A basic contract language for web services *

Samuele Carpineti and Cosimo Laneve

Department of Computer Science, University of Bologna, Italy.
{carpinet,laneve}@cs.unibo.it

Abstract. We design a schema language that includes channel schemas with capabilities of input, output, and input-output. These schemas may describe documents containing references to operations of remote services on the web. In this language, the subschema relation turns out to have an exponential cost. We therefore discuss a language restriction that admits a subschema relation with a polynomial cost.

1 Introduction

Several schema languages have been recently proposed for describing the tree-structure of XML documents. We recall DTD [12], XML-Schema [9], RELAX NG [5], and `XDuce` types [7] and we refer to [13] for an analysis of their expressiveness. These schema languages are used in `WSDL` [11, 10] documents that are interfaces of web-services describing the messages sent and/or received by the services and the informations for reaching the services (location, transport protocol, etc.). For example, the one-way operation in `WSDL` (we are omitting some details)

```
<portType name="op-one-way">
 <operation name="one-way">
     <input message="Real"/>
 </operation>
</portType>
<service name="one-way-service">
 <port name="op-one-way">
     <address location="http://example.com/op-one-way"/>
 </port>
</service>
```

is expressing that the reference at `http://example.com/op-one-way` may be invoked with documents of schema `Real`. `WSDL` documents are also used in repositories for selecting appropriate referencesIn this context, "`http://example.com/op-one-way`" may be returned to queries asking for references that can be invoked with `Integer` (because integers are also reals). A client receiving "`http://example.com/op-one-way`", besides invoking it, might forward the reference to a third party that, in turn, could invoke `op-one-way` with `Natural`.

* Aspects of this investigation were supported in part by a Microsoft initiative in concurrent computing and web services.

Yet, web-services technologies also require the possibility to express and communicate references to operations of remote services [15] and to verify that the receiver uses the service according to its contract (sending proper data and performing the permitted operations). In facts, these requirements are recognized in the new specification of `WSDL` [10], which extends the schemas with references to interfaces of web-services (called `portType`s). However this extension is by no means satisfactory because no mechanism for comparing schemas with references is provided at all.

We therefore design a basic schema language with references $\langle S \rangle^i$, $\langle S \rangle^o$, and $\langle S \rangle^{io}$, called *channel schemas*, that collect references of schema S and being respectively used to receive notifications, to invoke services, and for both. In our notation, the channel `http://example.com/op-one-way` has schema $\langle \texttt{Real} \rangle^o$. The assessment that (channel) schemas are used according to the `WSDL` description is given by a subschema relation $<:$. Following [2, 14], $<:$ is the largest relation satisfying the closure property "if $S <: T$ then every branch of the syntax tree of S is matched by those of T yielding pairs that are still in $<:$ ". This matching is actually weakened for tag-labelled branches because, in our schema language, union schemas may be nondeterministic. To illustrate the problem, let $S = a[\texttt{Int} + \texttt{String}], c[\texttt{Int}]$ and $T = a[\texttt{Int}], c[\texttt{Int}] + a[\texttt{String}], c[\texttt{Int}]$. It turns out that $S <: T$ however, to demonstrate this, one has to pick one addend of T, let it be $T' = a[\texttt{Int}], c[\texttt{Int}]$, compute the difference of S and T', and show that this difference is still in T. In this case the difference is $a[\texttt{String}], c[\texttt{Int}]$, which is clearly contained in T.

The relation $<:$ turns out to be computationally expensive – it has an exponential cost with respect to the sizes of the schemas [8]. This is an issue in web-services, where data coming from untrusted parties, such as `WSDL` documents, might be validated at run-time before processing. While validation has a polynomial cost with respect to the size of the datum in current schema languages, this is not so when data carry references. In these cases, validation has to verify that the schema of the reference conforms with some expected schema, thus reducing itself to the subschema relation. (In `XDuce` run-time subschema checks are avoided because programs are strictly coupled and typechecking guarantees that invalid values cannot be produced.)

To avoid significative run-time degradations of web-services technologies, we impose a language restriction to diminish the cost of the subschema relation. Specifically, following XML Schema, we constrain schemas to retain a deterministic model as regards tag-labelled transitions. The model is still nondeterministic with respect to channel-labelled transitions. The resulting schemas, called *labelled-determined*, are equipped with a subschema relation defined as a set of syntax-directed rules. We prove the equivalence of this subschema relation with $<:$ and we demonstrate that it has a polynomial cost with respect to the sizes of the schemas. This result extends to channel schemas the computational complexity of language difference for deterministic tree automata (and XML Schemas) computed in [8].

Related works. The schema language studied in this article is similar to those introduced in languages extending π-calculus with XML datatypes [3, 1, 4]. The design of the schema language of [3] has been strongly affected by this study. As a minor difference, channel schemas in [3] only have output capabilities. The schema language of [1] is simpler than the one in this paper. In particular labelled schemas have singleton labels and the subschema relation seems not powerful enough (for example $a[b[\,] + c[\,]] <: a[b[\,]] + a[c[\,]]$ does not hold in [1]).

The types in [4] include channels with capabilities, union, product, intersection and negation. The definition of subschema is semantic, by means of a set-inclusion on a set-theoretic model. Our schema language is simpler than [4] and the notion of subschema is quite different. For example, in our case, top and bottom are derived schemas and channel schemas may be nested at wish, while this is problematic in presence of recursion and intersection. The contribution [4] overlooks the restrictions for reducing the computational complexity of the subschema relation that turns out to be hyperexponential.

Structure of the paper. We proceed as follows. Section 2 reviews `WSDL` and describes how operations may be encoded in our schema language. The schema language with channels is formally described in Section 3. In Section 4 we define the subschema relation $<:$ and analyze some of its properties. In Section 5 we discuss the constraint of labelled-determinedness and design the alternative syntax-directed subschema definition. We also analyze its algorithmic cost. The appendix is devoted to the proof of equivalence of $<:$ and the syntax-directed subschema.

2 Encoding WSDL interfaces

`WSDL` documents are XML documents that consist of several parts. Among these parts, *ports* are logical groupings of operations that are defined by a name, an interaction pattern, and the schema of messages for invoking the operations and receiving back the answers. Operations may use four interaction patterns: *one-way*, *notification*, *request-response*, and *solicit-response*. The former twos model asynchronous unidirectional communications and require a single schema: in one-way, the schema describes the messages to invoke the operation; in notification, the schema describes the messages returned by the invocation. Request-response and solicit-response operations model two communication actions. Therefore they require two schemas. In request-response, the two schemas describe the messages to invoke the operation and to receive the answer, respectively; in solicit-response, schemas are in the other way around.

The one-way operation in `WSDL`1.1 (we are omitting some details of the `WSDL` document)

```
<portType name="op-one-way">
 <operation name="one-way">
     <input message="InvokeScm"/>
 </operation>
```

```
</portType>
<service name="one-way-service">
 <port name="op-one-way">
     <address location="http://example.com/op-one-way"/>
 </port>
</service>
```

is expressing that the reference at `http://example.com/op-one-way` may be invoked with documents of schema `InvokeScm`. Technically, the tag `<input message="`S`"/>` in the `WSDL` must be interpreted as a schema constructor collecting references that may be invoked with values of schema S, or with subsets of such values. Said otherwise, the constructor `<input message="..."/>` behaves contravariantly with respect to the argument schema. In our notation, introduced in the next section, the operation `one-way` has schema $\langle \texttt{InvokeScm} \rangle^o$.

The notification operation is defined by

```
<operation name="notification">
    <output message="ReturnScm"/>
</operation>
```

The intended meaning of this pattern is that the remote service is communicating the schema of the messages it will send back. To receive this message, the client service has to create a reference whose schema in our notation is (greater than) $\langle \texttt{ReturnScm} \rangle^i$. It is worth to remark that, operationally, the notification is equivalent to delivering a fresh reference of schema $\langle \texttt{ReturnScm} \rangle^i$ to the client. The capability "i" constrains the client to use the reference for receiving messages.

The request-response operation is defined by (as usual, some details of the `WSDL` document are omitted)

```
<portType name="op-request-response">
 <operation name="request-response">
     <input message="InvokeScm"/>
     <output message="ReturnScm"/>
 </operation>
</portType>
<service name="op-request-response">
 <port name="op-request-response">
    <address location="http://example.com/request-response"/>
 </port>
</service>
```

In this case, the connection with the service at `http://example.com/request-response` is *bidirectional*, that is two references are created: one for invoking the service and the other for receiving the return value. The two have schemas $\langle \texttt{InvokeScm} \rangle^o$ and $\langle \texttt{ReturnScm} \rangle^i$, respectively.

Finally, the solicit-response operation is described by

```
<operation name="solicit-response">
    <output message="ReturnScm"/>
    <input message="InvokeScm"/>
</operation>
```

Also in this case two references are created during the connection. The first reference is for receiving solicitations and is described by the schema $\langle \texttt{ReturnScm} \rangle^i$; the second reference is for responses and is described by $\langle \texttt{InvokeScm} \rangle^o$.

3 Schemas with channels

We use two disjoint countably infinite sets: the *tags*, ranged over by $a, b, \cdots$, and the *schema names*, ranged over by $\texttt{U}, \texttt{V}, \cdots$. The term κ is used to range over i, o, and io. The syntax of our language includes the categories of *labels* and *schemas* defined by the following rules

$L ::=$		**label**	$S ::=$		**schema**
	a	(tag)		$\bot$	(empty schema)
	$\sim$	(wildcard label)		$()$	(void schema)
	$L + L$	(union)		$\langle S \rangle^\kappa$	(channel schema)
	$L \setminus L$	(difference)		$L[S]\texttt{,}S$	(labelled sequence schema)
				$S + S$	(union schema)
				$\texttt{U}$	(schema name)

Labels. Labels specify collections of tags. The semantics of labels is defined by the following function $\widehat{\cdot}$:

$$\widehat{a} = \{a\} \qquad \widehat{\sim} = \{a, b, c, \cdots\} \qquad \widehat{L + L'} = \widehat{L} \cup \widehat{L'} \qquad \widehat{L \setminus L'} = \widehat{L} \setminus \widehat{L'}$$

($\sim$ represents the whole sets of tags). We write $a \in L$ for $a \in \widehat{L}$.

Schema. Schemas describe (XML) documents that are structurally similar. The schema $\bot$ describes the empty set of documents; $()$ describes the empty document; $\langle S \rangle^\kappa$ describes references whose messages have schema S and that may be used with *capability* $\kappa \in \{i, o, io\}$. The capabilities i, o, io mean that the reference can be used for performing inputs, outputs, and both inputs and outputs, respectively. The schema $L[S]\texttt{,}S'$ describes a sequence starting with a document having a tag in $\widehat{L}$ and a document of schema S as content, and followed by a document of schema S'. Finally $S + S'$ describes the set of documents belonging to S or S'. The schema name $\texttt{U}$ describes the set of documents such that $\texttt{U} = \mathcal{E}(\texttt{U})$, where $\mathcal{E}$ is a fixed mapping from names to schemas that fulfills the following *finiteness* and *guardedness properties*. Let $\texttt{names}(S)$ be the least set containing the schema names in S and such that if $\texttt{U} \in \texttt{names}(S)$ then $\texttt{names}(\mathcal{E}(\texttt{U})) \subseteq \texttt{names}(S)$. A map $\mathcal{E}$ is *finite* if, for every $\texttt{U} \in \texttt{dom}(\mathcal{E})$, the set $\texttt{names}(\texttt{U})$ is finite. A map $\mathcal{E}$ is *guarded* if every occurrence of $\texttt{U}$ in $\mathcal{E}(\texttt{U})$ is underneath a channel or labelled sequence schema constructor.

In the following, $L[()]$ and $L[S]\texttt{,}()$ are always abbreviated into $L[\,]$ and $L[S]$, respectively.

We illustrate the syntax by means of few sample schema name definitions. Let `Bool`, `Blist`, and `Btree` be such that

$$\begin{aligned}
\mathcal{E}(\texttt{Bool}) &= \mathit{true}[\,] + \mathit{false}[\,] \\
\mathcal{E}(\texttt{Blist}) &= () + \mathit{bool}[\texttt{Bool}]\texttt{,}\texttt{Blist} \\
\mathcal{E}(\texttt{Btree}) &= () + \mathit{val}[\texttt{Bool}]\texttt{,}\mathit{left}[\texttt{Btree}]\texttt{,}\mathit{right}[\texttt{Btree}] \\
\mathcal{E}(\texttt{Empty}) &= a[\texttt{Empty}]
\end{aligned}$$

The name `Bool` defines booleans that are encoded as tags *true* and *false* with content `()`. The name `Blist` defines any flat sequence of labelled documents containing booleans; `Btree` defines documents that are binary trees of booleans. The name `Empty` defines an empty set of documents because this set is the least solution of the equation $\texttt{Empty} = a[\texttt{Empty}]$. As such `Empty` is equal to $\bot$.

As regards channel schemas, $\langle\texttt{Bool}\rangle^o$ describes references that may be invoked with booleans; $\langle\texttt{Bool}\rangle^{io}$ contains references that may be invoked with booleans *and* may receive notifications carrying booleans. The name `NCbool` defined as

$$\mathcal{E}(\texttt{NCbool}) = \langle\texttt{Bool}\rangle^o + \langle\texttt{NCbool}\rangle^o$$

describes the references to be invoked with booleans or with references to be invoked with booleans, etc., till some finite but not bound depth. (The nesting of channel constructors in [4] is always bound.) We observe that a service querying a repository for references of schema $\langle\texttt{Bool}\rangle^o$ may get back a service of schema $\langle\texttt{Bool}\rangle^{io}$ or of schema `NCbool`. Conversely, if the query is about references of schema $\langle\texttt{Bool}\rangle^{io}$ then the repository will never return references of schema $\langle\texttt{Bool}\rangle^o$ nor `NCbool`.

Remarks.

1. According to the above grammar, sequences are lists of labelled elements concluded either by the void schema (the empty sequence), or by a channel schema, or by a name (we ignore sequences with a tailing $\bot$ because they are equivalent to $\bot$, see the forthcoming relation of subschema). Since schema names may only occur in tail position of sequences, it is not possible to define context-free schemas like $a[\,]^n\texttt{,}b[\,]^n$. Said otherwise, our grammars defines *tree regular schemas*, a class of languages that retain decision algorithms for language inclusion – the subschema relation [8].
2. The subschema language without channel schemas is closed under union, difference, and intersection [7]. Union closure is a consequence of the presence of union schemas; difference closure $S \setminus T$ follows by the fact that labels are represented as sets. For example $L[S]\texttt{,}S' \setminus L'[T]\texttt{,}T'$ is $(L \setminus L')[S]\texttt{,}S' + L[S \setminus T]\texttt{,}S' + L[S]\texttt{,}S' \setminus T'$. Intersection $S \cap T$ may be defined in terms of difference as $(S + T) \setminus (S \setminus T) \setminus (T \setminus S)$. This sublanguage has a decidable algorithm testing the emptiness of a schema. Thereafter $S <: T$ may be implemented as an emptiness test on $S \setminus T$. Channel schemas does not preserve the closures under difference and intersection. For this reason these operators are primitive in [4].

4 The subschema relation

The semantic definition of subschema in [6] does not adapt well to our language. In that paper, a language for *values* was introduced and a schema S was considered a subschema of T if the set of values described by S was contained in the set of values described by T. In our case values should contain references that do not carry any "structural" information about their schema. Therefore, in order to verify that a reference belongs to a schema S, we should verify the schema of the reference is a subschema of S. To circumvent this circularity we use an "operational" definition – a *simulation* relation – in the style of [2, 14].

The subschema relation uses handles to manifest all the branches of the syntax tree of a schema. Let μ range over $\mathtt{()}$, $\langle\rangle^\kappa(S)$, $L(S\ ;\ T)$ and let $S \downarrow \mu$, read S *has a handle* μ, be the least relation such that:

$\mathtt{()} \downarrow \mathtt{()}$	
$\langle S\rangle^\kappa \downarrow \langle\rangle^\kappa(S)$	
$L[S]\mathtt{,}T \downarrow L(S\ ;\ T)$	if $\widehat{L} \neq \emptyset$ and there are μ, μ' such that $S \downarrow \mu$ and $T \downarrow \mu'$
$S + T \downarrow \mu$	if $S \downarrow \mu$ or $T \downarrow \mu$
$\mathtt{U} \downarrow \mu$	if $\mathcal{E}(\mathtt{U}) \downarrow \mu$

We observe that $\bot$ has no handle. The schema $a[\ ]\mathtt{,}\bot$ has no handle as well; the reason is that a sequence has a handle provided that every element of the sequence has a handle. We also remark that a channel $\langle S\rangle^\kappa$ always retains a handle. A schema S is *not-empty* if and only if S has a handle; it is *empty* otherwise.

In the following definition we use the intersection operator on labels: $L \cap L' \stackrel{\text{def}}{=} \mathtt{~} \setminus ((\mathtt{~} \setminus L) + (\mathtt{~} \setminus L'))$.

Definition 1. *Let $\leq$ be the least partial order on capabilities such that $io \leq i$ and $io \leq o$. The* subschema relation $\mathtt{<:}$ *is the largest relation on schemas such that $S \mathtt{<:} T$ implies:*

1. *if $S \downarrow \mathtt{()}$ then $T \downarrow \mathtt{()}$;*
2. *if $S \downarrow \langle\rangle^\kappa(S')$ then $T \downarrow \langle\rangle^{\kappa'}(T')$ with $\kappa \leq \kappa'$ and one of the followings holds:*
   (a) *$\kappa' = o$ and $T' \mathtt{<:} S'$;*
   (b) *$\kappa' = i$ and $S' \mathtt{<:} T'$;*
   (c) *$\kappa' = io$ and $S' \mathtt{<:} T'$ and $T' \mathtt{<:} S'$;*
3. *if $S \downarrow L(S'\ ;\ S'')$ then $T \downarrow L'(T'\ ;\ T'')$ with $\widehat{L} \cap \widehat{L'} \neq \emptyset$ and:*
   (a) *either $\widehat{L} \subseteq \widehat{L'}$, $S' \mathtt{<:} T'$, and $S'' \mathtt{<:} T''$;*
   (b) *or $(L \setminus L')[S']\mathtt{,}S'' + (L \cap L')[R']\mathtt{,}S'' + (L \cap L')[S']\mathtt{,}R'' \mathtt{<:} T$, for some R' and R'' such that $S' \mathtt{<:} T' + R'$ and $S'' \mathtt{<:} T'' + R''$.*

The item 1 constraints greater schema to manifest a void handle if the smaller one retains such a handle. The item 2 reduces the subschema relation on channel schemas to the subschema of the arguments according to the capability. In case of output capability the relation is inverted on the arguments (contravariance), in case of input capability the relation is the same for the arguments (covariance),

in case of input-output capability the relation reduces to check the equivalence of the arguments (invariance). The item 3.a allows one to reduce the subschema relation to the schema arguments of handles -(- ; -) when the labels of the smaller schema are contained into those of the greater schema. The item 3.b is the problematic one: it weakens the item 3.a to those cases when the smaller schema shows up a handle $L(S'\ ;\ S'')$ and the greater one has no handle $L'(T'\ ;\ T'')$ with $L \subseteq L'$ and $S' <: T'$ and $S'' <: T''$. To explain item 3.a, we use a schema difference operator "\". (Contrary to [4], our schemas are not closed by difference. This operator is only used for the sake of explanation.) If $T \downarrow L'(T'\ ;\ T'')$, in order to prove $L[S'], S'' <: T$ one may reduce to demonstrate that $(L[S'], S'') \setminus (L'[T'], T'') <: T$. Such difference of labelled sequence schemas is equal to $(L \setminus L')[S'], S'' + (L \cap L')[S' \setminus T'], S'' + (L \cap L')[S'], (S'' \setminus T'')$. We observe that proving

$$(L \setminus L')[S'], S'' + (L \cap L')[S' \setminus T'], S'' + (L \cap L')[S'], (S'' \setminus T'') <: T$$

is equivalent to proving

$$\begin{array}{ll} & (L \setminus L')[S'], S'' + (L \cap L')[R'], S'' + (L \cap L')[S'], R'' <: T \\ \text{and} & S' <: T' + R' \\ \text{and} & S'' <: T'' + R'' \end{array}$$

that does not mention the difference operator. This is exactly what 3.b says. Let us illustrate 3.b for deriving $c[a[\ ] + b[\ ]], (d[\ ] + e[\ ]) <: T$, where $T = (c[a[\ ]], d[\ ]) + (c[b[\ ]], (d[\ ] + e[\ ]) + (c[a[\ ]], e[\ ])$. Since $T \downarrow c(a[\ ]\ ;\ d[\ ])$, by 3.b, one may reduce to verifying that $c[R'], (d[\ ] + e[\ ]) + c[a[\ ] + b[\ ]], R'' <: T$ with $R' = b[\ ]$ and $R'' = e[\ ]$. The relationship $c[b[\ ]], (d[\ ] + e[\ ]) <: T$ follows by 3.a because $c[b[\ ]], (d[\ ] + e[\ ])$ is the second addend of T. As regards $c[a[\ ] + b[\ ]], e[\ ] <: T$ we observe that $T \downarrow c(b[\ ]\ ;\ d[\ ] + e[\ ])$. This reduces to $c[a[\ ]], e[\ ] <: T$, which is true because $c[a[\ ]], e[\ ]$ is the third addend of T.

The schemas `Chan` and `Any` defined as:

$$\begin{array}{rcl} \mathcal{E}(\texttt{Chan}) & = & \langle \bot \rangle^o + \langle \texttt{Any} \rangle^i \\ \mathcal{E}(\texttt{Any}) & = & \texttt{()} + \texttt{\~{}}[\texttt{Any}], \texttt{Any} + \texttt{Chan} \end{array}$$

own relevant properties. `Chan` collects all the channel schemas, no matter what they can carry; `Any` collects all the documents, namely possibly empty sequences of documents, including channel schemas, no matter how they are labelled (the label "~"). We observe that $\langle \bot \rangle^o$ and $\langle \texttt{Any} \rangle^o$ are very different. $\langle \bot \rangle^o$ collects every reference with either capability "o" or "io", $\langle \texttt{Any} \rangle^o$ refers only to references where that arbitrary data can be sent. For instance $\langle a[\ ] \rangle^o$ is a subschema of $\langle \bot \rangle^o$ but not of $\langle \texttt{Any} \rangle^o$. The channel schemas $\langle \texttt{Any} \rangle^i$ and $\langle \bot \rangle^i$ are different as well. $\langle \texttt{Any} \rangle^i$ refers to references that may receive arbitrary data; $\langle \bot \rangle^i$ refers to a reference that cannot receive anything.

We also remark about differences between labelled schemas and channel schemas. Let $R = a[\texttt{Blist}] + a[\texttt{Btree}]$ and $R' = a[\texttt{Blist} + \texttt{Btree}]$. Then $R <: R'$ and $R' <: R$. However $Q = \langle \texttt{Blist} \rangle^\kappa + \langle \texttt{Btree} \rangle^\kappa$ is not subschema-equivalent to $Q' = \langle \texttt{Blist} + \texttt{Btree} \rangle^\kappa$. Let us discuss the case $\kappa = i$ that is similar to $L[\cdot]$

because covariant. It is possible to prove that $Q <: Q'$. However the converse is false because references in Q may be invoked only with documents that are lists of booleans or only with documents that are trees of booleans. Channels in Q' may be invoked with documents belonging either to `Blist` or to `Btree`.

A few properties of $<:$ are in order.

Proposition 1. *1.* $<:$ *is reflexive and transitive;*
2. (Contravariance of $\langle\cdot\rangle^o$*)* $S <: T$ *if and only if* $\langle T\rangle^o <: \langle S\rangle^o$*;*
3. (Covariance of $\langle\cdot\rangle^i$*)* $S <: T$ *if and only if* $\langle S\rangle^i <: \langle T\rangle^i$*;*
4. (Invariance of $\langle\cdot\rangle^{io}$*)* $S <: T$ *and* $T <: S$ *if and only if* $\langle S\rangle^{io} <: \langle T\rangle^{io}$*;*
5. If S *is empty then* $S <: \bot$*;*
6. For every S*,* $\bot <: S <: \texttt{Any}$ *and* $\langle S\rangle^\kappa <: \texttt{Chan}$ *and* $\langle \texttt{Any}\rangle^{io} <: \langle S\rangle^o$ *and* $\langle\bot\rangle^{io} <: \langle S\rangle^i$*.*

We discuss Proposition 1.5. The name $\bot$ (as well as `Empty`) has no handle; thereafter it is a subschema of any other schema. To prove $S <: \texttt{Any}$, consider the relation $\mathcal{R} = \{(S, \texttt{Any}) \mid S \text{ is a schema}\}$. It is easy to to prove that $\texttt{()} <: \texttt{Any}$ and that $L[S]\texttt{,}T <: \texttt{Any}$, for every L, S, and T. As regards channel schemas $\langle S\rangle^\kappa$, it sufficies to demonstrate that $\langle S\rangle^\kappa <: \texttt{Chan}$. By definition of $<:$, if $\kappa \leq o$ then $\langle S\rangle^\kappa \leq \langle S\rangle^o$. This fact, $\bot <: S$, and Proposition 1.2 yield $\kappa \leq o$ implies $\langle S\rangle^\kappa <: \langle\bot\rangle^o$. If $\kappa = i$ then $\langle S\rangle^i \downarrow \langle\rangle^i(S)$ and $\texttt{Chan} \downarrow \langle\rangle^i\texttt{Any}$, and we are reduced to $(S, \texttt{Any}) \in \mathcal{R}$, which is true. We are left with $\langle \texttt{Any}\rangle^{io} <: \langle S\rangle^o$ and $\langle\bot\rangle^{io} <: \langle S\rangle^i$. We detail the former, the last statement is similar. By Proposition 1.2 applied to $S <: \texttt{Any}$ we obtain $\langle \texttt{Any}\rangle^o <: \langle S\rangle^o$; then by Proposition 1.1 and definition of $<:$, we derive $\langle \texttt{Any}\rangle^{io} <: \langle S\rangle^o$.

4.1 Primitive types

The extension of our schema language with primitive types is not difficult. Consider the new syntax:

T ::=		**primitive types**	S ::=		**schema**
	`n`	(integer constant)		$\cdots$	
	`"s"`	(string constant)		`T`	(primitive types)
	`Int`	(integers)			
	`String`	(strings)			

The primitive types `n`, `"s"`, `Int`, and `String` respectively describe a specific integer, a specific string, the set of integers, and the set of strings. For example, the schema that collects integers and strings is $\texttt{Int} + \texttt{String}$; the schema that collects references with integer messages is $\langle\texttt{Int}\rangle^i + \langle\texttt{Int}\rangle^o$. As in XML-Schema, sequences of primitive types are not allowed: in our language every sequence must be composed by labelled elements (except the tailing one).

As regards the subschema relation, the handles are extended with $\texttt{T} \downarrow \texttt{T}$. Let $\leq_p$ be the least partial order on primitive types such that $\texttt{n} \leq_p \texttt{Int}$ and $\texttt{"s"} \leq_p \texttt{String}$. To define the subschema relation for the new language it sufficies to extend Definition 1 with

4. *if* $S \downarrow \mathtt{T}$ *then* $T \downarrow \mathtt{T}'$ *and* $\mathtt{T} \leq_p \mathtt{T}'$.

It follows that $1 + \mathtt{Int} <: \mathtt{Int}$ and $a[1 + \mathtt{"bye"}] <: a[1] + a[\mathtt{"bye"}]$ (the proofs are left to the reader).

5 Labelled-determined schema

The relation $<:$ can be verified in exponential time [8]. As we have discussed in the Introduction, this is problematic when $<:$ must be computed at run time, such as when received references must be validated. In this section we study a restriction of the schema language that bears a polynomial subschema algorithm (and validation program). The restriction prevents unions of schemas having a common starting tag and is similar to the restriction used in single-type tree grammars [13] such as XML-Schema. The restrictions also allows an alternative definition of subschema that, instead of examining the potentiality to produce handles, compares the syntactic structure of the schemas.

Definition 2. *The set* `ldet` *of* labelled-determined schemas *is the least set containing empty schemas and such that:*

1. $() \in \mathtt{ldet}$;
2. *if* $S \in \mathtt{ldet}$ *then* $\langle S \rangle^\kappa \in \mathtt{ldet}$;
3. *if* $S \in \mathtt{ldet}$ *and* $T \in \mathtt{ldet}$ *then* $L[S]\mathtt{,}T \in \mathtt{ldet}$;
4. *if* $S \in \mathtt{ldet}$ *and* $T \in \mathtt{ldet}$ *and, for every* $S \downarrow L(S' \,;\, S'')$ *and* $T \downarrow L'(T' \,;\, T'')$, $\widehat{L} \cap \widehat{L'} = \emptyset$ *then* $S + T \in \mathtt{ldet}$;
5. *if* $\mathcal{E}(\mathtt{U}) \in \mathtt{ldet}$ *then* $\mathtt{U} \in \mathtt{ldet}$.

For example, $\mathtt{Empty} \in \mathtt{ldet}$ because `ldet` is closed by empty schemas; if $S \in \mathtt{ldet}$ and $T \in \mathtt{ldet}$ then $a[S] + (\mathtt{\sim} \setminus a)[T] \in \mathtt{ldet}$ and $\mathtt{\sim}[S] + \langle S \rangle^\kappa + \langle T \rangle^{\kappa'} \in \mathtt{ldet}$. The last example displays that union of channel schemas does not invalidate labelled-determinedness. The schemas $a[\,] + (a + b)[\,]$ and $\langle a[\,] + \mathtt{\sim}[\,] \rangle^\kappa$ are not labelled-determined.

Of course, Definition 1 also holds for labelled-determined schemas. For these schemas $<:$ is much simpler. Item *3* of Definition 1 can be simplified to:

3. *if* $S \downarrow L(S' \,;\, S'')$ *then* $T \downarrow L'(T' \,;\, T'')$ *with* $\widehat{L} \cap \widehat{L'} \neq \emptyset$ *and* $S' <: T'$, $S'' <: T''$, *and* $(L \setminus L')[S']\mathtt{,}S'' <: T$.

Alternatively, one may also consider the following formulation of item 3 (this is the one that is used in the proof of Theorem 1):

3. *if* $S \downarrow L(S' \,;\, S'')$ *then there is* I *such that, for every* $i \in I$, $T \downarrow L_i(T_i' \,;\, T_i'')$, $\widehat{L} \cap \widehat{L_i} \neq \emptyset$, $\widehat{L} \subseteq \bigcup_{i \in I} \widehat{L_i}$, $S' <: T_i'$, *and* $S'' <: T_i''$.

However, labelled-determined schemas retain a different, more algorithmic definition of subschema. This definition is presented below as a set of syntax-directed rules defining a relation $S \precsim_{\mathtt{A}} T \Rightarrow \mathtt{A}'$ where $\mathtt{A}$ and $\mathtt{A}'$ are sets of pairs $(\mathtt{U}, R)$ – the first element is always a schema name – that are used to detect

termination. In what follows we abbreviate $S \precsim_{\mathtt{A}} T \Rightarrow \mathtt{A}'$ into $S \precsim_{\mathtt{A}} T$ when we are not interested in $\mathtt{A}'$.

Let $\mathtt{first}(S) \stackrel{\text{def}}{=} \sum_{S \downarrow L(S'\,;\,S'')} L$.

Definition 3. *The syntax-directed subschema relation $\precsim_{\mathtt{A}}$ is the smallest relation closed under commutativity of unions and under the rules in Table 1.*

Table 1. The subschema relation $\precsim_{\mathtt{A}}$.

$$\text{(VOID)}\ \ \mathtt{()} \precsim_{\mathtt{A}} \mathtt{()} \Rightarrow \mathtt{A} \qquad \text{(BOT)}\ \ \bot \precsim_{\mathtt{A}} T \Rightarrow \mathtt{A} \qquad \text{(LBOT)}\ \ \frac{S \precsim_{\mathtt{A}} \bot \Rightarrow \mathtt{A}'}{L[S]\mathtt{,}S' \precsim_{\mathtt{A}} T \Rightarrow \mathtt{A}'} \qquad \text{(SBOT)}\ \ \frac{S' \precsim_{\mathtt{A}} \bot \Rightarrow \mathtt{A}'}{L[S]\mathtt{,}S' \precsim_{\mathtt{A}} T \Rightarrow \mathtt{A}'}$$

$$\text{(CHAN-I)}\ \ \frac{\kappa \leq i \quad S \precsim_{\mathtt{A}} T \Rightarrow \mathtt{A}'}{\langle S\rangle^{\kappa} \precsim_{\mathtt{A}} \langle T\rangle^{i} \Rightarrow \mathtt{A}'} \qquad \text{(CHAN-O)}\ \ \frac{\kappa \leq o \quad T \precsim_{\mathtt{A}} S \Rightarrow \mathtt{A}'}{\langle S\rangle^{\kappa} \precsim_{\mathtt{A}} \langle T\rangle^{o} \Rightarrow \mathtt{A}'} \qquad \text{(CHAN-IO)}\ \ \frac{S \precsim_{\mathtt{A}} T \Rightarrow \mathtt{A}' \quad T \precsim_{\mathtt{A}'} S \Rightarrow \mathtt{A}''}{\langle S\rangle^{io} \precsim_{\mathtt{A}} \langle T\rangle^{io} \Rightarrow \mathtt{A}''}$$

$$\text{(RSEQ)}\ \ \frac{\widehat{L} \subseteq \widehat{L'} \qquad S \precsim_{\mathtt{A}} T \Rightarrow \mathtt{A}' \quad S' \precsim_{\mathtt{A}'} T' \Rightarrow \mathtt{A}''}{L[S]\mathtt{,}S' \precsim_{\mathtt{A}} L'[T]\mathtt{,}T' \Rightarrow \mathtt{A}''} \qquad \text{(UNIONR)}\ \ \frac{S \precsim_{\mathtt{A}} T \Rightarrow \mathtt{A}'}{S \precsim_{\mathtt{A}} T + T' \Rightarrow \mathtt{A}'} \qquad \text{(UNIONL)}\ \ \frac{S \precsim_{\mathtt{A}} T \Rightarrow \mathtt{A}' \quad S' \precsim_{\mathtt{A}'} T \Rightarrow \mathtt{A}''}{S + S' \precsim_{\mathtt{A}} T \Rightarrow \mathtt{A}''}$$

$$\text{(LSEQ)}\ \ \frac{L' = \mathtt{first}(T) \quad \emptyset \subsetneq \widehat{L} \cap \widehat{L'} \subsetneq \widehat{L} \qquad (L \cap L')[S]\mathtt{,}S' \precsim_{\mathtt{A}} T \Rightarrow \mathtt{A}' \quad (L \setminus L')[S]\mathtt{,}S' \precsim_{\mathtt{A}'} T' \Rightarrow \mathtt{A}''}{L[S]\mathtt{,}S' \precsim_{\mathtt{A}} T + T' \Rightarrow \mathtt{A}''}$$

$$\text{(NAMEL)}\ \ \frac{(\mathtt{U}, T) \in \mathtt{A}}{\mathtt{U} \precsim_{\mathtt{A}} T \Rightarrow \mathtt{A}} \qquad \text{(NAMEH)}\ \ \frac{\mathtt{A}' = \mathtt{A} \cup \{(\mathtt{U}, T)\} \quad \mathcal{E}(\mathtt{U}) \precsim_{\mathtt{A}'} T \Rightarrow \mathtt{A}''}{\mathtt{U} \precsim_{\mathtt{A}} T \Rightarrow \mathtt{A}''} \qquad \text{(NAMER)}\ \ \frac{S \precsim_{\mathtt{A}} \mathcal{E}(\mathtt{U}) \Rightarrow \mathtt{A}'}{S \precsim_{\mathtt{A}} \mathtt{U} \Rightarrow \mathtt{A}'}$$

The first four rules are simple and do not require any comment. Rules (CHAN-I), (CHAN-O), and (CHAN-IO) reduce subschema to the arguments of the channel constructors; they respectively establish covariant, contravariant, and invariant relationships on the arguments. Rules (RSEQ) and (LSEQ) define the subschema relation for sequences. The former applies if the arguments are already sequences. This rule, together with (UNIONR), permits to single out the sequence branch, if any, of the right argument. However, rules (RSEQ) and (UNIONR) do not suffice for proving that $\mathtt{\sim[()]\mathtt{,}()} <: a[\mathtt{()}]\mathtt{,}\mathtt{()} + (\mathtt{\sim} \setminus a)[\mathtt{()}]\mathtt{,}\mathtt{()}$. In this case $\mathtt{\sim}$ needs to be partitioned and this operation is performed by (LSEQ). It is worth noticing that rule (LSEQ), due to labelled-determinedness, only requires that $(L \cap L')[S]\mathtt{,}S' \precsim_{\mathtt{A}} T$, not just $(L \cap L')[R]\mathtt{,}R' \precsim_{\mathtt{A}} T$ with $S \not\precsim_{\mathtt{A}} R$ or $S' \not\precsim_{\mathtt{A}} R'$ (see item 3.b of Definition 1).

The last three rules are about schema names. Rule (NAMEL) derives a subschema $\mathtt{U} \precsim_{\mathtt{A}} T$ if the pair $(\mathtt{U}, T)$ is in the (hypothesis) set $\mathtt{A}$. Rule (NAMER) unfolds the name $\mathtt{U}$ when it is the right argument. Rule (NAMEH) is the unique one that uses an augmented set in the hypotheses. According to this rule, in order to prove that $\mathtt{U} \precsim_{\mathtt{A}} T$, one unfolds $\mathtt{U}$ and, at the same time, it is reminded that $\mathtt{U} \precsim_{\mathtt{A}} T$ is being proved. This rimind is stored in $\mathtt{A}'$. Such a machinery permits to avoid loops: if, during the proof of $\mathtt{U} \precsim_{\mathtt{A}} T$, one reduces to $\mathtt{U} \precsim_{\mathtt{A}'} T$ then it is possible to terminate (by rule (NAMEL)). This is the case, for example, when $\mathtt{U} \precsim_{\emptyset} \mathtt{V}$ must be proved, with $\mathcal{E}(\mathtt{U}) = \mathtt{()} + \mathtt{U}$ and $\mathcal{E}(\mathtt{V}) = \mathtt{()} + \mathtt{V}$.

The main result of this contribution is the equivalence between $\mathtt{<:}$ and $\precsim_{\mathtt{A}}$. The proof is technical and detailed in the Appendix.

Theorem 1. *Let S and T be labelled-determined and, for every $(\mathtt{U}, R) \in \mathtt{A}$, let $\mathtt{U}$ and R be labelled-determined and $\mathtt{U} \mathtt{<:} R$. Then $S \precsim_{\mathtt{A}} T$ if and only if $S \mathtt{<:} T$.*

5.1 The code of the syntax-directed subschema and its computational complexity

Next we design an algorithm for the syntax-directed subschema relationship and discuss its computational complexity. The algorithm `Alg` is detailed in Table 2. It is a boolean function using two sets of assumptions `At` and `Af` that are implemented as bi-dimensional associative arrays. `At`, similarly to `A`, stores schemas whose subschema relation is either verified or is being verified. However, unlike `A`, `At` also stores generic pairs of schemas, not just pairs $(\mathtt{U}, T)$. `Af` stores schemas whose subschema relation have been already verified to be false. The arrays `At` and `Af` improve the efficiency of `Alg` by preventing that the same subschema relation is verified twice.

`Alg` is initially invoked with every entry of the arrays `At` and `Af` set to `false` – `Alg` is computing $S \precsim_{\emptyset} T$–, with an environment `E` and with the two schemas S and T. `Alg` primarily verifies whether the subschema relation has been already computed – the checks on `At`$[S]$`[`T`]` and on `Af`$[S]$`[`T`]` at lines `(2)` and `(3)` –, and in case returns immediately. These checks implement rule (NAMEL). Otherwise, `Alg` sets `At`$[S]$`[`T`]` to `true`, meaning that the pair (S,T) is being verified, and begins the syntax-directed case analysis of the schemas (line `(5)`). The alternatives of the case analysis from line `(6)` to line `(15)` respectively implement the rules (VOID), (BOT), (CHAN-I), (CHAN-O), (CHAN-IO), (RSEQ) and (LSEQ) and (UNIONR), (NAMEH), (UNIONR), and (NAMER).

Line `(11)` deserves to be spelled out. When S is a labelled schema $L[S']\mathtt{,}S''$, the verification is delegated to the auxiliary boolean function `aux_Alg`. This function assumes that the label `L` is nonempty and is always contained into `first`(T), where T is the last argument of `aux_Alg`. Then `aux_Alg` verifies if T may be decomposed into $L'[T']\mathtt{,}T'' + R$ such that $L \cap L' \neq \emptyset$, $S' \mathtt{<:} T'$, and $S'' \mathtt{<:} T''$. In case, `aux_Alg` is recursively invoked with $L \setminus L'$ (see intruction `(3)` of `aux_Alg`, lines 3 and 4), otherwise `aux_Alg` returns `false`. The assignment `At`$[S]$`[`T`] := true` in line `(4)` guarantees the termination of the algorithm in case of nested recursive invocations of `Alg` (with same S and T). This is sound

Table 2. The syntax-directed subschema algorithm

```
    bool Alg(At, Af, E, S, T) {
(1)   bool res:= false ;
(2)   if (At[S][T]) res:= true ;
(3)   else if (Af[S][T]) res:= false ;
(4)   else At[S][T]:= true ;
(5)     case S , T of
(6)       () , (): res:= true ;
(7)       ⊥ , __: res:= true ;
(8)       ⟨S′⟩^κ , ⟨T′⟩^i: res:= (κ==i or κ==io) and Alg(At,Af,E,S′,T′) ;
(9)       ⟨S′⟩^κ , ⟨T′⟩^o: res:= (κ==o or κ==io) and Alg(At,Af,E,T′,S′) ;
(10)      ⟨S′⟩^io , ⟨T′⟩^io: res:= Alg(At,Af,E,S′,T′) and Alg(At,Af,E,T′,S′);
(11)      L[S′],S″ , __: if (L ⊆ first(T)) then
                              res:= aux_Alg(At,Af,E,L,S′,S″,T) ;
                         else res:= false ;
(12)      S′ + S″ , __: res:= Alg(At,Af,E,S′,T) and Alg(At,Af,E,S″,T) ;
(13)      U , __: res:= Alg(At,Af,E,E(U),T) ;
(14)      __ , T′ + T″: res:= Alg(At,Af,E,S,T′) or Alg(At,Af,E,S,T″) ;
(15)      __ , U : res:= Alg(At,Af,E,S,E(U)) ;
(16)    if (res == false) At[S][T]:= false ; Af[S][T]:= true ;
(17)  return res;
    }

    bool aux_Alg(At, Af, E, L, S′, S″, T) {
(1)   case T is
(2)     L'[T′],T″: return(Alg(At,Af,E,S′,T′) and Alg(At,Af,E,S″,T″)) ;
(3)     T′ + T″: if (L⊆first(T′)) return(aux_Alg(At,Af,E,L,S′,S″,T′)) ;
                 else if (L∩first(T′) == ∅)
                      return(aux_Alg(At,Af,E,L,S′,S″,T″));
                 else return(aux_Alg(At,Af,E,L∩first(T′),S′,S″,T′)
                                and aux_Alg(At,Af,E,L\first(T′),S′,S″,T″));
(4)     U: return(aux_Alg(At,Af,E,L,S′,S″,E(U)) ;
    }
```

because, by the guardedness property of $\mathcal{E}$, the recursive invocations in lines (13) or (15) must reduce to execute an instruction from line (6) to (11).

We also remark that Alg has no instruction for rules (LBOT) and (SBOT). Indeed, these rules entangle the algorithm (in (11) we should verify that schemas are not empty) and are useless if we assume that every empty schema is rewritten to $\bot$. Therefore, for the sake of correctness of Alg and its computational complexity we assume that empty schemas are always $\bot$. Later on, we discuss how a schema can be rewritten in order to conform with this constraint.

Proposition 2. Alg *terminates in polynomial time.*

Proof. Let $\mathfrak{t}(S)$ be the set of subterms of a schema S (see the Appendix for a formal definition) and let $|\cdot|$ be the cardinality function. The dimensions of the

arrays `At` and `Af` is $|\mathfrak{t}(S) \cup \mathfrak{t}(T)| \times |\mathfrak{t}(S) \cup \mathfrak{t}(T)| = |\mathfrak{t}(S) \cup \mathfrak{t}(T)|^2$. The reason is due to the contravariance of $\langle\cdot\rangle^o$ that may reduce $S <: T$ to $T' <: S'$ where $T' \in \mathfrak{t}(T)$. Let $\lceil\texttt{true}\rceil = 1$ and $\lceil\texttt{false}\rceil = 0$.

Let $\texttt{At}_i$ and $\texttt{Af}_i$ denote the arrays `At` and `Af` when one of them has been modified exactly i times. The following invariants are preserved at the end of every line of `Alg` and `aux_Alg`:

1. for every S,T: `(At[`S`][`T`] == false) or (Af[`S`][`T`] == false)`, that is `true` is never stored both in `Af[`S`][`T`]` and in `At[`S`][`T`]`;
2. for every S,T: if ($\texttt{Af}_i\texttt{[}S\texttt{][}T\texttt{] == true}$) then ($\texttt{Af}_{i+1}\texttt{[}S\texttt{][}T\texttt{] == true}$), that is `true` is never deleted from `Af`;
3. $\sum_{S,T}\lceil\texttt{At}_i\texttt{[}S\texttt{][}T\texttt{]}\rceil + \lceil\texttt{Af}_i\texttt{[}S\texttt{][}T\texttt{]}\rceil \leq \sum_{S,T}\lceil\texttt{At}_{i+1}\texttt{[}S\texttt{][}T\texttt{]}\rceil + \lceil\texttt{Af}_{i+1}\texttt{[}S\texttt{][}T\texttt{]}\rceil$ (i.e. the total number of `true`s either grows or remains the same)
4. if $\sum_{S,T}\lceil\texttt{At}_i\texttt{[}S\texttt{][}T\texttt{]}\rceil + \lceil\texttt{Af}_i\texttt{[}S\texttt{][}T\texttt{]}\rceil = \sum_{S,T}\lceil\texttt{At}_{i+1}\texttt{[}S\texttt{][}T\texttt{]}\rceil + \lceil\texttt{Af}_{i+1}\texttt{[}S\texttt{][}T\texttt{]}\rceil$ then $\sum_{S,T}\lceil\texttt{Af}_i\texttt{[}S\texttt{][}T\texttt{]}\rceil < \lceil\texttt{Af}_{i+1}\texttt{[}S\texttt{][}T\texttt{]}\rceil$ (i.e. when the total number of `true`s remains the same then the `true`s in `Af` strictly increase).

We observe that, in the worst case, the algorithm terminates when $\sum_{S,T}\lceil\texttt{At[}S\texttt{][}T\texttt{]}\rceil + \lceil\texttt{Af[}S\texttt{][}T\texttt{]}\rceil$ is equal to $|\mathfrak{t}(S) \cup \mathfrak{t}(T)|^2$. Invariants 3 and 4 guarantee terminations (the number of `true`s either grows or remains the same for at most $|\mathfrak{t}(S) \cup \mathfrak{t}(T)|^2$ times before terminating). Invariants 1 and 2 state that `true` is never set in the same entry twice and it is never assigned to the same entry of the two arrays. Therefore, there may be at most $|\mathfrak{t}(S) \cup \mathfrak{t}(T)|^2$ stores of `true` into `At` and each `true` may be "moved" at most once into `Af`. The cost of this movement is proportional to $\texttt{max}(|\mathfrak{t}(S)|, |\mathfrak{t}(T)|)$ because the body of `Alg` may parse the structure of one of the schema (function `aux_Alg`). This means that the total cost of `Alg` is $O(\texttt{max}(|\mathfrak{t}(S)|, |\mathfrak{t}(T)|) \times |\mathfrak{t}(S) \cup \mathfrak{t}(T)|^2)$. □

To rewrite empty schemas to $\bot$ we define an algorithm similar to `Alg`. The algorithm takes two associative boolean vectors of size $\mathfrak{t}(S)$, `Et` and `Ef`, that are initialized to `false` at the beginning. At each step `true` is either added to `Et` or moved into `Ef`. Base cases are: $\bot$, `()`, $\langle S\rangle^\kappa$, and S when either `Et[`S`]` or `Ef[`S`]`. In case of $\bot$ and in case of `Et[`S`]`, `true` is returned and `Et` is set to `true`; in the other cases `false` is returned, `Et` is set to `false`, and `Ef` to `true`. The recursive cases are for sequences, unions, and schema names. In every case the corresponding value of `Et` is set to `true` and subterms are checked. If the recursive calls determine that the schema is not empty (i.e. the schema definition is not empty for schema names; one of the components is not empty for unions; both the components are not empty for sequences) `Et` is set to `false` and `Ef` is set to `true`, otherwise the schema is considered empty. The cost of this algorithm is $O(|\mathfrak{t}(S)|)$. Once `Et` has been computed, the algorithm `Alg` may be modified to verify at every recursive call whether the arguments are empty or not, and in case replace them with $\bot$.

Acknowledgments. The authors thank Allen Brown for the interesting discussions about XML schema languages.

References

1. L. Acciai and M. Boreale. XPi: a typed process calculus for XML messaging. In *7th Formal Methods for Object-Based Distributed Systems (FMOODS'05)*, volume 3535 of *Lecture Notes in Computer Science*, pages 47 – 66. Springer-Verlag, 2005.
2. R. M. Amadio and L. Cardelli. Subtyping recursive types. *ACM Transactions on Programming Languages and Systems*, 15(4):575–631, September 1993.
3. A. Brown, C. Laneve, and L. Meredith. `PiDuce`: A process calculus with native `xml` datatypes. In *2nd International Workshop on Web Services and Formal Methods*, LNCS, 2005.
4. G. Castagna, R. D. Nicola, and D. Varacca. Semantic subtyping for the π-calculus. In *20th IEEE Symposium on Logic in Computer Science (LICS'05)*. IEEE Computer Society, 2005.
5. J. Clark and M. Murata. Relax ng specification. Available on: `http://www.oasis-open.org/committees/relax-ng/spec-20011203.html`, 2001. December, 3rd 2001.
6. H. Hosoya and B. C. Pierce. XDuce: A statically typed XML processing language. *ACM Transactions on Internet Technology (TOIT)*, 3(2):117–148, 2003.
7. H. Hosoya, J. Vouillon, and B. C. Pierce. Regular expression types for XML. *ACM SIGPLAN Notices*, 35(9):11–22, 2000.
8. H. Comon et al. Tree automata techniques and applications. At `www.grappa.univ-lille3.fr/tata`, October, 2002.
9. W3C XML Schema Working Group. XML Schema Part 2: Datatypes Second Edition. Available on: `http://www.w3.org/TR/2004/REC-xmlschema-2-20041028/datatypes.html`. W3C Recommendation - October, 28th 2004.
10. Web Services Description Working Group. Web Services Description Language (WSDL) Version 2.0 Part 0: Primer. Available on: `http://www.w3.org/TR/2005/WD-wsdl20-primer-20050510/`. W3C Working Draft 10 May 2005.
11. Web Services Description Working Group. Web Services Description Language (WSDL)1.1. Available on: `http://www.w3.org/TR/2001/NOTE-wsdl-20010315`. W3C Note 15 March 2001.
12. XML Protocol Working Group. Extensible markup language (xml) 1.0 (third edition). Available on: `http://www.w3.org/TR/2004/REC-xml-20040204`, 2004. February, 4th 2004.
13. M. Murata, D. Lee, and M. Mani. Taxonomy of XML schema languages using formal language theory. In *Extreme Markup Languages*, Montreal, Canada, 2001.
14. B. C. Pierce and D. Sangiorgi. Typing and subtyping for mobile processes. In *Logic in Computer Science*, 1993. Full version in *Mathematical Structures in Computer Science* , Vol. 6, No. 5, 1996.
15. S. Vinosky. Web service notifications. *IEEE Internet Computing magazine*, 2004.

A The equivalence of the two subschema relationships

Let $\|S\|_\emptyset$, called the *size* of S, be the function inductively defined as: $\|\bot\|_{\mathtt{X}} = 0$, $\|\mathtt{U}\|_{\mathtt{X}} = 0$ if $\mathtt{U} \in X$, $\|\mathtt{U}\|_{\mathtt{X}} = \|\mathcal{E}(\mathtt{U})\|_{\mathtt{X}\cup\{\mathtt{U}\}}$ if $\mathtt{U} \notin X$, $\|\langle S\rangle^\kappa\| = 1 + \|S\|_{\mathtt{X}}$, $\|S + S'\|_{\mathtt{X}} = 1 + \|S\|_{\mathtt{X}} + \|S'\|_{\mathtt{X}}$, and $\|L[S]\,,S'\|_{\mathtt{X}} = 1 + \|S\|_{\mathtt{X}} + \|S'\|_{\mathtt{X}}$. It is easy to verify that $\|S\|_\emptyset = 0$ implies that S is empty. In the following $\|S\|_\emptyset$ will be shortened into $\|S\|$. Let also $\mathfrak{t}(S)$, called the *set of subterms* of S, be the smallest set satisfying

the following equations: $\mathfrak{t}(\mathtt{()}) = \{\mathtt{()}\}$, $\mathfrak{t}(\bot) = \{\bot\}$, $\mathfrak{t}(\mathtt{U}) = \{\mathtt{U}\} \cup \{\mathfrak{t}(\mathcal{E}(\mathtt{U}))\}$, $\mathfrak{t}(\langle S\rangle^{\kappa}) = \{\langle S\rangle^{\kappa}\} \cup \mathfrak{t}(S)$, $\mathfrak{t}(L[S]\mathtt{,}T) = \{L[S]\mathtt{,}T\} \cup \mathfrak{t}(S) \cup \mathfrak{t}(T)$, and $\mathfrak{t}(S+T) = \{S+T\} \cup \mathfrak{t}(S) \cup \mathfrak{t}(T)$ We note that $\|S\|$ and $\mathfrak{t}(S)$ are different. For instance, $\|S+S\| = 2 * \|S\| + 1$ whilst $\mathfrak{t}(S+S) = \mathfrak{t}(S) \cup \{S+S\}$. We also note that $\mathtt{names}(S) = \{\mathtt{U} \mid \mathtt{U} \in \mathfrak{t}(S)\}$. Finally, let $\mathtt{lsubt}(S,T)$ be the smallest set containing $\mathfrak{t}(S)$, $\mathfrak{t}(T)$, and closed under the following property: if $L[Q]\mathtt{,}Q' \in \mathtt{lsubt}(S,T)$ and $L'[Q'']\mathtt{,}Q''' \in \mathtt{lsubt}(S,T)$ and $\emptyset \subsetneq L \setminus L' \subsetneq L$ then $(L \setminus L')[Q]\mathtt{,}Q' \in \mathtt{lsubt}(S,T)$ and $(L \cap L')[Q]\mathtt{,}Q' \in \mathtt{lsubt}(S,T)$. We observe that $\|S\|$, $\mathfrak{t}(S)$, $\mathtt{names}(S)$, and $\mathtt{lsubt}(S,T)$ are always finite.

The following properties are immediate consequences of the definition of $\precsim$.

Proposition 3. *(1)Let* $\mathtt{A} \subseteq \mathtt{A}'$. *If* $S \precsim_{\mathtt{A}} T$ *then* $S \precsim_{\mathtt{A}'} T$*; (2) If* $S \precsim_{\mathtt{A}} T \Rightarrow \mathtt{A}'$ *then* $\mathtt{A} \subseteq \mathtt{A}'$.

Lemma 1. *If* S *is empty then, for every* $\mathtt{A}$ *and* T, $S \precsim_{\mathtt{A}} T \Rightarrow \mathtt{A}'$

Proof. We construct a proof of $S \precsim_{\mathtt{A}} T \Rightarrow \mathtt{A}'$. The proof is defined by induction on $\|S\| + |(\mathtt{names}(S) \times \|T\|) \setminus \mathtt{A}|$.

The base cases are: $S = \bot$ and $S = \mathtt{U}$ with $(\mathtt{U}, T) \in \mathtt{A}$. The first follows from (BOT), the second from (NAMEL). The inductive cases are $\|S\| + |(\mathtt{names}(S) \times \|T\|) \setminus \mathtt{A}| = n+1$ and either (1) $S = S' + S''$ where both S' and S'' are empty, or (2) $S = L[S']\mathtt{,}S''$ where S' is empty, or (3) $S = L[S']\mathtt{,}S''$ where S'' is empty, or (4) $S = \mathtt{U}$ where $\mathcal{E}(\mathtt{U})$ is empty. Case (1) follows from the inductive hypothesis and from the rule (UNIONL). Cases (2) and (3) follow from inductive hypothesis and from (LBOT) and (SBOT) respectively. As regards case (4), note that, by definition of handle, $\mathcal{E}(\mathtt{U})$ is empty. Then we use either (NAMEL) and we conclude, or (NAMEH) and we are reduced to prove $\mathcal{E}(\mathtt{U}) \precsim_{\mathtt{A}'} T$, with $\mathtt{A}' = \mathtt{A} \cup \{(\mathtt{U}, T)\}$. This relationship follows by inductive hypothesis because $\|S\| + |(\mathtt{names}(S) \times \|T\|) \setminus \mathtt{A}'| = n$. □

Theorem 1. *Let* S *and* T *be labelled-determined and, for every* $(\mathtt{U}, R) \in \mathtt{A}$*, let* $\mathtt{U}$ *and* R *be labelled-determined and* $\mathtt{U} <: R$. *Then* $S \precsim_{\mathtt{A}} T \Rightarrow \mathtt{A}'$ *if and only if* $S <: T$.

Proof. **(⇒)** To prove that $S \precsim_{\mathtt{A}} T \Rightarrow \mathtt{A}'$ implies $S <: T$, we argue by induction on the proof of $S \precsim_{\mathtt{A}} T \Rightarrow \mathtt{A}'$. We focus on the interesting cases.

(lbot) According to (LBOT), the conclusion $L[S']\mathtt{,}S'' \precsim_{\mathtt{A}} T \Rightarrow \mathtt{A}'$ has premise $S' \precsim_{\mathtt{A}} \bot \Rightarrow \mathtt{A}'$. By inductive hypothesis applied to such, $S' <: \bot$. Thus $L[S']\mathtt{,}S''$ is an empty schema and $L[S']\mathtt{,}S'' <: T$ follows by Proposition 1 (items 1, 5 and 6).

(lseq) The conclusion $L[S']\mathtt{,}S'' \precsim_{\mathtt{A}} T' + T''$ of the proof is obtained by the hyphothesis

$$(L \cap L')[S]\mathtt{,}S' \precsim_{\mathtt{A}} T \Rightarrow \mathtt{A}' \tag{1}$$

$$(L \setminus L')[S]\mathtt{,}S' \precsim_{\mathtt{A}'} T' \Rightarrow \mathtt{A}'' \tag{2}$$

By inductive hyphothesis applied to (1) and (2) we obtain $(L \cap L')[S]\mathtt{,}S' <: T'$ and $(L \setminus L')[S]\mathtt{,}S' <: T''$. By definition of $<:$, the first subschema implies

$T' \downarrow L_i(R_i\ ;\ R'_i)$ for $i \in 1..h$ and $(\widehat{L} \cap \widehat{L'}) \subseteq \bigcup_{i\in 1..h} \widehat{L_i}$, $S' <: R_i$ and $S'' <: R'_i$. The subschema $(L \setminus L')[S]\texttt{,}S' <: T''$ implies $T'' \downarrow L_j(Q_j\ ;\ Q'_j)$ for $j \in 1..h'$ and $\widehat{L \setminus L'} \subseteq \bigcup_{j\in 1..h'} \widehat{L'_i}$, $S' <: Q_i$ and $S'' <: Q'_i$. (The definition of $<:$ is reformulated in this way for labelled-determined schemas: see Section 5.) Therefore, we can conclude $L[S']\texttt{,}S'' <: T$.

(namer) According to (NAMER), the conclusion $S \precsim_{\mathtt{A}} \mathtt{U}$ has premise $S \precsim_{\mathtt{A}} \mathcal{E}(\mathtt{U}) \Rightarrow \mathtt{A}'$. By the inductive hyphothesis, $S <: \mathcal{E}(\mathtt{U})$. Being $\mathcal{E}(\mathtt{U}) <: \mathtt{U}$, we conclude $S <: \mathtt{U}$ by transitivity.

($\Leftarrow$) Let $S <: T$ and, for every $(\mathtt{U}, R) \in \mathtt{A}$: $\mathtt{U} <: R$. To verify that $S \precsim_{\mathtt{A}} T \Rightarrow \mathtt{A}'$ we construct a proof tree. The argument is by induction on the structure of the triple $(n, \|S\|, \|T\|)$, where n is $|(\mathtt{names}(S + T) \times \mathtt{lsubt}(S + T)) \setminus \mathtt{A}|$. The base cases are: (1) $S = \bot$, then, we conclude by (BOT); (2) $T = \bot$, then, by Lemma 1, S is empty and $S <: T$ is immediate; (3) $S = \mathtt{U}$ and $n = 0$, then $(\mathtt{U}, T) \in \mathtt{A}$ and we conclude by (NAMEL); (4) $\|S\| = 0$ then S is empty and lemma 1 applies. The inductive cases are discussed with a case analysis on the structure of S.

- If $S = \mathtt{()}$ then $T \downarrow \mathtt{()}$. The proof of $\mathtt{()} \precsim_{\mathtt{A}} T \Rightarrow \mathtt{A}'$ is constructed by induction on the derivation of $T \downarrow \mathtt{()}$. Every application of "$T_1 + T_2 \downarrow \mathtt{()}$ if $T_1 \downarrow \mathtt{()}$ or $T_2 \downarrow \mathtt{()}$" corresponds to an instance of (UNIONR); every application of "$\mathtt{U} \downarrow \mathtt{()}$ if $\mathcal{E}(\mathtt{U}) \downarrow \mathtt{()}$" corresponds to an instance of (NAMER).
- If $S = \langle S' \rangle^{\kappa}$ then $T \downarrow \langle\rangle^{\kappa'}(T')$. The proof $\langle S' \rangle^{\kappa} \precsim_{\mathtt{A}} T$ distinguishes several sub-cases depending on the capabilities. When $\kappa = i$, $S' <: T'$ and, by inductive hypothesis, we obtain $S' \precsim_{\mathtt{A}} T' \Rightarrow \mathtt{A}'$. The proof of $S' \precsim_{\mathtt{A}} T' \Rightarrow \mathtt{A}'$ is extended to one of $\langle S' \rangle^{\kappa} \precsim_{\mathtt{A}} T \Rightarrow \mathtt{A}'$ by arguing on the derivation of $T \downarrow \langle\rangle^{i}(T')$. The details are similar to the case when the handle is $\mathtt{()}$. Same arguments apply when $\kappa = o$ and $\kappa = io$.
- If $S = L[S']\texttt{,}S''$, we assume that both S' and S'' are not empty, otherwise we conclude by Lemma 1. There are two subcases: (1) $T \downarrow L'(T'\ ;\ T'')$ with $\widehat{L} \subseteq \widehat{L'}$, $S' <: T'$, and $S'' <: T''$; (2) $T \downarrow L_i(T'_i\ ;\ T''_i)$, $\widehat{L} \cap \widehat{L'_i} \neq \emptyset$. $L \subseteq \bigcup_{i\in I} \widehat{L_i}$ with $|I| > 1$, $S' <: T'_i$, and $S'' <: T''_i$. In case (1), $S' \precsim_{\mathtt{A}} T' \Rightarrow \mathtt{A}'$ and of $S'' \precsim_{\mathtt{A}'} T'' \Rightarrow \mathtt{A}''$ follow by inductive hypothesis. Then we use the derivation of $T \downarrow L'(T'\ ;\ T'')$ to complete the proof as in the case of $\mathtt{()}$.
  In case (2), $T = T' + T''$ with $T' \downarrow L_i(T'_i\ ;\ T''_i)$, $i \in I'$ and $T'' \downarrow L_i(T'_i\ ;\ T''_i)$, $i \in I''$ where $I = I' \uplus I''$ and the labels are pairwise disjoint (because T is labelled-determined). We therefore have $S' <: T'_i$ and $S'' <: T''_i$ for every i. Let $\widehat{L'} = \bigcup_{i\in I'} \widehat{L_i}$, we may derive $(L \cap L')[S']\texttt{,}S'' <: T'$, $(L \setminus L')[S']\texttt{,}S'' <: T''$, $\emptyset \subsetneq \widehat{L} \cap \widehat{L'} \subsetneq \widehat{L'}$. We conclude by inductive hypothesis and (LSEQ).
- If $S = S' + S''$ then $S' <: T$ and $S'' <: T$. By inductive hypothesis it is possible to prove $S' \precsim_{\mathtt{A}} T \Rightarrow \mathtt{A}'$ and of $S'' \precsim_{\mathtt{A}'} T \Rightarrow \mathtt{A}''$. We conclude by (UNIONL).
- If $S = \mathtt{U}$, we may use the rules (NAMEH) or (NAMEL). (NAMEH) allows us to close the branch of the proof tree, (NAMEL) allows us to reduce to one of the previous cases. (NAMEL) unfolds the schema $\mathtt{U}$. Since there are finitely many constants in $\mathcal{E}(\mathtt{U})$ (because $\mathcal{E}$ is finite) (NAMEL) may be used finitely many times in a single branch of the proof tree of $S \precsim_{\mathtt{A}} T \Rightarrow \mathtt{A}'$ before saturating the set $\mathtt{A}$. □