Regular expression containment as a proof search problem

Vladimir Komendantsky

Regular expression containment as a proof search problem

Vladimir Komendantsky

School of Computer Science
University of St Andrews
St Andrews, KY16 9SX, UK
vk10@st-andrews.ac.uk

Abstract. Regular expression containment has recently enjoyed a renewed interest as a type-theoretic problem. By viewing regular expressions as types, it is possible to view containments as functions that coerce one regular expression into another. Containment of a given regular expression in another one is therefore a coercion inference problem. It is not straightforward how to apply here generic proof-theoretic approaches such as those based on the MALL with the least fixpoint. We propose an alternative approach, still a purely syntactic one, to decide containment in reasonable time. We employ the partial derivative construction that, for a regular expression E, yields a syntactic NFA recognising the language of E; and, for a containment $E \leq F$, it yields a simulation of E by F, with the empty simulation representing the absence of coercion. We are currently working on a proof of completeness of this optimised coercion search algorithm.

1 Introduction

Our general idea is that termination of proof search of containment can be proved by a method involving our finitary coinduction principle for containment (Theorem 2) defined in terms of Mirkin prebase [15].

We provide a system whose proof derivations *characterise* regular expression containments in a similar sense in which, for example, derivations in the intuitionistic natural deduction calculus for implication characterise intuitionistic implicational formulas. Let us illustrate the analogy by recalling the introduction rule for implication:

$$\frac{\begin{array}{c}[p]^{\psi} \\ \vdots \\ q\end{array}}{p \to q}$$

and comparing it with the following fixed point rule where $o(E)$ is the *constant part* of the regular expression E, namely, 1 if E is nullable, and 0 otherwise:

$$\frac{o(E) \leq o(F) \quad \begin{array}{c}[E \leq F]^{\psi} \\ \vdots \\ E'_0 \leq F'_0\end{array} \quad \ldots \quad \begin{array}{c}[E \leq F]^{\psi} \\ \vdots \\ E'_k \leq F'_k\end{array}}{E \leq F}\ \mathsf{FIX}^{\psi}$$

In each of the two cases, rule applications labelled with ψ discard assumptions having the same label. It turns out that the rule FIX^{ψ} is an introduction rule by behaviour rather than an elimination rule, which relates it directly to a sequent-style system. This is reasonable despite multiple premises of the fixed point rule. Indeed, in a suitable inference system, the conclusion can be annotated with a term bound by a least fixed point operator.

In this paper, a notion of natural deduction is provided for terminating proof search of regular expression containment. The cornerstone of the natural-deduction style system we call **NR** is a special computational fixed point rule FIX^{ψ} that, if interpreted premises first, computes a simulation between two regular expressions E and F, denoted $E \leq F$, given simulations between a certain combination of *partial derivatives* of these regular expressions, and between the constant parts of these regular expressions. This rule is a fixed point rule because every immediate subtree, whose root is a containment of two partial derivatives, has a premiss $E \leq F$ which is discarded as soon as the fixed point rule is applied. The fixed point rule, appropriately extended, provides an efficient proof search strategy for deciding regular expression containment.

Motivation. Efficient and complete proof search of regular expression containment is currently an open problem [11]. The minimal syntactic NFA construction of prebase used in this paper provides the necessary efficiency and we are working on a completeness proof by application of the coinduction principle for prebases (Corollary 2).

Outline. In Section 2, we recall basic notions of formal language theory such as non-deterministic automata or regular expressions. In Section 3, we recall the notion of Mirkin prebase [15] first, as an abstract inductive structure satisfying certain semantic properties, and second, as a syntactic construction parametrised by a given regular expression. Already for the abstract structure of prebase, we provide two very useful *finitary coinduction proof principles*: one, for equivalence of regular expressions (stated in terms of bisimulation), and another, for containment of regular expressions (stated in terms of simulation). In fact, the former subsumes the latter. In Section 4, we define the labelled natural-deduction style calculus **NR**, with context-free rules, and state that it is a sound and complete containment inference system with respect to the standard language semantics. We give conclusions in Section 6.

2 Preliminary notions

Let $A = \{a_0, \ldots, a_{n-1}\}$ be a finite set of symbols. In the paper, we assume that A is a given fixed alphabet and denote the i-th symbol of A by a_i. We employ coalgebraic definitions of automata which are quite well-established since [16] for our work with coinductive relations of simulation and bisimulation. A *non-deterministic automaton* with input alphabet A is a triple $\mathcal{Q} = \langle Q, o, t\rangle$ consisting of a possibly infinite set Q of states, an *output function* $o : Q \to 2$ and a *transition function* $t : Q \to (A \to 2^Q)$ where $2 = \{0, 1\}$, for regular expressions 0 and 1, is a set isomorphic to the set of boolean values, and 2^Q is the set of subsets of Q. The output function o indicates whether a state q is accepting, $o(q) = 1$, or not, $o(q) = 0$. The transition function t assigns a subset of states Q' to a state q after reading an input symbol a, which is denoted $t(q)(a) = Q'$.

A *simulation* between two automata $\mathcal{Q} = \langle Q, o, t\rangle$ and $\mathcal{Q}' = \langle Q', o', t'\rangle$ is a relation $R \subseteq Q \times Q'$ such that, for all $\langle q, q'\rangle \in R$ and $a \in A$, $o(q) \leq o'(q')$ and for every $r \in t(q)(a)$ there exists $r' \in t'(q')(a)$ such that $\langle r, r'\rangle \in R$. Existence of a simulation between $\mathcal{Q}$ and $\mathcal{Q}'$ means that $\mathcal{Q}'$ accepts at least the language accepted by $\mathcal{Q}$, since acceptance is preserved under simulation. We write $q \leq_{fin} q'$ if there is a finite simulation R such that $\langle q, q'\rangle \in R$. The relation $\leq_{fin}$ is the union of all finite simulations and therefore the greatest simulation, also called *finite similarity*. In the paper, infinite simulations do not arise because we only construct automata with finite sets of states.

A *bisimulation* between $\mathcal{Q} = \langle Q, o, t\rangle$ and $\mathcal{Q}' = \langle Q', o', t'\rangle$ is a relation $R \subseteq Q \times Q'$ such that, for all $\langle q, q'\rangle \in R$ and $a \in A$, $o(q) = o'(q')$ and for every $r \in t(q)(a)$ there exists $r' \in t'(q')(a)$ such that $\langle r, r'\rangle \in R$. If there exists a finite bisimulation R such that $\langle q, q'\rangle \in R$, we write $q \sim_{fin} q'$. A bisimulation is clearly a simulation. The converse holds only in case the inverse of the simulation is also a simulation.

Regular expressions are generated by the following grammar:

$$E, F ::= 0 \mid 1 \mid a_i \mid E + F \mid E \times F \mid E^*$$

for $i \in [0, n-1]$, where $a_i \in A$. Following the tradition, the operation $\times$ has greater preference over $+$. If expressions are not parenthesised explicitly, parentheses are assumed with association to the left. Let $||E||$ denote the number of distinct alphabet symbols in the regular expression E, and let $o(E)$ be 1 if $\mathsf{nullable}(E)$ is true, and 0 otherwise, where $\mathsf{nullable}$ is defined by recursion on E as follows:

$$\begin{array}{lcl}
\mathsf{nullable}(0) & = & \mathsf{true} \\
\mathsf{nullable}(1) & = & \mathsf{false} \\
\mathsf{nullable}(a_i) & = & \mathsf{false} \\
\mathsf{nullable}(E_1 + E_2) & = & \mathsf{nullable}(E_1) \mid\mid \mathsf{nullable}(E_2) \\
\mathsf{nullable}(E_1 \times E_2) & = & \mathsf{nullable}(E_1) \;\&\&\; \mathsf{nullable}(E_2) \\
\mathsf{nullable}(E_1^*) & = & \mathsf{true}
\end{array}$$

The language denoted by a regular expression is defined by recursion on the regular expression as follows:

$$\begin{array}{lcllcl} L(0) & = & \emptyset & L(1) & = & \{\varepsilon\} \\ L(a_i) & = & \{a_i\} & L(E+F) & = & L(E) \cup L(F) \\ L(E{\times}F) & = & L(E) \cdot L(F) & L(E^*) & = & \bigcup\limits_{i \in \mathbb{N}} L^i \end{array}$$

where $\cup$ is the union of sets, $\cdot$ is the operation of language concatenation:

$$L \cdot M = \{w \mid w = uv \wedge u \in L \wedge v \in M\}$$

and L^i is the i-th power of the language L:

$$L^0 = 1 \qquad L^{i+1} = L \cdot L^i$$

For regular expressions E and F, by $E =_L F$ and $E \leq_L F$ we denote equality of the languages denoted by E and F and, respectively, containment of the language denoted by E in the language denoted by F.

3 Prebases

The goal of this section is, first, to give a definition of a generic prebase [15, 7, 2, 1], and then, define a construction of a prebase for a given regular expression. This way we single out *what* properties a prebase should have from *how* it is constructed.

We start with an algorithmic motivation. First let us define a *monomial* to be a regular expression whose main operation is not $+$, and a *polynomial* to be a finite, possibly empty sum of monomials. We say that a monomial E is in *head normal form* if it is of the kind $a_j \times F$ for some symbol a_j and a regular expression F. Suppose we are given a polynomial $E = E_1 + \cdots + E_k$. The polynomial E can be reduced to a polynomial with all monomials in head normal form by recursive application of the following rewrite rule schemes to every monomial:

$$\begin{array}{lcl} (F_1 \times \cdots \times F_l \times F_{l+1}) \times G & \mapsto & (F_1 \times \cdots \times F_l) \times (F_{l+1} \times G) \\ (F_1 + \cdots + F_l) \times G & \mapsto & F_1 \times G + \cdots + F_l \times G \\ F^* \times G & \mapsto & F \times (F^* \times G) + G \\ F^* & \mapsto & F \times F^* + 1 \end{array} \tag{1}$$

Then, the set of monomials can be rearranged using the fact that the operation $+$ is associative, commutative and idempotent expressed by the following *ACI equivalences*:

$$\begin{array}{lcl} F + (G + H) & =_L & (F + G) + H \\ F + G & =_L & G + H \\ F + F & =_L & F \end{array} \tag{2}$$

as well as using the following equivalence from the structure of additive monoid on regular languages on A:

$$F =_L F + 0 \tag{3}$$

Therefore E can be reduced to a polynomial

$$E^1 = a_{i_1} \times E^1_1 + \cdots + a_{i_{p_1}} \times E^1_{p_1} + o(E)$$

Then, we apply the same recursive algorithm to reduce $E^1_1, \ldots, E^1_{p_1}$, and so on, until the obtained polynomial is of the form

$$E^q = a_{j_1} \times E^q_1 + \cdots + a_{j_{p_q}} \times E^q_{p_q} + o(E^q)$$

where $E^q_1, \ldots, E^q_{p_q}$ are already defined, that is, these regular expressions appear in the set

$$E^1_1, \ldots, E^1_{p_1}, \ldots, E^{q-1}_1, \ldots, E^{q-1}_{p_{q-1}}$$

Termination of this recursive process is the goal of Theorem 3 in this section, proved by structural induction on E.

Example 1. Consider a concrete alphabet $A = \{a, b\}$ and a regular expression $E = (a \times (a \times b)^* \times a)^*$ over this alphabet. Using the four rewrite rule schemes (1), the ACI-equivalences (2) and the equivalence (3), we obtain the following system of equations:

$$\begin{aligned}
E &=_L a \times (a \times b)^* \times a \times E + 1 \\
&=_L a \times E_1 + o(E), \\
E_1 &= (a \times b)^* \times a \times E \\
&=_L (a \times b) \times E_1 + a \times E \\
&=_L a \times E_2 + a \times E + o(E_1), \\
E_2 &= b \times E_1 \\
&=_L b \times E_1 + o(E_2)
\end{aligned}$$

Thus the regular expression E is described in terms of its output and the regular expression E_1 which denotes a residual language that results from taking those words in the language of E that start from the symbol a and removing that occurrence of a. In the terminology of this section, the prebase of E consists of regular expressions E, E_1 and E_2.

Now we will define this informal algorithm more formally using the notion of prebase. Note the difference in the style of definition: We state the desired properties first and associate these to a prebase, and then we provide an inductive construction satisfying the given properties in the proof of Theorem 3. We call this style of definition *declarative* as opposed to the *algorithmic* style of the informal definition.

Recall that we defined n to be the number of symbols in the given alphabet A. A *prebase* $\mathcal{P}$ is a triple $\langle m, P, M \rangle$ with the following components:

- m is a positive natural number.
- P is an m-tuple of regular expressions $P = \langle E_0, \ldots, E_{m-1} \rangle$ such that the following semantic language equivalences hold for $i \in [0, m-1]$ and $j \in [0, n-1]$:

$$E_i =_L a_0 \times \big(\sum M_{i,0}\big) + \cdots + a_{n-1} \times \big(\sum M_{i,n-1}\big) + o(E_i) \tag{4}$$

  where $M_{i,j}$ is a finite subset of P, that is,

$$M_{i,j} = \bigcup_{k \in I_{i,j}} E_k, \quad I_{i,j} \subseteq [0, m-1]$$

  and the regular expression denoted $\sum M_{i,j}$ is the sum of elements of $M_{i,j}$. The sum is 0 if and only if $M_{i,j}$ is empty.
- M is the *matrix* $[M]_{i,j}$ whose cells are these individual sets $M_{i,j}$. We introduce matrices to spell out the straightforward connection between prebases and non-deterministic finite automata.

Every prebase $\mathcal{P} = \langle m, P, M \rangle$, where $P = \langle E_0, \ldots, E_{n-1} \rangle$, corresponds to the finite non-deterministic automaton $\langle P, o, M \rangle$, where o is defined on regular expressions, which we will also denote by $\mathcal{P}$ since prebases and their non-deterministic finite automata provide two views on one and the same class of systems. The following notation is handy in the automaton view. Assume $M_{i,\overline{\varepsilon}}$, denotes the set $M_{i,j}$, where ε denotes the empty word. For a non-empty word $w = a_j u \in A^*$, let $M_{i,\overline{w}}$ denote the union of all sets $M_{i',\overline{u}}$ such that $E_{i'} \in M_{i',j}$.

In the following two theorems, we establish coinduction proof principles in terms of prebases.

Theorem 1 (Finitary coinduction principle for equivalence). *For all prebases $\mathcal{P} = \langle m, P, M \rangle$ and $\mathcal{P}' = \langle m', P', M' \rangle$, where $P = \langle E_0, \ldots, E_{m-1} \rangle$ and $P' = \langle E'_0, \ldots, E'_{m'-1} \rangle$, and natural numbers $i < m$, $i' < m'$,*

$$E_i \sim_{fin} E'_{i'} \quad \textit{if and only if} \quad E_i =_L E'_{i'}$$

Proof. For the "if" direction, assume $E_i =_L E'_{i'}$. Then $\{\langle F, F' \rangle \mid F \in M_{i,\overline{w}}, F' \in M'_{i',\overline{w}} \text{ and } w \in A^*\}$ can be easily checked to be a bisimulation between E_i and $E'_{i'}$. Its finiteness follows by the definition of prebase matrix. For the "only if" direction, consider induction on the length of a word w such that $M_{i,\overline{w}} \sim_{fin} M'_{i',\overline{w}}$. The base case follows immediately. For the inductive step, we rewrite with the inductive hypothesis in the prebase equation (4) to yield the required conclusion. □

In fact, the above proof can be transformed to a proof of the weaker finitary principle below, by replacing bisimulation with simulation.

Theorem 2 (Finitary coinduction principle for containment). *For all prebases $\mathcal{P} = \langle m, P, M \rangle$ and $\mathcal{P}' = \langle m', P', M' \rangle$, where $P = \langle E_0, \ldots, E_{m-1} \rangle$ and $P' = \langle E'_0, \ldots, E'_{m'-1} \rangle$, and natural numbers $i < m$, $i' < m'$,*

$$E_i \leq_{fin} E'_{i'} \quad \textit{if and only if} \quad E_i \leq_L E'_{i'}$$

We have defined the generic notion of prebase. Now we will construct a prebase for a given regular expression. In Theorem 3, the prebase algorithm by structural induction on a regular expression is an optimised version of the one given by Mirkin in the main theorem of [15]. The prebase tuple is defined for a given regular expression, as well as the corresponding matrix. Both tuple and matrix are parametrised by the given regular expression.

Theorem 3. *For any given regular expression E, we can construct a prebase $\mathcal{P}(E) = \langle m, P(E), M(E)\rangle$ such that*

(i) $E \in P(E)$*, namely,* $E = E_0$*;*
(ii) cells of the matrix of $P(E)$ *may contain elements of* $\langle E_1, \ldots, E_{m-1}\rangle$ *only;*
(iii) $|P(E)| \leq ||E|| + 1$*.*

We provide a proof outline of Theorem 3 in Appendix A. Now we state two straightforward corollaries of Theorems 1 and 2.

Corollary 1 (of Theorem 1). *For any regular expressions E, F and their prebases with tuples $P(E) = \langle E = E_0, \ldots, E_{m_E-1}\rangle$ and $P(F) = \langle F = F_0, \ldots, F_{m_F-1}\rangle$*

$$E_0 \sim_{fin} F_0 \quad \textit{if and only if} \quad E =_L F \ .$$

Corollary 2 (of Theorem 2). *For any regular expressions E, F and their prebases with tuples $P(E) = \langle E = E_0, \ldots, E_{m_E-1}\rangle$ and $P(F) = \langle F = F_0, \ldots, F_{m_F-1}\rangle$*

$$E_0 \leq_{fin} F_0 \quad \textit{if and only if} \quad E \leq_L F \ .$$

Corollary 2 is directly applied to derive the inference rule schema FIX^{ψ}.

4 Labelled natural deduction proof system NR

The proof system introduced below is based on Kozen's Kleene algebra axiomatisation of regular expression equivalence [14]. Although Kozen's formal notion of containment $E \leq F$ is $E + F = F$, we have containment as primitive, which splits every equality rule of the kind $E = F$ of Kleene algebra in two, $E \leq F$ and $F \leq E$, which we denote by $E \leq\geq F$ in Fig. 1.

Assume a countably infinite set of *labels* Ψ which is disjoint, for the purpose of disambiguation, with the set of regular expressions over A. *Formulas* of **NR** are regular expression containments. *Assumptions* are Ψ-labelled formulas. *Axioms* and *rules* are those defined in Figures 1 and 2. Idempotent semiring axioms in Figure 1 can be read both ways, since their intended meaning is the greatest bisimulation on the l.h.s. and the r.h.s. regular expressions, that is, a two-sided simulation. In Fig. 2, the notation $M(E)_{q,j} \leq M(F)_{r,j}$ is a shorthand for a set of derivations: For every j and $E' \in M(E)_{q,j}$ there exists $F' \in M(F)_{r,j}$ such that $E' \leq F'$.

Derivations in **NR** are proof trees whose leaves are axioms or assumptions such that different formulas in assumptions are labelled by different labels. The root formula in a tree is the conclusion of the proof derived by inference rules

Idempotent semiring axioms (bi-directional schemes):

$$E + (F + G) \leq\geq (E + F) + G$$
$$E + F \leq\geq F + E$$
$$E + 0 \leq\geq E$$
$$E + E \leq\geq E$$
$$E{\times}(F{\times}G) \leq\geq (E{\times}F){\times}G$$
$$1{\times}E \leq\geq E$$
$$E{\times}1 \leq\geq E$$
$$E{\times}(F + G) \leq\geq (E{\times}F) + (E{\times}G)$$
$$(E + F){\times}G \leq\geq (E{\times}G) + (F{\times}G)$$
$$0{\times}E \leq\geq 0$$
$$E{\times}0 \leq\geq 0$$

Containment rules:

$$E \leq E \qquad \frac{E \leq F \qquad F \leq G}{E \leq G}$$
$$\frac{E \leq F \qquad G \leq H}{E + G \leq F + H} \qquad \frac{E \leq F \qquad G \leq H}{E{\times}G \leq F{\times}H}$$

Kleene Algebra rules:

$$1 + (E{\times}E^*) \leq E^*$$
$$1 + (E^*{\times}E) \leq E^*$$
$$\frac{E{\times}F \leq F}{E^*{\times}F \leq F} \qquad \frac{E{\times}F \leq E}{E{\times}F^* \leq E}$$

Fig. 1. Regular expression containment axiomatisation $\mathbf{KA}_{\leq}$

Computational rules:

$$\frac{o(P(E)_q) \leq o(P(F)_r) \qquad \begin{array}{c}(E \leq F)^{\psi}\\ \vdots \\ M(E)_{q,j} \leq M(F)_{r,j}\end{array}}{P(E)_q \leq P(F)_r}\ \mathsf{FIX}^{\psi}_{q,r}$$

$$\frac{o(E) \leq o(F) \qquad \begin{array}{c}[E \leq F]^{\psi}\\ \vdots \\ M(E)_{0,j} \leq M(F)_{0,j}\end{array}}{E \leq F}\ \mathsf{FIX}^{\psi}_{0,0}$$

(Plus all axioms and rules of $\mathbf{KA}_{\leq}$)

Fig. 2. System **NR** (includes $\mathbf{KA}_{\leq}$)

of **NR**. There are two kinds of occurrences of assumptions with label ψ in a proof tree: *open*, if the path from the occurrence in question to the root does not contain an application of $\mathsf{FIX}^{\psi}_{0,0}$ which closes this assumption, and *closed* otherwise. Note that an application of the rule scheme $\mathsf{FIX}^{\psi}_{q,r}$, where $q \neq 0$ and $r \neq 0$, does not close the assumption $(E \leq F)^{\psi}$. Applications of the rule $\mathsf{FIX}^{\psi}_{0,0}$ are required to close (discharge) at least one non-empty class of open occurrences of the same formula labelled by ψ. The computational interpretation of the rule $\mathsf{FIX}^{\psi}_{0,0}$ is the unfolding of the greatest finite simulation construction that is obtained thanks to Corollary 2, if all the rules in the right branches are instances of $\mathsf{FIX}^{\psi}_{q,r}$.

By $\vdash_{\mathbf{NR}} E \leq F$ we denote that $E \leq F$ is a *theorem* of **NR**, that is, there is in **NR** a closed derivation ∇ with no open assumptions, and with the conclusion $E \leq F$.

Theorem 4. *The system* **NR** *is sound and complete with respect to regular expression containment. That is, for any regular expressions* E, F*,*

$$\vdash_{\mathbf{NR}} E \leq F \quad \textit{if and only if} \quad E \leq_L F$$

Proof (Outline). For the "if" direction, assume $E \leq_L F$. By Corollary 2, we have $E \leq_{fin} F$, from which we can extract a closed derivation in **NR**. For the "only if" direction, assume ∇ is a closed derivation with conclusion $E \leq F$. By induction on the height of ∇, a finite simulation $E \leq_{fin} F$ can be obtained as $\{E' \leq F' \mid E' \leq F' \in \nabla\}$ to yield the conclusion by application of Corollary 2.

5 Related work

Partial derivatives of regular expressions are rather widely known [3, 8], but much less so compared to (total) Brzozowski derivatives [6] that are employed to guarantee termination of regular expression matching, parsing and subtyping algorithms [9, 11]. An *algorithmic* definition for either total or partial derivative is treated as standard. There is also a very early and equivalent form of partial derivative defined *declaratively* [15, 7, 2, 1], by structural induction as opposed to structural recursion. The declarative method due to Mirkin is arguably better suited for provably-correct implementations in an inductive theorem prover [1, 13]. The algorithmic version of total derivative is used in [10] to define a

derivative	total	partial
algorithmic	Brzozowski [6]	Antimirov [3]
declarative	Spivak (circa 1963, [15])	Mirkin [15]

Fig. 3. Versions of derivatives of regular expressions.

natural-deduction style calculus for regular expression equivalence. This requires

to consider the infinite factor-automaton on regular expressions modulo idempotent semiring equivalence (called ACI^+-equivalence in [10]) because of the way monomials are presented in the linear form of a regular expression. On the contrary, partial derivatives, being certain ordered sets of monomials, do not require a factor-automaton.

Spivak's declarative analogue of Brzozowski derivative is also called the *base* B of a regular expression in [15], and is a special case of Mirkin prebase, the one where each cell of the prebase matrix in equation (4) of Section 3 is a singleton set containing a regular expression from the set B. A base of E corresponds to a deterministic automaton accepting the language denoted by E. Every prebase can be converted to an equivalent base by an analogue of the automaton determinisation algorithm.

Proof search of regular expression containment can be viewed from a type theoretic perspective if terms are assigned to regular-expressions-as-types, which is done in [11, 12]. The resulting axiomatic system is sequent-style rather than natural-deduction style, although the only essential difference concerns the fixed point rule. Even so, the fixed point rule of the sequent-style system does discard the fixed point assumption, although it is subject to syntactic guardedness checks in order to preserve soundness of the system, without a guarantee of recursively decreasing computation such as in the system **NR**.

A notion of proof search is provided in [11] by referring to Grabmayer's natural-deduction style calculus for regular expression equivalence [10] which is based on a finitary coinduction principle. This has a direct effect on the amount of purely syntactic proof rules to deal with ACI^+-equivalence. Our point of having a fixed point rule that guarantees termination is that proof search can be internal to the proof system rather than implemented outside the proof system. Grabmayer's coinduction principle is designed for equivalence proofs, not containment proofs. The latter observation has influenced a decision to seek for a computational fixed point rule with intrinsic proof search properties.

It is quite straightforward that certain basic problems on regular expressions such as matching can be encoded, by restricting the form of judgement, in intuitionistic linear logic (ILL) [5], that is, logic of linear connectives $\otimes, \multimap, \&, \oplus, !$ and logical constants $I, \mathbf{t}, \mathbf{f}$. Here we have the following correspondence between connectives and constants as in Figure 4. However, problems such as containment

	regular expressions	ILL
void	0	$\mathbf{f}$
alternation	$+$	$\oplus$
empty word	1	I
concatenation	$\times$	$\otimes$
iteration	*	$!$

Fig. 4. Embedding of the language of regular expressions into ILL.

or equivalence of regular expressions require constructing proofs as fixed points, for which the standard ILL is not sufficiently expressive since a higher-order fixed point operator would be required.

In this context, it is appropriate to mention that a linear logic with first-order fixed point operators was investigated in [4] in the context of a focused approach to terminating proof search. The logic μMALL is obtained by enriching the multiplicative additive fragment of linear logic (MALL) with equality, first-order quantifiers and certain least and greatest fixed point operators. The fixed point operators generalise the standard linear logic exponentials ! and ? in a way that allows to represent induction and coinduction by means of structural sequent rules. The problem with straigthforward iterative proof search in μMALL is that it is not practically possible because cut-free derivations are not analytic. Meanwhile, focused proof search allows to restrict the search space to resonable bounds and guarantees completeness (but not decidability).

6 Conclusions

We discussed an application of the finitary coinduction principle for prebases to deciding regular expression containment. The inference system **NR** can only be used for checking whether a proof derivation produces a valid coercion but also for generating such a coercion. We showed how the rule $\mathsf{FIX}^{\psi}_{0,0}$, given a containment problem $E \leq F$, extends to take into account the one-step relation between partial derivatives in the prebases of E and F, which means that this rule provides a complete proof search strategy for containment. A rigorous proof of completeness of this proof search strategy is a work in progress.

References

1. J. B. Almeida, N. Moreira, D. Pereira, and S. M. de Sousa. Partial derivative automata formalized in Coq. In *Implementation and Application of Automata 2010*, volume 6482/2011 of *Lecture Notes in Computer Science*, pages 59–68, 2011.
2. M. Almeida, N. Moreira, and R. Reis. Antimirov and Mosses' rewrite system revisited. *International Journal of Foundations of Computer Science*, 20(4):669–684, August 2009.
3. V. M. Antimirov. Partial derivatives of regular expressions and finite automaton constructions. *Theor. Comput. Sci.*, 155(2):291–319, 1996.
4. D. Baelde. *A linear approach to the proof-theory of least and greatest fixed points.* PhD thesis, École Polytechnique, 2008.
5. G. M. Bierman. *On Intuitionistic Linear Logic.* PhD thesis, Wolfson College, Cambridge, 1994.
6. J. A. Brzozowski. Derivatives of regular expressions. *J. ACM*, 11(4):481–494, 1964.
7. J. A. Brzozowski. Review: B. G. Mirkin, An algorithm for constructing a base in a language of regular expressions. *Journal of Symbolic Logic*, 36(4):694, December 1971.
8. J.-M. Champarnaud and D. Ziadi. From Mirkin's prebases to Antimirov's word partial derivatives. *Fundam. Inf.*, 45:195–205, January 2001.

9. N. A. Danielsson. Total parser combinators. In *Proceedings of the 15th ACM SIGPLAN international conference on Functional programming*, ICFP '10, pages 285–296, New York, NY, USA, 2010. ACM.
10. C. Grabmayer. Using proofs by coinduction to find "traditional" proofs. In J. L. Fiadeiro, editor, *Proceedings of CALCO 2005*, volume 3629 of *LNCS*, 2005.
11. F. Henglein and L. Nielsen. Declarative coinductive axiomatization of regular expression containment and its computational interpretation (preliminary version). Technical Report 612, Department of Computer Science, University of Copenhagen (DIKU), February 2010.
12. F. Henglein and L. Nielsen. Regular expression containment: Coinductive axiomatization and computational interpretation. In *Proc. 38th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL)*. ACM, January 2011.
13. V. Komendantsky. Computable partial derivatives of regular expressions, 2011. Submitted.
14. D. Kozen. A completeness theorem for Kleene algebras and the algebra of regular events. *Infor. and Comput.*, 110(2):366–390, 1994.
15. B. G. Mirkin. New algorithm for construction of base in the language of regular expressions. *Tekhnicheskaya Kibernetika*, 5:113–119, 1966. English translation in *Engineering Cybernetics*, No. 5, Sept.–Oct. 1966, pp. 110-116.
16. J. J. M. M. Rutten. Automata and coinduction (an exercise in coalgebra). In D. Sangiorigi and R. de Simone, editors, *CONCUR '98*, volume 1466 of *LNCS*, pages 194–218. Springer, 1998.

A Proof of Theorem 3

Proof (Outline). By induction on E, we define:
Basis.

1. $P(0) = \{0\}$ and $M(0) = \lambda\ i\ j.\ \emptyset$.
2. $P(1) = \{1\}$ and $M(1) = \lambda\ i\ j.\ \emptyset$.
3. $P(a_k) = \{a_k, 1\}$ and $M(a_k) = \lambda\ i\ j.\ \begin{cases}\text{if } i = 0 \text{ and } j = k \text{ then } \{1\}\\ \text{else } \emptyset.\end{cases}$

Induction step. Let us denote by $\boxdot$ the operation of vertical concatenation of two matrices of the same width. The width is not changing with induction step since it is fixed to be the number of symbols in A.

1. $P(E + F) = \{E_0 + F_0, E_1, \ldots, E_{|P(E)|-1}, F_1, \ldots, F_{|P(F)|-1}\}$.

$$M(E + F) = \left[M(E)_{0,j} \cup M(F)_{0,j}\right]_{0,j} \boxdot \left[M(E)\right]_{0<i,j} \boxdot \left[M(F)\right]_{0<i,j}$$

2. $P(E{\times}F) = \{E_0{\times}F_0, E_1{\times}F_0, \ldots, E_{|P(E)|-1}{\times}F_0, F_1, \ldots, F_{|P(F)|-1}\}$.

$$M(E{\times}F) = \begin{array}{l}\left[\left(\left(\sum M(E)_{i,j}\right){\times}F_0\right) \cup \left(o(E_i){\times}\left(\sum M(F)_{0,j}\right)\right)\right]_{i,j}\\ \boxdot\\ \left[M(F)\right]_{0<i,j}\end{array}$$

3. $P(E^*) = \{E_0^*, E_1 \times E_0^*, \ldots, E_{|P(E)|-1} \times E_0^*\}$ where E_0^* should be read as $(E_0)^*$.

$$M(E^*) = \begin{array}{c} \left[\left(\sum M(E)_{0,j}\right) \times E_0^*\right]_{0,j} \\ \boxdot \\ \left[\left(\left(\sum M(E)_{i,j}\right) \times E_0^*\right) \cup \left(o(E_i) \times \left(\sum M(E)_{0,j}\right) \times E_0^*\right)\right]_{0<i,j} \end{array}$$

Preservation of the property *(i)* is straightforward. The properties *(ii)* and *(iii)* follow by a direct inductive argument. □