Efficient Inclusion for a Class of XML Types with Interleaving and Counting

Dario Colazzo

Université Paris Sud, UMR CNRS 8623, Orsay F-91405 - France

Giorgio Ghelli

Dipartimento di Informatica - Università di Pisa - Italy

Carlo Sartiani

Dipartimento di Informatica - Università di Pisa - Italy

Abstract

Inclusion between XML types is important but expensive, and is much more expensive when unordered types are considered. We prove here that inclusion for XML types with interleaving and counting can be decided in polynomial time in presence of two important restrictions: no element appears twice in the same content model, and Kleene star is only applied to disjunctions of single elements.

Our approach is based on the transformation of each such content model into a set of constraints that completely characterizes the generated language. We then reduce inclusion checking to constraint implication. We exhibit a quadratic algorithm to perform inclusion checking on a RAM machine.

Key words:
PACS:

1 Introduction

XML schemas are an essential tool for the robustness of applications that involve XML data manipulation, transformation, integration, and, crucially,

Email addresses: dario.colazzo@lri.fr (Dario Colazzo), ghelli@di.unipi.it (Giorgio Ghelli), sartiani@di.unipi.it (Carlo Sartiani).

data exchange. To solve any static analysis problem that involves such types one must first be able to reason about their inclusion and equivalence.

XML schema languages are designed to describe ordered data, but they usually offer some (limited) support to deal with cases where the order among some elements is not constrained. These "unordered" mechanisms bring the language out of the well-understood realm of tree-grammars and tree-automata, and have been subject to little foundational study, with the important exception of a recent work by Gelade, Martens, and Neven [7]. Here, the authors study a wide range of schema languages, and show that the addition of interleaving (also called *shuffle*) and counting operators raises the complexity of inclusion checking from PSPACE (or EXPTIME, for Extended DTDs) to EXPSPACE. These are completeness results, hence this is really bad news.

A previous result by Mayer and Stockmeyer [11] had already shown that the inclusion of Regular Expressions with interleaving alone is complete in EXPSPACE, hence showing that counting is not essential for the high cost. Gelade et al.'s paper [7] concludes with: "It would therefore be desirable to find robust subclasses for which the basic decision problems are in PTIME". Such subclasses could be used either to design a new schema language, or to design adaptive algorithms, that use the PTIME algorithm whenever is possible, and resort to the full algorithm when needed. To this aim, it is important that (i) the subclass covers large classes of XML types used in practice, (ii) it is easy to verify whether a schema belongs to the subclass.

Our Contribution In this paper we define a class of Regular Expressions (REs) with interleaving and numerical constraints whose inclusion can be checked in polynomial time. Our class is based on the following two restrictions: each expression is *conflict-free* (or *single occurrence*) meaning that no symbol appears twice, and Kleene star is only applied to symbols or to disjunctions of symbols. We use the name "conflict-free types" for this class of REs with interleaving and counting. These restrictions are severe, but, as shown in [4] and [5], they are actually met by the vast majority of the schemas that are used in practice. [1] Polynomiality of inclusion of conflict-free types implies that XML types that use our conflict-free types as their content model can be checked for inclusion in PTIME as well; this result is well-known [7].

Our approach is based on the transformation of each type (i.e., each RE) into an equivalent set of constraints. Consider, for instance, the following type $T = (a\,[1..3] \cdot b\,[2..2]) + c\,[1..2]$, and the following properties for a word w in the language of T:

(1) lower-bound: at least one of a, b, and c appears in w;

[1] "More that 99% of the REs occurring in practical schemas", according to [5]

(2) cardinality: if a is in w, it appears 1, 2 or 3 times; if b is there, it appears twice; if c is there, it appears once or twice;
(3) upper-bound: no symbol out of $\{a, b, c\}$ is in w;
(4) exclusion: if one of a, b is in w, then c is not, and if c is in w then neither of a, b is in w;
(5) co-occurrence: if a is in w, then b is in w, and vice versa;[2]
(6) order: no occurrence of a may follow an occurrence of b.

It is easy to see that every w in T enjoys all of them. We will prove here that the opposite implication is true as well: every word that satisfies the six properties is indeed in T, i.e., that constraint set (1) – (6) is *complete* for T.

We will generalize this observation, and will associate a complete set of constraints, in the six categories above, to any conflict-free type (we will actually encode exclusion constraints as order constraints); as a consequence, type inclusion will be reduced to constraint implication. We then prove that each class of constraints can be checked independently, and provide an algorithm to verify type inclusion $T < U$ in time $O(|T| * |U| + |U|^2)$, through constraint implication.

For those classes of regular expressions that are characterized by corresponding classes of automata, the complexity of non-emptiness of binary intersection is usually lower or equal to that of inclusion, since the first is typically reduced to automata intersection, while the second is reduced to automata complement plus intersection. We prove here that things go the opposite way with our conflict-free types, since we prove intersection to be NP-complete. While the membership algorithm we defined in [9] may be regarded as automata-based, we have not been able to find a really natural class of automata for conflict-free types. This result about intersection complexity seems to suggest that such automata may be subject to surprising properties.

The ability to transform a type into a complete set of constraints expressed in a limited variable-free logic is used here to design an efficient inclusion algorithm. We used the same characterization to define a linear-time membership algorithm for the same class of types in [9] (membership is NP-complete for REs with interleaving). We also believe that the constraints approach could be useful for other tasks, such as inclusion of general REs with interleaving into conflict-free types, or path containment under a DTD based on our type language, but we leave this for future work.

Complexity Model To study the complexity of our algorithms we base our analysis on the RAM (*Random Access Machine*) model. As usual, we assume

[2] The term *co-occurrence constraint* has an unrelated meaning in [1]; we use it as in [12].

that read/write operations on the RAM memory and on the input/output device are performed in unit time, as well as comparisons and arithmetic operations. Under these hypotheses, our algorithms, expressed in a Pascal-like pseudocode, can be simulated by RAM programs with the same asymptotic complexity.

Paper Outline The paper is structured as follows. Section 2 describes the data model, the type language, and the constraint language we are using. Section 3 shows how types can be characterized in terms of constraints, and proves both correctness and completeness of this characterization. Section 4, then, shows how the constraint characterization of Section 3 can be exploited to derive a correct and complete inclusion-checking algorithm. In Section 5 we show that intersection is NP-complete. In Sections 6 and 7, finally, we briefly revise some related works and draw our conclusions.

2 Type Language and Constraint Language

2.1 The Type Language

Gelade, Martens, and Neven showed that, if inclusion for a given class of regular expressions with interleaving and numerical constraints is in the complexity class $\mathcal{C}$, and $\mathcal{C}$ is *closed under positive reductions* (a property enjoyed by PTIME), then the complexity of inclusion for DTDs and single-type EDTDs that use the same class of regular expressions is in $\mathcal{C}$ too [10,7]. Hence, we can focus our study on a class of regular expression over strings, and our PTIME result will immediately imply the same complexity for the inclusion problem of the corresponding classes of DTDs and single-type EDTDs. Single-type EDTDs are the theoretical counterpart of XML Schema definitions (see [7]).

We adopt the usual definitions for string concatenation $w_1 \cdot w_2$, and for the concatenation of two languages $L_1 \cdot L_2$. The *shuffle*, or *interleaving*, operator $w_1 \& w_2$ is also standard, and is defined as follows.

Definition 1 ($v \& w,\ L_1 \& L_2$) *Given a finite alphabet Σ, the shuffle set of two words $v, w \in \Sigma^*$, or two languages $L_1, L_2 \subseteq \Sigma^*$, is defined as follows; notice that each v_i or w_i may be the empty string ϵ.*

$$\begin{aligned} v \& w \quad &=_{def} \{v_1 \cdot w_1 \cdot \ldots \cdot v_n \cdot w_n \\ &\qquad\qquad \mid v_1 \cdot \ldots \cdot v_n = v,\ w_1 \cdot \ldots \cdot w_n = w,\ v_i \in \Sigma^*,\ w_i \in \Sigma^*,\ n > 0\} \\ L_1 \& L_2 &=_{def} \textstyle\bigcup_{w_1 \in L_1,\ w_2 \in L_2} w_1 \& w_2 \end{aligned}$$

Example 2 $(ab)\&(XY)$ *contains the permutations of* $abXY$ *where* a *comes before* b *and* X *comes before* Y*:*

$$(ab)\&(XY) = \{abXY, aXbY, aXYb, XabY, XaYb, XYab\}$$

When $v \in w_1\&w_2$, we say that v is a shuffle of w_1 and w_2; for example, $w_1\cdot w_2$ and $w_2\cdot w_1$ are shuffles of w_1 and w_2.

We define $\mathbb{N}_* = \mathbb{N} \cup \{*\}$, and extend the standard order among naturals with $n \leq *$ for each $n \in \mathbb{N}_*$. We consider the following type language for strings over an alphabet Σ:

$$T ::= \epsilon \ \mid\ a\,[m..n] \ \mid\ T + T \ \mid\ T\cdot T \ \mid\ T\&T \ \mid\ T!$$

where: $a \in \Sigma$, $m \in (\mathbb{N} \setminus \{0\})$, $n \in (\mathbb{N}_* \setminus \{0\})$, $n \geq m$, and, for any $T!$, at least one of the subterms of T has shape $a\,[m..n]$.

Note that expressions like $a\,[0..n]$ are not allowed due to the condition on m; of course, the type $a\,[0..n]$ can be equivalently represented by $a\,[1..n] + \epsilon$. The type $T!$ denotes $[\![T]\!]\setminus\{\epsilon\}$. The presence of an $a\,[m..n]$ subterm in $T!$ guarantees that T contains at least one word that is different from ϵ, hence $T!$ is never empty, hence no type in our language is empty (Lemma 4).

Definition 3 **($S(w), S(T), \mathit{Atoms}(T)$)** *For any string* w*,* $S(w)$ *is the set of all symbols appearing in* w*. For any type* T*,* $\mathit{Atoms}(T)$ *is the set of all atoms* $a\,[m..n]$ *appearing in* T*, and* $S(T)$ *is the set of all symbols appearing in* T*.*

The semantics of types is defined as follows ($|w|$ denotes the length of w).

$$\begin{aligned}
[\![\epsilon]\!] &= \{\epsilon\} \\
[\![a\,[m..n]]\!] &= \{w \mid S(w) = \{a\},\ m \leq |w| \leq n\} \\
[\![T_1 + T_2]\!] &= [\![T_1]\!] \cup [\![T_2]\!] \\
[\![T_1 \cdot T_2]\!] &= [\![T_1]\!] \cdot [\![T_2]\!] \\
[\![T_1\&T_2]\!] &= [\![T_1]\!]\&[\![T_2]\!] \\
[\![T!]\!] &= [\![T]\!] \setminus \{\epsilon\}
\end{aligned}$$

The following Lemma can be easily proved by structural induction.

Lemma 4 (Not empty) *For any type* T*:*

$$[\![T]\!] \neq \emptyset \tag{1}$$

$$a\,[m..n] \in \mathit{Atoms}(T) \ \Rightarrow \exists w \in [\![T]\!].\ a \in S(w) \tag{2}$$

It is worth noting that our type system lacks general repetition types T^* or T^+. Repetition types are here generalized by counting, which, is, however, restricted to symbols, so that, for example, types like $(a{\cdot}b)^*$ cannot be expressed. However, it has been found that DTD and XSD (XML Schema Definition) schemas use repetition almost exclusively as a^{op} or as $(a + \ldots + z)^{\mathrm{op}}$ (see [5]), which can be immediately translated to our system. For instance, $(a+\ldots+z)^*$ can be expressed as $(a^*\&\ldots\&z^*)$, where a^* is a shortcut for $a\,[1..*]+\epsilon)$, while $(a + \ldots + z)^+$ can be expressed as $(a^*\&\ldots\&z^*)!$.

Restrictions on types, similar to those we consider here, have already been considered in some previous works. Bex et al. [4] studied *simple expressions* (introduced by Martens et al. in [10]), which are concatenation of factors $(a_1^{\mathrm{op}_1} + \ldots + a_n^{\mathrm{op}_n})^{\mathrm{op}}$. Simple expressions can be trivially translated to our systems, and the authors measured that a 97% fraction of content models in XSD schemas consist of such expressions; however, they did not consider conflict-freedom. Chain Regular Expressions, defined in [5], is a similar class of expression, which satisfy conflict-freedom, hence all of them can be translated to our system, and the authors report measuring that a 99% of content models fall in this class.

We will use $\otimes$ to range over $\cdot$ and & when we need to specify common properties, such as, for example: $[\![T \otimes \epsilon]\!] = [\![\epsilon \otimes T]\!] = [\![T]\!]$. Some types contain the empty string ϵ, and are characterized as follows ($\mathrm{N}(T)$ is read as "T is nullable").

Definition 5 $\mathrm{N}(T)$ *is a predicate on types, defined as follows:*

$$\begin{aligned}
\mathrm{N}(\epsilon) &= \mathit{true}\\
\mathrm{N}(a\,[m..n]) &= \mathit{false}\\
\mathrm{N}(T!) &= \mathit{false}\\
\mathrm{N}(T + T') &= \mathrm{N}(T)\ \ \mathit{or}\ \ \mathrm{N}(T')\\
\mathrm{N}(T \otimes T') &= \mathrm{N}(T)\ \ \mathit{and}\ \ \mathrm{N}(T')
\end{aligned}$$

The following Lemma is immediate by structural induction in the definition of $\mathrm{N}(T)$.

Lemma 6 $\epsilon \in [\![T]\!]$ *iff* $\mathrm{N}(T)$.

We can now define the notion of *conflict-free* types.

Definition 7 (Conflict-free types) *A type T is* conflict-free *iff for each subexpression $(U + V)$ or $(U \otimes V)$: $S(U) \cap S(V) = \emptyset$.*

Equivalently, a type T is conflict-free if, for any two distinct subterms $a\,[m..n]$

and $a' [m'..n']$ that occur in T, a is different from a'.

Example 8 *Consider the following type:* $(a [1..1] \& b [1..1]) + (a [1..1] \& c [1..1])$*. This type generates the language* $\{ab, ba, ac, ca\}$*. This type is not conflict-free, since* $S(a [1..1] \& b [1..1]) \cap S(a [1..1] \& c [1..1]) = \{a\} \neq \emptyset$*.*

Consider now $a [1..1] \& (b [1..1] + c [1..1])$*; it generates the same language, but is conflict-free since* $a [1..1]$ *and* $(b [1..1] + c [1..1])$ *have no common symbols.*

Conflict-free DTDs have been considered many times before, because of their good properties and because of the high percentage of actual schemas that satisfy this constraint (see Section 6).

Hereafter, we will silently assume that every type is conflict-free.

2.2 The Constraint Language

We verify inclusion between T and U by translating them into constraint sets C_T and C_U and then by verifying that C_T implies C_U. Constraints are expressed using the following logic, where $a, b \in \Sigma$ and $A, B \subseteq \Sigma$, $m \in (\mathbb{N} \backslash \{0\})$, $n \in (\mathbb{N}_* \setminus \{0\})$, and $n \geq m$:

$$F \ ::= \ A^+ \ | \ A^+ \mapsto B^+ \ | \ a?[m..n] \ | \ \text{upper}(A) \ | \ a \prec b \ | \ F \wedge F' \ | \ \mathbf{true}$$

Satisfaction of a constraint F by a word w, written $w \models F$, is defined as follows.[3]

$$
\begin{array}{lcl}
w \models A^+ & \Leftrightarrow & S(w) \cap A \neq \emptyset, \text{ i.e. some } a \in A \text{ appears in } w \\
w \models A^+ \Mapsto B^+ & \Leftrightarrow & w \not\models A^+ \text{ or } w \models B^+ \\
w \models a?[m..n]\ (n \neq *) & \Leftrightarrow & \text{if } a \text{ appears in } w\text{, then it appears} \\
 & & \text{at least } m \text{ times and at most } n \text{ times} \\
w \models a?[m..*] & \Leftrightarrow & \text{if } a \text{ appears in } w\text{, then it appears} \\
 & & \text{at least } m \text{ times} \\
w \models \mathrm{upper}(A) & \Leftrightarrow & S(w) \subseteq A \\
w \models a \prec b & \Leftrightarrow & \text{there is no occurrence of } a \text{ in } w \text{ that follows} \\
 & & \text{an occurrence of } b \text{ in } w \\
w \models F_1 \wedge F_2 & \Leftrightarrow & w \models F_1 \text{ and } w \models F_2 \\
w \models \mathbf{true} & \Leftrightarrow & \text{always}
\end{array}
$$

The following special cases of the above definition are worth noticing.

$$
\begin{array}{lll}
\epsilon \not\models A^+ & \epsilon \models \mathrm{upper}(A) & \epsilon \models a?[m..n] \\
\epsilon \models a \prec b & b \models a \prec b & aba \not\models a \prec b \\
w \not\models \emptyset^+ & w \models \emptyset^+ \Mapsto A^+ & w \models \emptyset^+ \Mapsto \emptyset^+
\end{array}
$$

Observe that A^+ is monotone, i.e., $w \models A^+$ and w is a subword of w' imply that $w' \models A^+$, while $\mathrm{upper}(A)$ and $a \prec b$ are anti-monotone.

We use the following abbreviations:

$$
\begin{array}{rcl}
a^+ & =_{def} & \{a\}^+ \\
a \prec\succ b & =_{def} & (a \prec b) \wedge (b \prec a) \\
A \prec B & =_{def} & \displaystyle\bigwedge_{a \in A, b \in B} a \prec b \\
A \prec\succ B & =_{def} & \displaystyle\bigwedge_{a \in A, b \in B} a \prec\succ b
\end{array}
$$

The next propositions specify that $A \prec\succ B$ encodes mutual exclusion between sets of symbols.

[3] Notice that $A^+ \Mapsto b^+$ differs from the sibling constraint $A \Downarrow b$ of [13], since $A^+ \Mapsto b^+$ means "if one symbol of A is in w then b is in w", while $A \Downarrow b$ means "if *all* symbols of A are in w then b is in w".

Proposition 9 $w \models a \prec\succ b \;\Leftrightarrow\;$ *a and b are not both in* $S(w)$

Proposition 10 $w \models A \prec\succ B \;\Leftrightarrow w \not\models A^+ \wedge B^+$

Proof. By Proposition 9, we observe that $w \models A \prec\succ B$ holds if and only if, for each $a \in A,\ b \in B,\ \{a, b\} \not\subseteq S(w)$. This means that either $A \cap S(w) = \emptyset$ or $B \cap S(w) = \emptyset$, that is, $w \not\models A^+ \wedge B^+$. □

We extend the $S(\_)$ notation to formulas.

Definition 11 *$a \in S(F)$ if one of the following is a subterm of F: $a?[m..n]$, $a \prec b$, A^+, $A^+ \mapsto B^+$,* upper(A)*, where, in the last three cases, $a \in A$ or $a \in B$.*

The atomic operators are all mutually independent: only A^+ can force the presence of a symbol independently of any other, only $A^+ \mapsto B^+$ induces a positive correlation between the presence of two symbols, only $a?[m..n]$ can count, only upper(A) is affected by the presence of a symbol that is not in $S(F)$, and only $a \prec b$ is affected by order. However, combinations of the atomic operators can be mutually related (see Proposition 10, for example).

3 Characterization of Types as Constraints

3.1 Constraint Extraction

We first extend satisfaction from words to types, as follows.

Definition 12 $T \models F \;\Leftrightarrow\; \forall w \in [\![T]\!].\ w \models F$

We associate to each type T a formula $S^+(T)$ that tests for the presence of one of its symbols, as follows.

Definition 13 $S^+(T) \;=\; (S(T))^+$

We also define $\mathit{If}_T(F)$, which denotes either the formula F, or the formula **true**, depending on whether $\mathrm{N}(T)$ holds. We use it to express the lower-bound and co-occurrence constraints, since they only hold for non-nullable types.

Definition 14 (If-non-nullable) *$\mathit{If}_T(F)$ denotes the formula* **true** *if* $\mathrm{N}(T)$ *holds, and denotes the formula F otherwise.*

We can now endow a type T with five sets of constraints. We divide them in "nested constraints" (co-occurrence, order-and-exclusion), whose definition depends on the nesting structures of the type, and "flat constraints" (lower-bound, cardinality, and upper-bound), whose definition only depends on the leaves of the types, and on its nullability.

Definition 15 (Flat constraints)

$$\begin{array}{rrl}
\textit{Lower-bound:} & \mathit{SIf}(T) & =_{\mathit{def}}\ \mathit{If}_T(S^+(T)) \\
\textit{Cardinality:} & \mathrm{ZeroMinMax}(T) & =_{\mathit{def}}\ \bigwedge_{a[m..n]\in Atoms(T)} a?[m..n] \\
\textit{Upper-bound:} & \mathrm{upperS}(T) & =_{\mathit{def}}\ \mathrm{upper}(S(T)) \\
\textit{Flat constraints:} & \mathcal{FC}(T) & =_{\mathit{def}}\ \mathit{SIf}(T) \wedge \mathrm{ZeroMinMax}(T) \wedge \mathrm{upperS}(T)
\end{array}$$

Nested constraints are defined by induction over the type structure.

Definition 16 (Nested constraints)

Co-occurrence:

$$\begin{array}{ll}
\mathcal{CC}(T_1 + T_2) & =_{\mathit{def}}\ \mathcal{CC}(T_1) \wedge \mathcal{CC}(T_2) \\
\mathcal{CC}(T_1 \otimes T_2) & =_{\mathit{def}}\ \mathit{If}_{T_2}(S^+(T_1) \mapsto\!\!\!> S^+(T_2)) \\
 & \quad \wedge\ \mathit{If}_{T_1}(S^+(T_2) \mapsto\!\!\!> S^+(T_1)) \\
 & \quad \wedge\ \mathcal{CC}(T_1) \wedge \mathcal{CC}(T_2) \\
\mathcal{CC}(T!) & =_{\mathit{def}}\ \mathcal{CC}(T) \\
\mathcal{CC}(\epsilon) =_{\mathit{def}} \mathcal{CC}(a\,[m..n]) & =_{\mathit{def}}\ \mathbf{true}
\end{array}$$

Order and exclusion:

$$\begin{array}{ll}
\mathcal{OC}(T_1 + T_2) & =_{\mathit{def}}\ (S(T_1) \prec\succ S(T_2)) \wedge \mathcal{OC}(T_1) \wedge \mathcal{OC}(T_2) \\
\mathcal{OC}(T_1 \& T_2) & =_{\mathit{def}}\ \mathcal{OC}(T_1) \wedge \mathcal{OC}(T_2) \\
\mathcal{OC}(T_1 \cdot T_2) & =_{\mathit{def}}\ (S(T_1) \prec S(T_2)) \wedge \mathcal{OC}(T_1) \wedge \mathcal{OC}(T_2) \\
\mathcal{OC}(T!) & =_{\mathit{def}}\ \mathcal{OC}(T) \\
\mathcal{OC}(\epsilon) =_{\mathit{def}} \mathcal{OC}(a\,[m..n]) & =_{\mathit{def}}\ \mathbf{true}
\end{array}$$

Nested constraints:

$$\mathcal{NC}(T) \quad =_{\mathit{def}}\ \mathcal{CC}(T) \wedge \mathcal{OC}(T)$$

As a consequence of the above definition, nested constraints have the following property.

Proposition 17 **(**$\mathcal{NC}(T)$**)**

$$\begin{aligned}
\mathcal{NC}(T_1 + T_2) &= (S(T_1) \prec\succ S(T_2)) \wedge \mathcal{NC}(T_1) \wedge \mathcal{NC}(T_2)\\
\mathcal{NC}(T_1 \& T_2) &= (\mathit{If}_{T_2}(S^+(T_1) \Mapsto S^+(T_2))) \wedge (\mathit{If}_{T_1}(S^+(T_2) \Mapsto S^+(T_1))) \wedge\\
& \mathcal{NC}(T_1) \wedge \mathcal{NC}(T_2)\\
\mathcal{NC}(T_1 \cdot T_2) &= (S(T_1) \prec S(T_2))\\
& \wedge(\mathit{If}_{T_2}(S^+(T_1) \Mapsto S^+(T_2))) \wedge (\mathit{If}_{T_1}(S^+(T_2) \Mapsto S^+(T_1)))\\
& \wedge \mathcal{NC}(T_1) \wedge \mathcal{NC}(T_2)\\
\mathcal{NC}(T!) &= \mathcal{NC}(T)\\
\mathcal{NC}(\epsilon) &= \mathbf{true}\\
\mathcal{NC}(a\,[m..n]) &= \mathbf{true}
\end{aligned}$$

3.2 Correctness and Completeness of Constraints

We plan to prove the following theorem, that specifies that the constraint system completely captures the semantics of conflict-free types.

Theorem 18 *Given a conflict-free type T, it holds that:*

$$w \in [\![T]\!] \;\Leftrightarrow\; w \models \mathcal{FC}(T) \wedge \mathcal{NC}(T)$$

We first prove that constraints are complete, i.e., whenever w satisfies all the five groups of constraints associated with T, then $w \in [\![T]\!]$.

Proposition 19 **(**$\mathrm{ZeroMinMax}(T)$**)**

$$\begin{aligned}
&w \models \mathrm{ZeroMinMax}(T_1 + T_2) \Rightarrow w \models \mathrm{ZeroMinMax}(T_1) \wedge \mathrm{ZeroMinMax}(T_2)\\
&w \models \mathrm{ZeroMinMax}(T_1 \otimes T_2) \Rightarrow w \models \mathrm{ZeroMinMax}(T_1) \wedge \mathrm{ZeroMinMax}(T_2)
\end{aligned}$$

Proof. By definition of $\mathrm{ZeroMinMax}(T)$. □

Definition 20 (Word projection) *We define the projection of a word w onto A ($w|_A$) as the string obtained from w by removing all the symbols that are not in A.*

Proposition 21

$$w \models S^+(T_1) \;\wedge\; w \models \mathit{If}_{T_2}(S^+(T_1) \Mapsto S^+(T_2)) \;\Rightarrow\; w \models \mathit{SIf}(T_2)$$

We can now prove the crucial completeness theorem.

Theorem 22 (Completeness of constraints)

$$w \models (\mathcal{FC}(T) \wedge \mathcal{NC}(T)) \quad \Rightarrow \quad w \in [\![T]\!]$$

Proof.

For the sake of convenience, we will use ZMM-SIf(T) as a shortcut for ZeroMinMax$(T) \wedge$ *SIf*(T), so that we can rewrite the thesis as

$$w \models (\mathrm{upperS}(T) \wedge \text{ZMM-SIf}(T) \wedge \mathcal{NC}(T)) \quad \Rightarrow \quad w \in [\![T]\!]$$

We prove the following fact, by case inspection and structural induction on T.

$$w \models (\text{ZMM-SIf}(T) \wedge \mathcal{NC}(T)) \quad \Rightarrow \quad w|_{S(T)} \in [\![T]\!]$$

The theorem follows because $w \models \mathrm{upperS}(T)$ implies that $w = w|_{S(T)}$.

We first observe that $w|_{S(T)} = \epsilon$ and $w \models \mathit{SIf}(T)$ imply the thesis $w|_{S(T)} \in [\![T]\!]$. Indeed, $w|_{S(T)} = \epsilon$ implies that $w \not\models S^+(T)$, hence, the hypothesis $w \models \mathit{SIf}(T)$ implies that $\mathrm{N}(T)$ is true, which in turn implies that $\epsilon \in [\![T]\!]$, i.e. $w|_{S(T)} \in [\![T]\!]$.

Having dealt with the $w|_{S(T)} = \epsilon$ case, in the following we assume that $w|_{S(T)} = a_1 \cdot \ldots \cdot a_n$, where $n \neq 0$ (where, for each i and j, the symbol a_i may be either equal or different from a_j).

$\mathbf{T} = \epsilon$:

Trivial, as $w|_{S(\epsilon)} = \epsilon$ and $\epsilon \in [\![\epsilon]\!]$.

$\mathbf{T} = \mathbf{a}\,[\mathbf{m}..\mathbf{n}]$:

Since $\mathrm{N}(T)$ is false, $w \models$ ZMM-SIf(T) implies that $w \models \mathrm{ZeroMinMax}(T) \wedge S^+(T)$, i.e., $w \models \mathrm{ZeroMinMax}(a\,[m..n]) \wedge a^+$, i.e., $w \models a?[m..n] \wedge a^+$, hence $w|_{S(a[m..n])} \in [\![a\,[m..n]]\!]$.

$\mathbf{T} = \mathbf{T_1} + \mathbf{T_2}$:

Let $w|_{S(T)} = a_1 \cdot \ldots \cdot a_n$, and assume,without loss of generality, that $a_1 \in S(T_1)$.

By hypothesis and Proposition 17 we have that $w \models$ ZMM-SIf$(T_1 + T_2) \wedge (S(T_1) \prec\succ S(T_2)) \wedge \mathcal{NC}(T_1) \wedge \mathcal{NC}(T_2)$. As $w|_{S(T)} = a_1 \cdot \ldots \cdot a_n$ with $a_1 \in S(T_1)$, we also have that $w \models S^+(T_1)$.

This implies that $w \models \mathit{SIf}(T_1)$ (by definition of *SIf*()) and that $w \not\models S^+(T_2)$ (by Proposition 10). This, in turn, implies $w|_{S(T_1+T_2)} = w|_{S(T_1)}$ (*). By Proposition 19 and by $w \models \text{ZMM-SIf}(T_1 + T_2)$ we obtain that $w \models \text{ZeroMinMax}(T_1)$. Putting all together, $w \models \text{ZMM-SIf}(T_1) \wedge \mathcal{NC}(T_1)$.

By induction we have that $w|_{S(T_1)} \in [\![T_1]\!]$; hence, by (*), we get $w|_{S(T_1+T_2)} \in [\![T_1]\!]$, which, in turn, implies that $w|_{S(T_1+T_2)} \in [\![T_1 + T_2]\!]$.

$\mathbf{T = T_1 \cdot T_2}$:

We have two possible cases:

(1) $w|_{S(T)} = a_1 \cdot \ldots \cdot a_n$ and $a_1 \in S(T_1)$;
(2) $w|_{S(T)} = a_1 \cdot \ldots \cdot a_n$ and $a_1 \in S(T_2)$.

In both cases, by hypothesis and Proposition 17 we have that (*):

$$\begin{aligned} w \models \text{ZMM-SIf}(T_1 \cdot T_2) &\wedge (\mathit{If}_{T_2}(S^+(T_1) \Rrightarrow S^+(T_2))) \\ &\wedge (\mathit{If}_{T_1}(S^+(T_2) \Rrightarrow S^+(T_1))) \\ &\wedge (S(T_1) \prec S(T_2)) \\ &\wedge \mathcal{NC}(T_1) \wedge \mathcal{NC}(T_2) \end{aligned}$$

Case 1 ($w|_{S(T)} = a_1 \cdot \ldots \cdot a_n$ and $a_1 \in S(T_1)$).

By (*), since $w|_{S(T)} = a_1 \cdot \ldots \cdot a_n$ and $a_1 \in S(T_1)$, we have that $w \models S^+(T_1)$, which implies that $w \models \mathit{SIf}(T_1)$ (by definition of $\mathit{SIf}()$) and that $w \models \mathit{SIf}(T_2)$ (by hypothesis and by Proposition 21). By Proposition 19 we conclude that $w \models \text{ZMM-SIf}(T_1) \wedge \text{ZMM-SIf}(T_2)$.

Let us define $w_1 = w|_{S(T_1)}$ and $w_2 = w|_{S(T_2)}$. As $w \models \mathcal{NC}(T_1) \wedge \mathcal{NC}(T_2)$, by induction we obtain that $w_1 \in [\![T_1]\!]$ and $w_2 \in [\![T_2]\!]$.

By conflict-freedom, w_1 and w_2 do not contain any common symbols, hence, from the constraint $S(T_1) \prec S(T_2)$ we obtain that each symbol of w_1 precedes each symbol of w_2 in w. As a consequence, $w|_{S(T_1 \cdot T_2)} = w|_{S(T_1)} \cdot w|_{S(T_2)} = w_1 \cdot w_2$. Thus, $w|_{S(T_1 \cdot T_2)} \in [\![T_1 \cdot T_2]\!]$.

Case 2 ($w|_{S(T)} = a_1 \cdot \ldots \cdot a_n$ and $a_1 \in S(T_2)$).

By (*), since $w|_{S(T)} = a_1 \cdot \ldots \cdot a_n$ and $a_1 \in S(T_2)$, we obtain that $w \models S^+(T_2)$, which implies that $w \models \mathit{SIf}(T_1)$ (by Proposition 21) and that $w \models \mathit{SIf}(T_2)$ (by definition). By Proposition 19 we conclude that $w \models \text{ZMM-SIf}(T_1) \wedge \text{ZMM-SIf}(T_2)$. As $w \models \mathcal{NC}(T_1) \wedge \mathcal{NC}(T_2)$, by induction we obtain that $w|_{S(T_1)} \in [\![T_1]\!]$ and $w|_{S(T_2)} \in [\![T_2]\!]$.

$w \models (S(T_1) \prec S(T_2))$ and $a_1 \in S(T_2)$ imply that $w \not\models S^+(T_1)$, i.e., $w|_{S(T_1)} = \epsilon$. Hence, $w|_{S(T_1 \cdot T_2)} = w|_{S(T_2)} = \epsilon \cdot w|_{S(T_2)} = w|_{S(T_1)} \cdot w|_{S(T_2)}$. Hence, by $w|_{S(T_1)} \in [\![T_1]\!]$ and $w|_{S(T_2)} \in [\![T_2]\!]$, we conclude that $w|_{S(T_1 \cdot T_2)} \in [\![T_1 \cdot T_2]\!]$.

$\mathbf{T = T_1 \& T_2}$:

Let $w|_{S(T)} = a_1 \cdot \ldots \cdot a_n$ and assume, without loss of generality, that $a_1 \in S(T_1)$.

By hypothesis and Proposition 17 we have that:

$$\begin{aligned} w \models \text{ZMM-SIf}(T_1 \& T_2) \, & \wedge \, (\mathit{If}_{T_2}(S^+(T_1) \Rrightarrow S^+(T_2))) \\ & \wedge \, (\mathit{If}_{T_1}(S^+(T_2) \Rrightarrow S^+(T_1))) \\ & \wedge \, \mathcal{NC}(T_1) \wedge \, \mathcal{NC}(T_2) \end{aligned}$$

Since $w|_{S(T)} = a_1 \cdot \ldots \cdot a_n$, we have that $w \models S^+(T_1)$, from which we obtain that $w \models \mathit{SIf}(T_1)$ (by definition) and $w \models \mathit{SIf}(T_2)$.

By Proposition 19 it follows that $w \models \text{ZMM-SIf}(T_1) \wedge \text{ZMM-SIf}(T_2)$.

As $w \models \mathcal{NC}(T_1) \wedge \mathcal{NC}(T_2)$, by induction we obtain that $w_1 = w|_{S(T_1)} \in [\![T_1]\!]$ and that $w_2 = w|_{S(T_2)} \in [\![T_2]\!]$.

By the conflict freedom hypothesis, $S(T_1) \cap S(T_2) = \emptyset$, hence w is a shuffle of $w_1 \cdot w_2 \cdot w_3$, where the symbols in w_3 are not present in $S(T_1 \& T_2)$. As a consequence, $w|_{S(T_1 \& T_2)} \in w_1 \& w_2$, which implies that $w|_{S(T_1 \& T_2)} \in [\![T_1 \& T_2]\!]$.

$\mathbf{T = T_1!}$: By the hypothesis $w \models \mathit{SIf}(T)$ and by the fact that T is not nullable we deduce $w \neq \epsilon$ and $w \models S^+(T_1!)$, hence $w \models S^+(T_1)$. From $w \models \mathcal{NC}(T_1!)$ and $w \models \text{ZeroMinMax}(T_1!)$ we get $w \models \mathcal{NC}(T_1)$ and $w \models \text{ZeroMinMax}(T_1)$. Hence, by induction, $w|_{S(T_1)} \in [\![T_1]\!]$, and $w|_{S(T_1!)} \in [\![T_1!]\!]$ follows from $w \neq \epsilon$. □

In order to prove soundness, we use the following lemma specifying that the value of any formula F over w does not change if any letter a that is not in $S(F)$ is added or deleted from w, provided that F does not contain the upper(A) operator. Recall that upper(A) is only used to express upper-bound constraints.

Lemma 23 (Irrelevance) *Assume that* upper(A) *does not appear in F, for any A. Then, for any $B \supseteq S(F)$, and for any w:*

$$w \models F \;\Leftrightarrow\; w|_B \models F$$

Proof.

By induction on the structure of F, and by case analysis.

$F = A^+$: $w \models A^+$ iff $S(w) \cap A \neq \emptyset$: this condition is not affected by projecting a word onto $B \supseteq A$, hence $w \models F \Leftrightarrow w|_B \models F$.

$F = {A_1}^+ \mapsto\!\!> {A_2}^+$: $w \models {A_1}^+ \mapsto\!\!> {A_2}^+$ iff either $S(w) \cap A_1 = \emptyset$ or $S(w) \cap A_2 \neq \emptyset$: this condition is not affected by projecting a word onto $B \supseteq (A_1 \cup A_2)$.

$F = a \prec b$: $w \models a \prec b$ iff there is no occurrence of a in w that precedes one occurrence of b in w; this is not affected by projecting w onto $B \supseteq \{a, b\}$.

$F = a?[m..n]$: $w \models a?[m..n]$ iff, if a appears in w, then it appears at least m times and at most n times: this is not affected by projecting w onto $B \supseteq \{a\}$.

$F = F_1 \wedge F_2$: let $B \supseteq S(F)$; then, $B \supseteq S(F_1)$ and $B \supseteq S(F_2)$, hence the thesis follows by induction.

$F = \mathbf{true}$: trivial. $\square$

The soundness of our constraints is stated by Theorem 24.

Theorem 24 (Soundness)

$$w \in [\![T]\!] \Rightarrow w \models \mathcal{FC}(T) \wedge \mathcal{NC}(T)$$

Proof.

We first prove $w \in [\![T]\!] \Rightarrow w \models \mathcal{FC}(T)$.

The implication $w \in [\![T]\!] \Rightarrow w \models \text{upperS}(T)$ is immediate by structural induction on the definition of types.

For $w \in [\![T]\!] \Rightarrow w \models \mathit{SIf}(T)$, we have two cases. If $w = \epsilon$, then $w \in [\![T]\!]$ implies that $\text{N}(T)$, and therefore $\mathit{SIf}(T) = \mathbf{true}$, hence $w \models \mathit{SIf}(T)$. If $w \neq \epsilon$, then $S(w) \neq \emptyset$, hence, by $w \models \text{upperS}(T)$, w contains one symbol of $S(T)$, hence $w \models \mathit{SIf}(T)$.

For $w \in [\![T]\!] \Rightarrow w \models \text{ZeroMinMax}(T)$, we proceed by case inspection and induction on T.

$\mathbf{T} = \epsilon$: $w \in [\![T]\!]$ implies $w = \epsilon$, and, by definition, $\epsilon \models \text{ZeroMinMax}(T)$.

$\mathbf{T} = \mathbf{a}\,[\mathbf{m}..\mathbf{n}]$: Immediate.

$\mathbf{T = T_1 + T_2}$: Consider $w \in [\![T]\!]$, and assume, without loss of generality, that $w \in [\![T_1]\!]$. By induction we have $w \models \text{ZeroMinMax}(T_1)$. By $S(T_1) \cap S(T_2) = \emptyset$ and $w \models \text{upperS}(T_1)$, we also have that, for any $a\,[m..n] \in \mathit{Atoms}(T_2)$, $a \notin S(w)$, hence $w \models a?[m..n]$.

$\mathbf{T = T_1 \otimes T_2}$: Consider $w \in [\![T_1 \otimes T_2]\!]$; by definition, there exist $w_1 \in [\![T_1]\!]$ and $w_2 \in [\![T_2]\!]$ such that $w \in w_1 \& w_2$. By induction, $w_1 \models \text{ZeroMinMax}(T_1)$, i.e., for any $a\,[m..n] \in \mathit{Atoms}(T_1)$ $w_1 \models a?[m..n]$. From $S(T_1) \cap S(T_2) = \emptyset$ and $w_2 \models \text{upperS}(T_2)$, we deduce that $w|_{S(T_1)} = w_1$, hence, for any $a\,[m..n] \in \mathit{Atoms}(T_1)$, $w \models a?[m..n]$. In the same way, we prove that, for any $a\,[m..n] \in \mathit{Atoms}(T_2)$, $w \models a?[m..n]$, hence $w \models \text{ZeroMinMax}(T)$.

$\mathbf{T = T_1!}$: If $w \in [\![T]\!]$, then $w \in [\![T_1]\!]$ and $w \neq \epsilon$. By induction, $w \models \text{ZeroMinMax}(T_1)$, hence $w \models \text{ZeroMinMax}(T)$.

We now prove $w \in [\![T]\!] \Rightarrow w \models \mathcal{NC}(T)$.

We first observe that, for each T, $\epsilon \models \mathcal{NC}(T)$, because ϵ trivially satisfies any order constraint, and it also satisfies any non-trivial co-occurrence constraint $\mathit{If}_{T_2}(S^+(T_1) \mapsto S^+(T_2))$ by falsifying the hypothesis $S^+(T_1)$. This observation will be crucial in the $T_1 + T_2$ case.

We now proceed by case inspection and induction on T. The cases for ϵ and $a\,[m..n]$ are trivial.

$\mathbf{T = T_1 \& T_2}$: $\mathcal{NC}(T_1 \& T_2)$ is defined as follows.

$$(\mathit{If}_{T_2}(S^+(T_1) \mapsto S^+(T_2))) \wedge (\mathit{If}_{T_1}(S^+(T_2) \mapsto S^+(T_1))) \wedge \mathcal{NC}(T_1) \wedge \mathcal{NC}(T_2)$$

Consider $w \in [\![T]\!]$. We observe that, since $S(T_1) \cap S(T_2) = \emptyset$, we have $w|_{S(T_1)} \in [\![T_1]\!]$ and $w|_{S(T_2)} \in [\![T_2]\!]$, and therefore $w|_{S(T_1)} \models \mathcal{NC}(T_1)$ and $w|_{S(T_2)} \models \mathcal{NC}(T_2)$, by induction. Since $S(\mathcal{NC}(T_i)) \subseteq S(T_i)$, we have that $w|_{S(T_i)} \models \mathcal{NC}(T_i)$ implies $w \models \mathcal{NC}(T_i)$, for $i = 1, 2$, by Lemma 23. Similarly, $w|_{S(T_2)} \models \mathit{SIf}(T_2)$, which holds by correctness of flat constraints, implies that $w \models \mathit{SIf}(T_2)$, hence that $w \models \mathit{If}_{T_2}(S^+(T_1) \mapsto S^+(T_2))$. The constraint $w \models \mathit{If}_{T_1}(S^+(T_2) \mapsto S^+(T_1))$ is similar.

$\mathbf{T = T_1 \cdot T_2}$: This is similar to the previous case, but we also have to prove that $w \models S(T_1) \prec S(T_2)$. Assume $w \in [\![T]\!]$; then, there exist $w_1 \in [\![T_1]\!]$ and $w_2 \in [\![T_2]\!]$ such that $w = w_1 \cdot w_2$. Let $a_1 \in S(T_1)$ and $a_2 \in S(T_2)$; we must prove that no occurrence of a_1 is present in w after an occurrence of a_2. This follows immediately from the fact that every occurrence of a_1 is in w_1 and every occurrence of a_2 is in w_2, because of $w_i \models \text{upperS}(T_i)$, and because of $S(T_1) \cap S(T_2) = \emptyset$.

$\mathbf{T} = \mathbf{T_1} + \mathbf{T_2}$: We have:

$$\mathcal{NC}(T_1 + T_2) = (S(T_1) \prec\succ S(T_2)) \wedge \mathcal{NC}(T_1) \wedge \mathcal{NC}(T_2)$$

Consider $w \in [\![T_1 + T_2]\!]$. W.l.o.g., we can assume $w \in [\![T_1]\!]$. By induction we have $w \models \mathcal{NC}(T_1)$. Moreover, from $S(T_1) \cap S(T_2) = \emptyset$, we have that $w|_{S(T_2)} = \epsilon$, hence $w|_{S(T_2)} \models \mathcal{NC}(T_2)$, hence, by Lemma 23, $w \models \mathcal{NC}(T_2)$.

To derive $w \models S(T_1) \prec\succ S(T_2)$, we observe that by $w \in [\![T_1 + T_2]\!]$ and $S(T_1) \cap S(T_2) = \emptyset$ we have $w \not\models S(T_1)^+ \wedge S(T_2)^+$. Therefore, by Proposition 10 we obtain $w \models S(T_1) \prec\succ S(T_2)$.

$\mathbf{T} = \mathbf{T_1}!$: We have $[\![T]\!] \subseteq [\![T_1]\!]$, hence $w \in [\![T_1]\!]$, hence, by induction, $w \models \mathcal{NC}(T_1)$, hence $w \models \mathcal{NC}(T)$, by Proposition 17. □

This completes the proof of the soundness and completeness of the constraint characterization, i.e., Theorem 18.

4 Polynomial inclusion checking via constraint satisfaction

As shown in the previous section, type semantics can be characterized in terms of constraint satisfaction. As a consequence, type inclusion can also be characterized as constraint satisfaction, as follows.

Corollary 25

$$[\![T]\!] \subseteq [\![U]\!] \;\Leftrightarrow T \models \mathcal{CC}(U) \wedge T \models \mathcal{OC}(U) \wedge T \models \mathcal{FC}(U)$$

In the next sections we will show how each of the three properties in the right hand side can be verified in polynomial time.

4.1 Polynomial checking of $T \models \mathcal{CC}(U)$

We present here an algorithm to check whether $T \models \mathcal{CC}(U)$ in $O(|T|*|U|+|U|^2)$ time.

In [8], we observed that co-occurrence constraints are mutually related in the same way as functional dependencies, if one switches the left sides with the right sides. More precisely, one can define a notion of "backward closure" as follows.

Definition 26 *For any $B \subseteq \Sigma$ and type T, we indicate with $\mathsf{BC}(B)_T$ the backward closure of B with respect to T, and define it as:*

$$\mathsf{BC}(B)_T = \{a \in S(T) \ \mid \ T \models a^+ \mapsto\!\!\!\Rightarrow B^+\}$$

Knowledge of $\mathsf{BC}(B)_T$ is sufficient to decide whether $T \models A^+ \mapsto\!\!\!\Rightarrow B^+$.

Lemma 27 *For any A, B, T:*

$$\begin{aligned} & T \models A^+ \mapsto\!\!\!\Rightarrow B^+ && (1)\\ \Leftrightarrow\ \ & \forall a \in (A \cap S(T)).\ T \models a^+ \mapsto\!\!\!\Rightarrow B^+ && (2)\\ \Leftrightarrow\ \ & (A \cap S(T)) \subseteq\ \mathsf{BC}(B)_T && (3) \end{aligned}$$

Proof. (1) $\Leftrightarrow$ (2): $T \models A^+ \mapsto\!\!\!\Rightarrow B^+$
$\Leftrightarrow w \in [\![T]\!] \wedge w \models A^+ \Rightarrow w \models B^+$
$\Leftrightarrow \forall a \in A.\ w \in [\![T]\!] \wedge w \models a^+ \Rightarrow w \models B^+$
$\Leftrightarrow \forall a \in (A \cap S(T)).\ w \in [\![T]\!] \wedge w \models a^+ \Rightarrow w \models B^+$
$\Leftrightarrow \forall a \in (A \cap S(T)).\ T \models a^+ \mapsto\!\!\!\Rightarrow B^+$.

(2) $\Leftrightarrow$ (3): by definition of $\mathsf{BC}(B)_T$.

□

For $\mathsf{BC}(B)_T$, the following properties hold, which are proved by Lemma 28, but are anticipated here to to help the reader's intuition:

- transitivity: for any type $T = C[T_1 \otimes T_2]$, $\mathcal{CC}(T_1 \otimes T_2) = \mathit{If}_{T_2}(S^+(T_1) \mapsto\!\!\!\Rightarrow S^+(T_2)) \wedge \ldots$, hence:

$$\neg\mathrm{N}(T_2)\ \wedge\ S(T_2) \subseteq\ \mathsf{BC}(B)_T\ \Rightarrow\ S(T_1) \subseteq\ \mathsf{BC}(B)_T$$

- completeness: the repeated application of *transitivity* is a complete algorithm for computing $\mathsf{BC}(B)_T$.

According to these properties, $\mathsf{BC}(B)_T$ can be computed by the following algorithm, which "marks" a subterm T' of T as soon as it can conclude that $S(T') \subseteq \mathsf{BC}(B)_T$:

(1) mark all the leaves of the type whose symbol is in B, and visit the parse tree analyzing the parent of each marked node;
(2) if the parent of a marked T_2 with $\neg\mathrm{N}(T_2)$ is $T_1 \otimes T_2$ (or $T_2 \otimes T_1$), then mark it as well (thanks to transitivity) and continue the visit from its parent;

(3) when both children of a subterm T' are marked, then T' is marked too and the visit continues from T'.

Step (2) above does not mark T_1 and its descendants, but just $T_1 \otimes T_2$. As a consequence, not every $a\,[m..n]$ whose a is in $\mathsf{BC}(B)_T$ is marked. Instead, $\mathsf{BC}(B)_T$ corresponds to the set of all $a\,[m..n]$ with at least one ancestor that is marked (Lemma 28).

Having presented the idea, we can now start the formal treatment. We first define the set $B_T^{\uparrow}$, that represents the subterms of T that are marked by the above algorithm invoked with a B set of symbols. In this definition, $T = C[T']$ means that T' is a subterm of T.

$$\begin{array}{rlcl}
(\epsilon) & \text{for any } B, T & & \epsilon \in B_T^{\uparrow} \\
(a\,[m..n]) & T = C[a\,[m..n]],\ a \in B & \Rightarrow & a\,[m..n] \in B_T^{\uparrow} \\
(+) & T = C[T_1 + T_2],\ T_1 \in B_T^{\uparrow},\ T_2 \in B_T^{\uparrow} & \Rightarrow & T_1 + T_2 \in B_T^{\uparrow} \\
(\otimes) & T = C[T_1 \otimes T_2],\ T_1 \in B_T^{\uparrow},\ T_2 \in B_T^{\uparrow} & \Rightarrow & T_1 \otimes T_2 \in B_T^{\uparrow} \\
(\otimes \Mapsto) & T = C[T_1 \otimes T_2],\ \neg\mathrm{N}(T_2),\ T_2 \in B_T^{\uparrow} & \Rightarrow & T_1 \otimes T_2 \in B_T^{\uparrow} \\
(\otimes \Mapsfrom) & T = C[T_1 \otimes T_2],\ \neg\mathrm{N}(T_1),\ T_1 \in B_T^{\uparrow} & \Rightarrow & T_1 \otimes T_2 \in B_T^{\uparrow} \\
(!) & T = C[T_1!],\ T_1 \in B_T^{\uparrow} & \Rightarrow & T_1! \in B_T^{\uparrow}
\end{array}$$

Observe that $B_T^{\uparrow}$ is closed by type contexts, that is, for any type context $C[T]$, $T' \in B_T^{\uparrow}$ implies $T' \in B_{C[T]}^{\uparrow}$.

We use $S(B_T^{\uparrow})$ to denote $\cup_{T' \in B_T^{\uparrow}} S(T')$; the following lemma proves that $S(B_T^{\uparrow}) = \mathsf{BC}(B)_T$.

Lemma 28 *For each type T:*

$$(a \in S(T) \ \wedge\ T \models a^+ \Mapsto B^+) \quad \Leftrightarrow \quad a \in S(B_T^{\uparrow})$$

Proof.

We first consider the $\Rightarrow$ direction.

We prove the following property by induction on the structure of T and by case analysis (the interesting case being $T = T_1 \otimes T_2$):

$$\begin{array}{lcl}
(1)\ a \in S(T) \ \wedge\ T \models a^+ \Mapsto B^+ & \Rightarrow & a \in S(B_T^{\uparrow}) \\
(2)\ (\llbracket T \rrbracket \setminus \{\epsilon\}) \models B & \Rightarrow & T \in B_T^{\uparrow}
\end{array}$$

In line (2) we use the notation $W \models B$ to denote $\forall w \in W.\ w \models B$.

$\mathbf{T} = \epsilon$:

1: Trivial, as $a \notin S(\epsilon)$.

2: Trivial, since $\epsilon \in B^{\uparrow}_{U}$ for each U.

$\mathbf{T} = \mathbf{b}\,[\mathbf{m}..\mathbf{n}]$:

1: From $a \in S(T)$ we obtain that $a = b$. From $T \models b^{+} \mapsto B^{+}$ we have that $b \in B$, hence $b\,[m..n] \in B^{\uparrow}_{b[m..n]}$.

2: From $(\llbracket T \rrbracket \setminus \{\epsilon\}) \models B$ we deduce $b \in B$, hence $b\,[m..n] \in B^{\uparrow}_{b[m..n]}$.

$\mathbf{T} = \mathbf{T_1} + \mathbf{T_2}$:

Without loss of generality, we can assume that $a \in S(T_1)$.

1: From $T \models a^{+} \mapsto B^{+}$ we obtain that $T_1 \models a^{+} \mapsto B^{+}$, which, by induction, implies $a \in S(B^{\uparrow}_{T_1})$, hence, by context closure, $a \in S(B^{\uparrow}_{T})$.

2: From $(\llbracket T \rrbracket \setminus \{\epsilon\}) \models B$ we obtain that $(\llbracket T_1 \rrbracket \setminus \{\epsilon\}) \models B$ and $(\llbracket T_2 \rrbracket \setminus \{\epsilon\}) \models B$, which, by induction, imply $T_1 \in B^{\uparrow}_{T_1}$ and $T_2 \in B^{\uparrow}_{T_2}$, hence $(T_1 + T_2) \in B^{\uparrow}_{T_1+T_2}$.

$\mathbf{T_1} \otimes \mathbf{T_2}$:

Without loss of generality, we can assume that $a \in S(T_1)$.

1: We distinguish two cases:

1.1 $T_2 \not\models B$, i.e. $\exists w_2 \in \llbracket T_2 \rrbracket$ such that $w_2 \not\models B$;
1.2 $T_2 \models B$, i.e. $\forall w_2 \in \llbracket T_2 \rrbracket\ w_2 \models B$.

Case 1.1: We know that $\exists w_2 \in \llbracket T_2 \rrbracket$ such that $w_2 \not\models B$, i.e. w_2 does not contain any symbol in B. For each string $w \in \llbracket T_1 \rrbracket$, $w \cdot w_2 \in \llbracket T_1 \otimes T_2 \rrbracket$ hence, by hypothesis, $w \cdot w_2 \models a^{+} \mapsto B^{+}$. From $w_2 \not\models B$, we deduce that, for any such w, $w \models a^{+} \mapsto B^{+}$, hence $T_1 \models a^{+} \mapsto B^{+}$, hence $a \in S(B^{\uparrow}_{T_1})$ (by induction), hence $a \in S(B^{\uparrow}_{T})$ (by context closure).

Case 1.2: We know that $T_2 \models B$, hence $\llbracket T_2 \rrbracket \setminus \{\epsilon\} \models B$, hence, by induction, $T_2 \in B^{\uparrow}_{T_2}$, which entails $T_2 \in B^{\uparrow}_{T_1 \otimes T_2}$ by context closure. The hypothesis $T_2 \models B$ also implies that $\mathrm{N}(T_2)$ is *false*, hence, by definition of $B^{\uparrow}_{T_1 \otimes T_2}$, from $T_2 \in B^{\uparrow}_{T_1 \otimes T_2}$ we deduce $(T_1 \otimes T_2) \in B^{\uparrow}_{T_1 \otimes T_2}$, hence $S(T_1 \otimes T_2) \subseteq S(B^{\uparrow}_{T_1 \otimes T_2})$, hence $a \in S(B^{\uparrow}_{T_1 \otimes T_2})$, since $a \in S(T_1 \otimes T_2)$ by hypothesis.

2: We have three cases:

2.1 $\mathrm{N}(T_1)$ and $\mathrm{N}(T_2)$ both hold;
2.2 exactly one of $\neg\mathrm{N}(T_1)$ and $\neg\mathrm{N}(T_2)$ holds;
2.3 $\neg\mathrm{N}(T_1)$ and $\neg\mathrm{N}(T_2)$ both hold.

Case 2.1: $\epsilon \in [\![T_1]\!]$ implies that $[\![T_2]\!] \subseteq [\![T_1 \otimes T_2]\!]$, and $\epsilon \in [\![T_2]\!]$ implies that $[\![T_1]\!] \subseteq [\![T_1 \otimes T_2]\!]$, hence we can reason as in the case for $T_1 + T_2$.

Case 2.2: assume that $\neg\mathrm{N}(T_1)$ and $\mathrm{N}(T_2)$. As in the previous case, $\epsilon \in [\![T_2]\!]$ implies that $[\![T_1]\!] \subseteq [\![T_1 \otimes T_2]\!]$, hence $T_1 \in B^{\uparrow}_{T_1}$ by induction, hence, by $\neg\mathrm{N}(T_1)$, $(T_1 \otimes T_2) \in B^{\uparrow}_{T_1 \otimes T_2}$.

Case 2.3: We prove that either $([\![T_1]\!] \setminus \{\epsilon\}) \models B$ or $([\![T_2]\!] \setminus \{\epsilon\}) \models B$, and the thesis follows as in case 2.2. Assume, for a contradiction, that both $([\![T_1]\!] \setminus \{\epsilon\}) \not\models B$ and $([\![T_2]\!] \setminus \{\epsilon\}) \not\models B$ hold, then we have two non-empty words w_1 and w_2 such that $w_1 \not\models B$ and $w_2 \not\models B$, and hence $w_1 \cdot w_2 \not\models B$, which contradicts the hypothesis, since $w_1 \cdot w_2 \in [\![T_1 \otimes T_2]\!] \setminus \{\epsilon\}$.

$\mathbf{T} = \mathbf{T_1}!$:

1: From $T \models a^+ \mapsto B^+$ we obtain that $T_1 \models a^+ \mapsto B^+$, which, by induction, implies $a \in S(B^{\uparrow}_{T_1})$, hence $a \in S(B^{\uparrow}_{T})$.

2: From $[\![T!]\!] \setminus \{\epsilon\} \models B$ we obtain that $[\![T]\!] \setminus \{\epsilon\} \models B$, which, by induction, implies $T_1 \in B^{\uparrow}_{T_1}$ hence $(T_1!) \in B^{\uparrow}_{T_1!}$.

The $\Leftarrow$ direction is easy, and is proved by induction on the structure of T and by case analysis.

□

In Figure 1 we present the algorithm BACKWARDCLOSE(Type T, Set A) that marks any term in $A^{\uparrow}_{T}$. The algorithm works on a type T that has been ϵ-normalized, meaning that every subterm $\epsilon \otimes T$ or $T \otimes \epsilon$ is rewritten as T, and every $\epsilon + \epsilon$ is rewritten as ϵ, until no such terms remain; this can be done in linear time, and allows ϵ terms not to be marked. A type T is encoded by the arrays *NodeOfSymbol[]*, *Parent[]*, *Oper[]*, *LeftChild[]*, *RightChild[]*. For each symbol $a \in S(T)$, NodeOfSymbol[a] is the only node *nd* associated with a type $a\,[m..n]$, for some m and n, and is null if no such node exists[4]; for each node n in T, Parent[n] is either null or a pair (n_p,*pos*); n_p is the parent of n, while *pos* is *left* if n is the left child of n_p, and is *right* if it is the right child. The *Oper[]* array associates $\mathbf{true}_2$ to any $+$ node where no child is ϵ, $\mathbf{true}_1$

[4] Recall that a symbol appears at most once in each conflict-free type.

to a $+$ node where one child is ϵ. We use **true** to indicate that no constraint is associated with the node, while the pedices 2 and 1 refer to the number of children that have yet to be inserted into $A^{\uparrow}_{T}$ before the node can be inserted as well. An $\otimes$ node is associated with $\Leftrightarrow$ when no child is nullable, with $\Mapsfrom$ when the right child only is nullable, $\Mapsto$ when the left child only is nullable, with $\mathbf{true}_2$ when both children are nullable. *LeftChild[]* and *RightChild[]* are defined in the obvious way.

The algorithm differs from $\_^{\uparrow}_{T}$ in that it explicitly marks every subterm of each term in $B^{\uparrow}_{T}$; this is performed by the MarkDown routine, which propagates the mark down until it meets a node that is marked already. In this way, all the children of any node in $ToDo$ are marked, hence, thanks to the test in line 9, no node enters $ToDo$ twice.

BackwardClose first instantiates the above mentioned arrays in time $O(|T|)$. Then, the algorithm populates the $ToDo$ list in time $O(|A|)$; $ToDo$ contains all the nodes that must be visited. The body of the main loop (line 7 to line 23) needs constant time, apart from the calls to MarkDown which, collectively, mark less then $O(|T|)$ nodes. No node enters the $ToDo$ list twice, hence the main loop takes time $O(|T|)$, hence the complexity of the algorithm is $O(|A| + |T|)$.

The algorithm immediately satisfies the invariant that, whenever a node enters *ToDo*, then the corresponding T' is in $A^{\uparrow}_{T}$, and that, whenever a node is marked, then one of its ancestors is in $A^{\uparrow}_{T}$. To prove that every node in $A^{\uparrow}_{T}$ is eventually inserted in $ToDo$, we proceed by induction on the size of the corresponding subtree T', assuming that all the subtrees of T' that are in $A^{\uparrow}_{T}$ are eventually inserted in $ToDo$.

We can now define the algorithm CoImplies (Figure 2) to check whether $T \models \mathcal{CC}(U)$. The algorithm invokes BackwardClose once for each constraints $A^+ \Mapsto B^+$ in $\mathcal{CC}(U)$, and verifies whether each a in $A \cap S(T)$ belongs to $\mathsf{BC}(B)_T$. Each invocation of BackwardClose(Type T, Set B) is in $O(|T| + |U|)$, and the test for $(A \cap S(T)) \subseteq \mathsf{BC}(B)_T$ is in $O(|U|)$, hence in $O(|T| + |U|)$. Since $\mathcal{CC}(U)$ contains at most $2 * |U|$ constraints, the complexity of CoImplies is $O(|T| * |U| + |U|^2)$.

4.2 Polynomial checking of $T \models \mathcal{OC}(U)$

In the previous section we have shown how to deduce all the co-occurrence constraints that are satisfied by a type T, by computing a sort of backward transitive closure on $\mathcal{CC}(T)$. Order constraints are much simpler, since T does not satisfy any other order constraint apart from those that are actually present in $\mathcal{OC}(T)$; this is proved by the following theorem.

```
BackwardClose($Type\ T, Set\ A$)
(NodeOfSymbol[],Parent[],Oper[],LeftChild[],RightChild[]) := ReadType(T)
Marked[*] := false
ToDo := $\emptyset$
for $a$ in $A$
    do ToDo := ToDo ++ NodeOfSymbol[$a$]
while ToDo $\neq \emptyset$
    do pick $n$ from ToDo
       (parent,pos) := Parent[$n$]
       if (Marked[$n$])
          then break
          else  Marked[$n$] := true
       if (parent = null)
          then break
       case(Oper[parent],pos)
       when ($\mathbf{true}_{2}$,_) then Oper[parent] := $\mathbf{true}_{1}$
       when ($\mathbf{true}_{1}$,_) then ToDo := ToDo ++ parent
       when ($\Mapsto$,'left') or ($\Leftrightarrow$,'left')
                ToDo := ToDo ++ parent
                MarkDown(RightChild[parent])
       when ($\Lleftarrow$,'right') or ($\Leftrightarrow$,'right')
                ToDo := ToDo ++ parent
                MarkDown(LeftChild[parent])
       esac
```

Fig. 1. Backward closure algorithm.

```
CoImplies($Type\ T, Type\ U$)
for $A^{+} \Mapsto B^{+}$ in $\mathcal{CC}(U)$,
    do BackwardClose(T, B)
       if exists a in $A \cap S(T)$ with Marked[NodeOfSymbol[a]]=false
          then return false
return true
```

Fig. 2. Algorithm for implication of cooccurrence constraints.

Theorem 29 *If $a \neq b$, then*

$$T \models a \prec b \ \Leftrightarrow \ (a \prec b \in \mathcal{OC}(T) \ \vee \{a,b\} \not\subseteq S(T))$$

Proof.

The $\Leftarrow$ direction easily follows from the following facts:

- If $\{a,b\} \not\subseteq S(T)$, then each word w in T does not contain both a and b; this means that $w \models a \prec b$;

- If $a \prec b \in \mathcal{OC}(T)$, then $T \models a \prec b$ follows from the fact that $T \models \mathcal{OC}(T)$ (Theorem 18).

We focus now on the $\Rightarrow$ direction. We proceed by induction on the structure of T.

$\mathbf{T} = \epsilon$:

Trivial, as $S(T) = \emptyset$ entails $\{a, b\} \not\subseteq S(T)$.

$\mathbf{T} = \mathbf{c}\,[\mathbf{m}..\mathbf{n}]$:

Trivial, we immediately have $\{a, b\} \not\subseteq S(T)$ since $a \neq b$.

$\mathbf{T} = \mathbf{T_1} + \mathbf{T_2}$:

We suppose $\{a, b\} \subseteq S(T)$; the other case is trivial.

We have the following subcases:

Case 1. $\{a, b\} \subseteq S(T_1)$ or $\{a, b\} \subseteq S(T_2)$.

Case 2. $a \in S(T_1)$ and $b \in S(T_2)$, or vice versa.

For case 1 we only consider $\{a, b\} \subseteq S(T_1)$, the other one is similar. In this case we have $T_1 \models a \prec b$. By induction we have $(a \prec b \in \mathcal{OC}(T_1)\ \vee \{a, b\} \not\subseteq S(T_1))$, and, therefore $a \prec b \in \mathcal{OC}(T_1)$ (as $\{a, b\} \subseteq S(T_1)$), which entails that $a \prec b \in \mathcal{OC}(T)$, since $\mathcal{OC}(T) = F \wedge \mathcal{OC}(T_1)$.

For case 2, we only consider $a \in S(T_1)$ and $b \in S(T_2)$, the other one being identical. In this case we have that $a \prec\succ b = (a \prec b \wedge b \prec a) \in \mathcal{OC}(T)$.

$\mathbf{T_1}\&\mathbf{T_2}$:

We can have the following subcases:

Case 1. $\{a, b\} \subseteq S(T_1)$ or $\{a, b\} \subseteq S(T_2)$.

Case 2. $a \in S(T_1)$ and $b \in S(T_2)$, or vice versa.

Case 1 is similar to the corresponding one for case $T = T_1 + T_2$, while we can exclude case 2 because of the hypothesis $T \models a \prec b$ and of Lemma 4(2).

$\mathbf{T_1} \cdot \mathbf{T_2}$:

We can have the following subcases:

Case 1. $\{a, b\} \subseteq S(T_1)$ or $\{a, b\} \subseteq S(T_2)$.

Case 2. $a \in S(T_1)$ and $b \in S(T_2)$, or vice versa.

Case 1 is similar to the corresponding one for case $T = T_1 + T_2$.

For case 2 we can only have $a \in S(T_1)$ and $b \in S(T_2)$ (since the contrary would contradict $T \models a \prec b$). For this case, we have $a \prec b \in \mathcal{OC}(T)$ by definition of $\mathcal{OC}(T)$.

T!:

This case easily follows by induction once we recall that $[\![T!]\!] = [\![T]\!] \setminus \{\epsilon\}$, $S(T!) = S(T)$, and $\mathcal{OC}(T!) = \mathcal{OC}(T)$.

□

The previous theorem tells us that we can check $T \models \mathcal{OC}(U)$ in $O(|\mathcal{OC}(T)| \cdot |\mathcal{OC}(U)|)$ time, by checking whether each $a \prec b$ in $\mathcal{OC}(U)$ is also in $\mathcal{OC}(T)$, which gives us an upper bound $O(|T|^2 \cdot |U|^2)$, but we can actually do much better. Indeed, we can perform the checking in $O(|T| + |U|^2)$ time. Before defining the algorithm, we make some preliminary observations.

For each pair of leaves $a\,[m..n]$ and $b\,[m'..n']$ in the parse tree of T, let $LCA_T[a, b]$ be their common ancestor that is farthest from the root (the *Lowest Common Ancestor*). Order constraints correspond to the concatenation and union type operators:

- for each a and b in $S(T)$, $a \prec\succ b \in \mathcal{OC}(T)$ iff $LCA_T[a, b]$ is labeled by $+$;
- for each a and b in $S(T)$, $a \prec b \in \mathcal{OC}(T)$ iff $LCA_T[a, b] = +$ or a precedes b in T and $LCA_T[a, b] = \cdot$.

As a consequence, by Theorem 29, $T \models \mathcal{OC}(U)$ iff for each a and b in $S(U)$, such that a precedes b in U:

- if $LCA_U[a, b] = +$ then either $a \notin S(T)$ or $b \notin S(T)$ or $LCA_T[a, b] = +$;
- if $LCA_U[a, b] = \cdot$ then either $a \notin S(T)$ or $b \notin S(T)$ or $LCA_T[a, b] = +$ or $(LCA_T[a, b] = \cdot$ and a precedes b in $T)$.

Hence, we can verify whether $T \models \mathcal{OC}(U)$ via the following algorithm. We first build an array $LCA_T[a, b]$, which associates each a and b in $S(T)$ with the operator that labels the LCA of a and b in T, and similarly for U; this can be done in linear time $O(|T| + |U|)$ [3]. We then scan all the ordered pairs a, b of $S(U)$, checking the condition above, which can be done with $O(|U|^2)$ constant-time accesses to $LCA_T[\_, \_]$ and $LCA_U[\_, \_]$, which gives a $O(|T| + |U|^2)$ algorithm. The resulting algorithm is defined in Figure 3.

```
OrderImplies($Type\ \ T, Type\ \ U$)
build $LCA_T[\_,\_]$ and $LCA_U[\_,\_]$
for each leaf $a$ in $U$, leaf $b$ following $a$ in $U$
    do if $LCA_U[a,b] = + \ \wedge\ a \in S(T)\ \wedge\ b \in S(T) \wedge\ LCA_T[a,b] \neq +$
          then return false
       if $LCA_U[a,b] = \cdot\ \wedge\ a \in S(T)\ \wedge\ b \in S(T) \wedge\ LCA_T[a,b] \neq +$
                 $\wedge\ (LCA_T[a,b] \neq \cdot \vee a$ follows $b$ in $T)$
          then return false
return true
```

Fig. 3. Algorithm for implication of order constraints.

4.3 Polynomial checking of $T \models \mathcal{FC}(U)$

A systematic way to check flat constraint implication $T \models \mathcal{FC}(U)$ in polynomial time is directly illustrated by the following theorem, stating that, as expected, $T \models \mathcal{FC}(U)$ holds if and only if i) all cardinality constraints $a?[m..n]$ of T appear in U as well, possibly in a relaxed form, and ii) if $\mathrm{N}(T_1)$ holds, then $\mathrm{N}(T_2)$ is true as well.

Theorem 30

$$T \models \mathcal{FC}(U) \ \Leftrightarrow\ T \subseteq_{flat} U$$

where $T \subseteq_{flat} U$ *is defined as :*

$$\begin{gathered}(a?[m..n] \in Atoms(T) \Rightarrow \exists m' \leq m, n' \geq n.\ a\,[m'..n'] \in Atoms(U)) \\ \wedge\ (\mathrm{N}(T) \Rightarrow \mathrm{N}(U))\end{gathered}$$

Proof.

Recall that $\mathcal{FC}(U) = \mathit{SIf}(U) \wedge \mathrm{upperS}(U) \wedge \mathrm{ZeroMinMax}(U)$.

Case $T \models \mathcal{FC}(U)\ \Leftarrow\ T \subseteq_{flat} U$.

We first observe that $T \subseteq_{flat} U$ implies $S(T) \subseteq S(U)$. Recall that, by Theorem 24, $T \models \mathrm{upperS}(T) \wedge \text{ZMM-SIf}(T)$.

(1) $T \models \mathit{SIf}(U)$: if $\mathrm{N}(U)$, then $\mathit{SIf}(U) = \mathbf{true}$, hence the statement is trivial. Assume that $\mathrm{N}(T)$ is false. We have that $T \models \mathit{SIf}(T)$ and $w \in [\![T]\!]$ imply $w \models S^+(T)$, hence $w \models S^+(U)$ by $S(T) \subseteq S(U)$.

(2) $T \models \mathrm{upperS}(U)$: we must prove that $w \in [\![T]\!]$ and $w \models a^+$ imply that $a \in S(U)$; $T \models \mathrm{upperS}(T)$, $w \in [\![T]\!]$ and $w \models a^+$ imply that $a \in S(T)$, and $a \in S(U)$ follows from $S(T) \subseteq S(U)$.

(3) $T \models \mathrm{ZeroMinMax}(U)$: we must prove that, for any $w \in [\![T]\!]$ and $a\,[m..n] \in Atoms(U)$, $w \models a?[m..n]$. If $w \models a^+$, then, by $T \models \mathrm{upperS}(T) \wedge \mathrm{ZeroMinMax}(T)$,

$\exists m', n'$ such that $a\,[m'..n'] \in \mathit{Atoms}(T)$ and $w \models a?[m'..n']$. By hypothesis, $\exists m'' \leq m', n'' \geq n'$ such that $a\,[m''..n''] \in \mathit{Atoms}(U)$ hence, since U is conflict-free, we have that $m'' = m$ and $n'' = n$. Hence, $w \models a?[m'..n']$ and $m \leq m', n' \leq n$ imply $w \models a?[m..n]$.

Case $T \models \mathcal{FC}(U) \;\Rightarrow\; T \subseteq_{\mathit{flat}} U$.

We have that $T \models \mathit{SIf}(U) \wedge \mathrm{upperS}(U) \wedge \mathrm{ZeroMinMax}(U)$. In particular, $T \models \mathrm{upperS}(U)$ means that, for each $w \in [\![T]\!]$, we have $w \models \mathrm{upperS}(U)$; this entails $S(T) \subseteq S(U)$, so

$$\forall a?[m..n] \in \mathit{Atoms}(T).\ \exists a?[m'..n'] \in \mathit{Atoms}(U)$$

hence it remains to prove that it is always the case that $m' \leq m$ and $n' \geq n$. This follows by observing that, if we assume $n' < n$ or $m' > m$, then the hypothesis $T \models \mathcal{FC}(U)$ is contradicted. This is due to the following fact: if $a?[m..n] \in \mathit{Atoms}(T)$ and $n \neq *$, then we have two words w_m and w_n in $[\![T]\!]$ such that they contain m and n occurrences of a respectively; if $n = *$ and $n' \neq *$ (the remaining cases are trivial), then there exists w_n in $[\![T]\!]$, such that it contains more than n' occurrences of a. For these two words w_n and w_m we have that, under the hypothesis ($n' < n$ or $m' > m$), either $w_m \models \mathrm{ZeroMinMax}(U)$ or $w_n \models \mathrm{ZeroMinMax}(U)$ does not hold, and so $T \models \mathrm{ZeroMinMax}(U)$ would be contradicted.

It remains to prove that $\mathrm{N}(T) \Rightarrow \mathrm{N}(U)$. Assume that $\mathrm{N}(T)$ holds (the other case is trivial). This means that $\epsilon \in [\![T]\!]$ and $\epsilon \models \mathit{SIf}(U)$, by hypothesis.

If $S(U) = \emptyset$ (that is U is equivalent to ϵ), then $\mathrm{N}(U)$ holds and the thesis is proved. If $S(U) \neq \emptyset$, then it must be $\mathit{SIf}(U) = \mathbf{true}$, because otherwise $\epsilon \models \mathit{SIf}(U)$ would not hold, thus contradicting the hypothesis $T \models \mathcal{FC}(U)$

□

4.4 Wrapping up: the inclusion checking algorithm

We have provided algorithms to check $T \models \mathcal{CC}(U)$, $T \models \mathcal{OC}(U)$ and $T \models \mathcal{FC}(U)$. They can be used to build the inclusion checking algorithm illustrated in Figure 4. It first verifies whether $T \models \mathcal{FC}(U)$, in time $O(|T| + |U|)$ in the size of T and U, as illustrated in Section 4.3. Then the algorithm invokes CoImplies and OrderImplies; the $O(|T| * |U| + |U|^2)$ time complexity of the first dominates the $O(|T| + |U|^2)$ complexity of the second, hence this is the cost of the whole algorithm.

```
Sub($T$, $U$)
  $(Min_U[], Max_U[]) = \textsc{BuildMinMaxArrays}(U)$
  $flat$ = (every $a?[m..n] \in Atoms(T)$
            satisfies $(Min_U[a] \leq m) \wedge\ (Max_U[a] \geq n)) \wedge\ (\neg \mathrm{N}(T)\ \vee\ \mathrm{N}(U))$
  return $flat \wedge \textsc{CoImplies}(T, U) \wedge \textsc{OrderImplies}(T, U)$
```

Fig. 4. Inclusion checking algorithm.

5 Complexity of Binary Intersection

Checking the non-emptiness of binary intersection of REs is typically cheaper than inclusion checking, since intersection [5] is checked through automata product, while inclusion corresponds to automata complement plus product.

The situation is different for conflict-free types: while inclusion is in PTIME, intersection of two conflict-free expressions is NP-complete. This result is quite surprising, and it suggests that it makes sense to study such types with an approach that is *not* based on automata.

Interestingly, as shown by our proof, the & operator alone makes binary intersection NP-hard, even without counting or Kleene star.

We first prove that non-emptiness is in NP, which is simple, since one has just to guess a word and check whether it is in both types; we have still to show that the word to guess is not big. To this aim, we show that any word $w \in (\llbracket T \rrbracket \cap \llbracket U \rrbracket)$ can be cut down to a word that is smaller than $|T| + |U|$ and is still in $\llbracket T \rrbracket \cap \llbracket U \rrbracket$. To achieve this bound, we represent each word *abbbcc* as $a^1 b^3 c^2$, as specified by the following definition.

Definition 31 (Binary word representation) *A binary word representation is a sequence of symbol-integer pairs, written* $a_1^{m_1} \ldots a_k^{m_k}$*, where each* $m_i > 0$*, and each pair* $a_i^{m_i}$ *represents a sequence of* m_i *copies of* a_i*.*

We now show how to reduce $w \in \llbracket T \rrbracket$ into a short form that is still in $\llbracket T \rrbracket$. We consider a set f of pairs (a, m), that defines a partial function from Σ to $\mathbb{N}$. We define a function $Stem(w, f)$ that deletes from w the symbols that are not in f, and keeps just one repeated occurrence $a^{f(a)}$ for the others, so that $|Stem(w, f)| < |f|$.

Definition 32 (*Stem*(w, f)) *For any word* w *and set of pairs* $f = \{(a_1, m_1), \ldots, (a_k, m_k)\}$*,* $Stem(w, f)$ *is defined as follows, where, in the*

[5] In this section we use *intersection* as an abbreviation for "non-emptiness of binary intersection"

second line, $a^{f(a)}$ is defined as ϵ if a is not in f.

$$\begin{aligned} \mathit{Stem}(\epsilon, f) &=_{\mathit{def}} \epsilon \\ \mathit{Stem}(a^n \cdot w, f) &=_{\mathit{def}} a^{f(a)} \cdot \mathit{Stem}(w, f \setminus (a, f(a))) \end{aligned}$$

Lemma 33 specifies that $w \in [\![T]\!]$ implies $\mathit{Stem}(w, f) \in [\![T]\!]$, when f leaves one copy of each character, and assigns acceptable exponents.

Lemma 33 (*Stem*$(w, f) \in [\![T]\!]$)

$$\begin{aligned} & w \in [\![T]\!] \\ & \wedge\ S(f) \supseteq S(w) \\ & \wedge\ ((a, f(a)) \in f \Rightarrow a^{f(a)} \models \text{ZeroMinMax}(T)) \\ & \ \Rightarrow \mathit{Stem}(w, f) \in [\![T]\!]. \end{aligned}$$

Proof. We prove that $\mathit{Stem}(w, f)$ satisfies $\mathit{SIf}(T)$, ZeroMinMax(T), upperS(T), $\mathcal{CC}(T)$, $\mathcal{OC}(T)$, and we conclude by Theorem 18. By $S(f) \supseteq S(w)$, we deduce that no symbol is removed from w, hence $w \models \mathit{SIf}(T)$ implies that $\mathit{Stem}(w, f) \models \mathit{SIf}(T)$; for the same reason, $w \models \mathcal{CC}(T)$ implies that $\mathit{Stem}(w, f) \models \mathcal{CC}(T)$. From $a^{f(a)} \models$ ZeroMinMax(T) we deduce that $\mathit{Stem}(w, f) \models$ ZeroMinMax(T). From $S(\mathit{Stem}(w, f)) \subseteq S(w)$ we deduce that $\mathit{Stem}(w, f) \models$ upperS(T). Finally, if a pair of symbols a and b appear in $\mathit{Stem}(w, f)$ in this order, then the first occurrence of a precedes the first occurrence of b in w; hence, $w \models \mathcal{OC}(T)$ implies that $\mathit{Stem}(w, f) \models \mathcal{OC}(T)$. □

Theorem 34 (Upper bound) *Non emptiness of the intersection of two conflict-free types is in NP.*

Proof. We prove that, if a word w is in $[\![T]\!] \cap [\![U]\!]$, then a word w' whose binary representation is shorter than $|T| + |U|$ is in $[\![T]\!] \cap [\![U]\!]$ as well. The result follows, since non-emptiness can be checked by guessing a word shorter than $|T| + |U|$ and checking that this word satisfies all the constraints of T and U, which is clearly in PTIME.

Define f as follows:

$$f = \{(a, \max(m, m')) \mid a\,[m..n] \in \mathit{Atoms}(T),\ a\,[m'..n'] \in \mathit{Atoms}(U), \\ \max(m, m') \leq \min(n, n')\}$$

We show that f satisfies the conditions of Lemma 33 for both T and U. Let $a \in S(w)$; from $w \models$ upperS(T) and $w \models$ upperS(U) we deduce that $a\,[m..n] \in$

$Atoms(T)$ and $a\,[m'..n'] \in Atoms(U)$. From $a \in S(w)$, $w \models \mathrm{ZeroMinMax}(T)$ and $w \models \mathrm{ZeroMinMax}(U)$, we deduce that $\max(m, m') \leq k \leq \min(n, n')$, where $k = |w|_{S(\{a\})}|$, hence $S(f) \supseteq S(w)$. The conditions $(a, f(a)) \in f \Rightarrow a^{f(a)} \models \mathrm{ZeroMinMax}(T)$ and $(a, f(a)) \in f \Rightarrow a^{f(a)} \models \mathrm{ZeroMinMax}(U)$ hold by construction. By Lemma 33, we conclude that $Stem(w, f) \in [\![T]\!] \cap [\![U]\!]$. By construction, $|Stem(w, f)| \leq |f| \leq (|T| + |U|)$. □

We can now prove that the problem is NP-hard, by reduction to 3-SAT.

Theorem 35 (Lower bound) *Emptiness of the intersection of two conflict-free types is NP-hard, even if the types do not use counting and concatenation.*

Proof. Consider m boolean variables $x_1, \ldots, x_m$ and a formula $\phi = (a_{1,1} \vee a_{2,1} \vee a_{3,1}) \wedge \ldots \wedge (a_{n,1} \vee a_{n,2} \vee a_{n,3})$ where each literal $a_{i,j}$ is either a variable x_l or a negated variable $\neg x_l$; 3SAT is the problem of deciding, for such a ϕ, whether an assignment of boolean values to $x_1, \ldots, x_m$ exists that satisfies the formula. Satisfiability of ϕ can be encoded as the intersection of two conflict-free types T_1 and T_2 as exemplified below. Both types have one symbol for each occurrence of a literal in ϕ, hence their size is linear in $|\phi|$. Observe that each line in T_2 sums the positive occurrences of one variable with the negative ones, hence making the two mutually incompatible.

$$\phi = (x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee x_3 \vee x_4) \wedge (\neg x_2 \vee \neg x_3 \vee \neg x_4) \wedge (\neg x_1 \vee \neg x_3 \vee x_4)$$

$$T_1 = (a_{1,1} + a_{1,2} + a_{1,3})\ \&\ (a_{2,1} + a_{2,2} + a_{2,3})\ \&\ (a_{3,1} + a_{3,2} + a_{3,3})\ \&\ (a_{4,1} + a_{4,2} + a_{4,3})$$

$$\begin{aligned} T_2 = {} & ((a_{1,1}?) + (a_{2,1}?\,\&\,a_{4,1}?)) \\ & \&\ ((a_{1,2}?) + (a_{3,1}?)) \\ & \&\ ((a_{1,3}?\,\&\,a_{2,2}?) + (\ a_{3,2}?\,\&\,a_{4,2}?)) \\ & \&\ ((a_{2,3}?\,\&\,a_{4,3}?) + (a_{3,3}?)) \end{aligned}$$

A formal definition of the translation can be easily produced.

The formula ϕ is satisfiable iff it has a *witness*, i.e. a choice of literal instances, one from each factor, such that not two instances are contradictory, i.e. if x_i is chosen in a factor then $\neg x_i$ is not chosen in any other factor.

Any element of T_1 corresponds to a choice of literal instances, one from each factor. If the same list also belongs to T_2, then the choice is not contradictory: if the word contained both a true and a false instance of the same variable, it should contain two symbols that come from the left and from the right side of a + operator in the line of T_2 that corresponds to that variable. Hence, words in $[\![T_1]\!] \cap [\![T_2]\!]$ correspond to witnesses for ϕ. □

6 Related Work

This paper is an extended and revised version of [8]; in particular, while [8] provided a cubic inclusion algorithm, we provide here an algorithm that is quadratic, and whose crucial component (co-occurrence constraint implication) is simpler.

The properties of unordered XML types have been studied in several recent papers. In [6], the authors discuss the techniques and heuristics they used in implementing a type-checker, based on sheaves automata with Presburger arithmetic, for unordered XML types. The type language is an extension of the language we are considering here, and shares a similar restriction on the use of repetition types. The main purpose of the paper is to address scalability problems that naturally arise when working on XML types; as a consequence, they describe effective heuristics that improve scalability, but do not affect computational complexity.

Restrictions to regular expression languages that are similar to ours have been proposed many times. For example, conflict-free REs appear as "conflict-free DTDs" in the context of well-typed XML updates in [2], as "duplicate-free DTDs" in the context of path inclusion in [13], and as "single occurrence REs" in the context of DTD inference in [5]. The same restriction that we pose on Kleene-star can be found, for example, in [6]. Chain Regular Expressions (CHARE's) [5,7] are also strictly related. They are defined as concatenations of factors, where each factor has a shape $(a_1 + \ldots + a_n)$, $(a_1 + \ldots + a_n)?$, $(a_1 + \ldots + a_n)^*$ or $(a_1 + \ldots + a_n)^+$. As we discussed in Section 2.1, we can express in our language each of these classes of factors. Simple expressions [4] have a more general syntax than CHAREs but the same expressive power, hence can still be managed through our approach.

We have cited paper [7] many times, where the complexity of type inclusion is studied for many different dialects of REs with interleaving and/or counting, showing that inclusion complexity is almost invariably EXPSPACE-complete. In particular, this is shown to hold for chain-REs with counting, which are concatenations of CHARE factors, as defined above, and counting factors $(a_1 + \ldots + a_k)[m..n]$ (with $n \neq *$ and $m \geq 0$), with no interleaving operator. In a sense, this hints that the conflict-free restriction, rather than the Kleene-star restriction, is crucial for our PTIME result. In the same paper, the authors introduce a sublanguage of CHAREs with PTIME inclusion, but that fragment is quite trivial, since it only includes counting factors $(a_1 + \ldots + a_k)[m..n]$, with the further restriction that $m > 0$ and $n \neq *$, hence cannot express neither optionality nor unbounded repetition (neither * nor $^+$).

7 Conclusions

Inclusion for REs with interleaving, counting, or both, is EXPSPACE-complete, even if we consider the restricted subclass of CHAREs (with counting) [11,7]. This result easily extends to XML types featuring these operators. We have introduced here a restricted class of REs with interleaving and counting. Our restriction is severe, but it seems to match reasonably well the measured features of actual DTDs and XSDs found on the web, and is extremely easy to define and verify. For this class of REs, we have proved that inclusion can be decided in quadratic time, a complexity that is surprising low, and trivially extends to DTDs and XSDs that use REs of this class for their content models. In particular, we exhibited a quadratic algorithm for content-model inclusion. We also proved that intersection of two conflict-free types has not the same complexity as inclusion (unless P=NP) but is, quite surprisingly, NP-complete.

Our result is based on the transformation of our REs into sets of constraints which completely characterize the expressions and are easy to manipulate. We believe that this constraint-based approach could be fruitfully used for other analysis tasks, such as, for example, type normalization or path minimization under a DTD.

As a proof of concept of our approach, we implemented our algorithm and tested it against a set of randomly generated regular expression types. Our preliminary experimental evaluation showed that inclusion checking is very fast and is always performed in less than a second, even on very large types (e.g., more than 1000 terms). More information about our implementation can be found at `http://www.di.unipi.it/~sartiani/projects/xelf.html`.

7.1 Acknowledgments

We thank the anonymous referees of DBPL'07 and IS for their constructive comments and suggestions. We would also like to thank Luca Pardini for his work on the implementation of our approach.

References

[1] S. Amer-Yahia, S. Cho, L. V. S. Lakshmanan, D. Srivastava, Minimization of tree pattern queries., in: Proceedings of the 2001 ACM SIGMOD International Conference on Management of Data, Santa Barbara, California, May 21-24, 2001, 2001.

[2] D. Barbosa, A. O. Mendelzon, L. Libkin, L. Mignet, M. Arenas, Efficient incremental validation of XML documents., in: Proceedings of the 20th International Conference on Data Engineering, ICDE 2004, 30 March - 2 April 2004, Boston, MA, USA, IEEE Computer Society, 2004.

[3] M. A. Bender, M. Farach-Colton, The LCA problem revisited., in: G. H. Gonnet, D. Panario, A. Viola (eds.), LATIN, vol. 1776 of Lecture Notes in Computer Science, Springer, 2000.

[4] G. J. Bex, F. Neven, J. V. den Bussche, DTDs versus XML Schema: A practical study, in: Proceedings of the Seventh International Workshop on the Web and Databases, WebDB 2004, June 17-18, 2004, Maison de la Chimie, Paris, France, Colocated with ACM SIGMOD/PODS 2004, 2004.

[5] G. J. Bex, F. Neven, T. Schwentick, K. Tuyls, Inference of concise DTDs from XML data, in: U. Dayal, K.-Y. Whang, D. B. Lomet, G. Alonso, G. M. Lohman, M. L. Kersten, S. K. Cha, Y.-K. Kim (eds.), Proceedings of the 32nd International Conference on Very Large Data Bases, Seoul, Korea, September 12-15, 2006, ACM, 2006.

[6] J. N. Foster, B. C. Pierce, A. Schmitt, A logic your typechecker can count on: Unordered tree types in practice, in: Workshop on Programming Language Technologies for XML (PLAN-X), informal proceedings, 2007.

[7] W. Gelade, W. Martens, F. Neven, Optimizing schema languages for XML: Numerical constraints and interleaving, in: T. Schwentick, D. Suciu (eds.), Proceedings of the 11th International Conference on Database Theory - ICDT 2007, Barcelona, Spain, January 10-12, 2007, vol. 4353 of Lecture Notes in Computer Science, Springer, 2007.

[8] G. Ghelli, D. Colazzo, C. Sartiani, Efficient inclusion for a class of XML types with interleaving and counting, in: M. Arenas, M. I. Schwartzbach (eds.), Proceedings of the 11th International Symposium on Database Programming Languages, DBPL 2007, Vienna, Austria, September 23-24, 2007, Revised Selected Papers, vol. 4797 of Lecture Notes in Computer Science, Springer, 2007.

[9] G. Ghelli, D. Colazzo, C. Sartiani, Linear time membership for a class of xml types with interleaving and counting, in: PLAN-X 2008, Programming Language Technologies for XML, An ACM SIGPLAN Workshop colocated with POPL 2008, San Francisco, California, USA, January 9, 2008, 2008.

[10] W. Martens, F. Neven, T. Schwentick, Complexity of decision problems for simple regular expressions., in: J. Fiala, V. Koubek, J. Kratochvíl (eds.), Proceedings of the 29th International Symposium on Mathematical Foundations of Computer Science, MFCS 2004, Prague, Czech Republic, August 22-27, 2004, vol. 3153 of Lecture Notes in Computer Science, Springer, 2004.

[11] A. J. Mayer, L. J. Stockmeyer, Word problems — this time with interleaving, Inf. Comput. 115 (2) (1994) 293–311.

[12] H. S. Thompson, D. Beech, M. Maloney, N. Mendelsohn, XML Schema Part 1: Structures Second Edition, Tech. rep., World Wide Web Consortium, W3C Recommendation (Oct 2004).

[13] P. T. Wood, Containment for XPath fragments under dtd constraints, in: D. Calvanese, M. Lenzerini, R. Motwani (eds.), Proceedings of the 9th International Conference on Database Theory - ICDT 2003, Siena, Italy, January 8-10, 2003, vol. 2572 of Lecture Notes in Computer Science, Springer, 2003.