsilon \in \{r(\alpha)\}) = (\varepsilon \in L_{I,r}(\alpha))
\]

\[
\text{Null}_{I,r}(\emptyset) = 0 = (\varepsilon \in \emptyset) = (\varepsilon \in L_{I,r}(\emptyset))
\]

\[
\text{Null}_{I,r}(\alpha \cdot E_1) = (r(\alpha) == \varepsilon) \land \text{Null}_{I,r}(E_1)
\]

\[
= (r(\alpha) == \varepsilon) \land (\varepsilon \in L_{I,r}(E_1))
\]

\[
= (\varepsilon \in \{r(\alpha)\}) \land (\varepsilon \in L_{I,r}(E_1))
\]

\[
= (\varepsilon \in L_{I,r}(\alpha \cdot E_1))
\]

\[
\text{Null}_{I,r}(f(E_1, \ldots, E_k)) = f'(\text{Null}_{I,r}(E_1), \ldots, \text{Null}_{I,r}(E_k))
\]

\[
= f'(\varepsilon \in L_{I,r}(E_1), \ldots, \varepsilon \in L_{I,r}(E_k))
\]

\[
= (\varepsilon \in L_{I,r}(f(E_1, \ldots, E_k))
\]

\[
\text{Null}_{I,r}(F \cdot G) = \text{Null}_{I,r}(F) \land \text{Null}_{I,r}(G)
\]

\[
= (\varepsilon \in L_{I,r}(F)) \land (\varepsilon \in L_{I,r}(G))
\]

\[
= (\varepsilon \in L_{I,r}(F \cdot G))
\]

\[
\text{Null}_{I,r}(F^*) = 1 = (\varepsilon \in L_{I,r}(F^*))
\]

\[
\text{Null}_{I,r}(E_1 | \phi) = \text{Null}_{I,r}(E_1) \land \text{eval}_{I,r}(\phi)
\]

\[
= \varepsilon \in L_{I,r}(E_1) \land \text{eval}_{I,r}(\phi)
\]

\[
= \varepsilon \in L_{I,r}(E_1 | \phi)
\]

4.2 General Cases

As it was the case for derivating, the computation of the Null predicate needs assumptions to be made. However, we only need here to determine which variable symbols have to be transformed into the empty
word. Since we need to "erase" several symbols at the same time, we define several notations to perform the corresponding substitutions.

Let $E = (\Sigma, \Gamma, \mathcal{P}, \mathcal{F})$ be an expression environment. We denote by $\text{Sub}(\Gamma, \Sigma)$ the set of the functions from $\Gamma$ to $(\Sigma \cup \Gamma)^*$. Let $X \subseteq \Gamma$. We denote by $S_{X \leftarrow \varepsilon}$ the substitution defined by $S_{X \leftarrow \varepsilon}(y) = \{ \varepsilon \text{ if } y \in X, \}$ $\{ y \text{ otherwise.} \}$

Let $\alpha$ be a word in $\Gamma^*$. We denote by $\Gamma_\alpha$ the subset $\{ y \in \Gamma \mid \exists u, v \in \Gamma^*, \alpha = uv \}$ of $\Gamma$. We denote by $\top$ (resp. $\bot$) the $0$-ary boolean operator True (resp. False). Given a formula $\phi$, we denote by $\phi_{X \leftarrow \varepsilon}$ the formula inductively computed by:
\[
\phi_{X \leftarrow \varepsilon} = \begin{cases} 
\phi & \text{if } X = \emptyset, \\
(\phi_{x \leftarrow \varepsilon})_{X' \leftarrow \varepsilon} & \text{if } X = X' \cup \{ x \}.
\end{cases}
\]

The general computation of the Null predicate takes into account several informations of an expression: first, it needs to consider the expression itself, in order to determine if $\varepsilon$ may appear; but secondly, it has to consider the fact that several formulae that appear in the expression have to be satisfied, otherwise the language may be empty. Hence we first compute a particular indicator set, made of couples composed of a set of variable symbols that need to be erased and a formula that needs to be satisfied.

**Definition 18 ($S^\varepsilon(E)$).** Let $E = (\Sigma, \Gamma, \mathcal{P}, \mathcal{F})$ be an expression environment and let $E$ be a constrained expression over $E$. We denote by $S^\varepsilon(E)$ the subset of $2^\Gamma \times \mathcal{P}(\mathcal{F}(\Gamma))$ inductively defined by:
\[
S^\varepsilon(\alpha) = \begin{cases} 
\emptyset & \text{if } \alpha \notin \Gamma^*, \\
\{(\Gamma_\alpha, \top)\} & \text{otherwise},
\end{cases}
\]
\[
S^\varepsilon(\emptyset) = \emptyset,
\]
\[
S^\varepsilon(\alpha \cdot E_1) = \begin{cases} 
\emptyset & \text{if } \alpha \notin \Gamma^*, \\
(\{(\Gamma_\alpha, \top)\} \otimes S^\varepsilon(E_1)) & \text{otherwise},
\end{cases}
\]
\[
S^\varepsilon(E_1 + E_2) = S^\varepsilon(E_1) \cup S^\varepsilon(E_2),
\]
\[
S^\varepsilon(E_1 \cdot E_2) = S^\varepsilon(E_1) \otimes S^\varepsilon(E_2),
\]
\[
S^\varepsilon(E_1^*) = \{(\emptyset, \top)\},
\]
\[
S^\varepsilon(E_1 | \phi) = \bigcup_{(X, \phi) \in S^\varepsilon(E_1)} \{(X, (\phi \land \psi)_{X \leftarrow \varepsilon})\},
\]

where for any two subsets $S_1, S_2$ of $2^\Gamma \times \mathcal{P}(\mathcal{F}(\Gamma))$, $S_1 \otimes S_2 = \bigcup_{(X_1, \phi_1) \in S_1, (X_2, \phi_2) \in S_2} \{(X_1 \cup X_2, (\phi_1 \land \phi_2)_{(X_1 \cup X_2) \leftarrow \varepsilon})\}$.

Using this previous indicator set, it can be shown that the computation of the different predicates Null is equivalent to different satisfiability problems.

**Theorem 3.** Let $E = (\Sigma, \Gamma, \mathcal{P}, \mathcal{F})$ be an expression environment and let $E$ be a constrained expression over $E$. Let $I$ be an expression interpretation over $E$. Let $r$ be a realisation in $\text{Real}_{\varepsilon}(I)$. Then the two following conditions are equivalent:

- **Null$_{r,I}(E)$**
- there exists $(X, \phi)$ in $S^\varepsilon(E)$ such that the two following conditions are satisfied:
  - $\forall x \in \Gamma, x \in X \Rightarrow r(x) = \varepsilon,$
  - $\varepsilon = \text{eval}_{(I, r)}(\phi) = 1.$

**Proof.** By induction over the structure of $E$.

Let us say that a couple $(X, \phi)$ in $2^\Gamma \times \mathcal{P}(\mathcal{F}(\Gamma))$ satisfies the condition $\mathcal{C}$ if the two following conditions are satisfied:

- $\forall x \in \Gamma, x \in X \Rightarrow r(x) = \varepsilon,$
- $\varepsilon = \text{eval}_{(I, r)}(\phi) = 1.$

Hence the second condition of the equivalence can be rephrased as "there exists a couple $(X, \phi)$ in $S^\varepsilon(E)$ that satisfies $\mathcal{C}$", formally denoted by $\exists (X, \phi) \in S^\varepsilon(E) \mid \mathcal{C}$.

Null$_{r,I}(\alpha) \iff r(\alpha) = \varepsilon$
\[
\iff \alpha \in \Gamma^* \land x \in \Gamma_\alpha \Rightarrow r(x) = \varepsilon
\]
\[
\iff (\Gamma_\alpha, \top) \in S^\varepsilon(\alpha) \mid \mathcal{C}
\]
\[
\iff \exists (X, \phi) \in S^\varepsilon(\alpha) \mid \mathcal{C}
\]

Null$_{r,I}(\emptyset) = 0$ and $S^\varepsilon(\emptyset) = \emptyset$

\[17\]
Null_{I,t}(\alpha + E_1) \iff r(\alpha) = \varepsilon \land \varepsilon \in L_{I,t}(E_1)
\iff r(\alpha) = \varepsilon \land \exists (X, \phi) \in S'(E_1) \mid C
\iff \alpha \in I^* \land \varepsilon \in I_{\alpha} \Rightarrow r(x) = \varepsilon \land \exists (X, \phi) \in S'(E_1) \mid C
\iff (I_{\alpha}, \top) \mid C \land \exists (X, \phi) \in S'(E_1) \mid C
\iff (I_{\alpha} \cup X, (\top \land \phi)_{X \mapsto \varepsilon}) \in \{(I_{\alpha}, \top)\} \otimes S'(E_1) \mid C
\iff \exists (X, \phi) \in S'(\alpha \mapsto E_1) \mid C
Null_{I,t}(E_1 + E_2) \iff \varepsilon \in L_{I,t}(E_1) \lor \varepsilon \in L_{I,t}(E_2)
\iff \exists (X, \phi) \in S'(E_1) \mid C \lor \exists (X, \phi) \in S'(E_2) \mid C
\iff \exists (X, \phi) \in S'(E_1) \cup S'(E_2) \mid C
\iff \exists (X, \phi) \in S'(E_1 + E_2) \mid C
Null_{I,t}(E_1 \cdot E_2) \iff \varepsilon \in L_{I,t}(E_1) \land \varepsilon \in L_{I,t}(E_2)
\iff \exists (X_1, \phi_1) \in S'(E_1) \mid C \land \exists (X_2, \phi_2) \in S'(E_2) \mid C
\iff \exists (X_1, \phi_1) \in S'(E_1), \exists (X_2, \phi_2) \in S'(E_2) | (X_1 \cup X_2, (\phi_1 \land \phi_2)_{X_1 \cup X_2 \mapsto \varepsilon}) \mid C
\iff \exists (X, \phi) \in S'(E_1) \otimes S'(E_2) \mid C
\iff \exists (X, \phi) \in S'(E_1 \cdot E_2) \mid C
Null_{I,t}(E_1^*) = 1 \text{ and } \{0, \top\} \in S'(E_1^*) \mid C
Null_{I,t}(E_1 | \phi) \iff \varepsilon \in L_{I,t}(E_1) \land \text{eval}_{I,t}(\phi)
\iff \exists (X', \phi') \in S'(E_1) \mid C \land \text{eval}_{I,t}(\phi)
\iff \exists (X', \phi') \in S'(E_1) \mid C \land \text{eval}_{I,t}((\phi' \land \phi)_{X' \mapsto \varepsilon})
\iff \exists (X', \phi') \in S'(E_1 | \phi) \mid C
\textbf{Definition 19 (Null Predicate).} Let \( E = (\Sigma, \Gamma, \mathcal{P}, \mathcal{F}) \) be an expression environment and let \( E \) be a constrained expression over \( E \). Let \( I \) be an expression interpretation over \( E \). The boolean \( \text{Null}_I(E) \) is defined by:
$$\text{Null}_I(E) = (\varepsilon \in L_I(E)).$$

\textbf{Corollary 3 (of Theorem 3).} Let \( E = (\Sigma, \Gamma, \mathcal{P}, \mathcal{F}) \) be an expression environment and let \( E \) be a constrained expression over \( E \). Let \( I \) be an expression interpretation over \( E \). Then the two following conditions are equivalent:
- \( \text{Null}_I(E) = 1 \),
- there exists \((X, \phi)\) in \( S'(E) \) and \( r \) in \( \text{Real}_I(I) \) such that \( \text{eval}_{I,t}(\phi) = 1 \).

\textbf{Definition 20 (Null Predicate).} Let \( E = (\Sigma, \Gamma, \mathcal{P}, \mathcal{F}) \) be an expression environment and let \( E \) be a constrained expression over \( E \). Let \( I \) be an expression interpretation over \( E \). The boolean \( \text{Null}(E) \) is defined by:
$$\text{Null}(E) = (\varepsilon \in L(E)).$$

\textbf{Corollary 4 (of Theorem 3).} Let \( E = (\Sigma, \Gamma, \mathcal{P}, \mathcal{F}) \) be an expression environment and let \( E \) be a constrained expression over \( E \). Then the two following conditions are equivalent:
- \( \text{Null}(E) = 1 \)
- there exists \((X, \phi)\) in \( S'(E) \), \( I \) in \( \text{Int}(E) \) and \( r \) in \( \text{Real}_I(I) \) such that \( \text{eval}_{I,t}(\phi) = 1 \).

\textbf{Example 5.} Let us consider the expression \( E_1 = x^*y | \sim (f(x), f(y)) \) and its constrained derivative w.r.t. \( ab \) (Example 4):
$$\overline{\partial_{ab}}(E_1) = \begin{cases} (x^*y | \sim (f(abx), f(by))), \{(x, bx)\}, \\ (c^*y | \sim (f(a), f(by))), \{(x, \varepsilon)\}, \\ (y | \sim (f(a), f(by))), \{(x, \varepsilon), (y, by)\}, \\ (y | \sim (f(c), f(aby))), \{(y, by)\} \end{cases}$$

In order to decide whether \( ab \) belongs to \( L(E_1) \), let us test whether there is an expression \( E' \) in \( \overline{\partial_{ab}}(E_1) \) such that \( \text{Null}(E') = 1 \). Let us consider the expression \( E'_1 = x^*y | \sim (f(abx), f(y)) \). Since epsilon should be in matching, \( x^*y \) has to be nullable and both \( x \) and \( y \) have to be realized as epsilon. Consequently, the boolean formula, which has to be satisfied, is transformed into \( (f(ab), f(\varepsilon)) \). This step is exactly what the set \( S'(E'_1) \) computes:
S^ε(x^*y) = S^ε(x) ⊗ S^ε(b^*) ⊗ S^ε(y)
= \{(\{x\}, \top)\} ⊗ \{(\emptyset, \top)\} ⊗ \{(\{y\}, \top)\}
= \{(x, \top)\}
S^ε(E'_1) = \{(\{x, y\}, \sim (f(ab), f(\varepsilon)))\}

If there exists an interpretation I and a realization r associating x and y with \varepsilon such that eval(I,r)(\sim (f(ab), f(\varepsilon))) = 1, then \varepsilon belongs to L(E). Finally, considering the same interpretation and a realization r’ associating ab with x and \varepsilon with y, eval(I,r’)(\sim (f(x), f(y))) = eval(I,r)(\sim (f(ab), f(\varepsilon))) = 1 and then ab belongs to the (I, r’)-language denoted by E_1, i.e. [E_1]_{I,r'} = \{ab^*\varepsilon | \sim (f(ab), f(\varepsilon))\}.

5 Decidability Considerations

Previous section shows that the membership test is equivalent to a classical satisfiability problem. Moreover, it is well-known that such problems are generally undecidable.

Theorem 4. Let \mathcal{E} = (\Sigma, \Gamma, \mathcal{P}, \mathcal{F}) be an expression environment. Given a boolean formula \phi in \mathcal{P}(\mathcal{F}(\Gamma)) and an interpretation I in \text{Int}(\mathcal{E}), to determine whether or not there exists a realization r in Real(\Gamma)(I) such that eval(I,r)(\phi) = 1 is undecidable.

Proof. Let k be an integer. Let \mathcal{E} = (\Sigma, \{x_1, \ldots, x_k\}, \mathcal{P}, \mathcal{F}). Let \mathcal{S} be a system of diophantine equations with k variables and \mathcal{T} be a symbol in \mathcal{P}_k. Let us consider the expression interpretation I = (\Sigma^*, \mathcal{S}) such that \mathcal{G}(\mathcal{T}) = \{(w_1, \ldots, w_k) | (|w_1|, \ldots, |w_k|) is a solution of \mathcal{S}\}. Let \phi = \mathcal{P}(x_1, \ldots, x_k). Then there exists a solution \(n_1, \ldots, n_k\) for \mathcal{S} if and only if there exists a realization r that associates for any integer j in \{1, \ldots, k\} the variable \(x_j\) with a word \(w_j\) of length \(n_j\) such that eval(I,r)(\phi) = 1. The solvability of diophantine systems (a.k.a. the tenth problem of Hilbert) has been proved to be undecidable by Matiyasevich [14]. Hence to determine whether or not there exists a realization r in Real(\Gamma)(I) such that eval(I,r)(\phi) = 1 is undecidable.

However, given a boolean formula \phi in \mathcal{P}(\mathcal{F}(\Gamma)), to determine whether or not there exists an interpretation I in \text{Int}(\mathcal{E}) and a realization r in Real(\Gamma)(I) such that eval(I,r)(\phi) = 1 is decidable, only using propositional logic.

Theorem 5. Let \mathcal{E} = (\Sigma, \Gamma, \mathcal{P}, \mathcal{F}) be an expression environment. Given a boolean formula \phi in \mathcal{P}(\mathcal{F}(\Gamma)), to determine whether or not there exists an interpretation I in \text{Int}(\mathcal{E}) and a realization r in Real(\Gamma)(I) such that eval(I,r)(\phi) = 1 is decidable.

The next subsections are devoted to prove Theorem 5. We first show that any boolean formula can be transformed into a propositional formula (which is a boolean formula with only 0-ary predicate symbol). Then we show that any formula is equisatisfiable to its propositional form whenever there exists an evaluation which is an injection. We finally show that any formula admits an equivalent formula such that an injection exists.

5.1 Propositionalisation

The propositionalisation of a boolean formula is performed by replacing any predicate by a unique symbol; in fact, any predicate appearing in the formula is considered as a new symbol.

Definition 21 (Propositionalisation). Let \mathcal{E} = (\Sigma, \Gamma, \mathcal{P}, \mathcal{F}) be an expression environment. The propositionalisation of a boolean formula \phi in \mathcal{P}(\mathcal{F}(\Gamma)) is the transformation T inductively defined as follows:

\[ T(o(\phi_1, \ldots, \phi_k)) = o(T(\phi_1), \ldots, T(\phi_k)) \]
\[ T(P(t_1, \ldots, t_k)) = P(t_{i_1}, \ldots, t_{i_k}), \]

where k is any integer, P is any predicate symbol in \mathcal{P}_k, t_1, \ldots, t_k are any k elements in \mathcal{F}(\Gamma), o is any k-ary boolean operator associated with a mapping o' from \{0, 1\}^k to \{0, 1\} and \phi_1, \ldots, \phi_k are any k boolean formulae in \mathcal{P}(\mathcal{F}(\Gamma)). The symbol P(t_{i_1}, \ldots, t_{i_k}) is the propositional predicate symbol associated with the term P(t_1, \ldots, t_k).
**Definition 22 (Propositional Alphabet).** Let $E = (\Sigma, \Gamma, P, F)$ be an expression environment. The propositional alphabet of a boolean formula $\phi$ in $P(F(\Gamma))$ is the set $P'(\phi)$ inductively defined as follows:

$P'(o(\phi_1, \ldots, \phi_k)) = P'(\phi_1) \cup \cdots \cup P'(\phi_k),$

$P'(P(t_1, \ldots, t_k)) = \{P(t_1, \ldots, t_k)\},$

where $k$ is any integer, $P$ is any predicate symbol in $P_k$, $t_1, \ldots, t_k$ are any $k$ elements in $F(\Gamma)$, $o$ is any $k$-ary boolean operator associated with a mapping $o'$ from $\{0, 1\}^k$ to $\{0, 1\}$ and $\phi_1, \ldots, \phi_k$ are any $k$ boolean formulae in $P(F(\Gamma))$.

**Proposition 5.** Let $E = (\Sigma, \Gamma, P, F)$ be an expression environment. Let $\phi$ be a boolean formula in $P(F(\Gamma))$. Then:

$T(\phi)$ is a boolean formula in $(P'(\phi))(\emptyset)$.

Furthermore, $P'(\phi)$ is a finite set.

**Proof.** Inductively deduced from Definition 21 and from Definition 22.

Two of the main interests of these propositional formulae are that (I) they do not need realization to be evaluated (since there is no variable symbols nor terms) and (II) their satisfiability is decidable, using truth tables for example.

Let us now show that the Propositionalisation may produce an equisatisfiable formula.

### 5.2 Equisatisfiability of the Propositionalisation

Once the propositionalisation applied over a formula, it can be determined if the obtained formula is satisfiable. This first information may allow us to conclude in two cases.

**Proposition 6.** Let $E = (\Sigma, \Gamma, P, F)$ be an expression environment. Let $\phi$ be a boolean formula in $P(F(\Gamma))$. Let $I$ be an interpretation in $\text{Int}(E)$ and $r$ be a realization in $\text{Real}_I(F\Gamma)$. Let $I' = (\Sigma^*, \emptyset')$ be the expression interpretation over $E'$, such that for any symbol $P(t_1, \ldots, t_k)$ in $P'(\phi)$, $\text{eval}_{I', (P(t_1, \ldots, t_k))} = \text{eval}_{I, r}(P(t_1, \ldots, t_k))$.

Then:

$\text{eval}_{I, r}(\phi) = \text{eval}_{I'}(T(\phi)).$

**Proof.** By induction over the structure of $\phi$.

If $\phi = P(t_1, \ldots, t_k)$, then $\text{eval}_{I, r}(P(t_1, \ldots, t_k)) = \text{eval}_{I}(P(t_1, \ldots, t_k)).$

If $\phi = o(\phi_1, \ldots, \phi_k)$, then

$\text{eval}_{I, r}(o(\phi_1, \ldots, \phi_k)) = o'(\text{eval}_{I, r}(\phi_1), \ldots, \text{eval}_{I, r}(\phi_k))$

$= o'(\text{eval}_{I}(T(\phi_1)), \ldots, \text{eval}_{I}(T(\phi_k)))$

$= \text{eval}_{I}(o(T(\phi_1), \ldots, T(\phi_k)))$

$= \text{eval}_{I}(T(\phi)).$

**Corollary 5.** Let $E = (\Sigma, \Gamma, P, F)$ be an expression environment. Let $\phi$ be a boolean formula in $P(F(\Gamma))$. Then:

- If $T(\phi)$ is a contradiction, so is $\phi$.
- If $T(\phi)$ is a tautology, so is $\phi$.

However, the satisfiability of $T(\phi)$, when it is not a tautology, is not sufficient to conclude. Indeed, maybe two distinct predicates in $T(\phi)$ have to be evaluated differently but the predicates they are associated with cannot be in $\phi$. As an example, consider the formulae $\phi = P((x \cdot y) \cdot z) \land \neg P(x \cdot (y \cdot z))$ and $T(\phi) = P((x \cdot y) \cdot z) \land \neg P((x \cdot y) \cdot z)$. The formula $T(\phi)$ is satisfiable when $P((x \cdot y) \cdot z)$ is true and $P((x \cdot y) \cdot z)$ is not. However, whatever the realization considered, $P((x \cdot y) \cdot z)$ and $P((x \cdot y) \cdot z)$ will always be equi-evaluated. Let us formally define the notion of injection that separates two distinct terms while evaluating.

Let $E = (\Sigma, \Gamma, P, F)$ be an expression environment. Let $\phi$ be a boolean formula in $P(F(\Gamma))$. The set of the terms of $\phi$ is the set $\text{Term}(\phi)$ inductively defined by:

$\text{Term}(o(\phi_1, \ldots, \phi_k)) = \text{Term}(\phi_1) \cup \cdots \cup \text{Term}(\phi_k),$

$\text{Term}(P(t_1, \ldots, t_k)) = \{t_1, \ldots, t_k\},$

where $k$ is any integer, $P$ is any predicate symbol in $P_k$, $t_1, \ldots, t_k$ are any $k$ elements in $F(\Gamma)$, $o$ is any $k$-ary boolean operator associated with a mapping $o'$ from $\{0, 1\}^k$ to $\{0, 1\}$ and $\phi_1, \ldots, \phi_k$ are any $k$ boolean formulae in $P(F(\Gamma))$.  

20
Definition 23 (Injection). Let \( \mathcal{E} = (\Sigma, \Gamma, \mathcal{P}, \mathcal{F}) \) be an expression environment. Let \( T \) be a subset of \( \mathcal{F}(\Gamma) \). Let \( I \) be an interpretation in \( \text{Int}(\mathcal{E}) \) and \( r \) be a realization in \( \text{Real}_I(\mathcal{I}) \). The function \( \text{eval}_{I,r} \) is said to be an injection of \( T \) in \( \Sigma^* \) if:

for any two terms \( t_1 \) and \( t_2 \) in \( \text{Term}(\phi) \), \( \text{eval}_{I,r}(t_1) \neq \text{eval}_{I,r}(t_2) \).

As far as such an evaluation exists, let us show that the propositionalisation preserves the satisfiability.

Proposition 7. Let \( \mathcal{E} = (\Sigma, \Gamma, \mathcal{P}, \mathcal{F}) \) be an expression environment. Let \( \phi \) be a boolean formula in \( \mathcal{P}(\mathcal{F}(\Gamma)) \). Let \( I = (\Sigma^*, \mathfrak{F}) \) be an interpretation in \( \text{Int}(\mathcal{E}) \) and \( r \) be a realization in \( \text{Real}_I(\mathcal{I}) \) such that \( \text{eval}_{I,r} \) is an injection of \( \text{Term}(\phi) \) in \( \Sigma^* \). Let \( \mathcal{E}' = (\Sigma, \Gamma, \mathcal{P}', \emptyset) \). Let \( I' = (\Sigma^*, \mathfrak{F}') \) be an expression interpretation over \( \mathcal{E}' \). Let \( \mathcal{E}'' = (\Sigma^*, \mathfrak{F}'') \) be an expression interpretation over \( \mathcal{E} \) satisfying the two following conditions:

- for any function symbol \( f \) in \( \mathcal{F}_k \), \( \mathfrak{F}''(f) = \mathfrak{F}(f) \).
- for any predicate symbol \( P(t_1,\ldots,t_k) \) in \( \mathcal{P}(\phi) \), \( \text{eval}_{I}(P(t_1,\ldots,t_k)) = 1 \Leftrightarrow (\text{eval}_{I,r}(t_1),\ldots,\text{eval}_{I,r}(t_k)) \in \mathfrak{F}''(P) \).

Then:

\( \text{eval}_{I'}(\phi) \) is an injection of \( \text{Term}(\phi) \) in \( \Sigma^* \) such that \( \text{eval}_{I}(\phi) = \text{eval}_{I'}(T(\phi)) \).

Proof. (I) Let us show that \( \text{eval}_{I'} \) is an injection of \( \text{Term}(\phi) \) in \( \Sigma^* \). Let \( t \) be a term in \( \mathcal{F}(\Gamma) \). (a) Let us show by induction over the structure of \( t \) that \( \text{eval}_{I'}(t) = \text{eval}_{I,r}(t) \). (i) If \( t = x \in \Gamma \), then \( \text{eval}_{I'}(x) = r(x) = \text{eval}_{I,r}(x) \). (ii) Let us suppose that \( t = f(t_1,\ldots,t_k) \) with \( f \) any \( k \)-ary function symbol in \( \mathcal{F}_k \) and \( t_1,\ldots,t_k \) any \( k \) terms in \( \mathcal{F}(\Gamma) \). Then:

\[
\text{eval}_{I'}(f(t_1,\ldots,t_k)) = x_{k+1} \\
\Leftrightarrow (\text{eval}_{I'}(t_1),\ldots,\text{eval}_{I'}(t_k),x_{k+1}) \in \mathfrak{F}''(f) \\
\Leftrightarrow (\text{eval}_{I,r}(t_1),\ldots,\text{eval}_{I,r}(t_k),x_{k+1}) \in \mathfrak{F}(f) \text{ (by definition of } \mathfrak{F}'' \text{)} \\
\Leftrightarrow \text{eval}_{I,r}(f(t_1,\ldots,t_k)) = x_{k+1}.
\]

(b) As a direct consequence of (a), since \( \text{eval}_{I,r} \) is an injection of \( \text{Term}(\phi) \) in \( \Sigma^* \), so is \( \text{eval}_{I'} \).

(II) Let us show by induction over \( \phi \) that \( \text{eval}_{I'}(\phi) = \text{eval}_{I}(T(\phi)) \). (a) If \( \phi = P(t_1,\ldots,t_k) \) with \( P \) a \( k \)-ary predicate symbol in \( \mathcal{P}_k \), then \( \text{eval}_{I'}(P(t_1,\ldots,t_k)) = (\text{eval}_{I'}(t_1),\ldots,\text{eval}_{I'}(t_k)) \in \mathfrak{F}''(P) = \text{eval}_{I'}(P(t_1,\ldots,t_k)) \).

(b) Let us consider that \( \phi = o(\phi_1,\ldots,\phi_k) \). Then:

\[
\text{eval}_{I'}(o(\phi_1,\ldots,\phi_k)) = o'(\text{eval}_{I'}(\phi_1),\ldots,\text{eval}_{I'}(\phi_k)) \\
= o'(\text{eval}_{I}(T(\phi_1)),\ldots,\text{eval}_{I}(T(\phi_k))) \text{ (induction hypothesis)} \\
= \text{eval}_{I'}(o(T(\phi_1),\ldots,T(\phi_k))) \\
= \text{eval}_{I}(T(\phi)).
\]

Corollary 6. Let \( \mathcal{E} = (\Sigma, \Gamma, \mathcal{P}, \mathcal{F}) \) be an expression environment. Let \( \phi \) be a boolean formula in \( \mathcal{P}(\mathcal{F}(\Gamma)) \) such that there exists an injection of \( \text{Term}(\phi) \) in \( \Sigma^* \). Then:

\( T(\phi) \) is satisfiable if and only if \( \phi \) is.

Example 6. Let \( \mathcal{E} = (\Sigma, \Gamma, \mathcal{P}, \mathcal{F}) \) be the expression environment defined by:

- \( \Sigma = \{a, b, c\} \),
- \( \Gamma = \{x, y, z\} \),
- \( \mathcal{P} = \mathcal{P}_2 = \{\prec, \sim\} \),
- \( \mathcal{F}_0 = \Sigma \cup \{\varepsilon\}, \mathcal{F}_2 = \{g, \cdot\} \).

Let us consider the two boolean formulae defined by:

\[
\phi_1 = \prec(g(ab, x), abx) \land \neg(\sim(abx, g(a, bx))) \\
\phi_2 = \prec(\sim(ab, x), abx) \land \neg(\prec(abx, (a, bx))).
\]

The terms that appear in these two formulae are:

\[
\text{Term}(\phi_1) = \{g(ab, x), abc, g(a, bx)\}, \\
\text{Term}(\phi_2) = \{(ab, c), abc, (a, bc)\}.
\]

Let \( I = (\Sigma^*, \mathfrak{F}) \) be the expression interpretation defined by:

\[
\mathfrak{F}(\prec) = \mathfrak{F}(\sim) = \{(u, v) \mid |u| \leq |v|\}.
\]
It is not the case for any expression interpretation. Notice that finally, let us consider a realization $r$ that associates $c$ with $x$. Then
\[
eval(I, r)(g(ab, x)) = cab, \quad \eval(I', r)(abx) = abc, \quad \eval(I, r)(g(a, bx)) = bca,
\]
\[
\eval(I, r)(a, bx) = \eval(I, r)(cx) = abc.
\]
Consequently, $\eval(I, r)$ is an injection of $\text{Term}(\phi_1)$ in $\Sigma^*$ but it is not an injection of $\text{Term}(\phi_2)$ in $\Sigma^*$. Furthermore, it holds that
\[
\eval(I, r)(\phi_1) = \eval(I, r)(\langle cab, abc \rangle \land \neg(\langle abc, bca \rangle)) = \eval(I, r)(1 \land \neg(1)) = 0
\]
\[
\eval(I, r)(\phi_2) = \eval(I, r)(\langle abc, abc \rangle \land \neg(\langle abc, abc \rangle)) = 0
\]
Notice that $\phi_2$ is what can we call an expression contradiction, since it is a contradiction whenever the function $\cdot$ is interpreted as the catenation function, because of its associativity property. Consequently, for any expression interpretation $I'$ and any realization $r'$, $\eval_{I', r'}(\phi_2) = 0$.

It is not the case for $\phi_1$, since there exists an injection of its terms in $\Sigma^*$. Let us show that $\phi_1$ is satisfiable. First, we need to compute the propositionalised formula $T(\phi_1) = \langle g(ab, x), abx \land \neg(\langle abx, g(a, bx) \rangle)$ associated with $\phi_1$. It contains two predicates symbols, $\langle g(ab, x), abx$ and $\langle abx, g(a, bx)$. Then, let us consider an interpretation $I' = (\Sigma^*, \mathfrak{F}'')$ such that $eval_{I'}(\langle g(ab, x), abx \rangle) = 1$ and $eval_{I'}(\langle abx, g(a, bx) \rangle) = 0$. Consequently $eval_{I'}(T(\phi_1)) = 1$.

From this interpretation, we can construct the interpretation $I'' = (\Sigma^*, \mathfrak{F}'')$ defined by:
\[
- \mathfrak{F}''(\alpha) = \{\alpha\}, \text{ for any } \alpha \in \mathfrak{F}_0,
- \mathfrak{F}''(g) = \{(u, v, vu)\},
- \mathfrak{F}''(\cdot) = \{(u, v, u \cdot v)\},
- \mathfrak{F}''(\langle) = \{cab, abc\},
- \mathfrak{F}''(\sim) = \emptyset.
\]

Then:
\[
\eval_{I'', r}(\phi_1) = \eval_{I'', r}(\langle g(ab, x), abx \rangle \land \neg(\langle abx, g(a, bx) \rangle))
\]
\[
= \eval_{I'', r}(\langle cab, abc \rangle \land \neg(\langle abc, cab \rangle))
\]
\[
= \eval_{I'', r}(1 \land \neg(0)) = 1
\]

The existence of the injection allowed us to show that $\phi_1$ was satisfiable via the satisfiability of its propositionalised form. Notice that $T(\phi_2) = \langle \langle g(ab, x), abx \land \neg(\langle abx, g(a, bx) \rangle)$ is satisfiable too, since for any interpretation $I$ satisfying $eval_{I}(\langle g(ab, x), abx \rangle) = 1$ and $eval_{I}(\langle abx, g(a, bx) \rangle) = 0$, $eval_{I}(T(\phi_2)) = 1$. However, since there is no injection due to the associativity of $\cdot$ in any expression interpretation, the satisfiability of $T(\phi_2)$ does not allow us to conclude (see the notion of normalization in the next subsection).

### 5.3 Injections for non-Unary Alphabets via the Normalization

In this subsection, we show that any formula can be transformed into an equivalent one where the set of terms can be evaluated by an injection. In fact, we compute a normal form that consider the associativity of the catenation and the identity element $\varepsilon$. Notice that we do not consider unary alphabets where the catenation is also commutative.

**Definition 24 (Normalized Term).** Let $E = (\Sigma, \Gamma, P, F)$ be an expression environment. Let $t$ be a term in $F(\Gamma)$. The term $t$ is said to be normalized if the two following conditions are satisfied:

- any child of a concatenation node is not equal to $\varepsilon$;
- the root of the left child of any concatenation node in $t$ is not a concatenation node.

**Definition 25 (Normalization).** Let $E = (\Sigma, \Gamma, P, F)$ be an expression environment. The normalization of a term $t$ in $F(\Gamma)$ is the transformation $\cdot$ inductively defined as follows:

\[
x' = x
\]
Definition 26 (Left-Dot Level). Let $\mathcal{E} = (\Sigma, \Gamma, P, F)$ be an expression environment. Let $t$ be a term in $F(\Gamma)$. The left-dot level $\text{ldl}(t)$ is the integer inductively computed as follows:

$$
\text{ldl}(t) = \begin{cases} 
0 & \text{if } t = x \in \Gamma, \\
0 & \text{if } t = f(t_1, \ldots, t_k) \land f \in F_k \setminus \{\cdot\}, \\
1 + \text{ldl}(t_1) & \text{if } t = t_1 \cdot t_2,
\end{cases}
$$

where $x$ is any symbol in $\Gamma$, $f$ is any symbol in $F_k \setminus \{\cdot\}$ and $t_1, \ldots, t_k$ are any $k$ terms in $F(\Gamma)$.

Proposition 8. Let $\mathcal{E} = (\Sigma, \Gamma, P, F)$ be an expression environment. Let $t$ be a term in $F(\Gamma)$. Then:

- $t'$ is a normalized term.

Furthermore, whenever $t$ is a normalized term, then $t = t'$.

Proof. By induction over the structure of $t'$.

(I) If $t = x \in \Gamma$, then $t$ is normalized, $x = x'$ and then $t = t'$.

(II) If $t = f(t_1, \ldots, t_k)$ with $f$ any symbol in $F_k \setminus \{\cdot\}$, by induction hypothesis it holds that for any integer $j$ in $\{1, \ldots, k\}$, $t_j'$ is normalized and if $t_j$ is normalized, then $t_j = t_j'$. As a direct consequence, $t'$ is normalized and if $t$ is normalized, since it implies that for any integer $j$ in $\{1, \ldots, k\}$, $t_j$ is normalized, then $t = t'$.

(III) Let us suppose that $t = t_1 \cdot t_2$. (a) If $t_1 = \varepsilon$ (resp. $t_2 = \varepsilon$), then $t' = t_2'$ (resp. $t' = t_1'$). By induction hypothesis it holds that $t_2'$ (resp. $t_1'$) is normalized. As a consequence, $t'$ is normalized. Let us notice that in this case, $t$ is not normalized. (b) Let us suppose that $t_1 = x$ with $x \in \Gamma$. Hence, $t' = x \cdot t_2'$. According to induction hypothesis, $t_2'$ is normalized and if $t_2$ is normalized, then $t_2 = t_2'$. Since $x' = x$, then $t' = x \cdot t_2'$ is normalized and if $t = x \cdot t_2$ is normalized, then $t' = t$. (c) Let us suppose that $t_1 = f(r_1, \ldots, r_k)$ with $f$ any symbol in $F_k$ and that $t_2 \neq \varepsilon$. By recurrence over $\text{ldl}(t_1)$. (i) If $\text{ldl}(t_1) = 0$, then $t_1 = f(r_1, \ldots, r_k)$ with $f \neq \{\cdot\}$. Hence $t' = (t_1') \cdot (t_2')$. According to induction hypothesis, for any integer $j$ in $\{1, 2\}$, $t_j'$ is normalized and if $t_j$ is normalized, then $t_j = t_j'$. Since $t_j' = f(r'_1, \ldots, r'_k)$, $t_j'$ is normalized (since the left child of its concatenation root is not a concatenation node). Furthermore, if $t$ is normalized, since it implies that both $t_1$ and $t_2$ are normalized and that $t_1' = t_1$ and $t_2' = t_2$, it holds that $t = t'$. (ii) Let us suppose that $\text{ldl}(t_1) = m$ with $m > 0$. Then $t_1 = (t_3 \cdot t_4)$. As a consequence, $t' = (t_3 \cdot (t_4 \cdot t_2))'$. Let us notice that $\text{ldl}(t') = \text{ldl}(t) - 1$. According to recurrence hypothesis, $(t_3 \cdot (t_4 \cdot t_2))'$ is normalized. Let us notice that in this case, $t$ is not normalized.

Let us show now that the normalization preserves the evaluation.

Proposition 9. Let $\mathcal{E} = (\Sigma, \Gamma, P, F)$ be an expression environment. Let $t$ be a term in $F(\Gamma)$. Let $I$ be an interpretation in $\text{Int}(\mathcal{E})$ and $r$ be a realization in $\text{Real}_I(\Gamma)$. Then:

$$
\text{eval}(I, t) = \text{eval}(I, t').
$$

Proof. By induction over the structure of $t'$.

(I) If $t = x \in \Gamma$, then $t' = x = x$. Hence $\text{eval}(I, t) = \text{eval}(I, t')$.

(II) If $t = f(t_1, \ldots, t_k)$ with $f$ any symbol in $F_k \setminus \{\cdot\}$. By induction hypothesis, it holds that for any integer $j$ in $\{1, \ldots, k\}$, $\text{eval}(I)(t_j) = \text{eval}(I)(t_j')$. Hence:

$$
\text{eval}(I)(t) = x_{k+1} \text{ with } \text{eval}(I)(t_1), \ldots, \text{eval}(I)(t_k), x_{k+1} \in \mathbb{F}(f)
$$

$$
\text{eval}(I)(t') = x'_{k+1} \text{ with } \text{eval}(I)(t_1'), \ldots, \text{eval}(I)(t_k'), x'_{k+1} \in \mathbb{F}(f)
$$

Finally, according to Definition 2 it holds that $x_{k+1} = x'_{k+1}$.

(III) Let us suppose that $t = t_1 \cdot t_2$. (a) If $t_1 = \varepsilon$ (resp. $t_2 = \varepsilon$), then $t' = t_2'$ (resp. $t' = t_1'$). By induction hypothesis, $\text{eval}(I)(t_2) = \text{eval}(I)(t_2')$ (resp. $\text{eval}(I)(t_1) = \text{eval}(I)(t_1')$). Then:

$$
\text{eval}(I, t) = \text{eval}(I, t_1) \cdot \text{eval}(I, t_2) = \varepsilon \cdot \text{eval}(I, t_2') \text{ (resp. } \text{eval}(I, t_1') \cdot \varepsilon)
$$

$$
= \text{eval}(I, t').
$$
(b) Let us suppose that \( t_1 = x \) with \( x \in \Gamma \). Then \( t' = x \cdot t_2' \). By induction hypothesis, \( \text{eval}_{(I,r)}(t_2) = \text{eval}_{(I,r)}(t_2') \). Then:

\[
\begin{align*}
\text{eval}_{(I,r)}(t) &= \text{eval}_{(I,r)}(x) \cdot \text{eval}_{(I,r)}(t_2)
= r(x) \cdot \text{eval}_{(I,r)}(t_2')
= \text{eval}_{(I,r)}(x') \cdot t_2'
= \text{eval}_{(I,r)}(t').
\end{align*}
\]

(c) Let us suppose that \( t_1 = f(r_1, \ldots, r_k) \) with \( f \) any symbol in \( F_k \) and that \( t_2 \neq \varepsilon \). By recurrence over \( \text{ldl}(t_1) \).

(i) If \( \text{ldl}(t_1) = 0 \), then \( t_1 = f(r_1, \ldots, r_k) \) with \( f \neq \{\cdot \} \). Hence \( t' = (t_1)' \cdot (t_2)' \). According to induction hypothesis, for any integer \( j \in \{1,2\} \), \( \text{eval}_{(I,r)}(t_j) = \text{eval}_{(I,r)}(t_j') \). As a consequence,

\[
\text{eval}_{(I,r)}(t) = \text{eval}_{(I,r)}(t_1) \cdot \text{eval}_{(I,r)}(t_2)
= \text{eval}_{(I,r)}(t_1') \cdot \text{eval}_{(I,r)}(t_2')
= \text{eval}_{(I,r)}(t').
\]

(ii) Let us suppose that \( \text{ldl}(t_1) = m \) with \( m > 0 \). Then \( t_1 = (t_3 \cdot t_4) \). As a consequence, \( t' = (t_3 \cdot (t_4 \cdot t_2))' \).

Let us notice that \( \text{ldl}(t') = \text{ldl}(t) - 1 \). According to recurrence hypothesis, \( \text{eval}_{(I,r)}((t_3 \cdot (t_4 \cdot t_2))') = \text{eval}_{(I,r)}(t_1) \cdot \text{eval}_{(I,r)}((t_3 \cdot (t_4 \cdot t_2)))' \). Hence \( \text{eval}_{(I,r)}(t) = \text{eval}_{(I,r)}(t_1) \cdot \text{eval}_{(I,r)}(t_2') \).

Let \( \mathcal{E} = (\Sigma, \Gamma, P, F) \) be an expression environment. We denote by \( F(\Gamma)' \) the set of normalized terms in \( F(\Gamma) \).

**Definition 27 (Normalized Formula).** Let \( \mathcal{E} = (\Sigma, \Gamma, P, F) \) be an expression environment. Let \( \phi \) be a boolean formula in \( P(F(\Gamma)) \). The formula \( \phi \) is said to be normalized if any of the terms appearing in it is normalized (i.e. if it belongs to \( P(F(\Gamma)') \)).

**Definition 28 (Formula Normalization).** Let \( \mathcal{E} = (\Sigma, \Gamma, P, F) \) be an expression environment. The normalization of a boolean formula \( \phi \) in \( P(F(\Gamma)) \) is the transformation \( \phi' \) inductively defined as follows:

\[
\begin{align*}
(o(\phi_1, \ldots, \phi_k))' &= o(\phi_1', \ldots, \phi_k') \\
(P(t_1, \ldots, t_k))' &= P(t_1', \ldots, t_k'),
\end{align*}
\]

where \( k \) is any integer, \( P \) is any predicate symbol in \( P_k \), \( t_1, \ldots, t_k \) are any \( k \) elements in \( F(X) \), \( o \) is any \( k \)-ary boolean operator associated with a mapping \( o' \) from \( \{0,1\}^k \) to \( \{0,1\} \) and \( \phi_1, \ldots, \phi_k \) are any \( k \) boolean formulae over \( (P, X) \).

**Proposition 10.** Let \( \mathcal{E} = (\Sigma, \Gamma, P, F) \) be an expression environment. Let \( \phi \) be a boolean formula in \( P(F(\Gamma)) \). Then:

\( \phi' \) is a normalized formula.

**Proof.** By induction over the structure of \( \phi \), direct corollary of Proposition as an inductive extension of the normalization.

**Proposition 11.** Let \( \mathcal{E} = (\Sigma, \Gamma, P, F) \) be an expression environment. Let \( \phi \) be a boolean formula in \( P(F(\Gamma)) \). Let \( I \) be an interpretation in \( \text{Int}(\mathcal{E}) \) and \( r \) be a realization in \( \text{Real}_r(I) \). Then:

\( \text{eval}_{(I,r)}(\phi) = \text{eval}_{(I,r)}(\phi') \).

**Proof.** By induction over the structure of \( \phi \), direct corollary of Proposition as an inductive extension of the normalization.

**Example 7.** Let us consider the formula \( \phi_2 = \langle(ab, x), abx \rangle \land \neg\langle(ab, \cdot(ab, bx)) \rangle \) of Example Considering the catenation as right-associative, its normalized form is the formula \( \phi_2' = \langle(ab, abx) \rangle \land \neg\langle(ab, abx) \rangle \), that is a classical contradiction.

Let us now show how to compute an injection from a set of normalized terms.

**Definition 29 (Left, Right and Middle Word).** A word is a left word (resp. right word, middle word) of a term \( t \) in \( F(\Gamma) \) if it belongs to the set \( \text{LeftWord}(t) \) (resp. \( \text{RightWord}(t), \text{MiddleWords}(t) \)) computed as follows:

\[
\begin{align*}
\text{LeftWord}(x) &= \{x \mid x \in \Sigma \cup \{\varepsilon\}\}, \\
\text{LeftWord}(f(t_1, \ldots, t_k)) &= \emptyset, \\
\text{LeftWord}((t_1, t_2)) &= \begin{cases} \\
\text{LeftWord}(t_1), & \text{if } t_1 \notin \{\varepsilon\} \cup \Sigma \cup \emptyset \lor \text{LeftWord}(t_2) = \emptyset, \\
\text{LeftWord}(t_1) \cdot \text{LeftWord}(t_2), & \text{otherwise,}
\end{cases}
\end{align*}
\]
RightWord((t_1, t_2)) = \{ \text{RightWord}(t_1), \text{RightWord}(t_2) \} \setminus \{ \text{RightWord}(t_1) \cup \text{RightWord}(t_2) \} \\
MiddleWords((t_1, t_2)) = \{ \text{MiddleWords}(t_1), \text{MiddleWords}(t_2) \} \\
where x \text{ is an element in } F_0 \cup \Gamma, f \text{ is a function symbol in } F_k \setminus \{ \cdot \} \text{ and } t_1, \ldots, t_k \text{ are any } k \text{ terms in } F(\Gamma).

A word u is a factor of the term t if it is a factor of v where v \in \text{MiddleWords}(t).

Example 8. Let us illustrate the notion of factor:

– The factors of f(a, g(a, baxc)) are \{a, b, ba, c\}.
– The factors of a, g(a, baxc) are \{a, ba, c\}.
– The factors of f(a, a, baxc) are \{a, ab, aba, b, ba, c\}.

\[ \square \]

Definition 30 (Term Function). Let Term be the function from \( \Sigma^* \) to \( (\Sigma \cup \{ \cdot, \varepsilon \}) (\emptyset) \) inductively defined for any word w as follows:

\[ \text{Term}(w) = \left\{ \begin{array}{ll}
w & \text{if } w \in \Sigma \cup \{ \varepsilon \}, \\
(a, \text{Term}(w')) & \text{if } w = aw' \wedge a \in \Sigma \wedge w' \in \Sigma^*.
\end{array} \right. \]

Definition 31 (root Function). Let root be the function from \( F(\Gamma) \) to \( F \cup \Gamma \) defined for any term t as follows:

\[ \text{root}(t) = \left\{ \begin{array}{ll}
t & \text{if } t \in \Gamma \cup F_0, \\
f & \text{if } t = f(t_1, \ldots, t_k) \text{ with } f \in F_k.
\end{array} \right. \]

Lemma 4. Let \( \mathcal{E} = (\Sigma, \Gamma, P, F) \) be an expression environment such that \( \text{Card}(\Sigma) \geq 2 \). Let T be a finite subset of \( F(\Gamma)' \). Then there exists a word w in \( \Sigma^* \) such that for any term t in \( F(\Gamma)' \); for any two distinct terms \( t_1 \) and \( t_2 \) in \( F(\Gamma)' \), it holds:

\[ (t_{1, t} \text{–Term}(w))' \neq (t_{2, t} \text{–Term}(w))'. \]

Proof. Let \( w = ab^pa \) be such that \( p \) is the smallest integer such that any factor \( b^q \) of a term of T satisfies \( q < p \). Let us set \( s_1 = (t_{1, t} \text{–Term}(w)) \) and \( s_2 = (t_{1, t} \text{–Term}(w)) \).

If \( t \) is neither a subterm of \( t_1 \) nor of \( t_2 \), then \( s_1 = s_1, t_2 = s_2 = s_2 \) and thus \( s_1' \neq s_2' \).

Let \( t \) be a subterm of \( t_1 \) but not of \( t_2 \). There exists a factor \( ab^pa \) of \( s_1' \) whereas any factor \( b^q \) of \( s_2' \) (that is a factor of \( t_2 \)) satisfies \( q < p \). Then \( s_1' \neq s_2' \).

Let us suppose that \( t \) is a subterm of \( t_1 \) and of \( t_2 \).

1. Let us suppose that \( t_1 = t \). Then root(s_1') = \varepsilon \) since \( s_1' = \text{Term}(ab^pa) \).

(a) If \( t_2 = y \neq t \), then root(s_2') = s_2' = y. Hence \( s_1' \neq s_2' \).

(b) Let us suppose that \( t_2 = f(t_2, \ldots, t_k) \) with \( F_k \setminus \{ \cdot \} \). Then root(s_2') = f. Hence \( s_1' \neq s_2' \).

(c) Let us suppose that \( t_2 = (t_{2, t} \text{–Term}(w)) \). Since \( t_2 \) is normalized, then root(t_2) = \varepsilon .

i. If \( t_{21} = t \), then \( (t_{22, t})' = (a, \cdot (b, \ldots, b, \cdot (a, (t_{22, t})') \ldots)) \). Since \( t_{22} \neq \varepsilon \), then \( t_{22, t} \neq \varepsilon \) and then \( s_1' \neq s_2' \).

ii. If \( t_{21} \neq t \), then \( s_1' = (t_{22, t})', (t_{22, t})' \).

A. If \( t_{21} \neq a \) then \( s_1' \neq s_2' \).

25
B. Let us suppose that \( t_{21} = a \). Either \( (t_{22})' \neq \text{Term}(b^p a) \) and then \( s'_1 \neq s'_2 \) or \( (t_{22})' = \text{Term}(b^p a) \), and \( t_2 = \text{Term}(a b^p a) \) (Contradiction with the definition of \( p \)).

2. If \( f = (t_{11}, \ldots, t_{1k}) \) with \( F_k \setminus \{\cdot\} \). Then \( (t_{1i}w)^{'} = f((t_{11t_iw})^{'}\ldots, (t_{1kt_iw})^{'}\ldots) \) and root\( (s'_1) = f \).
   (a) Let us suppose that \( t_2 = t \). See case 1B.
   (b) Let us suppose that \( t_2 = y \in \Gamma \neq t \). Then root\( (s'_2) = s'_2 = y \). Hence \( s'_1 \neq s'_2 \).
   (c) Let us suppose that \( t_2 = g(t_{21}, \ldots, t_{2k}) \) with \( g \in F_k \) and \( g \neq f \). Then root\( (s'_2) = g \) and then \( s'_1 \neq s'_2 \).
   (d) Let us suppose that \( t_2 = f(t_{21}, \ldots, t_{2k}) \). Then \( (t_{2t_iw})^{'} = f((t_{21t_iw})^{'}\ldots, (t_{2kt_iw})^{'}\ldots) \). Since \( t_1 \neq t_2 \), there exists \( j \) in \( \{1, \ldots, k\} \) such that \( t_{1j} \neq t_{2j} \). According to induction hypothesis, \( (t_{1jt_iw})^{'} \neq (t_{2jt_iw})^{'} \), it holds that \( s'_1 \neq s'_2 \).

3. Let us suppose that \( t_1 = (t_{11}, t_{12}) \). Then root\( (t_{11}) \neq \cdot \).
   (a) If \( t_2 = t \), see case 1C.
   (b) If \( t_2 = y \in \Gamma \neq t \), then root\( (s'_2) = s'_2 = y \). Hence \( s'_1 \neq s'_2 \).
   (c) If \( t_2 = f(t_{21}, \ldots, t_{2k}) \) with \( f \in F_k \setminus \{\cdot\} \), see case 2C.
   (d) Let us suppose that \( t_2 = (t_{21}, t_{22}) \). Consequently root\( (t_{21}) \neq \cdot \).

   i. If \( t_{11} = t \) then \( s'_1 = (a, (b, \ldots, (b, (a, (t_{12t_iw})^{'}))\ldots)) \).
   A. If \( t_{21} = t \), then \( t_{12} \neq t_{22} \). According to induction hypothesis, \( (t_{12t_iw})^{'} \neq (t_{22t_iw})^{'} \). Since \( s'_2 = (a, (b, \ldots, (b, (a, (t_{22t_iw})^{'}))\ldots)), s'_1 \neq s'_2 \).
   B. If \( t_{21} = a \), then \( s'_1 = (a, (t_{22t_iw})^{'})) \). Either \( (t_{22t_iw})^{'} \) does not admit \( b^p \) as a prefix of a left word and then \( s'_1 \neq s'_2 \) or it does and then \( t_2 \) admits \( b^p \) as a factor (contradiction with the definition of \( p \)).
   C. If \( t_{21} = f(t_{211}, \ldots, t_{21k}) \), then the respective roots of the leftmost subterm of \( s'_1 \) and \( s'_2 \) are distinct. Hence \( s'_1 \neq s'_2 \).

   ii. Suppose that \( t_{11} = a \).
   A. If \( t_{21} = t \), see case 1D(iiB).
   B. If \( t_{21} = a \), then since \( t_1 \neq t_2 \), it holds that \( t_{12} \neq t_{22} \). By induction hypothesis, \( (t_{12t_iw})^{'} \neq (t_{22t_iw})^{'} \). Finally, since \( s'_2 = (a, (t_{22t_iw})^{'})), s'_1 \neq s'_2 \).
   C. If \( t_{21} = f(t_{211}, \ldots, t_{21k}) \), then root\( (s'_2) = f \neq \cdot = \text{root}(s'_1) \). Hence \( s'_1 \neq s'_2 \).

   iii. Suppose that \( t_{11} = f(t_{111}, \ldots, t_{11k}) \). Then \( s'_1 = (u_1, u_2) \) with root\( (u_1) = f \).
   A. If \( t_{21} = t \), see case 1D(iiiC).
   B. If \( t_{21} = y \in \Gamma \neq t \) or \( t_{21} = g(t_{211}, \ldots, t_{211}) \) with \( g \in F_k \), the respective roots of the leftmost subterm of \( s'_1 \) and \( s'_2 \) are distinct. Hence \( s'_1 \neq s'_2 \).
   C. If \( t_{21} = f(t_{211}, \ldots, t_{21k}) \), then \( s'_2 = (v_1, v_2) \) with root\( (v_1) = f \). Two cases can occur: either \( t_{12} \neq t_{22} \) or there exists \( j \) in \( \{1, \ldots, k\} \) such that \( t_{11j} \neq t_{21j} \). In the first (resp. second) case, it holds by induction that \( u_2 \neq v_2 \) (resp. \( u_1 \neq v_1 \)). Consequently, \( s'_1 \neq s'_2 \).

Definition 32 (Term Index). Let us define the index \( \text{Ind}(T) \) as the integer computed as follows:
\[
\text{Ind}(T) = \bigcup_{t \in T} \text{Ind}(T),
\]
where for any term \( t \),
\[
\text{Ind}(T) = \begin{cases} 0 & \text{if } t \in \Sigma \cup \{\varepsilon\}, \\ 1 & \text{if } t \in \Gamma' \cup F_0 \setminus (\Sigma \cup \{\varepsilon\}), \\ 1 + \text{Ind}(t_1) + \cdots + \text{Ind}(t_k) & \text{if } t = f(t_1, \ldots, t_k) \land f \in F_k \setminus \{\cdot\}, \\ \text{Ind}(t_1) + \text{Ind}(t_2) & \text{if } t = t_1 \cdot t_2. \\
\end{cases}
\]
Let us define for any two terms \( t_1 \) and \( t_2 \) the set \( T_{t_1 \leftarrow t_2} \) as follows:
\[
T_{t_1 \leftarrow t_2} = \bigcup_{t \in T} \{t': t' \in T_{t_1 \leftarrow t_2}\}
\]
where for any term \( t \),
\[
t_{t_1 \leftarrow t_2} = \begin{cases} t_2 & \text{if } t = t_1, \\ f(u_{1t_1 \leftarrow t_2}, \ldots, u_{kt_1 \leftarrow t_2}) & \text{if } t = f(u_1, \ldots, u_k) \land f \in F_k, \\ t & \text{otherwise}. \\
\end{cases}
\]
Let us define for any term \( t \) the depth \( D(t) \) as follows:
\[
d(t) = \begin{cases} 1 & \text{if } t \in F_0 \cup \Gamma, \\ 1 + \text{max}(d(t_1, \ldots, t_k)) & \text{if } t = f(t_1, \ldots, t_k) \land f \in F_k. \\
\end{cases}
\]
Proposition 12. Let $\mathcal{E} = (\Sigma, \Gamma, P, F)$ be an expression environment such that $\text{Card}(\Sigma) \geq 2$. Let $T$ be a finite subset of $F(\Gamma)'$. There exist $I$ an interpretation in $\text{Int}(\mathcal{E})$ and $r$ a realization in $\text{Real}_{F}(I)$ such that the function $\text{eval}_{I,r}$ is an injection of $T$ in $\Sigma^*$.

Proof. By recurrence over $\text{Ind}(T)$.

(I) If $\text{Ind}(T) = 0$ then $T \subset (\Sigma \cup \{\varepsilon\})(\emptyset)$. Let us show that for any two distinct terms $t_1$ and $t_2$ in $T$, for any interpretation $I$ and for any realization $r$, it holds that $\text{eval}_{I,r}(t_1) \neq \text{eval}_{I,r}(t_2)$. By recurrence over $d(t_1)$. Let $I$ be any interpretation and $r$ be any realization.

(a) Let us suppose that $d(t_1) = 1$. Then $t_1 \in \Sigma \cup \{\varepsilon\}$. Consequently, $|\text{eval}_{I,r}(t_1)| \leq 1$.

(i) If $d(t_2) = 1$, since $t_1 \neq t_2$, then $t_2 \in \Sigma \cup \{\varepsilon\} \setminus \{t_1\}$. Hence, $\text{eval}_{I,r}(t_1) \neq \text{eval}_{I,r}(t_2)$.

(ii) If $d(t_2) \neq 1$, then $t_2 = (a, s_2)$ with $a \in \Sigma$ and $s_2 \neq \varepsilon$. Hence $|\text{eval}_{I,r}(t_2)| > 1$. As a consequence, $|\text{eval}_{I,r}(t_1)| \neq |\text{eval}_{I,r}(t_2)|$ and then $\text{eval}_{I,r}(t_1) \neq \text{eval}_{I,r}(t_2)$.

(b) Let us suppose that $d(t_1) > 1$. Then $t_1 = (a, s_1)$ with $a \in \Sigma$.

(i) If $d(t_2) = 1$, then it is symmetrically equivalent to (Ia).

(ii) Let us suppose that $d(t_2) \neq 1$. Then $t_2 = (b, s_2)$ with $b \in \Sigma$. If $a \neq b$, then $a \cdot \text{eval}_{I,r}(s_1) \neq b \cdot \text{eval}_{I,r}(s_2)$ and then $\text{eval}_{I,r}(t_1) \neq \text{eval}_{I,r}(t_2)$. Otherwise, it holds $s_1 \neq s_2$. According to recurrence hypothesis, $\text{eval}_{I,r}(s_1) \neq \text{eval}_{I,r}(s_2)$ and consequently $a \cdot \text{eval}_{I,r}(s_1) \neq a \cdot \text{eval}_{I,r}(s_2)$. Consequently, $\text{eval}_{I,r}(t_1) \neq \text{eval}_{I,r}(t_2)$.

(II) Let us suppose that $\text{Ind}(T) \neq 0$.

(a) Let $x$ be a symbol in $\Gamma$ such that $x$ is a subterm of a term in $T$.

(i) According to Lemma 4, there exists $w \in \Sigma^*$ such that for any two terms $t_1$ and $t_2$ in $F(\Gamma)'$, it holds: $(t_1 x_{t_1} t_2)' \neq (t_{2 x_{t_2}})'$.

(ii) It holds by recurrence hypothesis that there exists $I$ an interpretation in $\text{Int}(\mathcal{E})$ and $r$ a realization in $\text{Real}_{F}(I)$ such that the function $\text{eval}_{I,r}$ is an injection of $T_{x_{t_1} t_2}$ in $\Sigma^*$. Let us consider the realization $r'$ defined for any symbol $y$ in $\Gamma$ as follows:

$$r'(y) = \begin{cases} w & \text{if } x = y, \\ r(y) & \text{otherwise.} \end{cases}$$

Let us show that $\text{eval}_{I,r'}$ is an injection of $T$ in $\Sigma^*$.

Let $t_1$ and $t_2$ be two terms in $T$. According to (II a i), $s_1' = (t_{1 x_{t_1}} t_2)' \neq (t_{2 x_{t_2}})'$. By definition of $r'$, $\text{eval}_{I,r'}(s_1') \neq \text{eval}_{I,r'}(s_2')$. Since by construction of $r'$, $\text{eval}_{I,r'}(t_1) = \text{eval}_{I,r'}(s_1')$ and since $\text{eval}_{I,r'}(t_2) = \text{eval}_{I,r'}(s_2')$, it holds that $\text{eval}_{I,r'}(t_1) \neq \text{eval}_{I,r'}(t_2)$.

(b) Let us suppose that there is no subterm of a term in $T$ that belongs to $\Gamma$. Let $t = f(t_1, \ldots, t_k)$ be a subterm in a term in $T$ such that $t_1, \ldots, t_k$ are $k$ terms in $(\Sigma \cup \{\varepsilon\})(\emptyset)$.

(i) According to Lemma 4, there exists $w \in \Sigma^*$ such that for any two terms $t_1$ and $t_2$ in $F(\Gamma)'$, it holds: $(t_{1 x_{t_1}} t_2)' \neq (t_{2 x_{t_2}})'$.

(ii) It holds by recurrence hypothesis that there exists $I = (\Sigma^*, \mathfrak{F})$ an interpretation in $\text{Int}(\mathcal{E})$ and $r$ a realization in $\text{Real}_{F}(I)$ such that the function $\text{eval}_{I,r}$ is an injection of $T_{t_{1 x_{t_1}} t_2}$ in $\Sigma^*$. Let us denote by $w_j$ the word $\text{eval}_{I,r}(t_j)$ for any integer $j$ in $\{1, \ldots, k\}$. Let us consider the interpretation $I' = (\Sigma^*, \mathfrak{F}')$ defined as follows:

1. for any predicate symbol $P$ in $\mathcal{P}$, $\mathfrak{F}(P) = \mathfrak{F}'(P)$,
2. for any function symbol $g$ in $\mathcal{F} \setminus \{f\}$, $\mathfrak{F}(g) = \mathfrak{F}'(g)$,
3. for any word $w_1, \ldots, w_k$ in $\Sigma^*$:
   \[ w_1, \ldots, w_k, u_{k+1} \in \mathfrak{F}'(f) \Leftrightarrow u_1, \ldots, u_k, u_{k+1} \in \mathfrak{F}(f) \wedge (u_1, \ldots, u_k) \neq (w_1, \ldots, w_k) ,\]
4. $w_1, \ldots, w_k, w \in \mathfrak{F}'(f)$

Let us show that $\text{eval}_{I',r}$ is an injection of $T$ in $\Sigma^*$.

Let $t_1$ and $t_2$ be two terms in $T$. According to (II b i), $s_1' = (t_{1 x_{t_1}} t_2)' \neq (t_{2 x_{t_2}})'$. By definition of $I'$, $\text{eval}_{I',r}(s_1') \neq \text{eval}_{I',r}(s_2')$. Since by construction of $I'$, $\text{eval}_{I',r}(t_1) = \text{eval}_{I',r}(s_1')$ and since $\text{eval}_{I',r}(t_2) = \text{eval}_{I',r}(s_2')$, it holds that $\text{eval}_{I',r}(t_1) \neq \text{eval}_{I',r}(t_2)$.

Example 9. Let us consider the terms $t_1 = (a, g(a, baxc))$ and $v_1 = f(a, (a, baxc))$ and their factors (Example 9).

- The factors of $t_1$ are $F_{v_1} = \{a, b, ba, c\}$.
The factors of \( v_1 \) are \( F_{v_2} = \{a, ab, aba, ba, bc\} \).

Consider the word \( w = ab^2a \) that is not in \( F_{v_1} \cup F_{v_2} \). Consider a realization \( r \) associating \( w_1 \) with \( x \). Let us substitute \( x \) with \( w_1 \) in \( t_1 \) and \( v_1 \):

\[
t_2 = t_1[\cdot\cdot\cdot]_{w_1} = \cdot(a, g(a, baab^2ac)), v_2 = v_1[\cdot\cdot\cdot]_{w_1} = f(a, abab^2ac).
\]

The word \( w_2 = ab^4a \) is neither a factor of \( t_2 \) nor of \( v_2 \). Consider an interpretation \( I = (\Sigma^*, \mathfrak{F}) \) where \( (a, abab^2ac, ab^4a) \in \mathfrak{F}(f) \). Let us substitute \( f(a, abab^2ac) \) with \( w_2 \) in \( t_2 \) and \( v_2 \):

\[
t_3 = t_2[\cdot\cdot\cdot]_{f(a, abab^2ac)}_{w_2} = \cdot(a, g(a, baab^2ac)), v_3 = v_2[\cdot\cdot\cdot]_{f(a, abab^2ac)}_{w_2} = ab^3a.
\]

The word \( w_3 = ab^4a \) is neither a factor of \( t_3 \) nor of \( v_3 \). Consider that the interpretation \( I = (\Sigma^*, \mathfrak{F}) \) satisfies \( (a, baab^2ac, ab^4a) \in \mathfrak{F}(g) \). Let us substitute \( g(a, baab^2ac) \) with \( w_3 \) in \( t_3 \) and \( v_3 \):

\[
t_4 = t_3[\cdot\cdot\cdot]_{g(a, baab^2ac)}_{w_3} = aab^4a, v_4 = v_3[\cdot\cdot\cdot]_{g(a, baab^2ac)}_{w_3} = ab^3a.
\]

Hence, since \( \text{eval}_{I,r}(t_1) = aab^4a \) and \( \text{eval}_{I,r}(v_1) = ab^3a \) are distinct, the function \( \text{eval}_{I,r} \) is an injection of \( \{t_1, v_1\} \) in \( \Sigma^* \).

As a conclusion, the following corollary holds from Proposition 7, Proposition 11, and Corollary 6.

**Corollary 7.** Let \( E = (\Sigma, \Gamma, P, F) \) be an expression environment such that \( \text{Card}(\Sigma) \geq 2 \). Let \( \phi \) be a boolean formula in \( F(\Gamma) \). Then the three following conditions are equivalent:

- \( \phi \) is satisfiable,
- \( \text{T}(\phi') \) is satisfiable.

**Corollary 8.** Let \( E = (\Sigma, \Gamma, P, F) \) be an expression environment and let \( E \) be a constrained expression over \( E \). Then the boolean Null\((E)\) can be computed.

**Corollary 9.** Let \( E = (\Sigma, \Gamma, P, F) \) be an expression environment, \( E \) be a constrained expression over \( E \) and \( w \) be a word in \( \Sigma^* \). The membership test of \( w \) in \( L(E) \) is decidable.

## 6 Conclusion and Perspectives

In this paper, we have extended the expressive power of (not necessarily) regular expressions by the addition of two new operators involving the zeroth order boolean formulae leading to the notion of constrained expression. We have presented a method in order to solve the membership problem in the general case where the interpretation is not fixed and when the alphabet is not unary.

A perspective is to consider the case of unary alphabets by extending the normalization defined in Subsection 5.3 with the commutativity of the catenation; indeed, as far as a unary alphabet is considered, two words commute. Hence, any term has to be sorted according to an order (e.g., the lexicographic order). We conjecture that Proposition 11 still holds for unary case, considering the word \( a^p \) with \( p > p' \) for any \( a^{p'} \) in a term in \( T \) instead of \( ab^p a \) (see proof of Lemma 3).

We have also shown that the membership problem is generally undecidable when the interpretation is fixed. However, we can express a sufficient condition for the membership test to be decidable: indeed, whenever the interpretation \( I \) is fixed, if it can be decided if a formula \( \phi \) is satisfiable (e.g., if there exists a realization \( r \) such that \( \text{eval}_{I,r}(\phi) = 1 \)), then (and trivially) the membership problem can be solved.

### References

1. R. Almeida, S. Broda, and N. Moreira. Deciding KAT and Hoare logic with derivatives. In M. Faella and A. Murano, editors, *GandALF*, volume 96 of *EPTCS*, pages 127–140, 2012.
2. V. Antimirov. Partial derivatives of regular expressions and finite automaton constructions. *Theoret. Comput. Sci.*, 155:291–319, 1996.
3. J. A. Brzozowski. Derivatives of regular expressions. *J. Assoc. Comput. Mach.*, 11(4):481–494, 1964.
4. J. A. Brzozowski. Regular-like expressions for some irregular languages. In *SWAT (FOCS)*, pages 278–286. IEEE Computer Society, 1968.
5. P. Caron, J.-M. Champarnaud, and L. Mignot. Partial derivatives of an extended regular expression. In A. Horia Dediu, S. Inenaga, and C. Martin-Vide, editors, *LATA*, volume 6638 of *Lecture Notes in Computer Science*, pages 179–191. Springer, 2011.
6. P. Caron, J.-M. Champarnaud, and L. Mignot. Multi-tilde-bar derivatives. In N. Moreira and R. Reis, editors, CIAA, volume 7381 of Lecture Notes in Computer Science, pages 321–328. Springer, 2012.

7. P. Caron, J.-M. Champarnaud, and L. Mignot. A general framework for the derivation of regular expressions. RAIRO Inform. Théor. Appl., Special Issue of Journées Montoises 2012, To appear, 2014.

8. J.-M. Champarnaud, J.-P. Dubernard, H. Jeanne, and L. Mignot. Two-sided derivatives for regular expressions and for hairpin expressions. In A. Horia Dediu, C. Martín-Vide, and B. Truthe, editors, LATA, volume 7810 of Lecture Notes in Computer Science, pages 202–213. Springer, 2013.

9. J.-M. Champarnaud, H. Jeanne, and L. Mignot. Derivatives of approximate regular expressions. Discrete Mathematics & Theoretical Computer Science, 15(2):95–120, 2013.

10. N. Chomsky. Three models for the description of language. IRE Trans. on Information Theory, 2(3):113–124, 1956.

11. V. M. Glushkov. On a synthesis algorithm for abstract automata. Ukr. Matem. Zhurnal, 12(2):147–156, 1960. In Russian.

12. H. Gruber and M. Holzer. From finite automata to regular expressions and back—a summary on descriptional complexity. EPTCS: AFL14, 51:25–48, 2014.

13. L. Ilie and S. Yu. Follow automata. Inf. Comput., 186(1):140–162, 2003.

14. Yu. V. Matiyasevich. Hilbert’s Tenth Problem. MIT Press Series in the Foundations of Computing, MIT Press, Cambridge, Massachusetts, 1993. With a foreword by Martin Davis.

15. R. F. McNaughton and H. Yamada. Regular expressions and state graphs for automata. IEEE Transactions on Electronic Computers, 9:39–57, March 1960.

16. M. Might, D. Darais, and D. Spiewak. Parsing with derivatives: a functional pearl. In M. M. T. Chakravarty, Z. Hu, and O. Danvy, editors, ICFP, pages 189–195. ACM, 2011.

17. J. M. Sempere. On a class of regular-like expressions for linear languages. Journal of Automata, Languages and Combinatorics, 5(3):343–354, 2000.

18. T. Skolem. The foundations of elementary arithmetic established by means of the recursive mode of thought, without the use of apparent variables ranging over infinite domains. In J.v. Heijenoort, editor, From Frege to Gödel, pages 302–333. Harvard, Iuniverse, 1967.

19. K. Thompson. Regular expression search algorithm. Comm. ACM, 11(6):419–422, 1968.