Experimental Evaluation of some Algorithms to Check Inclusion and Equivalence of Regular Expressions

Baudouin Le Charlier
Université catholique de Louvain, Belgium
baudouin.lecharlier@uclouvain.be

February 1, 2024

Abstract

1 Introduction

2 A programming framework for regular languages

3 The algorithms

3.1 The basic algorithm

A first, simple, algorithm to check inclusion or equivalence of two normalized expressions E and E' is depicted in Figure 1.

Correctness proof of the algorithm *basicCheck*

It is easily verified that the algorithm agrees with the following invariant.

1. $\forall \langle F, F' \rangle \in R_{dev} : \forall x \in \mathit{Letter} : \langle D_x\, F, D_x\, F' \rangle \in R_{tod} \cup R_{dev}$
2. $\forall \langle F, F' \rangle \in R_{dev} : \; o_F = o_{F'}$

Therefore, since $R_{tod} = \{\}$ when the algorithm returns *true*, the following two conditions hold whenever it returns this value.

1. $\forall \langle F, F' \rangle \in R_{dev} : \forall x \in \mathit{Letter} : \langle D_x\, F, D_x\, F' \rangle \in R_{dev}$
2. $\forall \langle F, F' \rangle \in R_{dev} : \; o_F = o_{F'}$

Those conditions imply that $\forall \langle F, F' \rangle \in R_{dev} : \; \mathcal{L}(F) = \mathcal{L}(F')$. Indeed, we can prove, by induction on the length of w that $\forall w \in \mathit{Letter}^* : \; w \in \mathcal{L}(F) \longleftrightarrow \; w \in \mathcal{L}(F')$. It is true when $w = 1$ because $o_F = o_{F'}$. Otherwise, let $w = x.u$ where x is a letter. Clearly,

$$\begin{array}{rcl} w \in \mathcal{L}(F) & \longleftrightarrow & x.u \in \mathcal{L}(F) \\ & \longleftrightarrow & u \in \mathcal{L}(D_x\, F) \\ & \longleftrightarrow & u \in \mathcal{L}(D_x\, F') \\ & \longleftrightarrow & x.u \in \mathcal{L}(F') \\ & \longleftrightarrow & w \in \mathcal{L}(F') \end{array}$$

Figure 1: The basic algorithm to check inclusion or equivalence ($basicCheck(E, E')$)

$R_{tod} := \{\langle E,\ E' \rangle\}$; $R_{dev} := \{\}$;
while $R_{tod} \neq \{\}$, do
  $\langle F, F' \rangle \Longleftarrow R_{tod}$;
  $tabD := computeDirectDerivatives(F)$;
  $tabD' := computeDirectDerivatives(F')$;
  if $uncompatible(tabD, tabD')$, do
    return *false*;
  for every letter x, do
    if $\langle tabD[x], tabD'[x] \rangle \notin (R_{tod} \cup R_{dev})$, do
      $R_{tod} \Longleftarrow \langle tabD[x], tabD'[x] \rangle$;
  $R_{dev} \Longleftarrow \langle F, F' \rangle$;
return *true*;

The identity $u \in \mathcal{L}(D_x F) \longleftrightarrow u \in \mathcal{L}(D_x F')$ is a consequence of the induction hypothesis applied to the pair $\langle D_x F, D_x F' \rangle$, which, actually, belongs to R_{dev}. The result just proved implies that $\mathcal{L}(E) = \mathcal{L}(E')$ since, of course, $\langle E, E' \rangle \in R_{dev}$.

To complete the proof, we still need to show that $\mathcal{L}(E) \neq \mathcal{L}(E')$ when the algorithm returns *false*, and that the algorithm always terminates. To prove the first claim, we can observe that the invariant of the algorithm implies that all pairs $\langle F, F' \rangle \in R_{dev} \cup R_{tod}$ are such that $F = D_w E$ and $F' = D_w E'$ for some $w \in Letter^*$. In fact, this is best proved by making it an additional condition to the invariant. Therefore, the algoritm returns *false* only if $\mathcal{L}(D_w E) \neq \mathcal{L}(D_w E')$ for some chain w, which implies that $\mathcal{L}(E) \neq \mathcal{L}(E')$. To prove termination, we note that the size of the set R_{dev} increases by one at each iteration of the algorithm but it cannot exceed $nD \times nD'$ where nD and nD' are the numbers of syntactic derivatives of E and E', respectively.

(End of correctness proof)

Given that all the algorithms presented later in this document aim to improve the *basicCheck* algorithm, it is interesting to show how it performs on an example where significant improvements are actually obtained later. Let us consider the expressions E_n of the form $(\mathrm{a}^* \mathrm{b})^* \mathrm{a}^n \mathrm{a}^*$ and E'_n of the form $(\mathrm{a} + \mathrm{b})^* \mathrm{a} (\mathrm{a} + \mathrm{b})^{n-1}$ where $n \geq 1$, from [1]. The relation computed by the version of Algorithm *basicCheck* that checks inclusion of E_n in E'_n is depicted in Figure 2 (for $n = 3$).
The same verification can be done by applying the version that checks equivalence of the expressions $E_n + E'_n$, and E'_n. The computed relation is presented in Figure 4. In both cases, the number of computed pairs in the relation R_{dev} is $8 = 2^3$. In general, it is equal to 2^n. Thus, for this class of examples, the time and space complexity of the algorithm is exponential. This is because the number of syntactic derivatives of E'_n is equal to 2^n and because each syntactic derivative of E_n and

Figure 2: Relation computed by Algorithm *basicCheck* ($\leq$) for $n = 3$

i	F_i	F'_i
1	(a*b)*aaaa*	(a + b)*a(a+ b)^2
2	aaa* + a*b(a*b)*aaaa*	(a+ b)^2 + (a + b)*a(a+ b)^2
3	aa* + a*b(a*b)*aaaa*	a + b + (a+ b)^2 + (a + b)*a(a+ b)^2
4	(a*b)*aaaa*	a + b + (a + b)*a(a+ b)^2
5	a* + a*b(a*b)*aaaa*	1 + a + b + (a+ b)^2 + (a + b)*a(a+ b)^2
6	(a*b)*aaaa*	1 + a + b + (a + b)*a(a+ b)^2
7	aaa* + a*b(a*b)*aaaa*	1 + (a+ b)^2 + (a + b)*a(a+ b)^2
8	(a*b)*aaaa*	1 + (a + b)*a(a+ b)^2

Figure 3: Relation computed by Algorithm *basicCheck* ($=$) for $n = 3$

i	F_i	F'_i
1	$E + E'$	E'
2	aaa* + (a + b)^2 + E' + a*bE	(a + b)^2 + E'
3	a + b + aa* + (a + b)^2 + E' + a*bE	a + b + (a + b)^2 + E'
4	a + b + E + E'	a + b + E'
5	1 + a + b + a* + (a + b)^2 + E' + a*bE	1 + a + b + (a + b)^2 + E'
6	1 + a + b + E + E'	1 + a + b + E'
7	1 + aaa* + (a + b)^2 + E' + a*bE	1 + (a + b)^2 + E'
8	$1 + E + E'$	$1 + E'$

E'_n must be used in at least one pair $\langle F, F' \rangle$ computed by the algorithm.
When the check reports *true*, an execution of Algorithm *basicCheck* boils down to compute DFAs for the expressions E and E' with the algorithm proposed in [**?**] and to check inclusion or equivalence with (the basic version of) Hopcroft and Karp's for DFA ([**?**]). However, Algorithm *basicCheck* generally is much faster and economical than building and using DFAs when the check reports *false*. Thus, in the rest of this paper, we are mainly interested by improving Algorithm *basicCheck* when the result of the check is *true*.

3.2 Improved algorithms for checking inclusion

In this section, we describe an improved algorithm to check inclusion of the languages defined by two normalized expressions E and E'. The algorithm uses ideas apparently first proposed by Antimirov in [1]. The same ideas were used later on in [] for checking inclusion of languages defined by NFAs. The method proposed by Antimirov consists of a set of rewrite rules and it lacks a definite strategy to use them. The correctness of the method is not formally proved either. In this section, full-fledged algorithms are described and proved correct.

Let us first give the main ideas of the method. First, it can be observed that, to prove that $E \leq E'$, it is sufficient to prove $E_1 \leq E'$, ..., $E_n \leq E'$ where E is syntactically equal to $E_1 + \ldots + E_n$ ($n \geq 0$). This may simplify the proof in some "trivial" ways since, for instance, some of the E_i can be direct sub-expressions of E' but the main improvement (in many cases) is due to the fact that we use syntactic derivatives of expressions that are unions of partial derivatives (see [3]). As proved in [], the number of partial derivatives of an expression is small and linear in the size of the expression.

Figure 4: Relation computed by AlgorithmeQ($\sim_e$) for $n = 3$

i	F_i	F'_i
1	$E + E'$	E'
2	$\mathrm{aaa}^* + (\mathrm{a} + \mathrm{b})^2 + E' + \mathrm{a}^*\mathrm{b}E$	$(\mathrm{a} + \mathrm{b})^2 + E'$
3	$\mathrm{a} + \mathrm{b} + \mathrm{aa}^* + (\mathrm{a} + \mathrm{b})^2 + E' + \mathrm{a}^*\mathrm{b}E$	$\mathrm{a} + \mathrm{b} + (\mathrm{a} + \mathrm{b})^2 + E'$
4	$\mathrm{a} + \mathrm{b} + E + E'$	$\mathrm{a} + \mathrm{b} + E'$
5	$1 + \mathrm{a} + \mathrm{b} + \mathrm{a}^* + (\mathrm{a} + \mathrm{b})^2 + E' + \mathrm{a}^*\mathrm{b}E$	$1 + \mathrm{a} + \mathrm{b} + (\mathrm{a} + \mathrm{b})^2 + E'$
6	$1 + \mathrm{a} + \mathrm{b} + E + E'$	$1 + \mathrm{a} + \mathrm{b} + E'$
7	$1 + \mathrm{aaa}^* + (\mathrm{a} + \mathrm{b})^2 + E' + \mathrm{a}^*\mathrm{b}E$	$1 + (\mathrm{a} + \mathrm{b})^2 + E'$
8	$1 + E + E'$	$1 + E'$

Thus, in the worse case, we only have to perform $O(\mathrm{size}(E) \times 2^{\mathrm{size}(E')})$ instead of $O(2^{\mathrm{size}(E)+\mathrm{size}(E')})$ comparisons. Still more interestingly, it can be the case that pairs $\langle P, F\rangle$ and $\langle P, G\rangle$ have to be checked where F and G are unions and the set of sub-expressions of F is a subset of the set of sub-expressions of G. Then the check of $\langle P, G\rangle$ "obviously" is useless and can be removed from what is to be done. This situation is likely to occur because F and G are syntactic derivatives of the same expression E', which are unions of partial derivatives of E'. As an example, let us look at Figure 2, we see that $F_1 = F_4 = F_6 = F_8$ and that F'_1 is a sub-expression of F'_4, F'_6 and F'_8. Therefore, it is useless to check the pairs $\langle F_i, F'_i\rangle$ for $i = 4, 6, 8$.

The improved algorithm to check inclusion is depicted in Figure 5. It makes use of the algorithm *checkSubs* ($\langle P, F\rangle$), which checks whether the pair $\langle P, F\rangle$ is subsumed by another pair in $R_{tod} \cup R_{dev}$, or alternatively detects that some such pairs are subsumed by the pair $\langle P, F\rangle$ itself. More precisely, let us note *minPairs* (P) the set of pairs $\langle P, F'\rangle$ such that $\langle P, F'\rangle \in (R_{tod} \cup R_{dev}) \setminus R_{sub}$. If *minPairs* ($P$) contains a pair $\langle P, F'\rangle$ such that $F' \subseteq F$, the algorithm *checkSubs* ($\langle P, F\rangle$) adds the pair $\langle P, F\rangle$ to R_{sub}; otherwise, it adds to R_{sub} all pairs $\langle P, F'\rangle$ such that $F \subseteq F'$.

Correctness proof of the algorithm iEA

The algorithm agrees with the following loop invariant.

1. $\forall \langle P, F'\rangle \in R_{dev} : \forall x \in \mathit{Letter} : \forall P' \in \mathcal{D}_x P :$
   $\exists \langle P', F''\rangle \in (R_{tod} \setminus R_{sub}) \cup R_{dev}$ such that $F'' \subseteq D_x F'$
2. $\forall \langle P, F'\rangle \in R_{dev} : \ o_P \leq o_{F'}$
3. $\forall \langle P, F'\rangle \in R_{sub} : \exists \langle P, F''\rangle \in (R_{tod} \setminus R_{sub}) \cup R_{dev} : \ F'' \subset F'$

Therefore, since $R_{tod} \setminus R_{sub} = \{\}$ when the algorithm returns *true*, the following postconditions hold whenever it returns this value.

1. $\forall \langle P, F'\rangle \in R_{dev} : \forall x \in \mathit{Letter} : \forall P' \in \mathcal{D}_x P : \exists \langle P', F''\rangle \in R_{dev}$ such that $F'' \subseteq D_x F'$
2. $\forall \langle P, F'\rangle \in R_{dev} : \ o_P \leq o_{F'}$

Those conditions imply that $\forall \langle P, F'\rangle \in R_{dev} : \ \mathcal{L}(P) \subseteq \mathcal{L}(F')$. Indeed, we can prove, by induction on the length of w that $\forall w \in \mathit{Letter}^* : \ w \in \mathcal{L}(P) \rightarrow \ w \in \mathcal{L}(F')$. It is true when $w = 1$ because $o_P \leq o_{F'}$. Otherwise, let $w = x.u$ where x is a letter. Since $w \in \mathcal{L}(P)$, there exists $P' \in \mathcal{D}_x P : u \in \mathcal{L}(P')$. Moreover, the first postcondition above implies that R_{dev} contains a

Figure 5: An improved algorithm to check inclusion (iEA (E, E'))

Let $P_1 + \ldots + P_n = E \setminus E'$;
$R_{tod} := \{\langle P_1,\ E'\rangle, \ldots, \langle P_n,\ E'\rangle\}$;
$R_{dev} := \{\}$; $R_{sub} := \{\}$;
while $R_{tod} \setminus R_{sub} \neq \{\}$, do
  $\langle P, F'\rangle \Longleftarrow R_{tod} \setminus R_{sub}$;
  $tabD \ := \ computeDirectDerivatives\,(P)$;
  $tabD' \ := \ computeDirectDerivatives\,(F')$;
  if $uncompatible\,(tabD\,, tabD')$, do
    return *false*;
  for every letter x, do
    let $P'_1 + \ldots + P'_{n'} = tabD\,[x] \setminus tabD'[x]$;
    for $i := 1$ to n', do
      if $\langle P'_i, tabD'[x]\rangle \ \notin \ (R_{tod} \cup R_{dev} \cup R_{sub})$, do
        $checkSubs\,(\langle P'_i, tabD'[x]\rangle)$;
        if $\langle P'_i, tabD'[x]\rangle \ \notin \ R_{sub}$, do
          $R_{tod} \Longleftarrow \langle P'_i, tabD'[x]\rangle$;
  $R_{dev} \Longleftarrow \langle P, F'\rangle$;
return *true*;

Figure 6: Relation computed by Algorithm iEA to check that (a*b)*aaaa* $\leq$ (a + b)*a(a + b)^2

i	P_i	F'_i
1	(a*b)*aaaa*	(a + b)*a(a + b)^2
2	aaa*	(a + b)^2 + (a + b)*a(a + b)^2
3	a*b(a*b)*aaaa*	(a + b)^2 + (a + b)*a(a + b)^2
4	aa*	a + b + (a + b)^2 + (a + b)*a(a + b)^2
5	a*	1 + a + b + (a + b)^2 + (a + b)*a(a + b)^2

Figure 7: Checking (a*b)*aaaa* $\leq$ (a + b)*a(a + b)^2 without subsumption

i	P_i	F'_i
1	(a*b)*aaaa*	(a + b)*a(a + b)^2
2	aaa*	(a + b)^2 + (a + b)*a(a + b)^2
3	a*b(a*b)*aaaa*	(a + b)^2 + (a + b)*a(a + b)^2
4	aa*	a + b + (a + b)^2 + (a + b)*a(a + b)^2
5	a*b(a*b)*aaaa*	a + b + (a + b)^2 + (a + b)*a(a + b)^2
6	(a*b)*aaaa*	a + b + (a + b)*a(a + b)^2
7	a*	1 + a + b + (a + b)^2 + (a + b)*a(a + b)^2
8	a*b(a*b)*aaaa*	1 + a + b + (a + b)^2 + (a + b)*a(a + b)^2
9	(a*b)*aaaa*	1 + a + b + (a + b)*a(a + b)^2
10	aaa*	1 + (a + b)^2 + (a + b)*a(a + b)^2
11	a*b(a*b)*aaaa*	1 + (a + b)^2 + (a + b)*a(a + b)^2
12	(a*b)*aaaa*	1 + (a + b)*a(a + b)^2

pair $\langle P', F''\rangle$ such that $F'' \subseteq D_x\, F'$. By induction hypothesis, $u \in \mathcal{L}(P') \to u \in \mathcal{L}(F'')$. Thus, $u \in \mathcal{L}(D_x\, F')$. And, finally, $w = x.u \in \mathcal{L}(F')$.

The result just proved implies that $\mathcal{L}(E) \subseteq \mathcal{L}(E')$: Let $P_1 + \ldots + P_n = E \setminus E'$. It is sufficient to prove that $\mathcal{L}(P_i) \subseteq \mathcal{L}(E')$ for all i such that $1 \leq i \leq n$. We can check that the condition $\langle P_i, E'\rangle \in R_{tod} \cup R_{dev} \cup R_{sub}$ is an invariant of the algorithm. When the algorithm terminates, the condition simplifies to $\langle P_i, E'\rangle \in R_{dev} \cup R_{sub}$. Then, either $\langle P_i, E'\rangle \in R_{dev}$, and the result has been proved above; otherwise $\langle P_i, E'\rangle \in R_{sub}$. Then, by the third condition of the invariant, R_{dev} contains a pair $\langle P_i, E''\rangle$ such that $E'' \subset E'$. Thus, $\mathcal{L}(P_i) \subseteq \mathcal{L}(E'') \subseteq \mathcal{L}(E')$.

The rest of the proof is similar to the rest of the proof of the algorithm *basicCheck*.

(End of correctness proof)

Let us compare the behaviours the algorithms *basicCheck* ($\leq$) and iEA on the running example of Section 1, by looking at Figures 2 and 6. The expression (a*b)*aaaa* has five partial derivatives and they are used exactly once in the relation of Figure 6. More generally, the expression (a*b)*a^{n}a* has $n + 2$ partial derivatives, so that the size of the table is linear in n while the size of the table computed by Algorithm *basicCheck* ($\leq$) is 2^n.

3.3 Improved algorithms for checking equivalence

3.3.1 A way to prove equivalence of regular expressions

Let $\sim$ be a relation between regular expressions such that

1. $E \sim E' \ \rightarrow\ \forall x \in \mathit{Letter} : D_x\, E \sim D_x\, E'$
2. $E \sim E' \ \rightarrow\ o_E = o_{E'}$

Then, for all expressions E and E', $\ E \sim E' \ \rightarrow\ \mathcal{L}(E) = \mathcal{L}(E')$.

The proof is similar to the correctness proof of Algorithm *basicCheck*.

Proof

We prove that, for very chain $w \in \mathit{Letter}^*$, $E \sim E' \ \rightarrow\ (w \in \mathcal{L}(E) \longleftrightarrow w \in \mathcal{L}(E'))$. The proof uses an induction on the length of w. The result is immediate for $w = 1$, since $E \sim E' \ \rightarrow\ o_E = o_{E'}$. Now, let us assume that $w = x.u$ where x is a letter. Clearly, $w \in \mathcal{L}(E)$ implies that $u \in \mathcal{L}(D_x\, E)$. As, $E \sim E'$, the condition $D_x\, E \sim D_x\, E'$ holds. Therefore, by induction hypothesis on w, one has $u \in \mathcal{L}(D_x\, E')$, which implies $w = x.u \in \mathcal{L}(E')$.

(End of proof)

3.3.2 The improved algorithm to check equivalence

The algorithm is depicted in Figure 8. It is parameterized by the relation $\sim$, which, in fact, evolves at each iteration and depends on the contents of the relations R_{tod} and R_{dev}. By the way, it can be observed that the algorithm trivially is equivalent to Algorithm *basicCheck*$\,(=)$ if we define the relation $\sim$ by $F \sim F' \longleftrightarrow \langle F, F' \rangle \in R_{tod} \cup R_{dev}$ (shortly : $\sim\ =\ R_{tod} \cup R_{dev}$). The improvements brought by the new, more general, version, come from using more powerfull versions of $\sim$, which allow us to put less pairs of derivatives in R_{tod}. We consider two more powerful versions.

1. ($\sim_e$) We define $\sim_e$ as the smallest equivalence relation containing $R_{tod} \cup R_{dev}$.
2. ($\sim_\oplus$) We define $\sim_\oplus$ as the smallest equivalence relation containing $R_{tod} \cup R_{dev}$ such that $E_1 \sim_\oplus E_1'$ and $E_2 \sim_\oplus E_2'$ implies that $E_1 \oplus E_1' \sim_\oplus E_2 \oplus E_2'$.

Now we can give a (generic) correctness proof of the algorithm eQ$\,(\sim)$.

Correctness proof of the algorithm eQ$\,(\sim)$

The algorithm respects the following invariant.

1. $\forall \langle F, F' \rangle \in R_{dev} :\ \forall x \in \mathit{Letter} :\ D_x\, F \sim D_x\, F'$
2. $\forall \langle F, F' \rangle \in R_{dev} :\ o_F = o_{F'}$

The algorithm add a pair $\langle F, F' \rangle$ in R_{dev} at each iteration. It is enough to prove that the invariant is maintained with this new pair. Of course, the second condition is maintained since the algorithm does not stop (returning *false*). As for the first condition, consider every pair of direct derivatives of F and F', they are noted $\mathit{tabD}\,[x]$ and $\mathit{tabD}'[x]$ in Figure 8. Either $\mathit{tabD}\,[x] \sim \mathit{tabD}'[x]$, then nothing is added to R_{tod}, but the required condition $D_x\, F \sim D_x\, F'$ already is verified beforehand; otherwise, the pair $\langle \mathit{tabD}\,[x], \mathit{tabD}'[x] \rangle$ is added to R_{tod} which implies that the condition $D_x\, F \sim D_x\, F'$ becomes true since, in any case, $R_{tod} \cup R_{dev} \ \subseteq\ \sim$. (Adding a pair to R_{tod} enlarges the relation $\sim$.)

Figure 8: The improved algorithm to check equivalence (eQ $(\sim)(E, E')$)

```
R_tod := {⟨E, E'⟩}; R_dev := {};
while R_tod ≠ {}, do
    ⟨F, F'⟩ ⟸ R_tod ;
    tabD := computeDirectDerivatives (F) ;
    tabD' := computeDirectDerivatives (F') ;
    if uncompatible (tabD , tabD'), do
        return false;
    for every letter x, do
        if tabD[x] ≁ tabD'[x], do
            R_tod ⟸ ⟨tabD[x], tabD'[x]⟩ ;
    R_dev ⟸ ⟨F, F'⟩ ;
return true;
```

When the algorithm terminates, returning *true*, we can prove that for all expressions F and F', $F \sim F' \;\; \rightarrow \;\; \forall x \in \mathit{Letter} : D_x\, F \sim D_x\, F'$. If $\sim = \sim_e$, it is a consequence of the fact that $\sim_e$ is the smallest equivalence relation containing R_{dev} (using transitivity of $\sim_e$). If $\sim = \sim_\oplus$ it can be proved using transitivity and the fact that, according to the definition of syntactic derivatives in [], the syntactic equality $D_x\,(F_1 \oplus F_2) = D_x\, F_1 \oplus D_x\, F_2$ holds for all normalized expressions F_1 and F_2. A detailed proof of this fact is given in Appendix A.1.

Finally, we can use the theorem of Section 3.3.1 to prove that all pairs in R_{dev} are language equivalent, thus, in particular, E and E'.

(End of correctness proof)

The above presentation does not make precise the way relations $\sim$ are practically computed and it ignores its cost. This is to be discussed in Section 4. But now, let us examine how the improved algorithm behaves on the running example of Section 1. The relation computed by Algorithm eQ $(\sim_\oplus)$ is depicted in Figure 9. The column i_b gives the index of the pairs $\langle F_{i_b}, F'_{i_b}\rangle$ in Figure 4. We see that four pairs are not put in the relation by the improved algorithm. Let us explain why. When the expressions F_4 and F'_4 are computed (as $D_b\, F_2$ and $D_b\, F'_2$), it is checked whether $F_4 \sim_\oplus F'_4$. It is the case because $F'_1 \subset F'_4$, which implies that $F'_4 \sim_\oplus (F'_4 \oplus F_1) = F_4$. Thus, the pair $\langle F_4, F'_4\rangle$ is not added to the relation. Later on, the expressions F_6 and F'_6 are computed (as $D_b\, F_3$ and $D_b\, F'_3$). It is found that $F'_1 \subset F'_6$, so that $F'_6 \sim_\oplus (F'_6 \oplus F_1) = F_6$. Finally, the expressions F_7, F'_7, F_8, and F'_8 are not computed at all, since they are derivatives of F_4, F'_4, F_6, and F'_6, which have not been put in the relation.

.

Figure 9: Relation computed by Algorithm eQ ($\sim_\oplus$) for $n = 3$

i	i_b	F_{i_b}	F'_{i_b}
1	1	$E + E'$	E'
2	2	$\mathrm{aaa}^* + (\mathrm{a} + \mathrm{b})^2 + E' + \mathrm{a}^*\mathrm{b}E$	$(\mathrm{a} + \mathrm{b})^2 + E'$
3	3	$\mathrm{a} + \mathrm{b} + \mathrm{aa}^* + (\mathrm{a} + \mathrm{b})^2 + E' + \mathrm{a}^*\mathrm{b}E$	$\mathrm{a} + \mathrm{b} + (\mathrm{a} + \mathrm{b})^2 + E'$
4	5	$1 + \mathrm{a} + \mathrm{b} + \mathrm{a}^* + (\mathrm{a} + \mathrm{b})^2 + E' + \mathrm{a}^*\mathrm{b}E$	$1 + \mathrm{a} + \mathrm{b} + (\mathrm{a} + \mathrm{b})^2 + E'$

4 Notes on the implementation

Before presenting the experimental evaluation of the algorithms described in Section 3, which is the main goal of this paper, it is necessary to give information about the programming framework used to implement the algorithms. A complete description of this framework, called *the background*, has not been published yet but it uses techniques similar to the ones explained in [8]. Partial descriptions of it are given in [6, 7]. Below key aspects of the background are pointed out.

The background contains the representation of a limited set of normalized expressions. Expressions are uniquely represented and identified by an integer. Identifiers of expressions range from 0 to a maximum value N, e.g. $5,000,000$. A global array of N arrays of integers is used to represent all expressions. The identifier *iExpr* of an expression E gives access to an array of integers containing the identifiers of its direct sub-expressions.[1] The types of expressions (0, 1, LETTER, UNION, ...) are provided by another array of length N. (Other global arrays are used to record additional information about the expressions but they are not used by the algorithms of this paper.) The identifier of an expression is computed from its type and the identifiers of its direct sub-expressions using hashing techniques, which allows normalization of (external) expressions in $O(n \log n)$ where n is the size of the expression to normalize. The background maintains a list of the integers actually used to identify expressions as well as a complementary list of all free identifiers. So, identifiers are attributed to expressions on demand. Therefore the same expression is usually given different identifiers at different system runs. When all or many identifiers are used, a kind of specialized garbage collector can be called to mark expressions necessary to complete the current task, and to return the identifiers of the others to the free list.

4.1 Implementation of the basic algorithm

To implement the basic algorithm (see Figure 1) in the context of the background, a global array of N integers is used to represent the sets of pairs R_{tod} and R_{dev}. The same array is reused at every execution of the algorithm. In fact, pairs of expressions are treated as a new kind of expression, of a new type.[2] The set R_{dev} is put in the first positions of the global array and the set R_{tod} in the following first thereafter. The pairs of derivatives are computed with a breadth-first strategy, i.e. (for two letters), with respect to a, b, aa, ab, ba, bb, ... Using a depht-first strategy would be possible but it is inadequate for the improved algorithms (see the next two subsections). The arrays of derivatives *tabD* and *tabD'* actually are arrays of integers (identifiers). Since the letters $a, b, \ldots$ are given identifiers $2, 3, \ldots$, getting the value $tabD[x]$ for a letter x is done in $O(1)$. The

[1] The system is implemented in Java, so arrays of (identifiers of) sub-expressions are Java arrays, of variable length, possibly equal to null when an identifier is not used.

[2] Such pairs can also be possibly viewed as so-called extended expressions (see e.g. [3, 5, 7]). For instance, Algorithm *basicCheck* ($\leq$) can be modified to compute a DFA for the extended expression $E \setminus E'$ (see [7]).

algorithm *computeDirectDerivatives* is fully described in [6]; the most optimized version is used here. It computes derivatives that are unions of partial derivatives (see [2, 6]), thus, in practice, sorted arrays of identifiers, which greatly contributes to the efficiency of the improved algorithms. The check $uncompatible\,(tabD\,, tabD')$ must at least compare $tabD\,[0]$ and $tabD'[0]$ (note that the identifiers of the expressions 0 and 1 actually are the integers 0 and 1) but additional simple checks can improve efficiency (see [10]). To check whether a pair of derivatives $\langle tabD\,[x], tabD'[x]\rangle$ belongs to $R_{tod} \cup R_{dev}$ we use another global array of N elements, reused at every execution of the algorithm. Conceptually, it is an array of booleans whose i-th element is set to true when the pair identified by i is put into R_{tod}. However, to not allocate a new array at each execution of the algorithm, an array of integers is used as well as a global integer that is increased by one at each execution, the new value playing the role of *true*, and the lower ones the role of *false*.

Using the above data structures, the implementation of the algorithm $basicCheck\,(E, E')$ can be estimated as (time) optimal since the total execution time is equal to the time spent building the relation R_{dev}, which is proportional to $\sharp R_{dev}$, plus the time necessary to compute all the derivatives of E and E', each derivative being computed only once (see [6]). Although $\sharp R_{dev}$ can be equal to $nD \times nD'$, where nD and nD' are the number of derivatives of E and E', it can be the case that it takes much more time to compute the derivatives than to compute the relation R_{dev} (see Section 5).

4.2 Implementation of Algorithm iEA (E, E')

To implement the algorithm of Figure 5, the same data structures as those described in the previous subsection are used, with a simple modification to take into account the set R_{sub}, and a new global array to implement the operation $checkSubs\,(\langle P, F\rangle)$. As for the set R_{sub}, we reuse the same global array used to check the presence of a pair in $R_{tod} \cup R_{dev}$ but we need a different value to indicate that a pair belongs to R_{sub}. The global integer playing the role of *true* is incremented two times. The first increment is used to identify pairs in $R_{tod} \cup R_{dev}$, not subsumed by another one in the same set. The second increment is used to identify pairs in R_{sub}. Those pairs can belong to $R_{tod} \cup R_{dev}$ or not: It does not matter, since they must be ignored by the algorithm.

For implementing the operation $checkSubs\,(\langle P, F\rangle)$, we use a single global array of N elements. Let P an expression occurring in at least a pair of $R_{tod} \cup R_{dev} \cup R_{sub}$. Its identifier iP is used a pointer in the global array to the first element of a list containing the identifiers of all pairs $\langle P, F_1\rangle$, $\ldots$, $\langle P, F_{n_P}\rangle$ belonging to $(R_{tod} \cup R_{dev}) \setminus R_{sub}$. Let pTab be the (name of the) global array and let iPF_j be the identifier of the pair $\langle P, F_j\rangle$. The following equalities hold:

$$\text{pTab}[iP] = iPF_1, \qquad \text{pTab}[iPF_j] = iPF_{j+1} \quad (1 \le j < n_P), \qquad \text{pTab}[iPF_{n_P}] = -1.$$

Using these representation conventions, the operation $checkSubs\,(\langle P, F\rangle)$ (see subsection 3.2) can be carried out through a single traversal of the list related to P. To check subsumption of F by a F_j in the list, or the contrary, an operation $subsumeTest\,(F, F')$ is used, which checks if one of the sets of sub-expressions of F and F' is a subset of the other. Its complexity is proportional to the number of sub-expressions of F and F'. A difficult question is to find a best order to maintain sorted the list related to P, in order to minimize the number of subsumption checks, on the average. The current strategy is to put the pair winning the subsumption check (be it F or one of the F_j's), at the beginning of the list, hoping it has a chance to win again at the next call of $checkSubs\,(\langle P, F'\rangle)$.

4.3 Implementations of Algorithm eQ $(\sim)(E, E')$

To implement the algorithm of Figure 8, the same data structures as those described in 4.1 are used. (Remember that the "improved" algorithm is just *basicCheck* (E, E') when $\sim$ is equal to $R_{tod} \cup R_{dev}$.) The only thing to make precise concerns the implementation of the relations $\sim_e$ and $\sim_\oplus$. Concerning $\sim_e$ it is natural to implement it with a UNION-FIND data structure [9]. So both the check $tabD[x] \not\sim_e tabD'[x]$ and the possible enlargement of the relation $\sim_e$ are done in (almost) $O(1)$.[3]

As for the relation $\sim_\oplus$, several possibilities have been investigated. Simply stated, there are two main options:

1. We compute the check $tabD[x] \sim_\oplus tabD'[x]$ from scratch, on the basis of the relation $R_{tod} \cup R_{dev}$, as suggested in [4].
2. We maintain a kind of UNION-FIND data structure to represent the relation $\sim_\oplus$, or a better approximation of it than merely $R_{tod} \cup R_{dev}$.

Now, let's take a look at these possibilities, from the simplest to the most elaborated. In order to check whether $F \sim_\oplus F'$, we can compute two longest expressions $\hat{F}$ and $\hat{F}'$ such that $F \sim_\oplus \hat{F}$ and $F' \sim_\oplus \hat{F}'$ [4]. Then $F \sim_\oplus F'$ if and only if $\hat{F} = \hat{F}'$ (i.e. their identifiers are equal). The computation of $\hat{F}$ (and similarly of $\hat{F}'$) can be done by iterating through the relation $R_{tod} \cup R_{dev}$, which is straightforward since it is contained in a prefix of an array. For every pair $\langle F_i, F'_i \rangle$, it can be checked if $F_i \subseteq \hat{F}$ (or, alternatively, $F'_i \subseteq \hat{F}$); if it is the case we execute the operation $\hat{F} := \hat{F} \oplus F'_i$ (alternatively, $\hat{F} := \hat{F} \oplus F_i$) and we mark the position i as successfully tested. We iterate on the relation $R_{tod} \cup R_{dev}$ until no unmarked position can be successfully tested. This may require at most $\sharp(R_{tod} \cup R_{dev})$ iterations in the whole relation, although it is rarely the case, in practice. In the following and, especially, in Section 5, this version of Algorithm eQ $(\sim_\oplus)$ is called E2. Moreover, the algorithms *basicCheck* $(=)$ and eQ $(\sim_e)$ are called E and E1, respectively.
A drawback of the above method is the fact that the expressions $\hat{F}$ and $\hat{F}'$ are forgotten after the test $\hat{F} = \hat{F}'$, so that part or all of the work done to compute them must possibly be redone at further iterations of the global algorithm. Thus, we can choose to keep track of these expressions by linking F to $\hat{F}$ and F' to $\hat{F}'$ when the pair $\langle F, F' \rangle$ is added into the relation $R_{tod} \cup R_{dev}$. In fact, we can do better: By adding $\langle F, F' \rangle$ into the relation, we enlarge the relation $\sim_\oplus$ in such a way that $F \sim_\oplus F'$, but since $F \sim_\oplus \hat{F}$ and $F' \sim_\oplus \hat{F}'$, we also have $F \sim_\oplus F' \sim_\oplus (F \oplus F') \sim_\oplus (\hat{F} \oplus \hat{F}')$. Thus, we can build a version of Algorithm eQ $(\sim_\oplus)$ such that both F and F' are linked to $(\hat{F} \oplus \hat{F}')$ when the pair $\langle F, F' \rangle$ is added into the relation $R_{tod} \cup R_{dev}$. This can shorten the computation of (a new value of) $\hat{F}$ (or $\hat{F}'$) if one of the two expressions F or F' is computed later on as a new derivative: The new computation can start from the old value of $\hat{F}$ (or $\hat{F}'$). To implement the link, a UNION-FIND structure similar to the one explained for $\sim_e$ is used. Let us note rep(E) the representative of E in the UNION-FIND structure. For the problem at hand, the equalities rep(F) = rep(F') = $(\hat{F} \oplus \hat{F}')$ are established.[4] Note that it is often the case that other expressions have the same representative

[3] In fact, the background contains a global UNION-FIND data structure, which is normally used to record classes of equivalent expressions in a simplification algorithm [7]. However, it cannot be straightforwardly used because the relation $\sim_e$ is not guaranteed to relate equivalent expressions until the algorithm returns true. So the following "trick" is used. A new "pseudo" type of expressions is introduced, called BOX. A boxed expression is of the form *box*(E), where E is a normal, unboxed, expression. The UNION-FIND data structure for $\sim_e$ is built inside the global UNION-FIND data structure of the background, using identifiers of boxed expressions. When the algorithm terminates, returning *true* or *false*, the identiers of all boxed expressions are removed from the UNION-FIND data structure and they are returned to the free list of identifiers. (This requires only one additional global variable to a linked list of identifiers of boxed expressions but we leave out further implementation details.)

[4] More exactly, boxed versions of the expressions are used.

than F or F' beforehand; their representatives automatically are updated in the same way as those of F and F', thanks to the UNION-FIND structure.
Still another idea, when a new pair $\langle F, F' \rangle$ such that $\hat{F} \neq \hat{F'}$ is computed, consists of adding $\langle \hat{F}, \hat{F'} \rangle$ to the relation $R_{tod} \cup R_{sub}$, instead of $\langle F, F' \rangle$. This maintains the invariant as well. Indeed, assume that $F = D_x\, G$ and $F' = D_x\, G'$ for some pair $\langle G, G' \rangle \in (R_{tod} \cup R_{dev})$. Adding $\langle \hat{F}, \hat{F'} \rangle$ to R_{tod} ensures that $\hat{F} \sim_{\oplus} \hat{F'}$ for the extended relation $\sim_{\oplus}$. Thus,

$$D_x\, G \;=\; F \;\sim_{\oplus}\; \hat{F} \;\sim_{\oplus}\; \hat{F'} \;\sim_{\oplus}\; F' \;=\; D_x\, G'.$$

The benefit of this version is that it does not need a UNION-FIND data structure so that no boxed expressions are needed either. However it adds to the relation $R_{tod} \cup R_{dev}$ pairs of larger expressions for which it takes more time to compute their derivatives. Keeping in mind the experimental evaluation of Section 5, this variant is called E3 in the following, while the variant using the UNION-FIND data structure is called E4.

Finally, we can go a step further to design a still more ambitious version ensuring that every expression F occurring in the relation $R_{tod} \cup R_{dev}$ is linked to its best possible (i.e. longest) representative in the relation $\sim_{\oplus}$. This is normally the case with Algorithm E4 for the expressions F and F' just after adding the new pair $\langle F, F' \rangle$ into $R_{tod} \cup R_{dev}$, but this may no longer be the case after some iterations of the algorithm because the relation $\sim_{\oplus}$ enlarges monotonically. This final version, which we call E5 in the following, has to maintain the following invariant.

1. $\forall X, X' \in dom(R_{tod} \cup R_{dev}) :\ X \subseteq \mathrm{rep}(X') \;\rightarrow\; \mathrm{rep}(X) \subseteq \mathrm{rep}(X')$
2. $\forall \langle X, Y \rangle \in R_{tod} \cup R_{dev} :\ X \oplus Y \subseteq \mathrm{rep}(X),\ \mathrm{rep}(Y)$
3. $\forall \langle X, Y \rangle \in R_{tod} \cup R_{dev} :\ \mathrm{rep}(X) = \mathrm{rep}(Y)$
4. $\forall X \in dom(R_{tod} \cup R_{dev}) :\ X \sim_{\oplus} \mathrm{rep}(X)$

Observe that the last three conditions are ensured by Algorithm E4. The additional first condition tells that no representative $\mathrm{rep}(X')$ can be increased since for all $X \subseteq \mathrm{rep}(X')$, the equality $\mathrm{rep}(X') = \mathrm{rep}(X') \oplus \mathrm{rep}(X)$ holds. To maintain the first condition when a new pair $\langle F, F' \rangle$ is added to $R_{tod} \cup R_{dev}$, we have to check all other pairs $\langle X, Y \rangle$ in the relation to see whether $F \subseteq \mathrm{rep}(X)$ or $F' \subseteq \mathrm{rep}(X)$ and $\mathrm{rep}(F) \not\subseteq \mathrm{rep}(X)$. For all such pairs, we execute $\mathrm{rep}(X) := \mathrm{rep}(X) \oplus \mathrm{rep}(F)$ so that $\mathrm{rep}(X)$ is increased. Such pairs $\langle X, Y \rangle$, for which $\mathrm{rep}(X)$ is increased, must then be taken into account, in the same way as $\langle F, F' \rangle$ until no remaining representative can be improved.

5 Experimental evaluation

5.1 Introduction

We compare the practical efficiency of the algorithms proposed before on several class of problems. First, we consider a class of problems where iEA is "exponentially faster" than iE and we show that E2, E3, E4 and E5 provide similar efficiency improvements. Second, we compare the efficiency of the main algorithms for comparing randomly generated expressions. Third, we assess the benefit of using some of the algorithms as subproblems in an algorithm simplifying regular expressions.
Note that all measurements are made with a background using $5,000,000$ identifiers.
Also, additional measurements and comparison of (variants) of the algorithms are proposed in the appendix. Altough they are significant and informative, they are left out of the main paper to focus on the most interesting issues.

Table 1: Checking (a* b)*a^{n}a* $\leq$ (a + b)*a(a + b)$^{n-1}$

n	t_{E}	t_{E1}	t_{E2}	t_{E3}	t_{E4}	t_{E5}	t_{iE}	t_{iEA}	$\sharp R_{\mathrm{exp}}$	$R_{\sim_\oplus}$	$\sharp R_{\mathrm{iEA}}$
1	77.5 μ	125 μ	92.1 μ	139 μ	136 μ	112 μ	88.2 μ	87.1 μ	2	2	3
2	183 μ	353 μ	154 μ	161 μ	241 μ	198 μ	151 μ	124 μ	4	3	4
3	276 μ	445 μ	299 μ	266 μ	480 μ	280 μ	179 μ	117 μ	8	4	5
4	730 μ	659 μ	324 μ	406 μ	429 μ	344 μ	336 μ	134 μ	16	5	6
5	725 μ	887 μ	388 μ	258 μ	432 μ	398 μ	452 μ	221 μ	32	6	7
6	1.29 m	1.57 m	394 μ	327 μ	589 μ	561 μ	796 μ	150 μ	64	7	8
7	2.53 m	3.15 m	606 μ	419 μ	588 μ	566 μ	1.18 m	178 μ	128	8	9
8	3.83 m	4.42 m	609 μ	468 μ	686 μ	734 μ	2.47 m	182 μ	256	9	10
9	6.48 m	8.02 m	476 μ	469 μ	692 μ	736 μ	2.93 m	192 μ	512	10	11
10	12.6 m	13.3 m	554 μ	583 μ	976 μ	710 μ	5.34 m	217 μ	1.02 K	11	12
11	25.1 m	28.7 m	711 μ	572 μ	1.49 m	1.21 m	10.8 m	251 μ	2.04 K	12	13
12	41.3 m	49.5 m	1.24 m	579 μ	1.66 m	1.35 m	19.6 m	250 μ	4.09 K	13	14
13	79.9 m	102 m	1.34 m	1.04 m	1.78 m	1.70 m	47.4 m	257 μ	8.19 K	14	15
14	139 m	190 m	1.36 m	1.02 m	1.80 m	1.70 m	97.6 m	286 μ	16.3 K	15	16
15	283 m	279 m	1.48 m	982 μ	2.03 m	1.93 m	168 m	305 μ	32.7 K	16	17
16	414 m	455 m	1.72 m	1.10 m	2.11 m	2.09 m	300 m	510 μ	65.5 K	17	18
17	847 m	933 m	1.95 m	1.32 m	2.38 m	2.07 m	444 m	593 μ	131 K	18	19
18	1.77 s	1.95 s	2.28 m	4.95 m	2.62 m	2.26 m	911 m	325 μ	262 K	19	20
19	3.40 s	3.79 s	2.12 m	990 μ	2.95 m	2.43 m	1.95 s	385 μ	524 K	20	21
20	6.08 s	$---$	2.43 m	1.03 m	2.95 m	2.31 m	3.78 s	520 μ	1.04 M	21	22

5.2 A class of problems where inclusion can be checked in linear time

We consider the class of expressions already used to differentiate the various algorithms in Section 3. Remember that we note E_n the expression (a* b)*a^{n}a* and E'_n the expression (a + b)*a(a + b)$^{n-1}$. We apply the algorithms iE and iEA to check that $E_n \leq E'_n$ for $n = 1$ to $n = 20$. We also apply the algorithms E, E1, E2, E3, E4 and E5 to the expressions $E_n + E'_n$ and E'_n, to perform that same check. The execution times and the sizes of the relations computed by the algorithms are reported in Table 1. The meaning of the first nine columns is obviously indicated. The column $\sharp R_{\mathrm{exp}}$ is the size of the relations computed by the algorithms E, E1, and iE; it is always equal to 2^n. The column $\sharp R_{\sim_\oplus}$ is the size of the relation computed by the algorithms E2, E3, E4, E5. The column $\sharp R_{\mathrm{iEA}}$ is the size of the relation computed by the algorithm iEA. We clearly see that E, E1 and iE have an exponential complexity (with respect to n) in both time and space, while the other algorithms behave linearly on this class of problems. Times measurements are not very reliable with java on my machine, especially when n is small. Neverteless, we see that iEA is at least to time faster than the algorithms based on congruence closure. Moreover, the algorithm E1, which only checks equivalence by transitivity is not faster than E and is unable to complete the check for $n = 20$ due to a lack of expression identifiers, in spite of using gargage collection.

5.3 Random expressions

In this subsection, the efficiency of the algorithms presented in this paper is discussed by applying them to randomly generated expressions of various sizes, from 10 to 2560. The expressions use two letters and they do not contain any occurrence of the symbol 0. They are generated in such a way that all expressions of this form have the same probability of being chosen (for a given size). Normal

Table 2: Statistics on Algorithms iE and iEA when $E \leq E'$

	Algorithm iE					Algorithm iEA					
ℓ	t_{med}	$t_{9/10}$	t_{max}	t_{deriv}	$\sharp R$	t_{med}	$t_{9/10}$	t_{max}	t_{deriv}	t_{subs}	$\sharp R$
10	$146\,\mu$	$161\,\mu$	$2.71\,m$	$125\,\mu$	4	$156\,\mu$	$161\,\mu$	$2.71\,m$	$96.7\,\mu$	$3.82\,\mu$	3
20	$195\,\mu$	$260\,\mu$	$2.76\,m$	$116\,\mu$	9	$214\,\mu$	$231\,\mu$	$2.86\,m$	$93.3\,\mu$	$7.59\,\mu$	6
40	$267\,\mu$	$496\,\mu$	$1.22\,m$	$211\,\mu$	23	$286\,\mu$	$441\,\mu$	$967\,\mu$	$168\,\mu$	$16.9\,\mu$	13
80	$503\,\mu$	$844\,\mu$	$1.26\,m$	$391\,\mu$	56	$506\,\mu$	$810\,\mu$	$1.42\,m$	$290\,\mu$	$38.9\,\mu$	39
160	$1.24\,m$	$2.25\,m$	$4.85\,m$	$949\,\mu$	159	$970\,\mu$	$1.82\,m$	$3.73\,m$	$525\,\mu$	$92.7\,\mu$	91
320	$4.21\,m$	$6.66\,m$	$10.5\,m$	$3.32\,m$	806	$2.26\,m$	$4.06\,m$	$7.78\,m$	$1.10\,m$	$387\,\mu$	305
640	$32.3\,m$	$73.5\,m$	$102\,m$	$25.5\,m$	$6.65\,K$	$9.34\,m$	$21.3\,m$	$37.4\,m$	$3.99\,m$	$2.76\,m$	$1.26\,K$
1280	$394\,m$	$669\,m$	$2.57\,s$	$287\,m$	$117\,K$	$57\,m$	$159\,m$	$487\,m$	$12.5\,m$	$34.9\,m$	$5.87\,K$
2560	$8.07\,s$	$---$	$---$	$6.01\,s$	$1.17\,M$	$380\,m$	$895\,m$	$3.44\,s$	$63.6\,m$	$286\,m$	$21.4\,K$

(non normalized) expressions are first generated and they are normalized before the algorithms are applied to them. (Normalization reduces the size of the expressions by approximatively 25%, on the average.) For each size, three kinds of pairs of expressions are selected. The first set only contains pairs $\langle E, E' \rangle$ such that $E \not\leq E'$ (i.e. $\mathcal{L}(E) \not\subseteq \mathcal{L}(E')$. The second set contains pairs such that $E \leq E'$. The third set contains pairs of equivalent expressions. The three sets contains 50 pairs of independently generated expressions. The sets are created by generating expressions completely randomly and selecting pairs of consecutive expressions meeting a given condition. For selecting equivalent pairs, it is checked that both are equivalent to (a + b)* .

5.3.1 Checking inclusion

We first compare the efficiency of Algorithms iE and iEA on expressions such that $E \leq E'$ in Table 16. The first column ℓ indicates the size of the expressions to compare. For both algorithms, the columns t_{med}, $t_{9/10}$, and t_{max} give the median execution time, the ninth decile, and the maximum value of the execution time, respectively; the columns t_{deriv} are the average time spent in Algorithm *computeDirectDerivatives*; the columns $\sharp R$ show the number of pairs in the relation R_{dev} when the algorithms terminate. Finally, t_{subs} is the average of the time spent checking subsumption in Algorithm iEA (i.e. in the calls to *checkSubs*). It can be observed that the speed-up brought by Algorithm iEA over iE is significant and that it increases with the size of the expressions. The time spent computing derivatives is reduced by almost two orders of magnitude for $\ell = 2560$. In the case of iE, approximatively 75% of the execution time is spent computing derivatives, for all sizes, while it becomes almost marginal for iEA when ℓ grows. However, for large values of ℓ, the time spent in checking subsumption increases up to 75% of the total execution time. As for the size of the relation, it grows much quicker for iE than for iEA, up to 25% of the total number of identifiers available in the background (for $\ell = 2560$), while it remains under 0.5% for iEA. The average size of the relation nevertheless grows exponentially for iEA but much more slowly than for iE. Note that iE is only able to check 33 pairs for $\ell = 2560$, so the values indicated for $\sharp R$ are underestimated. More pairs should be needed for some problems, possibly more than the number of available identifiers.

Now, let us compare the same algorithms for pairs of expressions E, E' such that $E \not\leq E'$. It is reasonable to think that this is a frequently occurring situation, with a much higher probability than the case $E \leq E'$, analyzed above, which can be seen as the worst case scenario. Statistics on such pairs are given in Table 3. It can be observed that the two algorithms behaves similarly

Table 3: Statistics on Algorithms iE and iEA when $E \not\leq E'$

	Algorithm iE					Algorithm iEA					
ℓ	t_{med}	$t_{9/10}$	t_{max}	t_{deriv}	$\sharp R$	t_{med}	$t_{9/10}$	t_{max}	t_{deriv}	t_{subs}	$\sharp R$
10	$49.3\,\mu$	$78.6\,\mu$	$164\,\mu$	$41.4\,\mu$	2	$76.2\,\mu$	$108\,\mu$	$240\,\mu$	$30.3\,\mu$	$2.73\,\mu$	2
20	$84.9\,\mu$	$132\,\mu$	$208\,\mu$	$73.7\,\mu$	2	$155\,\mu$	$171\,\mu$	$2.43\,m$	$99.8\,\mu$	$4.57\,\mu$	3
40	$120\,\mu$	$260\,\mu$	$338\,\mu$	$106\,\mu$	2	$145\,\mu$	$280\,\mu$	$522\,\mu$	$71.8\,\mu$	$8.84\,\mu$	4
80	$191\,\mu$	$462\,\mu$	$799\,\mu$	$171\,\mu$	4	$234\,\mu$	$521\,\mu$	$931\,\mu$	$128\,\mu$	$16\,\mu$	7
160	$237\,\mu$	$600\,\mu$	$1.90\,m$	$214\,\mu$	4	$298\,\mu$	$807\,\mu$	$2.21\,m$	$165\,\mu$	$20.5\,\mu$	9
320	$380\,\mu$	$842\,\mu$	$5.33\,m$	$339\,\mu$	8	$411\,\mu$	$897\,\mu$	$2.46\,m$	$233\,\mu$	$33.7\,\mu$	15
640	$304\,\mu$	$711\,\mu$	$1.97\,m$	$278\,\mu$	3	$391\,\mu$	$627\,\mu$	$4.39\,m$	$212\,\mu$	$31.4\,\mu$	10
1280	$445\,\mu$	$1.81\,m$	$2.68\,m$	$417\,\mu$	5	$525\,\mu$	$1.50\,m$	$3.14\,m$	$299\,\mu$	$36.1\,\mu$	22
2560	$628\,\mu$	$967\,\mu$	$7.80\,m$	$588\,\mu$	11	$696\,\mu$	$1.07\,m$	$10.3\,m$	$318\,\mu$	$73.6\,\mu$	35

Table 4: Statistics on Algorithms E and E4 when $\mathcal{L}(E) = \mathcal{L}(E')$

	Algorithm E				Algorithm E4				
ℓ	t_{med}	$t_{9/10}$	t_{max}	t_{deriv}	t_{med}	$t_{9/10}$	t_{max}	t_{deriv}	t_{subst}
10	$53.5\,\mu$	$131\,\mu$	$2.60\,m$	$96.5\,\mu$	$77.2\,\mu$	$208\,\mu$	$2.70\,m$	$92.2\,\mu$	$3.61\,\mu$
20	$213\,\mu$	$511\,\mu$	$745\,\mu$	$211\,\mu$	$180\,\mu$	$380\,\mu$	$580\,\mu$	$117\,\mu$	$8.51\,\mu$
40	$192\,\mu$	$412\,\mu$	$3.19\,m$	$266\,\mu$	$273\,\mu$	$621\,\mu$	$3.33\,m$	$236\,\mu$	$18.1\,\mu$
80	$313\,\mu$	$702\,\mu$	$2.63\,m$	$418\,\mu$	$642\,\mu$	$1.14\,m$	$3.55\,m$	$338\,\mu$	$57\,\mu$
160	$891\,\mu$	$1.34\,m$	$7.36\,m$	$1.01\,m$	$1.66\,m$	$2.75\,m$	$7.63\,m$	$692\,\mu$	$288\,\mu$
320	$3.18\,m$	$7.03\,m$	$17\,m$	$3.39\,m$	$4.66\,m$	$7.86\,m$	$20.1\,m$	$1.42\,m$	$1.20\,m$
640	$19.1\,m$	$38.6\,m$	$122\,m$	$20.2\,m$	$17.4\,m$	$41.6\,m$	$63.7\,m$	$4.48\,m$	$6.30\,m$
1280	$273\,m$	$1.07\,s$	$5.71\,s$	$367\,m$	$68\,m$	$119\,m$	$170\,m$	$10.4\,m$	$22.4\,m$
2560	$7.55\,s$	$18.8\,s$	$30\,s$	$7.50\,s$	$180\,m$	$432\,m$	$1.30\,s$	$26.5\,m$	$95.3\,m$

and that they are reasonably efficient on these test data: Most of the checks require less than a millisecond.

5.3.2 Checking equivalence

The algorithms for checking equivalence are now compared on a set of pairs of equivalent expressions (actually equivalent to (a + b)*). The basic algorithm E is compared to Algorithm E4, which appears to be the most efficient on the average, in Table 4. The name of the columns are as in the previous tables, except t_{subst} which gives the average time spent in checking for some expressions X and X' that $X \subseteq \text{rep}(X')$ in order to possibly increase $\text{rep}(X')$. We observe that both algorithms have similar execution times when ℓ is less or equal to 320 but Algorithm E4 behaves much more better for greater expressions. The time spent computing derivatives increases almost linearly for E4, while it increases exponentially for E. For largest values of ℓ, the time spent in subsumption checks becomes the largest part of the execution time.
More statistics are given in Table 5. Here, the execution times are correlated to $\sharp R$, i.e. the size of the relation R_{dev}, computed by the algorithms. First, it can be seen that the size of the relation is much bigger for E than for the other algorithms. It is almost equal to 50% of the number of available identifiers. In fact, when $\ell = 2560$, Algorithm E is able to complete the test only for 36

Table 5: Statistics on Algorithms for cheking equivalence when $\mathcal{L}(E) = \mathcal{L}(E')$

	E		E1		E2		E3	E4	E5
ℓ	t_{avrg}	$\sharp R$	t_{avrg}	$\sharp R$	t_{avrg}	$\sharp R$	t_{avrg}	t_{avrg}	t_{avrg}
10	$110\,\mu$	3	$134\,\mu$	3	$133\,\mu$	3	$161\,\mu$	$145\,\mu$	$151\,\mu$
20	$247\,\mu$	5	$186\,\mu$	5	$205\,\mu$	5	$221\,\mu$	$213\,\mu$	$216\,\mu$
40	$297\,\mu$	10	$273\,\mu$	9	$428\,\mu$	7	$455\,\mu$	$396\,\mu$	$426\,\mu$
80	$469\,\mu$	21	$467\,\mu$	19	$878\,\mu$	13	$908\,\mu$	$731\,\mu$	$876\,\mu$
160	$1.17\,m$	77	$1.16\,m$	52	$3\,m$	26	$2.38\,m$	$1.97\,m$	$2.04\,m$
320	$4.19\,m$	565	$3.68\,m$	234	$11.8\,m$	50	$7.61\,m$	$5.44\,m$	$6.50\,m$
640	$23.8\,m$	$3.07\,K$	$20\,m$	873	$42.2\,m$	95	$27.8\,m$	$22.3\,m$	$24\,m$
1280	$531\,m$	$159\,K$	$288\,m$	$11\,K$	$158\,m$	183	$84.5\,m$	$71.2\,m$	$91.9\,m$
2560	$9.79\,s$	$2.35\,M$	$9.28\,s$	$157\,K$	$756\,m$	337	$351\,m$	$265\,m$	$292\,m$

Table 6: Time to compute derivatives when $\mathcal{L}(E) = \mathcal{L}(E')$

ℓ	E	E1	E2	E3	E4	E5
10	$96.5\,\mu$	$96.9\,\mu$	$94.7\,\mu$	$107\,\mu$	$92.2\,\mu$	$92.2\,\mu$
20	$211\,\mu$	$125\,\mu$	$128\,\mu$	$127\,\mu$	$117\,\mu$	$113\,\mu$
40	$266\,\mu$	$189\,\mu$	$244\,\mu$	$259\,\mu$	$236\,\mu$	$234\,\mu$
80	$418\,\mu$	$339\,\mu$	$340\,\mu$	$427\,\mu$	$338\,\mu$	$378\,\mu$
160	$1.01\,m$	$874\,\mu$	$655\,\mu$	$919\,\mu$	$692\,\mu$	$641\,\mu$
320	$3.39\,m$	$2.82\,m$	$1.68\,m$	$2.31\,m$	$1.42\,m$	$1.42\,m$
640	$20.2\,m$	$17\,m$	$3.60\,m$	$7.23\,m$	$4.48\,m$	$4.03\,m$
1280	$367\,m$	$265\,m$	$8.45\,m$	$19.4\,m$	$10.4\,m$	$13.3\,m$
2560	$7.50\,s$	$8.76\,s$	$22\,m$	$70.7\,m$	$26.5\,m$	$26.7\,m$

pairs of expressions, so that the actual value of $\sharp R$ is underestimated (as well as the value of t_{avrg}). Looking to the figures for Algorithm E1, we see that the size of $\sharp R$ remains much more smaller than for E: Only 3% of the number of available identifiers, for $\ell = 2560$. Nevertheless, the execution times for E1 are not much better. This is explained by the fact that, for both algorithms, most of the execution time is spent computing derivatives, which are memoized. The time spent building R_{dev} is almost marginal, even for E. The results for the four versions of Algorithm eQ $(\sim_\oplus)$ are more striking. Note that they always compute relations R_{dev} of exactly the same size and that this size grows less than linearly.[5] Algorithm E2, which recomputes the largest expression $\hat{F}$ such that $F \sim_\oplus \hat{F}$, from scratch, as in [4], is more than two times less efficient than the three other versions. The execution times of these best versions clearly are less than quadratic.

Tables 6 and 7 provide information on how the execution time of the algorithms is distributed between the main tasks: Computing derivatives, checking inclusion $(\subseteq)$, and building new expressions (mainly $\oplus$). Table 6 depicts the times spent by the various algorithms to compute derivatives. Algorithms E et E1 compute all derivatives of E and E' exactly once and the times should be the same. The differences in Table 6 are partly due to the low precision of the method used to measure time in java, but also to the fact that using memoization has a cost, which is higher for E. The timings for E2, E4 and E5 also should be equal. The timings for E3 are higher because the pairs

[5]Of course, this is true only on the average.

Table 7: Statistics on Algorithms for cheking equivalence when $\mathcal{L}(E) = \mathcal{L}(E')$

	E2		E3		E4		E5	
ℓ	n_{subst}	t_{subst}	n_{subst}	t_{subst}	n_{subst}	t_{subst}	n_{subst}	t_{subst}
10	6	$4.33\,\mu$	15	$6.50\,\mu$	5	$3.61\,\mu$	6	$4.34\,\mu$
20	34	$9.49\,\mu$	58	$11.6\,\mu$	23	$8.51\,\mu$	31	$9.15\,\mu$
40	177	$28.9\,\mu$	185	$31.1\,\mu$	90	$18.1\,\mu$	113	$23.2\,\mu$
80	812	$97.1\,\mu$	653	$85.9\,\mu$	440	$57\,\mu$	562	$74.1\,\mu$
160	$4.28\,K$	$587\,\mu$	$2.47\,K$	$336\,\mu$	$2.03\,K$	$288\,\mu$	$2.53\,K$	$341\,\mu$
320	$22.8\,K$	$3.15\,m$	$11.7\,K$	$1.51\,m$	$10\,K$	$1.20\,m$	$12.4\,K$	$1.56\,m$
640	$88.8\,K$	$12.6\,m$	$42.7\,K$	$6.80\,m$	$42.1\,K$	$6.30\,m$	$50.4\,K$	$7.55\,m$
1280	$318\,K$	$48.6\,m$	$144\,K$	$20.6\,m$	$165\,K$	$22.4\,m$	$196\,K$	$27.8\,m$
2560	$1.21\,M$	$222\,m$	$527\,K$	$88.6\,m$	$656\,K$	$95.3\,m$	$783\,K$	$109\,m$

Table 8: Statistics on Algorithms for cheking equivalence when $\mathcal{L}(E) = \mathcal{L}(E')$

Alg	t_{avg}	t_{med}	$t_{9/10}$	t_{max}	t_{deriv}	t_{subst}	$\sharp R$	$\max(\sharp R)$	ℓ_1	ℓ_2
E	$18.1\,m$	$1.61\,m$	$30.7\,m$	$494\,m$	$15.6\,m$		$2.29\,K$	$155\,K$	387	388
E1	$19.2\,m$	$2.03\,m$	$27.6\,m$	$505\,m$	$15.6\,m$		840	$27\,K$	387	388
E2	$134\,m$	$6.39\,m$	$143\,m$	$7.14\,s$	$2.76\,m$	$44.9\,m$	134	$2.01\,K$	578	318
E3	$102\,m$	$5.33\,m$	$137\,m$	$5.76\,s$	$4.41\,m$	$30.5\,m$	134	$2.01\,K$	578	318
E4	$135\,m$	$5.09\,m$	$116\,m$	$7.70\,s$	$2.46\,m$	$41.5\,m$	134	$2.01\,K$	578	318
E5	$172\,m$	$6.18\,m$	$160\,m$	$9.67\,s$	$2.67\,m$	$52.8\,m$	134	$2.01\,K$	578	318
EA	$7.91\,m$	$1.69\,m$	$11.3\,m$	$218\,m$	$2.81\,m$	$2.16\,m$	$1.10\,K$	$18.1\,K$	578	318

of expressions added to $R_{tod} \cup R_{dev}$ are of the form $\langle \hat{F}, \hat{F'} \rangle$ instead of $\langle F, F' \rangle$. Expressions such as $\hat{F}$ and $\hat{F}'$ are larger so their derivation takes more time. Nevertheless, we can see that the timings for E2 to E5 grow less than quadratically. Table 7 contains statistics on the numbers (n_{subst}) and timings (t_{subst}) of inclusion checks ($\subseteq$) executed by the algorithms, on the average. Because it computes expressions $\hat{F}$ from scratch, Algorithm E2 performs about twice as many checks as the others, and its execution times are proportionally increased. Algorithm E3 is best on this issue because derivatives of pairs of expressions $\langle F, F' \rangle$ chosen in R_{tod} are very frequently found syntactically equal, so that the computation of $\hat{F}$ and $\hat{F}'$ is unnecessary.

Putting together the figures from Tables 5, 6, and 7, we conclude that for Algorithms E and E1, most of the execution time is spent computing derivatives, whose number grows exponentially whith ℓ. For expressions denoting the same regular language, they virtually compute the DFAs of both expressions while executing the algorithm of [**?**] on them, in parallel. Computing the DFAs takes most of the time. As for Algorithms E2 to E5, the time spent computing derivatives is just a small part of the total execution time, which becomes proportionally smaller when ℓ grows. The time spent checking inclusion is greater and it proportionally grows with ℓ. The rest of the time is spent creating expressions, mostly through the operation $\oplus$.

Table 9: Statistics on Algorithms for cheking equivalence when $\mathcal{L}(E) = \mathcal{L}(E')$

Alg	t_{avg}	t_{med}	$t_{9/10}$	t_{max}	t_{deriv}	t_{subst}	$t_{\oplus}$	$\sharp R$	$\max(\sharp R)$	ℓ_1	ℓ_2
E	$21.8\,m$	$1.63\,m$	$33.2\,m$	$387\,m$	$19.5\,m$	$0\,\mu$	$1.98\,m$	$2.53\,K$	$153\,K$	381	382
E1	$22.1\,m$	$1.90\,m$	$38.6\,m$	$316\,m$	$18.7\,m$	$0\,\mu$	$2.71\,m$	$1.14\,K$	$26.8\,K$	381	382
E2	$144\,m$	$5.46\,m$	$165\,m$	$6.27\,s$	$2.59\,m$	$49.3\,m$	$2.74\,m$	150	$2.02\,K$	356	641
E3	$110\,m$	$4.07\,m$	$115\,m$	$4.90\,s$	$4.31\,m$	$31.5\,m$	$960\,\mu$	150	$2.02\,K$	356	641
E4	$163\,m$	$4.68\,m$	$173\,m$	$7.48\,s$	$2.85\,m$	$47.2\,m$	$2.20\,m$	150	$2.02\,K$	356	641
E5	$202\,m$	$5.80\,m$	$190\,m$	$9.18\,s$	$2.88\,m$	$59.8\,m$	$2.18\,m$	150	$2.02\,K$	356	641
EA	$7.09\,m$	$1.36\,m$	$9.97\,m$	$200\,m$	$2.52\,m$	$1.92\,m$	$323\,\mu$	$1.01\,K$	$16.8\,K$	356	641

Table 10: Statistics on Algorithms for cheking equivalence when $\mathcal{L}(E) = \mathcal{L}(E')$

Alg	t_{avg}	t_{med}	$t_{9/10}$	t_{max}	t_{deriv}	t_{subst}	$t_{\oplus}$	$\sharp R$	$\max(\sharp R)$	ℓ_1	ℓ_2
E	$21.8\,m$	$1.63\,m$	$33.2\,m$	$387\,m$	$1.52\,m$	$0\,\mu$	$256\,\mu$	146	$153\,K$	381	382
E1	$22.1\,m$	$1.90\,m$	$38.6\,m$	$316\,m$	$1.45\,m$	$0\,\mu$	$354\,\mu$	137	$26.8\,K$	381	382
E2	$144\,m$	$5.46\,m$	$165\,m$	$6.27\,s$	$689\,\mu$	$1.45\,m$	$648\,\mu$	58	$2.02\,K$	356	641
E3	$110\,m$	$4.07\,m$	$115\,m$	$4.90\,s$	$1.06\,m$	$894\,\mu$	$305\,\mu$	58	$2.02\,K$	356	641
E4	$163\,m$	$4.68\,m$	$173\,m$	$7.48\,s$	$644\,\mu$	$1.18\,m$	$431\,\mu$	58	$2.02\,K$	356	641
E5	$202\,m$	$5.80\,m$	$190\,m$	$9.18\,s$	$730\,\mu$	$1.43\,m$	$388\,\mu$	58	$2.02\,K$	356	641
EA	$7.09\,m$	$1.36\,m$	$9.97\,m$	$200\,m$	$758\,\mu$	$105\,\mu$	$157\,\mu$	301	$16.8\,K$	356	641

Table 11: Statistics on Algorithms for cheking equivalence when $\mathcal{L}(E) = \mathcal{L}(E')$

Alg	t_{avg}	t_{med}	$t_{9/10}$	t_{max}	t_{deriv}	t_{subst}	$\sharp R$	$\max(\sharp R)$	ℓ_1	ℓ_2
E	$24.3\,m$	$1.64\,m$	$46.8\,m$	$433\,m$	$21.7\,m$		$2.53\,K$	$153\,K$	381	382
E1	$22.3\,m$	$1.79\,m$	$40.8\,m$	$316\,m$	$18.9\,m$		$1.14\,K$	$8.15\,K$	356	641
E2	$147\,m$	$5.43\,m$	$151\,m$	$6.36\,s$	$2.57\,m$	$50.9\,m$	150	$2.02\,K$	356	641
E3	$113\,m$	$4.67\,m$	$113\,m$	$5.04\,s$	$4.41\,m$	$32.7\,m$	150	$2.02\,K$	356	641
E4	$162\,m$	$4.87\,m$	$158\,m$	$7.47\,s$	$2.87\,m$	$47.2\,m$	150	$2.02\,K$	356	641
E5	$248\,m$	$6.42\,m$	$188\,m$	$11.7\,s$	$3.39\,m$	$67.6\,m$	150	$2.02\,K$	356	641
EA	$9.30\,m$	$1.63\,m$	$13.8\,m$	$218\,m$	$3.52\,m$	$2.31\,m$	$1.01\,K$	$16.8\,K$	356	641

Table 12: Statistics on Algorithms for cheking equivalence when $\mathcal{L}(E) = \mathcal{L}(E')$

Alg	t_{avg}	t_{med}	$t_{9/10}$	t_{max}	t_{deriv}	t_{subst}	$\sharp R$	$\max(\sharp R)$	ℓ_1	ℓ_2
E	$22.4\,m$	$1.52\,m$	$32.9\,m$	$570\,m$	$19.6\,m$		$2.34\,K$	$153\,K$	381	382
E1	$21.1\,m$	$2.08\,m$	$28.9\,m$	$361\,m$	$17.7\,m$		965	$10.5\,K$	582	320
E2	$151\,m$	$5.77\,m$	$135\,m$	$7.24\,s$	$3.07\,m$	$50\,m$	151	$2.01\,K$	582	320
E3	$116\,m$	$4.97\,m$	$124\,m$	$5.67\,s$	$4.60\,m$	$33.7\,m$	151	$2.01\,K$	582	320
E4	$174\,m$	$5.26\,m$	$149\,m$	$8.53\,s$	$3.24\,m$	$47.4\,m$	151	$2.01\,K$	582	320
E5	$219\,m$	$6.47\,m$	$224\,m$	$10.7\,s$	$3.28\,m$	$61.7\,m$	151	$2.01\,K$	582	320
EA	$7.48\,m$	$1.70\,m$	$14.4\,m$	$196\,m$	$2.78\,m$	$1.93\,m$	$1.05\,K$	$17.4\,K$	582	320

Table 13: Statistics on Algorithms for simplifying expressions

Alg	iEA — E1				iEA — E4				iEA — EA			
ℓ	t_{avg}	t_{med}	$t_{9/10}$	t_{max}	t_{avg}	t_{med}	$t_{9/10}$	t_{max}	t_{avg}	t_{med}	$t_{9/10}$	t_{max}
10	280 μ	170 μ	494 μ	6.91 m	294 μ	169 μ	526 μ	6.35 m	256 μ	148 μ	447 μ	6.48 m
20	839 μ	439 μ	1.66 m	7.39 m	601 μ	440 μ	1.05 m	4.85 m	586 μ	383 μ	987 μ	6.83 m
40	1.58 m	1 m	3.27 m	8.96 m	1.60 m	1.07 m	2.92 m	9.71 m	1.32 m	933 μ	2.35 m	7.77 m
80	2.49 m	1.58 m	4.55 m	14.8 m	3.02 m	1.96 m	5.73 m	19.7 m	2.22 m	1.64 m	3.95 m	15.5 m
160	5.67 m	3.41 m	9.27 m	105 m	7.75 m	4.02 m	12.8 m	106 m	5.11 m	3.44 m	9.10 m	42.9 m
320	18.3 m	8.10 m	31.1 m	425 m	42 m	8.86 m	51.2 m	2.04 s	13.6 m	6.89 m	25.1 m	201 m
640	39.2 m	14.5 m	60.5 m	556 m	114 m	20.3 m	113 m	1.70 s	50.1 m	14.8 m	56.5 m	878 m
1280	177 m	30.9 m	99.8 m	9.02 s	762 m	55.7 m	359 m	55.2 s	222 m	37.2 m	113 m	15.7 s
2560	227 m	62.7 m	310 m	3.79 s	3.81 s	130 m	2.31 s	157 s	1.50 s	65 m	289 m	116 s

Table 14: Statistics on Algorithms for simplifying expressions

Alg	iE — E1				iE — E4				iE — EA			
ℓ	t_{avg}	t_{med}	$t_{9/10}$	t_{max}	t_{avg}	t_{med}	$t_{9/10}$	t_{max}	t_{avg}	t_{med}	$t_{9/10}$	t_{max}
10	289 μ	194 μ	509 μ	6.20 m	276 μ	143 μ	510 μ	6.45 m	419 μ	222 μ	618 μ	13.1 m
20	577 μ	439 μ	1.08 m	5.17 m	561 μ	391 μ	1.04 m	4.88 m	582 μ	406 μ	980 μ	5.28 m
40	1.41 m	945 μ	2.56 m	8.67 m	1.46 m	916 μ	2.84 m	9.39 m	1.31 m	883 μ	2.49 m	7.53 m
80	2.56 m	1.77 m	4.86 m	13.6 m	2.99 m	1.96 m	5.69 m	17.7 m	2.45 m	1.81 m	4.85 m	14.3 m
160	6.06 m	3.20 m	9.92 m	123 m	8.04 m	4.26 m	13.8 m	121 m	6.45 m	4.06 m	12.4 m	48.1 m
320	15.4 m	7.43 m	24.4 m	337 m	42.4 m	8.24 m	57.1 m	2.04 s	14.5 m	7.34 m	26.6 m	205 m
640	38.1 m	13.8 m	61.3 m	507 m	107 m	18.2 m	85.6 m	1.64 s	58.2 m	15 m	59.2 m	1.37 s
1280	268 m	34.7 m	149 m	16.2 s	825 m	59.3 m	382 m	53.2 s	299 m	37.9 m	132 m	15.4 s
2560	478 m	72.8 m	364 m	20.6 s	64.5 s	155 m	2.04 s	6065 s	1.70 s	80.6 m	703 m	102 s

Table 15: Statistics on Algorithms for simplifying expressions

Alg	iE — E				iEA — E				iEA — E2			
ℓ	t_{avg}	t_{med}	$t_{9/10}$	t_{max}	t_{avg}	t_{med}	$t_{9/10}$	t_{max}	t_{avg}	t_{med}	$t_{9/10}$	t_{max}
10	228 μ	131 μ	362 μ	6.09 m	227 μ	141 μ	383 μ	6.01 m	246 μ	162 μ	430 μ	5.87 m
20	479 μ	350 μ	835 μ	5.27 m	504 μ	356 μ	848 μ	4.85 m	558 μ	396 μ	1.02 m	4.81 m
40	1.06 m	757 μ	1.88 m	7.02 m	1.15 m	856 μ	1.88 m	7.16 m	1.33 m	880 μ	2.50 m	9.12 m
80	2.07 m	1.46 m	3.90 m	10.9 m	2.19 m	1.55 m	4.11 m	12.2 m	2.37 m	1.55 m	4.69 m	13.8 m
160	4.73 m	2.63 m	6.91 m	92.2 m	4.76 m	2.80 m	6.97 m	85.3 m	7.84 m	4.49 m	15 m	116 m
320	14.1 m	7.07 m	21.8 m	290 m	13 m	6.29 m	22.2 m	288 m	59.9 m	10.4 m	63.3 m	3.32 s
640	35 m	12.9 m	55.8 m	537 m	34.5 m	12.9 m	53.1 m	533 m	130 m	18.7 m	112 m	1.99 s
1280	315 m	33.8 m	219 m	19.5 s	174 m	31.4 m	112 m	8.99 s	892 m	57.7 m	401 m	66.8 s
2560	455 m	71.1 m	344 m	21.7 s	198 m	52.8 m	251 m	3.32 s	3.65 s	151 m	1.64 s	169 s

Table 16: Statistics on Algorithms for simplifying expressions

Alg	iEA — E3				iEA — E5				iE — E3			
ℓ	t_{avg}	t_{med}	$t_{9/10}$	t_{max}	t_{avg}	t_{med}	$t_{9/10}$	t_{max}	t_{avg}	t_{med}	$t_{9/10}$	t_{max}
10	$300\,\mu$	$153\,\mu$	$415\,\mu$	$11.2\,m$	$272\,\mu$	$157\,\mu$	$529\,\mu$	$6.14\,m$	$242\,\mu$	$155\,\mu$	$398\,\mu$	$6.12\,m$
20	$585\,\mu$	$400\,\mu$	$1\,m$	$5.08\,m$	$607\,\mu$	$411\,\mu$	$1.22\,m$	$5.18\,m$	$546\,\mu$	$400\,\mu$	$1.18\,m$	$4.94\,m$
40	$1.64\,m$	$962\,\mu$	$3.36\,m$	$10.4\,m$	$1.63\,m$	$1.01\,m$	$3.60\,m$	$10.6\,m$	$1.34\,m$	$871\,\mu$	$2.63\,m$	$7.85\,m$
80	$2.88\,m$	$1.83\,m$	$5.68\,m$	$18.5\,m$	$3.03\,m$	$2\,m$	$5.82\,m$	$24.3\,m$	$2.60\,m$	$1.77\,m$	$5.48\,m$	$15.1\,m$
160	$8.83\,m$	$3.96\,m$	$15.3\,m$	$175\,m$	$9.21\,m$	$5\,m$	$16.1\,m$	$133\,m$	$8.62\,m$	$4.10\,m$	$14.1\,m$	$177\,m$
320	$29.8\,m$	$9.24\,m$	$58.7\,m$	$856\,m$	$47.7\,m$	$9.70\,m$	$75.7\,m$	$2.30\,s$	$30.6\,m$	$9.73\,m$	$56.1\,m$	$816\,m$
640	$420\,m$	$19.1\,m$	$118\,m$	$31.9\,s$	$131\,m$	$20.5\,m$	$113\,m$	$2.32\,s$	$394\,m$	$19.1\,m$	$98\,m$	$30.8\,s$
1280	$540\,m$	$62.3\,m$	$321\,m$	$35.9\,s$	$1\,s$	$59.8\,m$	$477\,m$	$74.6\,s$	$625\,m$	$59.8\,m$	$409\,m$	$28.7\,s$
2560	$3.45\,s$	$126\,m$	$1.99\,s$	$87.2\,s$	$5.01\,s$	$157\,m$	$1.88\,s$	$195\,s$	$34.3\,s$	$140\,m$	$2.37\,s$	$3110\,s$

5.3.3 Using the algorithms to simplify expressions

6 Conclusion

References

[1] Valentin M. Antimirov. Rewriting regular inequalities (extended abstract). In Horst Reichel, editor, *Fundamentals of Computation Theory, 10th International Symposium, FCT '95, Dresden, Germany, August 22-25, 1995, Proceedings*, volume 965 of *Lecture Notes in Computer Science*, pages 116–125. Springer, 1995.

[2] Valentin M. Antimirov. Partial derivatives of regular expressions and finite automaton constructions. *Theor. Comput. Sci.*, 155(2):291–319, 1996.

[3] Valentin M. Antimirov and Peter D. Mosses. Rewriting extended regular expressions. *Theoretical Computer Science*, 143:195–209, 1994.

[4] Filippo Bonchi and Damien Pous. Checking NFA equivalence with bisimulations up to congruence. In Roberto Giacobazzi and Radhia Cousot, editors, *The 40th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL '13, Rome, Italy - January 23 - 25, 2013*, pages 457–468. ACM, 2013.

[5] Janusz A. Brzozowski. Derivatives of regular expressions. *J. ACM*, 11(4):481–494, 1964.

[6] Baudouin Le Charlier. Another look at derivatives of regular expressions, with an experimental evaluation. *SSRN*, 2023.

[7] Baudouin Le Charlier. A program that simplifies regular expressions (tool paper). *CoRR*, abs/2307.06436, 2023.

[8] Baudouin Le Charlier and Mêton Mêton Atindehou. A data structure to handle large sets of equal terms. In James H. Davenport and Fadoua Ghourabi, editors, *7th International Symposium on Symbolic Computation in Software Science, SCSS 2016, Tokyo, Japan, March 28-31, 2016*, volume 39 of *EPiC Series in Computing*, pages 81–94. EasyChair, 2016.

[9] Bernard A. Galler and Michael J. Fischer. An improved equivalence algorithm. *Commun. ACM*, 7(5):301–303, 1964.

[10] Stefan Kahrs and Colin Runciman. Simplifying regular expressions further. *J. Symb. Comput.*, 109:124–143, 2022.

Table 17: Statistics on Algorithms iEADFS and iEAO when $E_1 \leq E_2$

	Algorithm iADFS					Algorithm iEAO				
ℓ	t_{mean}	t_{max}	t_{deriv}	t_{subs}	$\sharp R$	t_{mean}	t_{max}	t_{deriv}	t_{subs}	$\sharp R$
10	$154\,\mu$	$2.79\,m$	$87.8\,\mu$	$3.12\,\mu$	3	$174\,\mu$	$2.54\,m$	$99.3\,\mu$	$2.89\,\mu$	2
20	$171\,\mu$	$466\,\mu$	$79.3\,\mu$	$6.35\,\mu$	6	$259\,\mu$	$2.92\,m$	$160\,\mu$	$6.44\,\mu$	5
40	$355\,\mu$	$2.99\,m$	$206\,\mu$	$19\,\mu$	13	$349\,\mu$	$2.99\,m$	$219\,\mu$	$12.5\,\mu$	11
80	$600\,\mu$	$1.39\,m$	$312\,\mu$	$42.6\,\mu$	38	$589\,\mu$	$1.44\,m$	$362\,\mu$	$27.7\,\mu$	28
160	$1.03\,m$	$3.74\,m$	$546\,\mu$	$95.3\,\mu$	95	$1.22\,m$	$3.96\,m$	$862\,\mu$	$53.6\,\mu$	73
320	$3.18\,m$	$10.3\,m$	$1.42\,m$	$696\,\mu$	447	$3.86\,m$	$13.5\,m$	$3.01\,m$	$154\,\mu$	280
640	$21.6\,m$	$208\,m$	$5.99\,m$	$10.2\,m$	$2.90\,K$	$25\,m$	$101\,m$	$21.2\,m$	$927\,\mu$	$1.63\,K$
1280	$391\,m$	$5.60\,s$	$52\,m$	$290\,m$	$33.3\,K$	$189\,m$	$925\,m$	$166\,m$	$9.28\,m$	$14.2\,K$
2560	$6.49\,s$	$10\,s$	$570\,m$	$5.50\,s$	$314\,K$	$3.29\,s$	$10\,s$	$2.92\,s$	$200\,m$	$155\,K$

A Proofs

A.1 The relation $\sim_\oplus$ fulfills the conditions required by Theorem 3.3.1

Proof

When Algorithm eQ ($\sim_\oplus$) terminates, returning *true*, the following postconditions hold.

1. $\forall\langle F, F'\rangle \in R_{dev}: \forall x \in \mathit{Letter}: D_x\, F \sim_\oplus D_x\, F'$
2. $\forall\langle F, F'\rangle \in R_{dev}: o_F = o_{F'}$

We have to show that the quantification $\forall\langle F, F'\rangle \in R_{dev}$ can be generalized to $\forall\langle F, F'\rangle : F \sim_\oplus F'$. Let us fix two expressions F and F' such that $F \sim_\oplus F'$. By definition of $\sim_\oplus$, there exist expressions $F_0, \ldots, F_n$, $X_0, \ldots, X_{n-1}$, $Y_0, \ldots, Y_{n-1}$ ($n \geq 0$), and $F'_0, \ldots, F'_{n'}$, $X'_0, \ldots, X'_{n'-1}$, $Y'_0, \ldots, Y'_{n'-1}$ ($n' \geq 0$), such that $F_n = F'_{n'}$, and

$$\begin{array}{lllll} F = F_0, & \langle X_i, Y_i\rangle \in R_{dev}, & X_i \subseteq F_i, & F_{i+1} = F_i \oplus Y_i, & (0 \leq i < n) \\ F' = F'_0, & \langle X'_i, Y'_i\rangle \in R_{dev}, & X'_i \subseteq F'_i, & F'_{i+1} = F'_i \oplus Y'_i, & (0 \leq i < n') \end{array}$$

We can prove by induction on i ($0 \leq i \leq n$) that $D_x\, F \sim_\oplus D_x\, F_i$. The result is clear for $i = 0$. Let $i \geq 0$. Then, $D_x\, F_{i+1} = (D_x\, F_i \oplus D_x\, Y_i) \sim_\oplus (D_x\, F_i \oplus D_x\, X_i)$ (because $\langle X_i, Y_i\rangle \in R_{dev}$ implies $D_x\, X_i \sim_\oplus D_x\, Y_i$). But the condition $X_i \subseteq F_i$ implies that $D_x\, F_i \oplus D_x\, X_i = D_x\,(F_i \oplus X_i) = D_x\, F_i$. Therefore, $D_x\, F_{i+1} = (D_x\, F_i \oplus D_x\, Y_i) \sim_\oplus D_x\, F_i \sim_\oplus D_x\, F$. So, we have: $D_x\, F \sim_\oplus D_x\, F_n$.
In the same way, we can prove that $D_x\, F' \sim_\oplus D_x\, F'_{n'}$. But, since $F_n = F'_{n'}$, we can conclude that $D_x\, F \sim_\oplus D_x\, F'$ by transitivity of $\sim_\oplus$.
We also have to prove, for the same F and F', that $o_F = o_{F'}$. It is easy to show that $o_F = o_{F_i}$ for the expressions F_i, above. Assuming it is true for i, since $o_{X_i} = o_{Y_i}$ and $X_i \subseteq F_i$, we can conclude that $o_{Y_i} \leq o_{F_i}$. Therefore, $o_{F_{i+1}} = o_{F_i}$, and, finally, $o_F = o_{F'}$.

(End of proof)

Table 18: Statistics on Algorithms iEA, iEADFS and iEAO when $E_1 \leq E_2$

	Algorithm iEA				Algorithm iEADFS				Algorithm iEAO			
ℓ	$\sharp subst$	$\sharp subs$	$\sharp s_a$	$\sharp s_b$	$\sharp subst$	$\sharp subs$	$\sharp s_a$	$\sharp s_b$	$\sharp subst$	$\sharp subs$	$\sharp s_a$	$\sharp s_b$
10	0	3	0	0	0	3	0	0	0	3	0	0
20	2	6	0	1	2	6	0	1	1	6	0	0
40	10	17	0	3	10	17	1	4	3	13	0	1
80	61	53	3	14	59	52	4	14	14	33	1	4
160	273	141	8	49	255	143	10	48	41	87	2	15
320	$1.80\,K$	493	35	187	$3.09\,K$	699	52	252	263	336	13	56
640	$15.1\,K$	$2.18\,K$	136	922	$54.4\,K$	$4.72\,K$	343	$1.82\,K$	$1.64\,K$	$2\,K$	61	367
1280	$213\,K$	$10.3\,K$	627	$4.43\,K$	$1.93\,M$	$56.6\,K$	$3.16\,K$	$23.2\,K$	$21.5\,K$	$18.1\,K$	609	$3.92\,K$
2560	$1.86\,M$	$39.9\,K$	$2.55\,K$	$18.5\,K$	$34.9\,M$	$517\,K$	$30.1\,K$	$203\,K$	$355\,K$	$207\,K$	$7.55\,K$	$47.5\,K$

Table 19: Statistics on Algorithms iECADR and ECADR

	iECADR ($E_1 \not\leq E_2$)			ECADR ($E_1 \not\leq E_2$)			iECADR ($E_1 \leq E_2$)			ECADR ($E_1 = E_2$)		
ℓ	t_{mean}	t_{deriv}	$\sharp R$	t_{mean}	t_{deriv}	$\sharp R$	t_{mean}	t_{deriv}	$\sharp R$	t_{mean}	t_{deriv}	$\sharp R$
10	$256\,\mu$	$145\,\mu$	5	$214\,\mu$	$120\,\mu$	3	$224\,\mu$	$76.9\,\mu$	5	$244\,\mu$	$127\,\mu$	2
20	$281\,\mu$	$143\,\mu$	9	$272\,\mu$	$130\,\mu$	7	$298\,\mu$	$140\,\mu$	10	$267\,\mu$	$119\,\mu$	7
40	$533\,\mu$	$318\,\mu$	20	$482\,\mu$	$278\,\mu$	15	$650\,\mu$	$358\,\mu$	24	$507\,\mu$	$264\,\mu$	12
80	$960\,\mu$	$637\,\mu$	58	$738\,\mu$	$451\,\mu$	38	$1.34\,m$	$844\,\mu$	61	$745\,\mu$	$429\,\mu$	23
160	$2.92\,m$	$2.18\,m$	209	$1.49\,m$	$1.05\,m$	101	$3.07\,m$	$2.15\,m$	170	$1.50\,m$	$1.04\,m$	59
320	$14.4\,m$	$12.2\,m$	831	$4.15\,m$	$3.31\,m$	306	$17.2\,m$	$13.6\,m$	838	$4.15\,m$	$3.13\,m$	245
640	$177\,m$	$167\,m$	$8.75\,K$	$28.9\,m$	$26.1\,m$	$1.67\,K$	$185\,m$	$163\,m$	$6.82\,K$	$20.4\,m$	$17.3\,m$	888
1280	$3.59\,s$	$3.48\,s$	$80.8\,K$	$226\,m$	$216\,m$	$10.1\,K$	$4\,s$	$3.79\,s$	$81.6\,K$	$289\,m$	$265\,m$	$11\,K$
2560	$9.83\,s$	$9.72\,s$	$113\,K$	$6.33\,s$	$6.16\,s$	$129\,K$	$10\,s$	$10\,s$	$89.4\,K$	$5.89\,s$	$5.56\,s$	$121\,K$

Table 20: Statistics on Algorithms E1, E2 and E3 when $E_1 = E_2$

	Algorithm E1			Algorithm E2				Algorithm E3			
ℓ	t_{mean}	t_{deriv}	$\sharp R$	t_{mean}	t_{deriv}	t_{subst}	$\sharp R$	t_{mean}	t_{deriv}	t_{subst}	$\sharp R$
10	$177\,\mu$	$97.3\,\mu$	3	$299\,\mu$	$115\,\mu$	$23.2\,\mu$	6	$205\,\mu$	$51.4\,\mu$	$8.46\,\mu$	3
20	$203\,\mu$	$115\,\mu$	5	$470\,\mu$	$145\,\mu$	$54.8\,\mu$	9	$297\,\mu$	$117\,\mu$	$22.6\,\mu$	5
40	$394\,\mu$	$211\,\mu$	9	$906\,\mu$	$228\,\mu$	$144\,\mu$	15	$773\,\mu$	$208\,\mu$	$81.8\,\mu$	9
80	$564\,\mu$	$384\,\mu$	19	$1.66\,m$	$404\,\mu$	$424\,\mu$	23	$1.31\,m$	$347\,\mu$	$229\,\mu$	16
160	$1.14\,m$	$851\,\mu$	52	$4.66\,m$	$923\,\mu$	$1.59\,m$	45	$4.24\,m$	$663\,\mu$	$1.19\,m$	36
320	$3.70\,m$	$2.99\,m$	234	$18.2\,m$	$2.36\,m$	$7.85\,m$	87	$20\,m$	$1.80\,m$	$7.90\,m$	79
640	$20\,m$	$17.7\,m$	873	$60\,m$	$7.20\,m$	$27.3\,m$	152	$281\,m$	$5.02\,m$	$105\,m$	216
1280	$284\,m$	$270\,m$	$11\,K$	$195\,m$	$18.7\,m$	$89.6\,m$	290	$1.08\,s$	$14.6\,m$	$559\,m$	478
2560	$6\,s$	$5.80\,s$	$114\,K$	$800\,m$	$70.8\,m$	$362\,m$	507	$1.90\,s$	$40.7\,m$	$768\,m$	611

Table 21: Statistics on Algorithms E and EOPT when $E_1 \not\leq E_2$

	Algorithm E					Algorithm EOPT					
ℓ	t_{mean}	$t_{9/10}$	t_{max}	t_{deriv}	$\sharp R$	t_{mean}	$t_{9/10}$	t_{max}	t_{deriv}	t_{subst}	$\sharp R$
10	$135\,\mu$	$137\,\mu$	$2.57\,m$	$84\,\mu$	1	$89.3\,\mu$	$133\,\mu$	$257\,\mu$	$34.3\,\mu$	$0.43\,\mu$	1
20	$111\,\mu$	$159\,\mu$	$276\,\mu$	$57.1\,\mu$	1	$136\,\mu$	$242\,\mu$	$389\,\mu$	$60.5\,\mu$	$2.74\,\mu$	1
40	$194\,\mu$	$272\,\mu$	$2.56\,m$	$126\,\mu$	1	$210\,\mu$	$350\,\mu$	$2.55\,m$	$80.2\,\mu$	$3.15\,\mu$	1
80	$197\,\mu$	$342\,\mu$	$849\,\mu$	$119\,\mu$	2	$214\,\mu$	$436\,\mu$	$936\,\mu$	$117\,\mu$	$4.52\,\mu$	2
160	$271\,\mu$	$434\,\mu$	$1.78\,m$	$171\,\mu$	2	$357\,\mu$	$709\,\mu$	$3.48\,m$	$174\,\mu$	$23.3\,\mu$	2
320	$301\,\mu$	$437\,\mu$	$2.04\,m$	$191\,\mu$	2	$369\,\mu$	$516\,\mu$	$4.61\,m$	$190\,\mu$	$27.1\,\mu$	2
640	$237\,\mu$	$518\,\mu$	$651\,\mu$	$129\,\mu$	2	$270\,\mu$	$560\,\mu$	$825\,\mu$	$133\,\mu$	$5.10\,\mu$	2
1280	$341\,\mu$	$813\,\mu$	$2.42\,m$	$222\,\mu$	2	$386\,\mu$	$797\,\mu$	$3.26\,m$	$215\,\mu$	$15.4\,\mu$	2
2560	$350\,\mu$	$509\,\mu$	$3.29\,m$	$221\,\mu$	2	$369\,\mu$	$612\,\mu$	$3.29\,m$	$216\,\mu$	$4.97\,\mu$	2

Table 22: Statistics on Algorithms EA and EOPT when $E_1 = E_2$

	Algorithm EA						Algorithm EOPT					
ℓ	t_{mean}	$t_{9/10}$	t_{max}	t_{deriv}	t_{subst}	$\sharp R$	t_{mean}	$t_{9/10}$	t_{max}	t_{deriv}	t_{subst}	$\sharp R$
10	$195\,\mu$	$326\,\mu$	$2.71\,m$	$48.6\,\mu$	$2.48\,\mu$	6	$141\,\mu$	$242\,\mu$	$455\,\mu$	$46.9\,\mu$	$3.88\,\mu$	3
20	$201\,\mu$	$284\,\mu$	$513\,\mu$	$96\,\mu$	$5.89\,\mu$	13	$256\,\mu$	$430\,\mu$	$570\,\mu$	$116\,\mu$	$8.18\,\mu$	5
40	$395\,\mu$	$567\,\mu$	$3.22\,m$	$232\,\mu$	$11.5\,\mu$	28	$498\,\mu$	$725\,\mu$	$3.17\,m$	$252\,\mu$	$24.1\,\mu$	8
80	$631\,\mu$	$1\,m$	$1.84\,m$	$330\,\mu$	$22.9\,\mu$	63	$826\,\mu$	$1.14\,m$	$2.69\,m$	$342\,\mu$	$87.9\,\mu$	13
160	$1.24\,m$	$1.80\,m$	$3.67\,m$	$618\,\mu$	$56.6\,\mu$	151	$2.09\,m$	$3.36\,m$	$9.02\,m$	$646\,\mu$	$507\,\mu$	26
320	$2.90\,m$	$4.59\,m$	$9.94\,m$	$1.27\,m$	$330\,\mu$	532	$5.69\,m$	$8.92\,m$	$21\,m$	$1.43\,m$	$1.90\,m$	51
640	$9.96\,m$	$16.6\,m$	$30.1\,m$	$4.10\,m$	$1.71\,m$	$1.56\,K$	$25.7\,m$	$48.8\,m$	$92.9\,m$	$4.31\,m$	$11.7\,m$	97
1280	$61.9\,m$	$125\,m$	$805\,m$	$14.8\,m$	$24.6\,m$	$6.55\,K$	$81.7\,m$	$123\,m$	$214\,m$	$9.52\,m$	$39.5\,m$	185
2560	$870\,m$	$1.49\,s$	$10\,s$	$117\,m$	$425\,m$	$33.3\,K$	$341\,m$	$489\,m$	$1.36\,s$	$25.1\,m$	$186\,m$	340

Table 23: Simplification of Regular Expressions with Algorithms iEA and EOPT

			Use of IEA				Use of EOPT			
ℓ	t_{mean}	ℓ_{sim}	n_{true}	t_{true}	n_{false}	t_{false}	n_{true}	t_{true}	n_{false}	t_{false}
10	$373\,\mu$	5	114	$645\,\mu$	384	$3.29\,m$	21	$266\,\mu$	344	$4.27\,m$
20	$696\,\mu$	9	383	$1.74\,m$	$1.37\,K$	$7.81\,m$	57	$539\,\mu$	$1.14\,K$	$11.5\,m$
40	$1.29\,m$	17	991	$2.85\,m$	$3.74\,K$	$15.6\,m$	172	$2.75\,m$	$3.19\,K$	$27.7\,m$
80	$2.24\,m$	24	$2.17\,K$	$6.41\,m$	$8.13\,K$	$31.4\,m$	378	$5.34\,m$	$7.11\,K$	$59.6\,m$
160	$4.37\,m$	40	$4.52\,K$	$16\,m$	$17\,K$	$63.3\,m$	772	$23.1\,m$	$14.5\,K$	$157\,m$
320	$10.9\,m$	45	$8.85\,K$	$37.9\,m$	$33.9\,K$	$158\,m$	$1.56\,K$	$141\,m$	$29\,K$	$448\,m$
640	$23.1\,m$	53	$17.3\,K$	$104\,m$	$65.9\,K$	$284\,m$	$2.91\,K$	$354\,m$	$56.7\,K$	$1.06\,s$
1280	$58.2\,m$	68	$33.4\,K$	$278\,m$	$130\,K$	$554\,m$	$5.85\,K$	$2.08\,s$	$114\,K$	$2.07\,s$
2560	$112\,m$	58	$64\,K$	$624\,m$	$249\,K$	$987\,m$	$11\,K$	$3.88\,s$	$213\,K$	$4.36\,s$

Table 24: Simplification of Regular Expressions with Algorithms iEA and EA

			Use of IEA				Use of EA			
ℓ	t_{mean}	ℓ_{sim}	n_{true}	t_{true}	n_{false}	t_{false}	n_{true}	t_{true}	n_{false}	t_{false}
10	$352\,\mu$	5	113	$605\,\mu$	374	$2.88\,m$	21	$244\,\mu$	337	$3.56\,m$
20	$677\,\mu$	9	387	$1.63\,m$	$1.38\,K$	$6.59\,m$	58	$469\,\mu$	$1.14\,K$	$9.32\,m$
40	$1.24\,m$	17	$1.01\,K$	$2.90\,m$	$3.81\,K$	$14.2\,m$	180	$1.94\,m$	$3.26\,K$	$24.7\,m$
80	$2.08\,m$	25	$2.18\,K$	$6.89\,m$	$8.20\,K$	$30.9\,m$	371	$4.69\,m$	$7.13\,K$	$49\,m$
160	$4.26\,m$	41	$4.57\,K$	$15.4\,m$	$17.2\,K$	$95.1\,m$	779	$14.1\,m$	$14.6\,K$	$121\,m$
320	$7.59\,m$	45	$8.94\,K$	$34.9\,m$	$34.2\,K$	$132\,m$	$1.58\,K$	$44.9\,m$	$29.2\,K$	$241\,m$
640	$15.4\,m$	53	$17.6\,K$	$101\,m$	$67.2\,K$	$275\,m$	$2.97\,K$	$87.5\,m$	$57.8\,K$	$569\,m$
1280	$33.6\,m$	67	$34\,K$	$268\,m$	$132\,K$	$628\,m$	$5.94\,K$	$250\,m$	$116\,K$	$1.28\,s$
2560	$61.4\,m$	58	$64.8\,K$	$682\,m$	$254\,K$	$1.08\,s$	$11.2\,K$	$621\,m$	$218\,K$	$2.23\,s$

Table 25: Simplification of Regular Expressions with Algorithms iE and E

			Use of iE				Use of E			
ℓ	t_{mean}	ℓ_{sim}	n_{true}	t_{true}	n_{false}	t_{false}	n_{true}	t_{true}	n_{false}	t_{false}
10	$345\,\mu$	5	49	$613\,\mu$	502	$2.27\,m$	21	$217\,\mu$	337	$2.58\,m$
20	$613\,\mu$	9	173	$1.68\,m$	$1.79\,K$	$4.83\,m$	57	$422\,\mu$	$1.14\,K$	$5.99\,m$
40	$1.16\,m$	17	437	$3.39\,m$	$4.89\,K$	$11.9\,m$	179	$1.99\,m$	$3.23\,K$	$16\,m$
80	$1.86\,m$	25	$1.02\,K$	$10.4\,m$	$10.4\,K$	$25.7\,m$	372	$3.81\,m$	$7.11\,K$	$31.3\,m$
160	$3.51\,m$	41	$2.08\,K$	$40.1\,m$	$22\,K$	$49.4\,m$	779	$15.2\,m$	$14.6\,K$	$69.4\,m$
320	$7.44\,m$	45	$4.14\,K$	$122\,m$	$43.4\,K$	$111\,m$	$1.57\,K$	$42\,m$	$29\,K$	$154\,m$
640	$16\,m$	53	$8.07\,K$	$434\,m$	$85.3\,K$	$243\,m$	$2.93\,K$	$82\,m$	$57.2\,K$	$328\,m$
1280	$237\,m$	68	$15.2\,K$	$9.17\,s$	$172\,K$	$424\,m$	$6.03\,K$	$10.5\,s$	$117\,K$	$572\,m$
2560	$222\,m$	66	$29.8\,K$	$9.17\,s$	$333\,K$	$781\,m$	$11.5\,K$	$1.25\,s$	$222\,K$	$1\,s$

Table 26: Simplification of Regular Expressions with Algorithms iECADR and ECADR

			Use of iECADR				Use of ECADR			
ℓ	t_{mean}	ℓ_{sim}	n_{true}	t_{true}	n_{false}	t_{false}	n_{true}	t_{true}	n_{false}	t_{false}
10	$517\,\mu$	5	107	$1.63\,m$	362	$8.81\,m$	20	$8.29\,\mu$	326	$5.29\,m$
20	$899\,\mu$	9	360	$3.40\,m$	$1.32\,K$	$19.3\,m$	55	$96.1\,\mu$	$1.10\,K$	$12.6\,m$
40	$1.67\,m$	17	950	$6.58\,m$	$3.57\,K$	$45.2\,m$	161	$795\,\mu$	$3.04\,K$	$32.9\,m$
80	$3.27\,m$	24	$2.09\,K$	$16.3\,m$	$8\,K$	$115\,m$	363	$2.43\,m$	$6.92\,K$	$76.8\,m$
160	$8.40\,m$	40	$4.34\,K$	$58.8\,m$	$16.7\,K$	$334\,m$	730	$9.40\,m$	$14.1\,K$	$253\,m$
320	$27\,m$	45	$8.61\,K$	$178\,m$	$33.3\,K$	$1.47\,s$	$1.49\,K$	$26.1\,m$	$28.2\,K$	$710\,m$
640	$53.7\,m$	52	$16.9\,K$	$472\,m$	$64.6\,K$	$2.81\,s$	$2.74\,K$	$47.5\,m$	$55.2\,K$	$1.62\,s$
1280	$467\,m$	67	$33.2\,K$	$8.48\,s$	$130\,K$	$22.4\,s$	$5.75\,K$	$1.42\,s$	$112\,K$	$12\,s$
2560	$1.33\,s$	65	$66\,K$	$11.1\,s$	$259\,K$	$81.5\,s$	$11.3\,K$	$934\,m$	$218\,K$	$16\,s$

Table 27: Simplification of Regular Expressions with Algorithms iES and ES

			Use of iES				Use of ES			
ℓ	t_{mean}	ℓ_{sim}	n_{true}	t_{true}	n_{false}	t_{false}	n_{true}	t_{true}	n_{false}	t_{false}
10	$444\,\mu$	5	114	$414\,\mu$	372	$1.13\,m$	21	$194\,\mu$	337	$1.73\,m$
20	$633\,\mu$	10	390	$1.08\,m$	$1.39\,K$	$3.53\,m$	55	$392\,\mu$	$1.15\,K$	$4.74\,m$
40	$2.09\,m$	18	979	$2.95\,m$	$3.80\,K$	$15.1\,m$	170	$2.23\,m$	$3.30\,K$	$28.2\,m$
80	$2.84\,m$	28	$2.16\,K$	$5.19\,m$	$8.17\,K$	$22.9\,m$	336	$5.05\,m$	$7.39\,K$	$47\,m$
160	$4.13\,m$	47	$4.52\,K$	$10.3\,m$	$17.4\,K$	$58.2\,m$	699	$11.8\,m$	$15.3\,K$	$127\,m$
320	$8.34\,m$	62	$8.92\,K$	$22.4\,m$	$35.2\,K$	$124\,m$	$1.33\,K$	$28.2\,m$	$31.4\,K$	$301\,m$
640	$13.6\,m$	72	$17.5\,K$	$40.2\,m$	$69.5\,K$	$210\,m$	$2.60\,K$	$56.3\,m$	$62.2\,K$	$532\,m$
1280	$30.3\,m$	107	$34.4\,K$	$72.5\,m$	$138\,K$	$475\,m$	$5.25\,K$	$123\,m$	$126\,K$	$1.35\,s$
2560	$54.2\,m$	100	$65.8\,K$	$167\,m$	$265\,K$	$922\,m$	$9.84\,K$	$242\,m$	$237\,K$	$2.42\,s$

Table 28: Simplification of Regular Expressions with Algorithms iES and ES (+ memo)

			Use of iES				Use of ES			
ℓ	t_{mean}	ℓ_{sim}	n_{true}	t_{true}	n_{false}	t_{false}	n_{true}	t_{true}	n_{false}	t_{false}
10	$572\,\mu$	5	114	$676\,\mu$	372	$19.4\,m$	21	$270\,\mu$	337	$2.73\,m$
20	$954\,\mu$	10	390	$1.12\,m$	$1.39\,K$	$27.2\,m$	55	$366\,\mu$	$1.15\,K$	$4.88\,m$
40	$1.32\,m$	18	979	$1.87\,m$	$3.80\,K$	$21.8\,m$	170	$1.43\,m$	$3.30\,K$	$15.3\,m$
80	$1.85\,m$	28	$2.16\,K$	$3.98\,m$	$8.17\,K$	$31.5\,m$	336	$2.59\,m$	$7.39\,K$	$31.2\,m$
160	$3.36\,m$	47	$4.52\,K$	$9.05\,m$	$17.4\,K$	$55\,m$	699	$4.92\,m$	$15.3\,K$	$80.4\,m$
320	$5.35\,m$	62	$8.92\,K$	$14.8\,m$	$35.2\,K$	$94.4\,m$	$1.33\,K$	$9.31\,m$	$31.4\,K$	$125\,m$
640	$9.50\,m$	72	$17.5\,K$	$33.5\,m$	$69.5\,K$	$160\,m$	$2.60\,K$	$19.2\,m$	$62.2\,K$	$258\,m$
1280	$19.6\,m$	107	$34.4\,K$	$74.3\,m$	$138\,K$	$367\,m$	$5.25\,K$	$43.8\,m$	$126\,K$	$614\,m$
2560	$33.9\,m$	100	$65.8\,K$	$164\,m$	$265\,K$	$605\,m$	$9.84\,K$	$69.9\,m$	$237\,K$	$961\,m$

Table 29: Relation computed by Algorithms iEA and iEADFS

i	$E_{1\,i}$	$E_{2\,i}$
1	(a*b)*a^{4}a*	(a + b)*a(a + b)^3
2	a^{3}a*	(a + b)^3 + (a + b)*a(a + b)^3
3	a*b(a*b)*a^{4}a*	(a + b)^3 + (a + b)*a(a + b)^3
4	a^{2}a*	(a + b)^2 + (a + b)^3 + (a + b)*a(a + b)^3
5	aa*	a + b + (a + b)^2 + (a + b)^3 + (a + b)*a(a + b)^3
6	a*	1 + a + b + (a + b)^2 + (a + b)^3 + (a + b)*a(a + b)^3

Table 30: Relation computed by Algorithm iEANS

i	$E_{1\,i}$	$E_{2\,i}$
1	(a*b)*a^{4}a*	(a + b)*a(a + b)^3
2	a^{3}a*	(a + b)^3 + (a + b)*a(a + b)^3
3	a*b(a*b)*a^{4}a*	(a + b)^3 + (a + b)*a(a + b)^3
4	a^{2}a*	(a + b)^2 + (a + b)^3 + (a + b)*a(a + b)^3
5	a*b(a*b)*a^{4}a*	(a + b)^2 + (a + b)^3 + (a + b)*a(a + b)^3
6	(a*b)*a^{4}a*	(a + b)^2 + (a + b)*a(a + b)^3
7	aa*	a + b + (a + b)^2 + (a + b)^3 + (a + b)*a(a + b)^3
8	a*b(a*b)*a^{4}a*	a + b + (a + b)^2 + (a + b)^3 + (a + b)*a(a + b)^3
9	(a*b)*a^{4}a*	a + b + (a + b)^2 + (a + b)*a(a + b)^3
10	a^{3}a*	a + b + (a + b)^3 + (a + b)*a(a + b)^3
11	a*b(a*b)*a^{4}a*	a + b + (a + b)^3 + (a + b)*a(a + b)^3
12	(a*b)*a^{4}a*	a + b + (a + b)*a(a + b)^3
13	a*	1 + a + b + (a + b)^2 + (a + b)^3 + (a + b)*a(a + b)^3
14	a*b(a*b)*a^{4}a*	1 + a + b + (a + b)^2 + (a + b)^3 + (a + b)*a(a + b)^3
15	(a*b)*a^{4}a*	1 + a + b + (a + b)^2 + (a + b)*a(a + b)^3
16	a^{3}a*	1 + a + b + (a + b)^3 + (a + b)*a(a + b)^3
17	a*b(a*b)*a^{4}a*	1 + a + b + (a + b)^3 + (a + b)*a(a + b)^3
18	(a*b)*a^{4}a*	1 + a + b + (a + b)*a(a + b)^3
19	a^{2}a*	1 + (a + b)^2 + (a + b)^3 + (a + b)*a(a + b)^3
20	a*b(a*b)*a^{4}a*	1 + (a + b)^2 + (a + b)^3 + (a + b)*a(a + b)^3
21	(a*b)*a^{4}a*	1 + (a + b)^2 + (a + b)*a(a + b)^3
22	a^{3}a*	1 + (a + b)^3 + (a + b)*a(a + b)^3
23	a*b(a*b)*a^{4}a*	1 + (a + b)^3 + (a + b)*a(a + b)^3
24	(a*b)*a^{4}a*	1 + (a + b)*a(a + b)^3

Table 31: Relation computed by Algorithm iEAO

i	$E_{1\,i}$	$E_{2\,i}$
1	(a*b)*a^{4}a*	(a + b)*a(a + b)^3
2	a^{3}a* + a*b(a*b)*a^{4}a*	(a + b)^3 + (a + b)*a(a + b)^3
3	a^{2}a* + a*b(a*b)*a^{4}a*	(a + b)^2 + (a + b)^3 + (a + b)*a(a + b)^3
4	aa* + a*b(a*b)*a^{4}a*	a + b + (a + b)^2 + (a + b)^3 + (a + b)*a(a + b)^3
5	a* + a*b(a*b)*a^{4}a*	1 + a + b + (a + b)^2 + (a + b)^3 + (a + b)*a(a + b)^3

Table 32: Relation computed by Algorithm iEP

i	$E_{1\,i}$	$E_{2\,i}$
1	(a*b)*a^{4}a*	(a + b)*a(a + b)^3
2	a^{3}a* + a*b(a*b)*a^{4}a*	(a + b)^3 + (a + b)*a(a + b)^3
3	a^{2}a* + a*b(a*b)*a^{4}a*	(a + b)^2 + (a + b)^3 + (a + b)*a(a + b)^3
4	(a*b)*a^{4}a*	(a + b)^2 + (a + b)*a(a + b)^3
5	aa* + a*b(a*b)*a^{4}a*	a + b + (a + b)^2 + (a + b)^3 + (a + b)*a(a + b)^3
6	(a*b)*a^{4}a*	a + b + (a + b)^2 + (a + b)*a(a + b)^3
7	a^{3}a* + a*b(a*b)*a^{4}a*	a + b + (a + b)^3 + (a + b)*a(a + b)^3
8	(a*b)*a^{4}a*	a + b + (a + b)*a(a + b)^3
9	a* + a*b(a*b)*a^{4}a*	1 + a + b + (a + b)^2 + (a + b)^3 + (a + b)*a(a + b)^3
10	(a*b)*a^{4}a*	1 + a + b + (a + b)^2 + (a + b)*a(a + b)^3
11	a^{3}a* + a*b(a*b)*a^{4}a*	1 + a + b + (a + b)^3 + (a + b)*a(a + b)^3
12	(a*b)*a^{4}a*	1 + a + b + (a + b)*a(a + b)^3
13	a^{2}a* + a*b(a*b)*a^{4}a*	1 + (a + b)^2 + (a + b)^3 + (a + b)*a(a + b)^3
14	(a*b)*a^{4}a*	1 + (a + b)^2 + (a + b)*a(a + b)^3
15	a^{3}a* + a*b(a*b)*a^{4}a*	1 + (a + b)^3 + (a + b)*a(a + b)^3
16	(a*b)*a^{4}a*	1 + (a + b)*a(a + b)^3

Table 33: Relation computed by Algorithm EA

i	$E_{1\,i}$	$E_{2\,i}$
1	(a*b)*a^{4}a*	(a + b)*a(a + b)^3
2	a^{3}a*	(a + b)^3 + (a + b)*a(a + b)^3
3	a*b(a*b)*a^{4}a*	(a + b)^3 + (a + b)*a(a + b)^3
4	a^{2}a*	(a + b)^2 + (a + b)^3 + (a + b)*a(a + b)^3
5	aa*	a + b + (a + b)^2 + (a + b)^3 + (a + b)*a(a + b)^3
6	a*	1 + a + b + (a + b)^2 + (a + b)^3 + (a + b)*a(a + b)^3

Table 34: Relation computed by Algorithms EOPT and E2

i	$E_{1\,i}$	$E_{2\,i}$
1	35 + 41	41
2	31 + 38 + 41 + 45	38 + 41
3	30 + 37 + 38 + 41 + 45	37 + 38 + 41
4	2 + 3 + 29 + 37 + 38 + 41 + 45	2 + 3 + 37 + 38 + 41
5	1 + 2 + 3 + 28 + 37 + 38 + 41 + 45	1 + 2 + 3 + 37 + 38 + 41

Table 35: Relation computed by Algorithms E and E1

i	$E_{1\,i}$	$E_{2\,i}$
1	35 + 41	41
2	31 + 38 + 41 + 45	38 + 41
3	30 + 37 + 38 + 41 + 45	37 + 38 + 41
4	35 + 37 + 41	37 + 41
5	2 + 3 + 29 + 37 + 38 + 41 + 45	2 + 3 + 37 + 38 + 41
6	2 + 3 + 35 + 37 + 41	2 + 3 + 37 + 41
7	2 + 3 + 31 + 38 + 41 + 45	2 + 3 + 38 + 41
8	2 + 3 + 35 + 41	2 + 3 + 41
9	1 + 2 + 3 + 28 + 37 + 38 + 41 + 45	1 + 2 + 3 + 37 + 38 + 41
10	1 + 2 + 3 + 35 + 37 + 41	1 + 2 + 3 + 37 + 41
11	1 + 2 + 3 + 31 + 38 + 41 + 45	1 + 2 + 3 + 38 + 41
12	1 + 2 + 3 + 35 + 41	1 + 2 + 3 + 41
13	1 + 30 + 37 + 38 + 41 + 45	1 + 37 + 38 + 41
14	1 + 35 + 37 + 41	1 + 37 + 41
15	1 + 31 + 38 + 41 + 45	1 + 38 + 41
16	1 + 35 + 41	1 + 41

Table 36: Relation computed by Algorithm E3

i	$E_{1\,i}$	$E_{2\,i}$
1	35 + 41	41
2	31 + 35 + 38 + 41 + 45	35 + 38 + 41
3	30 + 31 + 35 + 37 + 38 + 41 + 45	31 + 35 + 37 + 38 + 41 + 45
4	2 + 3 + 29 + 30 + 31 + 35 + 37 + 38 + 41 + 45	2 + 3 + 30 + 31 + 35 + 37 + 38 + 41 + 45
5	1 + 2 + 3 + 28 + 29 + 30 + 31 + 35 + 37 + 38 + 41 + 45	1 + 2 + 3 + 29 + 30 + 31 + 35 + 37 + 38 + 41 + 45