

# A Coinductive axiomatization of XML subtyping

Lasse Nielsen

14th January 2008

## Contents

<b>1</b>	<b>Introduction</b>	<b>5</b>
1.1	Resumé (Dansk) . . . . .	5
1.2	Summary (English) . . . . .	6
1.3	Prerequisites . . . . .	7
1.4	Notation . . . . .	7
1.4.1	The error element: $\top$ . . . . .	7
<b>2</b>	<b>Xml-sequences and Xml-types</b>	<b>9</b>
<b>3</b>	<b>Substitution and <math>\alpha</math>-equivalence</b>	<b>15</b>
<b>4</b>	<b>Semantics of Xml-types</b>	<b>18</b>
<b>5</b>	<b>Xml-data</b>	<b>22</b>
<b>6</b>	<b>Tagging</b>	<b>25</b>
<b>7</b>	<b>The Empty-sequence Property</b>	<b>30</b>
<b>8</b>	<b>Type Splitting</b>	<b>34</b>
<b>9</b>	<b>Restricted Tagging and Data Simplification</b>	<b>37</b>
<b>10</b>	<b>Data Splitting and Unsplitting</b>	<b>42</b>
<b>11</b>	<b>Derivatives</b>	<b>48</b>
<b>12</b>	<b>Simulations</b>	<b>56</b>
<b>13</b>	<b>Axiomatization</b>	<b>59</b>

---

<b>14 Coercions</b>	<b>62</b>
<b>15 Runtime Analysis</b>	<b>66</b>
<b>16 Implementation</b>	<b>70</b>
<b>17 Applications</b>	<b>71</b>
<b>18 Summing Up</b>	<b>73</b>
18.1 Related Work . . . . .	73
18.2 Future Work . . . . .	75
<b>19 Appendix</b>	<b>76</b>
19.1 Technical Proofs . . . . .	76
19.2 Program Code . . . . .	141
19.3 Output . . . . .	148
19.4 Xml Data . . . . .	152

## References

- [HVP] Regular Expression Types for XML  
By Haruo Hosoya, Jerome Voullion and Benjamin C. Pierce.
- [BH98] Coinductive Axiomatization of Recursive Type Equality and Subtyping  
By Michael Brandt and Fritz Henglein
- [LS04] An Implementation of Subtyping Among Regular Expression Types  
By Kenny Zhuo Ming Lu and Martin Sulzmann
- [LS05] A Type-Safe Embedding of XDuce into ML  
By Kenny Zhuo Ming Lu and Martin Sulzmann
- [BCF03] CDuce: an XML-centric general-purpose language  
By Véronique Benzaken and Giuseppe Castagna and Alain Frisch
- [F06] OCaml + XDuce  
By Alain Frisch
- [CLRS] Introduction to Algorithms  
Second Edition  
By Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest and Clifford Stein.
- [CA-REG] Coinductive Axiomatizations of Equality of Regular Expressions  
By Michael Nissen and Lasse Nielsen
- [TAPL] Types and Programming Languages  
By Benjamin C. Pierce.
- [HU79] Introduction to Automata Theory, Languages, and Computation  
By J. E. Hopcroft and J. D. Ullman.  
Addison-Wesley, 1979.
- [FSPL] The Formal Semantics of Programming Languages - An Introduction.  
By Glynn Winskel.
- [P75] Call-by-name, call-by-value and the lambda calculus.  
Theoretical Computer Science 1 (1975), page 125-159.  
By G. D. Plotkin.
- [CF58] Combinatory Logic, Vol. 1, North Holland, Amsterdam (1958).  
By H. B. Curry and R. Feys.
- [MAT2AL] Matematik 2AL, Algebra, 2.udgave (Danish).  
By Anders Thorup.
- [W3SCHOOLS.COM] Link: [http://www.w3schools.com/xml/cd\\_catalog.xml](http://www.w3schools.com/xml/cd_catalog.xml).
- [XCOMPACT] Link: <http://www.plan-x.org/projects/xcompact/>
- [BZIP2] Homepage: <http://www.bzip.org/>

[SML] The Definition of Standard ML  
By R. Milner and M. Tofte and R. Harper.

[MOSML] Moscow ML  
Homepage: <http://www.dina.kvl.dk/~sestoft/mosml.html>

# 1 Introduction

## 1.1 Resumé (Dansk)

Dette projekt omhandler koinduktive aksiomatiseringer af typeegenskaber, med fokus på subtypning af xml-typer.

Projektet kan opdeles i tre dele.

Første del af projektet definerer problemet formelt.

I denne del definerer vi xml-sekvenser, xml-typer og xml-data. Vi definerer semantikken af en xml-type som de xml-sekvenser den accepterer, og definerer hvordan vi givet en xml-type kan bruge xml-data til at beskrive hver enkelt af de xml-sekvenser den accepterer.

Til slut defineres subtypning af xml-typer ud fra deres semantik.

Anden del af projektet definerer en løsning på subtypnings-problemet, og vise at den sund og fuldstændig.

I denne del beskriver vi en metode til at lave en xml-type om til en form, der direkte beskriver hovede og hale af de xml-sekvenser den accepterer.

Vi bruger denne metode til at definere afledte af en xml-type, og udnytter endeligheden af de afledte til at definere en sund og fuldstændig aksiomatisering af subtypning for xml-typer.

Når dette er gjort definerer vi en oversættelse (coercions) fra afledninger af subtypning af xml-typer til funktioner, der oversætter xml-data for den første type til den anden type.

Til slut viser vi, at de oversættelser der bliver konstruerede er sunde og fuldstændige, og at deres køretid som funktion af størrelsen af xml-data er begrænset af et 4.grads polynomium.

Tredje del af projektet implementerer og afprøver løsningen.

I denne del implementerer vi systemet i SML, og laver en mindre afprøvning af implementationen.

## 1.2 Summary (English)

This project addresses the subject of coinductive axiomatizations of type properties, and it focuses on subtyping of xml-types.

The project contains three parts.

The first part formally defines the problem.

In this part xml-sequences, xml-types and xml-data are defined. The semantics of an xml-type is defined as the set of xml-sequences that inhabit it, and it is defined how we for any xml-type can use xml-data to describe any xml-sequence that inhabit it.

Finally subtyping of xml-types is defined using their semantics.

The second part defines a solution to the subtyping problem, and proves that it is sound and complete.

In this part we define a method to split the xml-type into a form that directly describes the heads and tails of the xml-sequences that inhabit it.

We use this method to define derivatives of xml-types, and use the finiteness of the derivatives to define a sound and complete axiomatization of subtyping of xml-types.

When this is done, we define a way to translate a derivation of subtyping into a function that convert xml-data for the specialized type to xml-data for the generalized type using coercions. Finally we show that the defined conversions are sound and complete, and that their runningtime as a function of the size of the xml-data is limited by a 4th degree polynomial.

The third part implements the solution in SML and tests the implementation.

### 1.3 Prerequisites

In order to fully understand the proof and the theoretic results in this project, the reader needs to be familiar with the following subjects.

- The type theory from [TAPL, Chapter I - IV].
- Mathematics corresponding to a mathematical bachelors degree.

### 1.4 Notation

Most of the notation in this project will be introduced the first time it is used, but it seems appropriate to introduce the following definitions before we start.

If  $t$  is an element of an abstract grammar, we will use the notation  $|t|$  to denote the size of the element (the number of constructors used to create the element).

When defining functions, we will use a mixture of SML and mathematical notation. There are two exceptions to this.

One exception is that we will use something similar to Haskell notation to define lists in the same way as we define sets. An example of this could be the expression

$$L_2 = [ 2 \cdot n \mid n \in L_1 ]$$

and this defines  $L_2$  as the list that contains the elements of  $L_1$  multiplied by 2, preserving the order of the elements.

The other exception is that we will use the well known `fix` expression when we define coercions.

#### 1.4.1 The error element: $\top$

Some functions need the possibility of stopping execution, and return an error, if it cannot do anything meaningful with the given arguments.

This can be solved by letting the function be undefined on the meaningless arguments.

In this project, this is done using an error element  $\top$  and `let – in – end` expressions.

An expression of the form

```
let  $x_1 = \text{exp}_1, x_2 = \text{exp}_2$  in  $\text{exp}$  end
```

evaluates by first evaluating  $\text{exp}_1$  and  $\text{exp}_2$ . If one of the expressions evaluates to  $\top$  then the entire expression evaluates to  $\top$ .

If none of the expressions evaluate to  $\top$  then  $\text{exp}$  is evaluated, with  $x_1$  replaced by the result of  $\text{exp}_1$  and  $x_2$  replaced by the result of  $\text{exp}_2$ . The result of the entire expression is the result of  $\text{exp}$ .

#### Example 1.1

*If we try to evaluate*

```
let  $x_1 = \text{true}, x_2 = \top$  in  $x_1 \wedge x_2$  end
```

we get that the expression for  $x_2$  is  $\top$  and therefore the entire expression evaluates to  $\top$ .

If we try to evaluate

`let  $x_1 = \text{true}$ ,  $x_2 = \top$  in  $x_1$  end`

we also get that the expression for  $x_2$  is  $\top$  and therefore the entire expression evaluates to  $\top$  even though  $x_2$  is never used.

If we try to evaluate

`let  $x_1 = \text{true}$ ,  $x_2 = \text{false}$  in  $\top$  end`

we get that neither the expression for  $x_1$  nor  $x_2$  evaluate to  $\top$  and therefore we need to look at the contained expression, but since it is  $\top$  the entire expression evaluates to  $\top$ .

Finally, if we try to evaluate

`let  $x_1 = \text{true}$ ,  $x_2 = \text{false}$  in  $x_1 \wedge x_2$  end`

we get that neither the expression for  $x_1$  nor  $x_2$  evaluate to  $\top$  and therefore we need to look at the contained expression,  $x_1 \wedge x_2$ . Therefore the entire expression evaluates to  `$\text{true} \wedge \text{false} = \text{false}$` .

## 2 Xml-sequences and Xml-types

In this section, we define xml-sequences and xml-types. We also provide some examples to give an intuitive understanding of xml-sequences and xml-types.

We start by defining xml-sequences.

### Definition 2.1

The set **XMLSeq** of xml-sequences contains exactly the elements that can be constructed using the following grammar

$$x ::= \varepsilon \mid \langle l \rangle x_1 \langle /l \rangle x_2$$

where  $l$  can be any element from a countable infinite set **XMLTag** of tagnames.

In the constructor  $\langle l \rangle x_1 \langle /l \rangle x_2$  it is important to note that the start tag, and the end tag must be use the same tagname.

The xml-sequence  $\varepsilon$  is the empty sequence.

The xml-sequence  $\langle l \rangle x_1 \langle /l \rangle x_2$  is the sequence where the first element (the head) is a tree named  $l$  that contains the xml-sequence  $x_1$ , and the rest of the sequence (the tail) is  $x_2$ .

We observe, that this definition can be extended with simple values such as integers and strings. One way to do that is by extending the syntax of xml-sequences to the following

$$x' ::= \varepsilon \mid \langle l \rangle x'_1 \langle /l \rangle x'_2 \mid \langle l \rangle s \langle /l \rangle x'_2 \mid \langle l \rangle i \langle /l \rangle x'_2$$

where  $s$  can be any element from the set of quoted strings, and  $i$  can be any element from  $\mathbb{N}$ .

In order to limit the number of cases in the proofs, we will only consider the original definition of xml-sequences.

An xml-sequence is a way to store data in a structured way. The following example shows an xml-sequence, together with a graphical illustration.

### Example 2.2 (A simple xml-sequence)

Consider the following xml-sequence

$$x = \langle \text{gender} \rangle \langle \text{male} \rangle \varepsilon \langle / \text{male} \rangle \varepsilon \langle / \text{gender} \rangle \langle \text{age} \rangle \langle s \rangle \varepsilon \langle / s \rangle \langle s \rangle \varepsilon \langle / s \rangle \langle s \rangle \varepsilon \langle / s \rangle \varepsilon \langle / \text{age} \rangle \varepsilon.$$

In Figure 1 we have visualized  $x$ .

We see, that there are two main trees. The first is named **gender**, and the second is named **age**.

The **gender** tree has one empty sub-tree named **male**, while the **age** tree has a sequence of empty sub-trees, all named **s**.

There are many ways to interpret this xml-sequence. One way is that it contains data about a person (A male of age 3).

For our purpose, the interpretation is not important.

The main thing is that xml-sequences represent data in a tree-structured way.

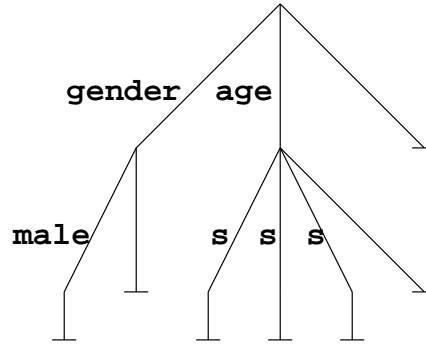


Figure 1: A graphical illustration of the xml-sequence in this example.

Next we define xml-types.

### Definition 2.3

The set **XMLType-unrestricted** of unrestricted xml-types contain exactly the elements that can be constructed using the following grammar

$$t, u ::= 0 \mid 1 \mid l\langle t_1 \rangle \mid t_1 t_2 \mid t_1 + t_2 \mid \mu X.t_1 \mid X$$

where  $l$  can be any element from **XMLTag**, and  $X$  can be any element from a countable infinite set **Vars** of variable names.

The binding priorities of the xml-type constructors are as follows.  $t_1 t_2$  binds tighter than  $t_1 + t_2$  which binds tighter than  $\mu X.t_1$ .

An xml-type of the form  $\mu X.t_1$  is called a  $\mu$ -type.

As usual types represent sets of elements (xml-sequences).

$0$  is the empty (uninhabited) xml-type.

$1$  is the unit type. The only element of type  $1$  is  $\varepsilon$ .

$l\langle t_1 \rangle$  represents the set of xml-sequences consisting of a single tree named  $l$ , containing a subtree of type  $t_1$ .

The sequence xml-type  $t_1 t_2$  is the set of xml-sequences, that can be split into two sequences, where the first is of type  $t_1$  and the second of type  $t_2$ .

The sum xml-type  $t_1 + t_2$  is the union of  $t_1$  and  $t_2$ .

Finally  $\mu X.t_1$  is a recursive xml-type, which we must *unfold* in order to find the xml-sequences it contains.

We define the formal semantics of an xml-type as the set of xml-sequences that inhabit it in Section 4.

### Example 2.4 (An xml-type)

$$t = \text{gender}\langle \text{male}\langle 1 \rangle + \text{female}\langle 1 \rangle \rangle \text{age}\langle \mu X.1 + \text{s}\langle 1 \rangle X \rangle.$$

We now give an intuitive explanation of what kind of xml-sequences that inhabits this xml-type.

We see that *xml*-sequences of type  $t$  must have exactly two trees, **gender** and **age**.

The **gender** tree must have exactly one of two sub-trees. It can either be an empty tree named **male** or an empty tree named **female**.

The **age** tree must contain an *xml*-sequence of type  $\mu X.1 + \mathbf{s}\langle 1 \rangle X$ .

Since this is a  $\mu$ -type we unfold it, so the *xml*-sequence must be of type  $1 + \mathbf{s}\langle 1 \rangle \mu X.1 + \mathbf{s}\langle 1 \rangle X$ .

This means that it must either be the empty *xml*-sequence, or an *xml*-sequence starting with an empty tree named **s** followed by an *xml*-sequence of type  $\mu X.1 + \mathbf{s}\langle 1 \rangle X$ .

We can now see, that the content of the **age** tree must be an *xml*-sequence of zero or more empty trees named **s**.

This explains intuitively what *xml*-sequences of this type must fulfill, and we can check that the *xml*-sequence from Example 2.2 fulfills this, and therefore that *xml*-sequence intuitively inhabits this *xml*-type.

We will use a variation of the well known notion of free variables in a term.

The variation is that we do not include variables that are inside a tree. It is defined as below.

**Definition 2.5 (Unguarded Free Variables)**

$$\begin{aligned}
 \mathbf{fv}'(0) &= \{\} \\
 \mathbf{fv}'(1) &= \{\} \\
 \mathbf{fv}'(X) &= \{X\} \\
 \mathbf{fv}'(l\langle t_1 \rangle) &= \{\} \quad // \text{This is the variation.} \\
 \mathbf{fv}'(t_1 t_2) &= \mathbf{fv}'(t_1) \cup \mathbf{fv}'(t_2) \\
 \mathbf{fv}'(t_1 + t_2) &= \mathbf{fv}'(t_1) \cup \mathbf{fv}'(t_2) \\
 \mathbf{fv}'(\mu X.t_1) &= \mathbf{fv}'(t_1) \setminus \{X\}
 \end{aligned}$$

We will use this variation of free variables to define tail-recursive *xml*-types.

If we use the traditional definition of free variables to do this, the definition of tail-recursive *xml*-types will be too restrictive.

We will still use the common notion of free variables (where  $\mathbf{fv}(l\langle t_1 \rangle) = \mathbf{fv}(t_1)$ ) in some places. We write  $\mathbf{fv}(t)$  for the free variables in  $t$ .

**Definition 2.6 (Tail Recursiveness)**

We define  $\mathbf{tr}_\chi$  where  $\chi$  is a finite set of variables by structural induction on  $t$ .

$$\begin{aligned}
 \mathbf{tr}_\chi(0) &= \mathbf{true} \\
 \mathbf{tr}_\chi(1) &= \mathbf{true} \\
 \mathbf{tr}_\chi(l\langle t_1 \rangle) &= \mathbf{tr}_\chi(t_1) \\
 \mathbf{tr}_\chi(t_1 t_2) &= \mathbf{fv}'(t_1) \cap \chi = \{\} \wedge \mathbf{tr}_\chi(t_1) \wedge \mathbf{tr}_\chi(t_2) \\
 \mathbf{tr}_\chi(t_1 + t_2) &= \mathbf{tr}_\chi(t_1) \wedge \mathbf{tr}_\chi(t_2) \\
 \mathbf{tr}_\chi(\mu X.t_1) &= \mathbf{tr}_{\chi \cup \{X\}}(t_1) \\
 \mathbf{tr}_\chi(X) &= \mathbf{true}
 \end{aligned}$$

We now define

$$\mathbf{tr}(t) = \mathbf{tr}_{\{\}}(t).$$

We require for all xml-types that  $\mathbf{tr}(t) = \mathbf{true}$ .

We now give some examples to illustrate this property.

**Example 2.7 (A tail-recursive xml-type)**

Consider the xml-type

$$t = \mu X.1 + l\langle X \rangle X.$$

We now evaluate  $\mathbf{tr}(t)$ .

$$\begin{aligned} \mathbf{tr}(\mu X.1 + l\langle X \rangle X) &= \mathbf{tr}_{\{\}}(\mu X.1 + l\langle X \rangle X) \\ &= \mathbf{tr}_{\{X\}}(1 + l\langle X \rangle X) \\ &= \mathbf{tr}_{\{X\}}(1) \wedge \mathbf{tr}_{\{X\}}(l\langle X \rangle X) \\ &= \mathbf{tr}_{\{X\}}(l\langle X \rangle X) \\ &= \mathbf{fv}'(l\langle X \rangle) \cap \{X\} = \{\} \wedge \mathbf{tr}_{\{X\}}(l\langle X \rangle) \wedge \mathbf{tr}_{\{X\}}(X) \\ &= \{\} \cap \{X\} = \{\} \wedge \mathbf{tr}_{\{X\}}(l\langle X \rangle) \wedge \mathbf{tr}_{\{X\}}(X) \\ &= \mathbf{tr}_{\{X\}}(l\langle X \rangle) \wedge \mathbf{tr}_{\{X\}}(X) \\ &= \mathbf{tr}_{\{X\}}(X) \wedge \mathbf{tr}_{\{X\}}(X) \\ &= \mathbf{true} \wedge \mathbf{tr}_{\{X\}}(X) \\ &= \mathbf{tr}_{\{X\}}(X) \\ &= \mathbf{true} \end{aligned}$$

Therefore  $t$  is a tail-recursive xml-type.

**Example 2.8 (A non tail-recursive type)**

If we consider the type  $t = \mu X.1 + (l_1\langle 1 \rangle(Xl_2\langle 1 \rangle))$ .

We can start by evaluating  $\mathbf{tr}(t)$ .

$$\begin{aligned} \mathbf{tr}(\mu X.1 + (l_1\langle 1 \rangle(Xl_2\langle 1 \rangle))) &= \mathbf{tr}_{\{\}}(\mu X.1 + (l_1\langle 1 \rangle(Xl_2\langle 1 \rangle))) \\ &= \mathbf{tr}_{\{X\}}(1 + (l_1\langle 1 \rangle(Xl_2\langle 1 \rangle))) \\ &= \mathbf{tr}_{\{X\}}(1) \wedge \mathbf{tr}_{\{X\}}(l_1\langle 1 \rangle(Xl_2\langle 1 \rangle)) \\ &= \mathbf{tr}_{\{X\}}(l_1\langle 1 \rangle(Xl_2\langle 1 \rangle)) \\ &= \mathbf{fv}'(l_1\langle 1 \rangle) \cap \{X\} = \{\} \wedge \mathbf{tr}_{\{X\}}(l\langle 1 \rangle) \wedge \mathbf{tr}_{\{X\}}(Xl_2\langle 1 \rangle) \\ &= \mathbf{tr}_{\{X\}}(l\langle 1 \rangle) \wedge \mathbf{tr}_{\{X\}}(Xl_2\langle 1 \rangle) \\ &= \mathbf{tr}_{\{X\}}(1) \wedge \mathbf{tr}_{\{X\}}(Xl_2\langle 1 \rangle) \\ &= \mathbf{tr}_{\{X\}}(Xl_2\langle 1 \rangle) \\ &= \mathbf{fv}'(X) \cap \{X\} = \{\} \wedge \mathbf{tr}_{\{X\}}(X) \wedge \mathbf{tr}_{\{X\}}(l_2\langle 1 \rangle) \\ &= \mathbf{false} \wedge \mathbf{tr}_{\{X\}}(X) \wedge \mathbf{tr}_{\{X\}}(l_2\langle 1 \rangle) \\ &= \mathbf{false} \end{aligned}$$

So according to the definition,  $t$  is not a tail-recursive xml-type.

The reason why we do not allow these xml-types is not that they don't make any sense. Non tail-recursive xml-types are useful, and can describe some sets of xml-sequences that cannot be described using tail-recursive xml-types.  $\mu X.1 + (l_1\langle 1 \rangle(Xl_2\langle 1 \rangle))$  is a good example of this, because it describes all the xml-sequences that start with a number of  $\langle l_1 \rangle \varepsilon \langle /l_1 \rangle$  trees, and ends with the same number of  $\langle l_2 \rangle \varepsilon \langle /l_2 \rangle$  trees. This relation between the number of trees cannot be described using tail-recursive xml-types.

If we want to use a tail-recursive xml-type that allows these xml-sequences, we have to use something like  $(\mu X.1 + l_1\langle 1 \rangle X)(\mu X.1 + l_2\langle 1 \rangle X)$ , which allows more xml-sequences than the non tail-recursive version.

We want to decide subtyping of xml-types.

The reason why we only allow tail-recursive xml-types is that the original problem is undecidable.

The reason for this is that [HVP] states that subtyping of xml-types is equivalent to language inclusion of context free languages, and [HVP] cites [HU79] to have proved that this is undecidable.

### Definition 2.9 (Contractiveness)

We will require all xml-types to be contractive.

This property is defined using the following function.

$$\begin{aligned}
\text{contractive}(0) &= \text{true} \\
\text{contractive}(1) &= \text{true} \\
\text{contractive}(l\langle t_1 \rangle) &= \text{contractive}(t_1) \\
\text{contractive}(t_1 t_2) &= \text{contractive}(t_1) \wedge \text{contractive}(t_2) \\
\text{contractive}(t_1 + t_2) &= \text{contractive}(t_1) \wedge \text{contractive}(t_2) \\
\text{contractive}(\mu X.l\langle t_1 \rangle) &= \text{contractive}(t_1) \\
\text{contractive}(\mu X.t_1 t_2) &= \text{contractive}(t_1) \wedge \text{contractive}(t_2) \\
\text{contractive}(\mu X.t_1 + t_2) &= \text{contractive}(t_1) \wedge \text{contractive}(t_2) \\
\text{contractive}(X) &= \text{true} \\
\text{contractive}(\_) &= \text{false}
\end{aligned}$$

We can see from the definition of `contractive` that all we require is that the xml-type does not contain any sub-term of the form  $\mu X.0$ ,  $\mu X.1$ ,  $\mu X.Y$  or  $\mu X.\mu Y.t'$ .

Terms that contain these sub-terms are not necessary, because there always exist simpler, contractive xml-types that describe the same sets of xml-sequences.

### Definition 2.10

The set `XMLType` contain exactly the elements in `XMLType-unrestricted` that are tail-recursive and contractive.

This means that

$$t \in \text{XMLType} \Leftrightarrow t \in \text{XMLType-unrestricted} \quad \wedge \text{tr}(t) \wedge \text{contractive}(t).$$

We will use both  $t$  and  $u$  to represent xml-types, and we will declare it explicitly if we use unrestricted xml-types.

It is worth noting, that the definition of xml-types easily can be extended with simple types as integer and string.

$$t' ::= 0 \mid 1 \mid l\langle t'_1 \rangle \mid l\langle \text{string} \rangle \mid l\langle \text{int} \rangle \mid t'_1 t'_2 \mid t'_1 + t'_2 \mid \mu X.t'_1 \mid X$$

In order to limit the number of cases in the proofs, we will not consider these types.

Much like [CA-REG], we need to generalize our types to sets of types, in order to ensure that we can generate finite proof trees.

In this case, it is convenient to represent them as lists of types.

This leads to the following definition.

**Definition 2.11**

$$\mathbf{XMLTypes} = \mathbf{XMLType} \textit{ list}$$

We will use  $T$  and  $U$  to represent elements of  $\mathbf{XMLTypes}$ .

We could have chosen to have generalized the xml-types to sets of xml-types instead of lists of xml-types, and this would save us a lot of bookkeeping in some of the proofs.

But the extra informations in the lists become necessary because we want to work with xml-data that uses an xml-type to describe an xml-sequence.

Therefore we have to endure the extra bookkeeping, so that the data can refer to the exact xml-type.

We have now introduced xml-sequences and xml-types and given examples of how they are interpreted.

### 3 Substitution and $\alpha$ -equivalence

We have now introduced xml-types.

Substitution plays an important role in xml-types, since we have to *unfold*  $\mu$ -terms in order to understand their semantics.

We use  $t[t'/X]$  to denote the result of replacing free occurrences of  $X$  in  $t$  with  $t'$ .

We use capture avoiding substitution, which means that if a variable is free in  $t'$  then it must not be bound in the result. To ensure this, it is necessary to use renaming of bound variables.

In many cases it is sufficient to have an intuitive understanding of substitution, but in this project, we will need to use some properties of the substitution that are not easy to justify using just an intuitive definition.

Therefore we introduce the following definition of capture avoiding substitution.

**Definition 3.1 (Substitution)**

We define  $t[t'/X]$  by induction on  $|t|$ .

$$\begin{aligned}
0[t'/X] &= 0 \\
1[t'/X] &= 1 \\
Y[t'/X] &= \begin{cases} Y & \text{if } Y \neq X \\ t' & \text{otherwise} \end{cases} \\
l\langle t_1 \rangle[t'/X] &= l\langle t_1[t'/X] \rangle \\
t_1 t_2[t'/X] &= (t_1[t'/X])(t_2[t'/X]) \\
t_1 + t_2[t'/X] &= (t_1[t'/X]) + (t_2[t'/X]) \\
\mu Y.t_1[t'/X] &= \mu Z.(t_1[Z/Y][t'/X]) \quad \text{where } Z \notin \text{fv}(t') \cup \text{fv}(\mu Y.t_1) \cup \{X\}
\end{aligned}$$

We have defined  $t[t'/X]$  by induction on  $|t|$ , so we have to consider the final case, since substitution is used recursively on  $t_1[Z/Y]$  instead of just on  $t_1$ .

It is however clear that  $|t_1[Z/Y]| = |t_1|$  and therefore this is acceptable.

Notice that  $t_1 t_2[t'/X] = (t_1 t_2)[t'/X]$ ,  $t_1 + t_2[t'/X] = (t_1 + t_2)[t'/X]$  and  $\mu Y.t_1[t'/X] = (\mu Y.t_1)[t'/X]$ . We will write  $t_1(t_2[t'/X])$ ,  $t_1 + (t_2[t'/X])$  and  $\mu Y.(t_1[t'/X])$  otherwise.

We notice that we need to select an unused variable in the final case.

This can make the substitution non-deterministic, although only up to  $\alpha$ -equivalence, which we define below.

We can fix this by ordering the variables, and make the substitution use the first variable that fulfills the requirement.

It is however more convenient to relax the equality of xml-types to consider xml-types as equal even though some bound variables have been renamed.

Therefore we introduce the following relation on xml-types, and use this to define equality on xml-types.

**Definition 3.2** ( $\alpha$ -equivalence)

Two xml-types  $t, t'$  are  $\alpha$ -equivalent if and only if  $\vdash t \equiv t'$ , where  $\vdash \cdot \equiv \cdot$  is defined by.

$$\begin{array}{c}
 \text{equiv-0} \frac{}{\vdash 0 \equiv 0} \\
 \\
 \text{equiv-1} \frac{}{\vdash 1 \equiv 1} \\
 \\
 \text{equiv-tree} \frac{\vdash t_1 \equiv t'_1}{\vdash l\langle t_1 \rangle \equiv l\langle t'_1 \rangle} \\
 \\
 \text{equiv-seq} \frac{\vdash t_1 \equiv t'_1 \quad \vdash t_2 \equiv t'_2}{\vdash t_1 t_2 \equiv t'_1 t'_2} \\
 \\
 \text{equiv-sum} \frac{\vdash t_1 \equiv t'_1 \quad \vdash t_2 \equiv t'_2}{\vdash t_1 + t_2 \equiv t'_1 + t'_2} \\
 \\
 \text{equiv-mu} \frac{\vdash t_1[Z/X] \equiv t'_1[Z/Y]}{\vdash \mu X.t_1 \equiv \mu Y.t'_1} \quad Z \notin \text{fv}(t_1) \cup \text{fv}(t'_1) \\
 \\
 \text{equiv-var} \frac{}{\vdash X \equiv X}
 \end{array}$$

This definition of substitution and  $\alpha$  equivalence is very close to and clearly equivalent to the one used in [P75] for the  $\lambda$ -calculus. In [P75] they cite [CF58] to have proved that this is a *good* definition.

We have not been able to find a copy of [CF58], so it is not clear exactly what they mean by a *good* definition<sup>1</sup>, but this must include that  $\alpha$ -equivalence is an equivalence-relation, and that the substitution is well-defined.

Since  $\vdash \cdot \equiv \cdot$  is an equivalence-relation, [MAT2AL, Tal 6.5] yields that we can consider types equal up to  $\alpha$ -equivalence.

We will therefore write  $t = t'$  if and only if  $\vdash t \equiv t'$ .

Now we can see that the  $\mu$ -case in the definition of substitution does not change the  $\mu$ -term, it only chooses a representation of the  $\mu$ -term, where the substitution can proceed without capturing free variables.

Strictly speaking, we will have to check that all functions that takes xml-types as arguments are well-defined in the way that if the function is given the same arguments ( $\alpha$ -equivalent xml-types) then it returns the same result.

We will not prove this for all the functions we define, but we will now prove it for the substitution.

Since this result is well-known, some details in the following proof have been left out.

---

<sup>1</sup>[TAPL] has a good introduction to the properties we should expect from substitution and  $\alpha$ -equivalence.

**Lemma 3.3 (Substitution is well-defined)**

$$\vdash t \equiv t_\alpha \wedge \vdash t' \equiv t'_\alpha \Rightarrow \vdash t[t'/X] \equiv t_\alpha[t'_\alpha/X]$$

*This is proved in Proof 19.1 on page 76.*

We have now defined substitution and  $\alpha$ -equivalence for xml-types.

We will now prove a lemma, that is used many times in the rest of this project.

**Lemma 3.4 (Commutation of substitutions)**

$$X \neq Y \wedge X \notin \text{fv}(t_Y) \Rightarrow t[t_X/X][t_Y/Y] = t[t_Y/Y][t_X[t_Y/Y]/X]$$

*This is proved in proof 19.2 on page 77.*

A final note on substitution is that if  $t$  and  $t'$  are tail recursive and contractive then  $t[t'/X]$  is also tail recursive and contractive.

This can be proved by induction on  $|t|$ .

We have now defined substitution and  $\alpha$ -equivalence, and we are therefore ready to define the semantics of xml-types.

## 4 Semantics of Xml-types

In this section we define the semantics of xml-types as the set of xml-sequences that inhabits it.

Then we generalize the semantics to lists of xml-types, and use that to express the semantical subtype property we wish to investigate.

Before we define the semantics of xml-types we need to define appending of two xml-sequences, much like appending of lists in [SML].

### Definition 4.1

We define  $x_1 x_2 \in \mathbf{XMLSeq}$  for all  $x_1, x_2 \in \mathbf{XMLSeq}$  by structural induction on  $x_1$ .

$$\begin{aligned} \varepsilon x_2 &= x_2 \\ \langle l \rangle x_{11} \langle /l \rangle x_{12} x_2 &= \langle l \rangle x_{11} \langle /l \rangle (x_{12} x_2) \end{aligned}$$

It is easy to see that  $x_1 x_2 = \varepsilon \Leftrightarrow x_1 = x_2 = \varepsilon$ .

We get by structural induction that

$$\begin{aligned} x \varepsilon &= x \\ x_1 (x_2 x_3) &= (x_1 x_2) x_3. \end{aligned}$$

Below we give a big-step semantics of xml-types.

### Definition 4.2 (inhabits)

We define the binary relation  $\vdash x$  *inhabits*  $t$  where  $x \in \mathbf{XMLSeq}$  and  $t \in \mathbf{XMLType}$  by

$$\begin{aligned} \text{in-1} &\frac{}{\vdash \varepsilon \text{ inhabits } 1} & \text{in-tree} &\frac{\vdash x_1 \text{ inhabits } t_1}{\vdash \langle l \rangle x_1 \langle /l \rangle \varepsilon \text{ inhabits } l \langle t_1 \rangle} \\ \text{in-seq} &\frac{\vdash x_1 \text{ inhabits } t_1 \quad \vdash x_2 \text{ inhabits } t_2}{\vdash x_1 x_2 \text{ inhabits } t_1 t_2} \\ \text{in-sum1} &\frac{\vdash x_1 \text{ inhabits } t_1}{\vdash x_1 \text{ inhabits } t_1 + t_2} & \text{in-sum2} &\frac{\vdash x_2 \text{ inhabits } t_2}{\vdash x_2 \text{ inhabits } t_1 + t_2} \\ \text{in-mu} &\frac{\vdash x_1 \text{ inhabits } t_1 [\mu X.t_1/X]}{\vdash x_1 \text{ inhabits } \mu X.t_1} \end{aligned}$$

We can now define the set of xml-sequences that inhabits an xml-type in the following way.

### Definition 4.3 (Semantics of xml-types)

We define  $\text{in}[[t]]$  as the set of xml-sequences that inhabits  $t$  for all  $t \in \mathbf{XMLType}$ .

$$\text{in}[[t]] = \{x \mid x \text{ inhabits } t\}$$

### Observation 4.4 (Semantics is well-defined)

$$\vdash t \equiv t' \Rightarrow \text{in}[[t]] = \text{in}[[t']]$$

Each inclusion can be proved by induction on the derivation of  $\vdash x$  *inhabits*  $t$ .

We have now defined the semantics of an xml-type.

We now give an example of this semantics, by finding the set of xml-sequences that inhabits an xml-type.

**Example 4.5**

We will now find the semantics of the following contractive and tail-recursive xml-type.

$$t = \mu X.1 + l\langle X \rangle X.$$

This means that we will determine what xml-sequences that inhabit  $t$ .

Since  $t$  is a  $\mu$ -term, we need to unfold it, to find the xml-sequences that inhabit it.

We therefore need to determine what xml-sequences that inhabits the unfolded xml-type  $1 + l\langle t \rangle t$ .

This xml-type is of the form  $t_1 + t_2$  and therefore we need to find the xml-sequences that inhabit either  $t_1$  or  $t_2$ .

It is easy to find the xml-sequences that inhabit  $t_1 = 1$  since this is exactly the empty xml-sequence  $\varepsilon$ .

The xml-sequences that inhabits  $t_2 = l\langle t \rangle t$  are xml-sequences of the form  $\langle 1 \rangle x_1 \langle /1 \rangle x_2$  where  $x_1$  and  $x_2$  inhabit  $t$ .

Therefore the xml-sequences that inhabit  $t$  are  $\varepsilon$  and xml-sequences of the form  $\langle 1 \rangle x_1 \langle /1 \rangle x_2$  where  $x_1, x_2$  inhabit  $t$ .

This means that the xml-sequences that inhabit  $t$  are  $\varepsilon$ ,  $\langle 1 \rangle \varepsilon \langle /1 \rangle \varepsilon$ ,  $\langle 1 \rangle \langle 1 \rangle \varepsilon \langle /1 \rangle \varepsilon \langle /1 \rangle \varepsilon$ ,  $\langle 1 \rangle \varepsilon \langle /1 \rangle \langle 1 \rangle \varepsilon \langle /1 \rangle \varepsilon$  and so on.

There are many equivalent ways to define the semantics of xml-types, and our definition has been chosen in order to prove the correspondence with the tagging function in Section 6 as efficiently as possible, without actually using the tagging function to define the semantics.

An alternative way to define the semantics could be the following denotational semantics, which is equivalent to the above.

**Comment 4.6**

This is a denotational semantics, equivalent to the above operational semantics.

$$\begin{aligned} \mathbf{in}[0]_f &= \{\} \\ \mathbf{in}[1]_f &= \{\varepsilon\} \\ \mathbf{in}[l\langle t_1 \rangle]_f &= \{\langle 1 \rangle x_1 \langle /1 \rangle \varepsilon \mid x_1 \in \mathbf{in}[t_1]_f\} \\ \mathbf{in}[t_1 t_2]_f &= \{x_1 x_2 \mid x_1 \in \mathbf{in}[t_1]_f \wedge x_2 \in \mathbf{in}[t_2]_f\} \\ \mathbf{in}[t_1 + t_2]_f &= \{x_1 \mid x_1 \in \mathbf{in}[t_1]_f\} \cup \{x_2 \mid x_2 \in \mathbf{in}[t_2]_f\} \\ \mathbf{in}[\mu X.t_1]_f &= \bigcup_{i=0}^{\infty} \mathbf{in}[t_1]_{f^i} \quad \text{where } f^0 = f[X \mapsto \{\}] \quad \text{and } f^{i+1} = f[X \mapsto \mathbf{in}[t_1]_{f^i}] \\ \mathbf{in}[X]_f &= f(X) \end{aligned}$$

We now define the semantics for lists of xml-types.

**Definition 4.7**

We define  $IN[[T]]$  for all  $T \in \mathbf{XMLTypes}$  by induction on the length of  $T$ .

$$\begin{aligned} IN[[\ ]] &= \{ \} \\ IN[[t_1 :: T_1]] &= in[[t_1]] \cup IN[[T_1]] \end{aligned}$$

We observe that the semantics of a list of xml-types is the union of the semantics of its elements.

We will now formalize and prove this observation.

**Definition 4.8 (Translating lists to sets)**

$$\begin{aligned} \overline{\ ]} &= \{ \} \\ \overline{x :: xs} &= \{x\} \cup \overline{xs} \end{aligned}$$

**Lemma 4.9**

$$IN[[T]] = \bigcup_{t \in \overline{T}} in[[t]]$$

We prove this by induction on the length of the list  $T$ .

*Start:*  $T = \ ]$

In this case  $IN[[T]] = \{ \} = \bigcup_{t \in \{ \}} in[[t]] = \bigcup_{t \in \overline{T}} in[[t]]$ .

*Induction Hypothesis:*

If  $T = t_1 :: T_1$  then we can assume that  $IN[[T_1]] = \bigcup_{t \in \overline{T_1}} in[[t]]$ .

*Step:*  $T = t_1 :: T_1$

In this case the equality can be proved by the following rewritings.

$$\begin{aligned} IN[[T]] &= IN[[t_1 :: T_1]] \\ &= in[[t_1]] \cup IN[[T_1]] \\ \text{(IH)} &= in[[t_1]] \cup \bigcup_{t \in \overline{T_1}} in[[t]] \\ &= \bigcup_{t \in \{t_1\} \cup \overline{T_1}} in[[t]] \\ &= \bigcup_{t \in \overline{t_1 :: T_1}} in[[t]] \end{aligned}$$

■

Now we are ready to define the semantical subtyping problem for lists of xml-types in the following way.

**Definition 4.10**

$$\models T_1 <: T_2 \Leftrightarrow \text{IN}[[T_1]] \subseteq \text{IN}[[T_2]]$$

We have now defined semantics for xml-types and lists of xml-types, and seen that the definition of the semantics of lists of xml-types matches the intuitive understanding, that an xml-sequence inhabits a list of xml-types if and only if it inhabits one of its xml-type elements.

Finally we used the semantics of lists of xml-types to express the semantical subtype property.

## 5 Xml-data

We have now introduced xml-sequences and xml-types.

We will in this section introduce *type-specific* representations of xml-sequences.

Type-specific representations are bound to an xml-type, and are used to represent an xml-sequence in the semantics of that xml-type. The type-specific representation holds the information necessary to describe the exact xml-sequence it represents.

If the xml-type is of the form  $t_1 + t_2$  this representation must describe if the data uses the type  $t_1$  or  $t_2$ .

Also the data must include the values in the leaf nodes, but since the type systems only allow the unit type in the leaves, there is only one possible value.

The type-specific representation does not need to hold information about the used tag-names, since they can be found in the xml-type.

We will now define the language of the type-specific representations for xml-types.

### Definition 5.1

The set **XMLData** of type-specific representations of xml-sequences for xml-types contains exactly the elements that can be constructed using the following grammar

$$d ::= \bullet \mid \langle d_1 \rangle \mid d_1 d_2 \mid d_1 \diamond \mid \diamond d_2$$

$\bullet$  is used for the xml-type 1, and represents the empty xml-sequence.

$\langle d_1 \rangle$  is used for xml-types of the form  $l\langle t_1 \rangle$ , and represents  $\langle 1 \rangle x_1 \langle /1 \rangle \varepsilon$  where  $d_1$  represents  $x_1$  for  $t_1$ .

$d_1 d_2$  is used for xml-types of the form  $t_1 t_2$ , and represents  $x_1 x_2$  when  $d_1$  represents  $x_1$  for  $t_1$  and  $d_2$  represents  $x_2$  for  $t_2$ .

$d_1 \diamond$  is used for xml-types of the form  $t_1 + t_2$ , and represents  $x_1$  when  $d_1$  represents  $x_1$  for  $t_1$ .

$\diamond d_2$  is used for xml-types of the form  $t_1 + t_2$ , and represents  $x_2$  when  $d_2$  represents  $x_2$  for  $t_2$ .

This means that given an xml-type, xml-data can be used to represent the xml-sequences that inhabits that type.

The following example will illustrate this, and give some idea of how we combine an xml-type with xml-data to obtain an xml-sequence.

### Example 5.2 (An example of xml-data)

We consider

$$d = \langle \langle \bullet \rangle \diamond \rangle \langle \diamond \langle \langle \bullet \rangle \diamond \rangle \langle \langle \langle \bullet \rangle \diamond \rangle \langle \langle \bullet \rangle \langle \bullet \diamond \rangle \rangle \rangle \rangle$$

and

$$t = \text{gender} \langle \text{male} \langle 1 \rangle + \text{female} \langle 1 \rangle \rangle \text{age} \langle \mu X.1 + s \langle 1 \rangle X \rangle$$

from Example 2.4.

Now we will try to deduce in an intuitive way what xml-sequence  $d$  represents.

Since  $t$  is of the form  $(l_1 \langle t_1 \rangle)(l_2 \langle t_2 \rangle)$ , and  $d$  is of the form  $\langle d_1 \rangle \langle d_2 \rangle$ , we know that the resulting xml-sequence consists of two trees, the first one named *gender*, because  $l_1 = \text{gender}$

and the second one named **age**, because  $l_2 = \mathbf{age}$ .

The content of the **gender** tree is the xml-sequence we get by combining  $d_1 = \langle \bullet \rangle \diamond$  with  $t_1 = \mathbf{male}\langle 1 \rangle + \mathbf{female}\langle 1 \rangle$ . Since  $t_1$  is of the form  $t_{11} + t_{12}$  and  $d_1$  is of the form  $d_{11} \diamond$  the result is the same as combining  $t_{11} = \mathbf{male}\langle 1 \rangle$  with  $d_{11} = \langle \bullet \rangle$ .

The result of this is a single tree named **male** and the content of this we get by combining  $\bullet$  with 1 and this is the empty xml-sequence  $\varepsilon$ .

We have now found that the first tree must be  $\langle \mathbf{gender} \rangle \langle \mathbf{male} \rangle \varepsilon \langle / \mathbf{male} \rangle \varepsilon \langle / \mathbf{gender} \rangle \varepsilon$ .

We get the content of the second tree by combining  $d_2 = \diamond(\langle \bullet \rangle(\diamond(\langle \bullet \rangle(\diamond(\langle \bullet \rangle(\bullet \diamond))))))$  with  $t_2 = \mu X.1 + \mathbf{s}\langle 1 \rangle X$ .

Since  $t_2$  is a  $\mu$ -term, we must unfold it in order to continue.

We therefore use  $t'_2 = (1 + \mathbf{s}\langle 1 \rangle X)[\mu X.1 + \mathbf{s}\langle 1 \rangle / X] = 1 + \mathbf{s}\langle 1 \rangle t_2$ .

Since  $d_2$  is of the form  $\diamond d_{22}$  and  $t'_2$  is of the form  $t_{21} + t_{22}$  we get that the result is the same as we get by combining  $d_{22}$  with  $t_{22}$ .

The result of this starts with a tree named **s**, and the content of this is  $\varepsilon$ .

In the same way we get that there are two more empty trees named **s** in this sequence, and the rest of the xml-sequence is the result of combining  $d_{23} = \bullet \diamond$  with  $t_2 = \mu X.1 + \mathbf{s}\langle 1 \rangle X$ .

Again we must unfold  $t_2$  to  $t'_2 = 1 + \mathbf{s}\langle 1 \rangle t_2$  in order to continue.

Since  $d_{23}$  is of the form  $\bullet \diamond$  and  $t'_2$  is of the form  $t_{23} + t_{24}$  the result is the same as combining  $\bullet$  with  $t_{23} = 1$  and this is the empty xml-sequence  $\varepsilon$ .

We can now combine these results to obtain the entire xml-sequence and this is

$$\langle \mathbf{gender} \rangle \langle \mathbf{male} \rangle \varepsilon \langle / \mathbf{male} \rangle \varepsilon \langle / \mathbf{gender} \rangle \langle \mathbf{age} \rangle \langle \mathbf{s} \rangle \varepsilon \langle / \mathbf{s} \rangle \langle \mathbf{s} \rangle \varepsilon \langle / \mathbf{s} \rangle \langle \mathbf{s} \rangle \varepsilon \langle / \mathbf{s} \rangle \varepsilon \langle / \mathbf{age} \rangle \varepsilon,$$

which is the xml-sequence from Example 2.2.

This Example is shown more formally in Example 6.4.

It is worth noting, that the definition of xml-data easily can be extended with simple types of data as integers and strings. One way to do this could be to extend the definition to

$$d' ::= \bullet \mid \langle d'_1 \rangle \mid \langle s \rangle \mid \langle i \rangle \mid d'_1 d'_2 \mid d'_1 \diamond \mid \diamond d'_2$$

where  $s$  can be any element from the set of quoted strings, and  $i$  can be any element from  $\mathbb{N}$ .

In order to limit the number of cases in the proofs, we will only consider the original definition of xml-data.

We are now ready to define the type-specific representations for lists of xml-types.

Data for a list of xml-types should include the position of the exact xml-type in the list, and the actual xml-data.

### Definition 5.3

$$\mathbf{XMLDatas} = \mathbb{N} \times \mathbf{XMLData}$$

We will use  $D$  to represent elements of  $\mathbf{XMLDatas}$ .

We have now defined the language of type-specific representations of xml-sequences for xml-types and lists of xml-types.

## 6 Tagging

We will in this section define how to create an xml-sequence from an xml-type and a type-specific representation.

We will then prove that the semantics of an xml-type contains exactly the xml-sequences that can be represented for that xml-type.

We will start by defining a degenerated size-function for xml-types.

### Definition 6.1

$$ismu(t) = \begin{cases} 1 & \text{if } t \text{ is of the form } \mu X.t' \\ 0 & \text{otherwise} \end{cases}$$

We are now ready to define the tagging function.

### Definition 6.2 (tag)

We will now define  $\mathbf{tag}(t, d)$  by induction on  $2 \cdot |d| + ismu(t)$ .

```

fun tag(1      , ●)      = ε
  | tag(l⟨t₁⟩   , ⟨d₁⟩)   = let x₁ = tag(t₁, d₁) in <l>x₁</l>ε end
  | tag(t₁t₂    , d₁d₂)   = let x₁ = tag(t₁, d₁), x₂ = tag(t₂, d₂) in x₁ x₂ end
  | tag(t₁ + t₂ , d₁◇)    = tag(t₁, d₁)
  | tag(t₁ + t₂ , ◇d₂)    = tag(t₂, d₂)
  | tag(μX.t₁   , d)      = tag(t₁[μX.t₁/X], d)
  | tag(-       , -)      = ⊤

```

### Observation 6.3 (tag is well-defined)

If  $t \equiv t'$  then  $\mathbf{tag}(t, d) = \mathbf{tag}(t', d)$ . This can be proved by induction on  $2 \cdot |d| + ismu(t)$ .

Before we prove that the  $\mathbf{tag}$  function is sound and complete with respect to the semantics of an xml-type, we will show an example of how  $\mathbf{tag}$  works.

### Example 6.4 (Tagging)

We consider the type  $t = \mathbf{gender}\langle \mathbf{male}\langle 1 \rangle + \mathbf{female}\langle 1 \rangle \rangle \mathbf{age}\langle \mu X.1 + \mathbf{s}\langle 1 \rangle X \rangle$  from Example 2.4, and the data  $d = \langle \langle \bullet \rangle \diamond \rangle \langle \diamond \langle \bullet \rangle \rangle \langle \diamond \langle \bullet \rangle \rangle \langle \diamond \langle \bullet \rangle \rangle \langle \bullet \diamond \rangle \rangle$  from Example 5.2.



$$\begin{aligned}
&= \text{let } x_1 = \langle \text{gender} \rangle \langle \text{male} \rangle \varepsilon \langle / \text{male} \rangle \varepsilon \langle / \text{gender} \rangle \varepsilon, \\
&\quad x_2 = \text{let } x'_2 = \text{let } x_{21} = \langle s \rangle \varepsilon \langle / s \rangle \varepsilon, \\
&\quad\quad x_{22} = \langle s \rangle \varepsilon \langle / s \rangle \langle s \rangle \varepsilon \langle / s \rangle \varepsilon \\
&\quad\quad\quad \text{in } x_{21} \ x_{22} \text{end} \\
&\quad\quad \text{in } \langle \text{age} \rangle x'_2 \langle / \text{age} \rangle \varepsilon \text{end} \\
&\text{in } x_1 \ x_2 \text{end} \\
&= \text{let } x_1 = \langle \text{gender} \rangle \langle \text{male} \rangle \varepsilon \langle / \text{male} \rangle \varepsilon \langle / \text{gender} \rangle \varepsilon, \\
&\quad x_2 = \text{let } x'_2 = \langle s \rangle \varepsilon \langle / s \rangle \langle s \rangle \varepsilon \langle / s \rangle \langle s \rangle \varepsilon \langle / s \rangle \varepsilon \\
&\quad\quad \text{in } \langle \text{age} \rangle x'_2 \langle / \text{age} \rangle \varepsilon \text{end} \\
&\text{in } x_1 \ x_2 \text{end} \\
&= \text{let } x_1 = \langle \text{gender} \rangle \langle \text{male} \rangle \varepsilon \langle / \text{male} \rangle \varepsilon \langle / \text{gender} \rangle \varepsilon, \\
&\quad x_2 = \langle \text{age} \rangle \langle s \rangle \varepsilon \langle / s \rangle \langle s \rangle \varepsilon \langle / s \rangle \langle s \rangle \varepsilon \langle / s \rangle \varepsilon \langle / \text{age} \rangle \varepsilon \\
&\quad \text{in } x_1 \ x_2 \text{end} \\
&= \langle \text{gender} \rangle \langle \text{male} \rangle \varepsilon \langle / \text{male} \rangle \varepsilon \langle / \text{gender} \rangle \langle \text{age} \rangle \langle s \rangle \varepsilon \langle / s \rangle \langle s \rangle \varepsilon \langle / s \rangle \langle s \rangle \varepsilon \langle / s \rangle \varepsilon \langle / \text{age} \rangle \varepsilon.
\end{aligned}$$

We can therefore see, that  $d$  is the type-specific representation of the xml-sequence from Example 2.2 for the type  $t$ .

The **tag** function can fail, if the given type-specific representation does not match the given xml-type, or if the given xml-type is not contractive. We will now show this in an example.

#### Example 6.5 (Tagging errors)

The first case where **tag** fails is if the xml-data and the xml-type does not match.

There are many examples of this, but we will look at the following case.

$$t = 1 + 1 \quad \text{and} \quad d = \bullet.$$

In this case  $\text{tag}(t, d)$  only matches the final case, and therefore  $\text{tag}(t, d) = \top$ .

Another example where **tag** fails is if it is given an unrestricted xml-type that is not contractive. In this case there is a possibility that the **tag** function will go into a never-ending loop. We will consider the following case.

$$t = \mu X.X \quad \text{and} \quad d = \bullet.$$

In this case  $\text{tag}(t, d) = \text{tag}(\mu X.X, d) = \text{tag}(X[\mu X.X/X], d) = \text{tag}(\mu X.X, d)$ .

Therefore  $\text{tag}(t, d)$  never terminates.

Now we can describe the semantics of an xml-type by using **tag**, because it is exactly the xml-sequences that can be obtained by tagging it with some xml-data.

Now we will prove this in the following soundness and completeness theorems for **tag**.

#### Theorem 6.6 (Soundness of tag)

$$\text{tag}(t, d) \neq \top \Rightarrow \vdash \text{tag}(t, d) \text{ inhabits } t$$

This is proved in Proof 19.3 on page 80.

**Theorem 6.7 (Completeness of tag)**

$$\vdash x \text{ inhabits } t \Rightarrow \exists d. \text{tag}(t, d) = x$$

This is proved in Poof 19.4 on page 81.

Now we collect the results in the following corollary.

**Corollary 6.8 (Correctness of tag)**

$$\text{in}[[t]] = \{x \in \mathbf{XMLSeq} \mid \exists d \in \mathbf{XMLData}. \text{tag}(t, d) = x\}$$

We prove each inclusion of this equality separately.

If  $x \in \text{in}[[t]]$  then  $\vdash x \text{ inhabits } t$ . Therefore Theorem 6.7 yields that there is a  $d$  such that  $\text{tag}(t, d) = x$ .

Therefore  $x \in \{x \in \mathbf{XMLSeq} \mid \exists d \in \mathbf{XMLData}. \text{tag}(t, d) = x\}$ .

If  $x \in \{x \in \mathbf{XMLSeq} \mid \exists d \in \mathbf{XMLData}. \text{tag}(t, d) = x\}$  then there is a  $d$  such that  $\text{tag}(t, d) = x \neq \top$  and therefore Theorem 6.6 yields that  $\vdash x \text{ inhabits } x$  and therefore  $x \in \text{in}[[t]]$ .

We have now proved both inclusions and can therefore conclude that

$$\text{in}[[t]] = \{x \in \mathbf{XMLSeq} \mid \exists d \in \mathbf{XMLData}. \text{tag}(t, d) = x\}.$$

■

Now we are ready to generalize tagging to lists of types.

**Definition 6.9**

$$\begin{aligned} \text{TAG}(t_1 :: T_1, (0, d)) &= \text{tag}(t_1, d) \\ \text{TAG}(t_1 :: T_1, (n, d)) &= \text{TAG}(T_1, (n-1, d)) \\ \text{TAG}([], (n, d)) &= \top \end{aligned}$$

We see that  $\text{TAG}(T, (n, d))$  works by finding the  $n$ 'th element  $t$  of  $T$  and return  $\text{tag}(t, d)$ . Since this is a simple generalization, the correctness proof follows from the correctness of the  $\text{tag}$  function. This is done in the following corollary.

**Corollary 6.10 (Correctness of TAG)**

$$\forall T \in \mathbf{XMLTypes}. \text{IN}[[T]] = \{x \in \mathbf{XMLSeq} \mid \exists D \in \mathbf{XMLDatas}. \text{TAG}(T, D) = x\}$$

This is proved by the following equalities

$$\begin{aligned}
& IN[[T]] \\
(\text{Lemma 4.9}) &= \bigcup_{t \in \bar{T}} in[[t]] \\
(\text{Corollary 6.8}) &= \bigcup_{t \in \bar{T}} \{x \in \mathbf{XMLSeq} \mid \exists d \in \mathbf{XMLData}. tag(t, d) = x\} \\
&= \bigcup_{n \in \mathbb{N}} \{x \in \mathbf{XMLSeq} \mid \exists d \in \mathbf{XMLData}. TAG(T, (n, d)) = x\} \\
&= \{x \in \mathbf{XMLSeq} \mid \exists n \in \mathbb{N}. \exists d \in \mathbf{XMLData}. TAG(T, (n, d)) = x\} \\
&= \{x \in \mathbf{XMLSeq} \mid \exists D \in \mathbf{XMLDats}. TAG(T, D) = x\}.
\end{aligned}$$

■

We have now defined the tagging function, and generalized it to work on lists of xml-types. We have also proved that the xml-sequences we can create with the tagging functions correspond with the semantics of the xml-types.

## 7 The Empty-sequence Property

[CA-REG] introduced the *empty-word property* as a boolean function that determines if the empty word could be derived from the given regular expression. We need something similar for this problem, but because we want to extend the axiomatization with translation of the type-specific representations, it is necessary to return a proof that the empty xml-sequence is derivable for the given type in the form of a type-specific representation of  $\varepsilon$ .

This leads to the following definition.

**Definition 7.1 (The empty-sequence property:  $\text{esp}$ )**

We define  $\text{esp}(t)$  by structural induction on  $t$ .

$$\begin{aligned}
 \text{esp}(0) &= \{\} \\
 \text{esp}(1) &= \{\bullet\} \\
 \text{esp}(l\langle t_1 \rangle) &= \{\} \\
 \text{esp}(t_1 + t_2) &= \text{if } \text{esp}(t_1) \neq \{\} \\
 &\quad \text{then } \{d_1 \diamond \mid d_1 \in \text{esp}(t_1)\} \\
 &\quad \text{else } \{\diamond d_2 \mid d_2 \in \text{esp}(t_2)\} \\
 \text{esp}(t_1 t_2) &= \{d_1 d_2 \mid d_1 \in \text{esp}(t_1) \wedge d_2 \in \text{esp}(t_2)\} \\
 \text{esp}(\mu X.t_1) &= \text{esp}(t_1) \\
 \text{esp}(X) &= \{\}
 \end{aligned}$$

**Observation 7.2 ( $\text{esp}$  is well-defined)**

$$\vdash t \equiv t' \Rightarrow \text{esp}(t) = \text{esp}(t')$$

If we generalize the observation to accept renaming of unbound variables, then it can be proved by structural induction on  $t$ .

Next we give an example of how the empty-sequence property works.

**Example 7.3**

We consider the case where

$$t = \mu X.l\langle 1 \rangle X + 1.$$

In this case

$$\begin{aligned}
& \mathbf{esp}(t) \\
&= \mathbf{esp}(\mu X.l\langle 1 \rangle X + 1) \\
&= \mathbf{esp}(l\langle 1 \rangle X + 1) \\
&= \{d_1 \diamond \mid d_1 \in \mathbf{esp}(l\langle 1 \rangle X)\} \cup \{\diamond d_2 \mid d_2 \in \mathbf{esp}(1)\} \\
&= \{d_1 \diamond \mid d_1 \in \{d_1 d_2 \mid d_1 \in \mathbf{esp}(l\langle 1 \rangle)\} \wedge d_2 \in \mathbf{esp}(X)\} \cup \{\diamond d_2 \mid d_2 \in \mathbf{esp}(1)\} \\
&= \{d_1 \diamond \mid d_1 \in \{d_1 d_2 \mid d_1 \in \{\} \wedge d_2 \in \{\}\}\} \cup \{\diamond d_2 \mid d_2 \in \mathbf{esp}(1)\} \\
&= \{d_1 \diamond \mid d_1 \in \{\}\} \cup \{\diamond d_2 \mid d_2 \in \mathbf{esp}(1)\} \\
&= \{\} \cup \{\diamond d_2 \mid d_2 \in \mathbf{esp}(1)\} \\
&= \{\diamond d_2 \mid d_2 \in \{\bullet\}\} \\
&= \{\diamond \bullet\}.
\end{aligned}$$

Now we check that the result is valid, so we need to check that

$$\begin{aligned}
& \mathbf{tag}(t, \diamond \bullet) \\
&= \mathbf{tag}(\mu X.l\langle 1 \rangle X + 1, \diamond \bullet) \\
&= \mathbf{tag}(l\langle 1 \rangle t + 1, \diamond \bullet) \\
&= \mathbf{tag}(1, \bullet) \\
&= \varepsilon.
\end{aligned}$$

Now we prove that  $\mathbf{esp}$  is sound, but instead of proving that  $\mathbf{esp}(t) \neq \{\} \Rightarrow \varepsilon \in \mathbf{in}[[t]]$ , we prove that  $\forall d \in \mathbf{esp}(t). \mathbf{tag}(t, d) = \varepsilon$ .

Since Corollary 6.8 proves that  $\mathbf{in}[[t]] = \{x \in \mathbf{XMLSeq} \mid \exists d \in \mathbf{XMLData}. \mathbf{tag}(t, d) = x\}$  this is a stronger result.

Since all recursive calls in  $\mathbf{esp}(t)$  uses a sub-term of  $t$ , this can be proved by structural induction on  $t$ . The only non trivial case is the  $\mu$ -case, and we will therefore need the following lemma.

**Lemma 7.4**

$$\mathbf{tag}(t, d) \neq \top \Rightarrow \mathbf{tag}(t, d) = \mathbf{tag}(t[t'/X], d)$$

This is proved in Proof 19.5 on page 82.

We are now ready to prove that  $\mathbf{esp}$  is sound.

**Lemma 7.5 (Soundness of  $\mathbf{esp}$ )**

$$\forall t \in \mathbf{XMLType}. \forall d \in \mathbf{esp}(t). \mathbf{tag}(t, d) = \varepsilon$$

This is proved in Proof 19.6 on page 84.

We will now prove that  $\mathbf{esp}$  is complete.

We can again do this by structural induction on  $t$ .

The only non trivial case is the  $\mu$ -case, and we will therefore need the following lemmas.

**Lemma 7.6**

$$\mathbf{tag}(t[t'/X], d) = \varepsilon \Rightarrow \mathbf{tag}(t, d) = \varepsilon \vee \exists d'. \mathbf{tag}(t', d') = \varepsilon \wedge |d'| \leq |d|$$

This is proved in Proof 19.7 on page 85.

**Lemma 7.7**

$$\mathbf{tag}(\mu X.t_1, d) = \varepsilon \Rightarrow \exists d'. \mathbf{tag}(t_1, d') = \varepsilon$$

This is proved in Proof 19.8 on page 88.

Now we are ready to prove that **esp** is complete.

**Lemma 7.8 (Completeness of esp)**

$$\forall t \in \mathbf{XMLType}. \varepsilon \in \mathbf{in}[t] \Rightarrow \mathbf{esp}(t) \neq \{\}$$

This is proved in Proof 19.9 on page 88.

We will now generalize the *empty sequence property* to work on lists of xml-types.

**Definition 7.9 (ESP : XMLTypes  $\rightarrow$   $\mathcal{P}$ (XMLDatas))**

$$\begin{aligned} \mathbf{ESP}(\[]) &= \{\} \\ \mathbf{ESP}(t_1 :: T_1) &= \{(0, d_1) \mid d_1 \in \mathbf{esp}(t_1)\} \cup \{(n+1, d_1) \mid (n, d_1) \in \mathbf{ESP}(T_1)\} \end{aligned}$$

We will now prove that **ESP** is sound in the same way as **esp**.

**Corollary 7.10 (Soundness of ESP)**

$$\forall T \in \mathbf{XMLTypes}. D \in \mathbf{ESP}(T) \Rightarrow \mathbf{TAG}(T, D) = \varepsilon$$

We will prove this by induction on the length of  $T$ .

*Start:*  $T = []$

In this case  $\mathbf{ESP}(T) = \{\}$ , so the lemma is trivially true.

*Induction Hypothesis:*

If  $T = t_1 :: T_1$  then we can assume that  $\forall D_1 \in \mathbf{ESP}(T_1). \mathbf{TAG}(T_1, D_1) = \varepsilon$ .

*Step:*  $T = t_1 :: T_1$

In this case  $\mathbf{ESP}(T) = \{(0, d_1) \mid d_1 \in \mathbf{esp}(t_1)\} \cup \{(n+1, d_1) \mid (n, d_1) \in \mathbf{ESP}(T_1)\}$ , so there are two cases for  $D \in \mathbf{ESP}(T)$ .

If  $D \in \{(0, d_1) \mid d_1 \in \mathbf{esp}(t_1)\}$  then  $\mathbf{TAG}(T, D) = \mathbf{TAG}(t_1 :: T_1, (0, d_1)) = \mathbf{tag}(t_1, d_1)$ .

Since  $d_1 \in \mathbf{esp}(t_1)$ , Lemma 7.5 yields that  $\mathbf{tag}(t_1, d_1) = \varepsilon$ .

If  $D \in \{(n+1, d_1) \mid (n, d_1) \in \mathbf{ESP}(T_1)\}$  then the induction hypothesis yields that

$\forall (n, d_1) \in \mathbf{ESP}(T_1). \mathbf{TAG}(T_1, (n, d_1)) = \varepsilon.$

Therefore we get that  $\mathbf{TAG}(T, D) = \mathbf{TAG}((t_1 :: T_1), (n + 1, d_1)) = \mathbf{TAG}(T_1, (n, d_1)) = \varepsilon.$

We can now conclude that  $\forall T \in \mathbf{XMLTypes}. D \in \mathbf{ESP}(T) \Rightarrow \mathbf{TAG}(T, D) = \varepsilon.$  ■

The normal way to define completeness for ESP would be that  $\varepsilon \in \mathbf{IN}[[T]] \Rightarrow \mathbf{ESP}(T) \neq \{\}$ , but we will need a stronger completeness when we get to translation of xml-data, and we will now prove this stronger completeness.

The reason for this is that we need to ensure that we find representations of the empty xml-sequence with all possible indices.

This makes the lemma more complex to express, which means that the induction hypothesis is a bit complicated.

The main complication is however the induction step, which have to keep track of all the indices when the induction hypothesis is used.

### Lemma 7.11 (Completeness of ESP)

$\forall T = [t_0, t_1, \dots, t_k] \in \mathbf{XMLTypes}. \forall i \in \{0, 1, \dots, k\}. \varepsilon \in \mathbf{in}[[t_i]] \Rightarrow \{(n, d) \in \mathbf{ESP}(T) \mid n = i\} \neq \{\}$

*This is proved in Proof 19.10 on page 89.*

We have now defined a function that determines the *empty sequence property* for xml-types and lists of xml-types. We have proved that the functions are sound and complete.

In the case where  $\varepsilon$  inhabits the given xml-type the functions return a type-specific representation of  $\varepsilon$  for the given type.

## 8 Type Splitting

In this section, we will introduce a central concept in the axiomatization.

This is a function that splits an xml-type into a list of xml-types that are of a desired form.

We will then generalize the type-splitting to work on lists of xml-types.

We will use Section 10 to prove in a constructive way, that the semantics except for the empty sequence is preserved by the generalized type splitting.

In [CA-REG] the basis of the axiomatization of equality on regular expressions is the concept of derivatives, and the type-splitting will help express the derivatives of xml-types and lists of xml-types. It is not possible to find the derivative for every way it is possible to start an xml-sequence, simply because it is not always the case that there are only finite many ways to start the xml-sequences that inhabits an xml-type.

Type-splitting will help us solve this problem, because the form of the elements in the result expresses how the inhabited xml-sequences can start, and how the xml-sequences then can continue.

In order to define type-splitting, it is necessary to keep track of a sequence of substitutions, and we will use a list of substitutions to represent this.

We now define this formally.

### Definition 8.1

If  $\varphi$  is a list of substitutions, then we define  $t[\varphi]$  by induction on the length of  $\varphi$ .

$$\begin{aligned} t[\ ] &= t \\ t[\ \mu X.t_X/X :: \varphi_1 ] &= t[\mu X, t_X/X][\varphi_1]. \end{aligned}$$

We will use  $\text{dom}(\varphi)$  to denote the set of variables that are substituted by  $\varphi$ .

We now define the type-splitting for xml-types.

### Definition 8.2 (Type Splitting: $\text{Split}_\varphi$ )

We now define  $\text{Split}_\varphi(t)$  where  $\varphi$  is a list of substitutions by structural induction on  $t$ .

$$\begin{aligned} \text{Split}_\varphi(0) &= \[] \\ \text{Split}_\varphi(1) &= \[] \\ \text{Split}_\varphi(X) &= \[] \\ \text{Split}_\varphi(l\langle t_1 \rangle) &= [l\langle t_1[\varphi] \rangle 1] \\ \text{Split}_\varphi(t_1 t_2) &= \text{if } \text{esp}(t_1) \neq \{ \} \\ &\quad \text{then } [l\langle t_{11} \rangle(t_{12}(t_2[\varphi])) \mid l\langle t_{11} \rangle t_{12} \in \text{Split}_\varphi(t_1)] @ \text{Split}_\varphi(t_2) \\ &\quad \text{else } [l\langle t_{11} \rangle(t_{12}(t_2[\varphi])) \mid l\langle t_{11} \rangle t_{12} \in \text{Split}_\varphi(t_1)] \\ \text{Split}_\varphi(t_1 + t_2) &= \text{Split}_\varphi(t_1) @ \text{Split}_\varphi(t_2) \\ \text{Split}_\varphi(\mu X.t_1) &= \text{Split}_{\mu X.t_1/X :: \varphi}(t_1) \end{aligned}$$

**Observation 8.3 (Split is well-defined)**

If  $\vdash t \equiv t'$  and  $\vdash t[\varphi] \equiv t'[\varphi']$  then  $\mathbf{Split}_\varphi(t) = \mathbf{Split}_{\varphi'}(t')$  If we generalize the observation to allow renaming of unbound variables in  $t$  but not in  $t[\varphi]$  then this observation can be proved by structural induction on  $t$ .

It is necessary to use  $\varphi$  in  $\mathbf{Split}$  in order to ensure that  $\mathbf{Split}$  always terminates.

When we split an xml-type we will always use  $\mathbf{Split}$  with the empty list of substitutions. Therefore we will use the following definition.

**Definition 8.4 (Split)**

$$\mathbf{Split}(t) = \mathbf{Split}_\square(t)$$

We will now use an example to show how  $\mathbf{Split}$  works.

**Example 8.5 (An example of Split)**

We consider

$$t = \mu X.1 + l\langle X \rangle X.$$

We will now evaluate  $\mathbf{Split}(t)$ .

$$\begin{aligned} \mathbf{Split}(\mu X.1 + l\langle X \rangle X) &= \mathbf{Split}_\square(\mu X.1 + l\langle X \rangle X) \\ &= \mathbf{Split}_{[t/X]}(1 + l\langle X \rangle X) \\ &= \mathbf{Split}_{[t/X]}(1) @ \mathbf{Split}_{[t/X]}(l\langle X \rangle X) \\ &= \square @ \mathbf{Split}_{[t/X]}(l\langle X \rangle X) \\ &= \mathbf{Split}_{[t/X]}(l\langle X \rangle X) \\ &= \text{if } \mathbf{esp}(l\langle X \rangle) \neq \{\} \\ &\quad \text{then } [l'\langle t_1 \rangle(t_2(X[ [t/X] ]))] \mid l'\langle t_1 \rangle t_2 \in \mathbf{Split}_{[t/X]}(l\langle X \rangle) @ \mathbf{Split}_{[t/X]}(X) \\ &\quad \text{else } [l'\langle t_1 \rangle(t_2(X[ [t/X] ]))] \mid l'\langle t_1 \rangle t_2 \in \mathbf{Split}_{[t/X]}(l\langle X \rangle) \\ &= \text{if } \{\} \neq \{\} \\ &\quad \text{then } [l'\langle t_1 \rangle(t_2(X[ [t/X] ]))] \mid l'\langle t_1 \rangle t_2 \in \mathbf{Split}_{[t/X]}(l\langle X \rangle) @ \mathbf{Split}_{[t/X]}(X) \\ &\quad \text{else } [l'\langle t_1 \rangle(t_2(X[ [t/X] ]))] \mid l'\langle t_1 \rangle t_2 \in \mathbf{Split}_{[t/X]}(l\langle X \rangle) \\ &= [l'\langle t_1 \rangle(t_2(X[ [t/X] ]))] \mid l'\langle t_1 \rangle t_2 \in \mathbf{Split}_{[t/X]}(l\langle X \rangle) \\ &= [l'\langle t_1 \rangle(t_2 t) \mid l'\langle t_1 \rangle t_2 \in \mathbf{Split}_{[t/X]}(l\langle X \rangle)] \\ &= [l'\langle t_1 \rangle(t_2 t) \mid l'\langle t_1 \rangle t_2 \in [l\langle X[ [t/X] ]1]] \\ &= [l\langle t \rangle(1t)] \end{aligned}$$

We now inspect the result.

The first thing we notice is that the element(s) is of the form  $l\langle t_1 \rangle t_2$  for some  $l, t_1$  and  $t_2$ .

This is a general result, that all elements in  $\mathbf{Split}(t)$  will be on this form, and it is also the key point of introducing  $\mathbf{Split}$ .

The second thing we notice is that intuitively, the semantics of  $t$  is the same as the semantics of  $\mathbf{Split}(t)$  with the exception of the empty sequence.

This is also a general result which we will prove in Section 10.

The last thing that we will mention in this example is that if we consider the direct sub-terms of the element(s) in  $\mathbf{Split}(t)$  they are  $t$  and  $1t$ , and if we evaluate  $\mathbf{Split}(t)$  and  $\mathbf{Split}(1t)$  we will find that they returned the same as  $\mathbf{Split}(t)$ .

This can be generalized to the result, that there are finitely many derivatives of a term, which are exactly the terms that can be reached by recursively calculating  $\mathbf{Split}$  and find the direct sub-terms of the elements in the result.

We will formalize and prove this in Section 11.

We have now defined  $\mathbf{Split}$ .

We will now prove that all elements in the result of  $\mathbf{Split}$  is of the desired form.

**Lemma 8.6 (Split is well-formed)**

$$\forall t' \in \overline{\mathbf{Split}_\varphi(t)}.t' = l'(t'_1)t'_2 \text{ for some } l', t'_1 \text{ and } t'_2.$$

This is proved in Proof 19.11 on page 90.

In order for  $\mathbf{Split}$  to be useful, the semantics of the result must correspond to the semantics of the given xml-type. Since  $\varepsilon$  cannot inhabit any xml-type of the form of the elements, the best we can hope for is that the semantics of  $\mathbf{Split}(t)$  is the same as the semantics of  $t$ , except for the empty xml-sequence  $\varepsilon$ .

We will use the Section 10 to introduce functions that translates data between  $t$  and  $\mathbf{Split}(t)$ , and from  $\mathbf{Split}(t)$  to  $t$ , and use this to prove in a constructive way that except for the empty xml-sequence, the semantics of an xml-type is preserved by  $\mathbf{Split}$ .

We will now generalize  $\mathbf{Split}$  to work on lists of xml-types.

**Definition 8.7**

$$\begin{aligned} \mathbf{SPLIT}(\[]) &= \[] \\ \mathbf{SPLIT}(t_1 :: T_1) &= \mathbf{Split}(t_1) @ \mathbf{SPLIT}(T_1) \end{aligned}$$

Since  $\mathbf{SPLIT}$  creates its result by appending results of  $\mathbf{Split}$  we know that all the elements in the result is of the form  $l(t_1)t_2$ .

We have now defined  $\mathbf{Split}$ , proved that the elements in the result are of the desired form, and generalized  $\mathbf{Split}$  to work on lists of xml-types.

## 9 Restricted Tagging and Data Simplification

We have now defined a way to split an xml-type to a list of xml-types on a certain form. We are of course interested in translating xml-data between the original xml-type and the splitted list of xml-types. We will in this section introduce two functions that are important in this translation.

If we try to translate the xml-data from the original xml-type to the splitted list of xml-types, there is a problem which is illustrated by the following example.

### Example 9.1 (Redundant data)

We consider  $t = \mu X.1 + (1 + l\langle 1 \rangle)X$  and  $d = \diamond((\bullet\diamond)(\diamond((\diamond\langle \diamond \rangle)(\bullet\diamond))))$ .

First we try to evaluate  $\text{tag}(t, d)$ .

$$\begin{aligned}
& \text{tag}(t, d) \\
&= \text{tag}(\mu X.1 + (1 + l\langle 1 \rangle)X, \diamond((\bullet\diamond)(\diamond((\diamond\langle \diamond \rangle)(\bullet\diamond)))) \\
&= \text{tag}(1 + (1 + l\langle 1 \rangle)t, \diamond((\bullet\diamond)(\diamond((\diamond\langle \diamond \rangle)(\bullet\diamond)))) \\
&= \text{tag}((1 + l\langle 1 \rangle)t, (\bullet\diamond)(\diamond((\diamond\langle \diamond \rangle)(\bullet\diamond))) \\
&= \text{tag}(t, \diamond((\diamond\langle \diamond \rangle)(\bullet\diamond))) \\
&= \text{tag}(\mu X.1 + (1 + l\langle 1 \rangle)X, \diamond((\diamond\langle \diamond \rangle)(\bullet\diamond))) \\
&= \text{tag}(1 + (1 + l\langle 1 \rangle)t, \diamond((\diamond\langle \diamond \rangle)(\bullet\diamond))) \\
&= \text{tag}((1 + l\langle 1 \rangle)t, (\diamond\langle \diamond \rangle)(\bullet\diamond)) \\
&= \langle 1 \rangle_\varepsilon \langle /1 \rangle_\varepsilon \text{tag}(t, \bullet\diamond) \\
&= \langle 1 \rangle_\varepsilon \langle /1 \rangle_\varepsilon \text{tag}(\mu X.1 + (1 + l\langle 1 \rangle)X, \bullet\diamond) \\
&= \langle 1 \rangle_\varepsilon \langle /1 \rangle_\varepsilon \text{tag}(1 + (1 + l\langle 1 \rangle)t, \bullet\diamond) \\
&= \langle 1 \rangle_\varepsilon \langle /1 \rangle_\varepsilon \varepsilon \\
&= \langle 1 \rangle_\varepsilon \langle /1 \rangle_\varepsilon
\end{aligned}$$

We notice, that before  $\text{tag}$  produces anything, it unfolds the  $\mu$ -term two times.

When we translate the data to the  $\text{Split}(t)$  we have to construct xml-data that only unfolds the  $\mu$ -term once before it produces the first tree, because  $\text{Split}$  has been defined to stop before it unfolds the same  $\mu$ -term twice, in order to ensure that  $\text{Split}(t)$  always terminates.

We will in this section introduce a way to remove the redundant data that is illustrated in the previous example.

**Definition 9.2 (simplify)**

We will now define  $\text{simplify}_{\mu X.t',d'}(t, d)$  for all  $X, t', d', t$  and  $d$ , by induction on  $2 \cdot |d| + \text{ismu}(t)$ .

$$\begin{aligned}
 \text{simplify}_{\mu X.t',d'}(1, \bullet) &= d' \\
 \text{simplify}_{\mu X.t',d'}(l\langle t_1 \rangle, \langle d_1 \rangle) &= d' \\
 \text{simplify}_{\mu X.t',d'}(t_1 t_2, d_1 d_2) &= \text{if } \text{tag}(t_1, d_1) = \varepsilon \\
 &\quad \text{then } \text{simplify}_{\mu X.t',d'}(t_2, d_2) \\
 &\quad \text{else } d' \\
 \text{simplify}_{\mu X.t',d'}(t_1 + t_2, d_1 -) &= \text{simplify}_{\mu X.t',d'}(t_1, d_1) \\
 \text{simplify}_{\mu X.t',d'}(t_1 + t_2, -d_2) &= \text{simplify}_{\mu X.t',d'}(t_2, d_2) \\
 \text{simplify}_{\mu X.t',d'}(\mu Y.t_1, d) &= \text{if } \mu Y.t_1 = \mu X.t' \\
 &\quad \text{then } \text{simplify}_{\mu Y.t_1,d}(t_1[\mu Y.t_1/Y], d) \\
 &\quad \text{else } \text{simplify}_{\mu X.t',d'}(t_1[\mu Y.t_1/Y], d) \\
 \text{simplify}_{\mu X.t',d'}(-, -) &= d'
 \end{aligned}$$

**Observation 9.3 (simplify is well-defined)**

If  $\vdash \mu X.t' \equiv \mu Y.t'_\alpha$  and  $\vdash t \equiv t_\alpha$  then  $\text{simplify}_{\mu X.t',d'}(t, d) = \text{simplify}_{\mu Y.t'_\alpha,d'}(t_\alpha, d)$ .

Now we give an example to show how `simplify` works.

**Example 9.4 (An example of simplify)**

We consider  $t = \mu X.1 + (1 + l\langle 1 \rangle)X$  and  $d = \diamond((\bullet \diamond)(\diamond((\diamond \langle \diamond \rangle)(\bullet \diamond))))$ .

Example 9.1 showed how the redundancy in  $d$  made `tag` unfold the  $\mu$ -term twice before it produced a tree.

We will now see how `simplify` ensures that `tag` only unfolds the  $\mu$ -term once, because it removes the redundant part of  $d$ .

$$\begin{aligned}
 &\text{simplify}_{t,d}(1 + (1 + l\langle 1 \rangle)X[t/X], d) \\
 &= \text{simplify}_{t,d}(1 + (1 + l\langle 1 \rangle)t, \diamond((\bullet \diamond)(\diamond((\diamond \langle \diamond \rangle)(\bullet \diamond)))))) \\
 &= \text{simplify}_{t,d}((1 + l\langle 1 \rangle)t, (\bullet \diamond)(\diamond((\diamond \langle \diamond \rangle)(\bullet \diamond)))) \\
 (\text{tag}(1 + l\langle 1 \rangle, \bullet \diamond) = \varepsilon) &= \text{simplify}_{t,d}(t, \diamond((\diamond \langle \diamond \rangle)(\bullet \diamond))) \\
 &= \text{simplify}_{t,d}(\mu X.1 + (1 + l\langle 1 \rangle)X, \diamond((\diamond \langle \diamond \rangle)(\bullet \diamond))) \\
 (t = \mu X.1 + (1 + l\langle 1 \rangle)X) &= \text{simplify}_{t, \diamond((\diamond \langle \diamond \rangle)(\bullet \diamond))}(1 + (1 + l\langle 1 \rangle)t, \diamond((\diamond \langle \diamond \rangle)(\bullet \diamond))) \\
 &= \text{simplify}_{t, \diamond((\diamond \langle \diamond \rangle)(\bullet \diamond))}((1 + l\langle 1 \rangle)t, (\diamond \langle \diamond \rangle)(\bullet \diamond)) \\
 &= \text{simplify}_{t, \diamond((\diamond \langle \diamond \rangle)(\bullet \diamond))}((1 + l\langle 1 \rangle)t, (\diamond \langle \diamond \rangle)(\bullet \diamond)) \\
 (\text{tag}(1 + l\langle 1 \rangle, \diamond \langle \diamond \rangle) \neq \varepsilon) &= \diamond \langle \diamond \rangle (\bullet \diamond).
 \end{aligned}$$

Now we just need to check that the result represents the same xml-sequence as the original

xml-data.

$$\begin{aligned}
& \text{tag}(1 + (1 + l\langle 1 \rangle)X[t/X], d) \\
&= \text{tag}(1 + (1 + l\langle 1 \rangle)t, \diamond((\bullet \diamond)(\diamond((\diamond \langle \diamond \rangle)(\bullet \diamond)))))) \\
&= \text{tag}((1 + l\langle 1 \rangle)t, (\bullet \diamond)(\diamond((\diamond \langle \diamond \rangle)(\bullet \diamond)))) \\
&= \text{let } x_1 = \text{tag}(1 + l\langle t_1 \rangle, \bullet \diamond), x_2 = \text{tag}(t, \diamond((\diamond \langle \diamond \rangle)(\bullet \diamond))) \text{ in } x_1 \ x_2 \text{ end} \\
(\text{tag}(1 + l\langle 1 \rangle, \bullet \diamond) = \varepsilon) &= \text{tag}(t, \diamond((\diamond \langle \diamond \rangle)(\bullet \diamond))).
\end{aligned}$$

We will now prove that the data simplification does not change the xml-sequence represented by the xml-data.

**Lemma 9.5 (Soundness of simplify)**

$$\text{tag}(t, d) = \text{tag}(\mu X.t', d') \Rightarrow \text{tag}(\mu X.t', \text{simplify}_{\mu X.t', d'}(t, d)) = \text{tag}(t, d)$$

This is proved in Proof 19.12 on page 91.

We have now defined `simplify`, but in order to prove that the functions that use `simplify` to translate xml-data from an xml-type to the splitted list of xml-types is sound and complete, we need a version of `tag` that uses `simplify` in the same way as the translation, and keeps track of a sequence of substitutions in the same way as `Split`.

More precisely, we will need the following definition in order to express some of the lemmas in a way that allows a proof by induction.

**Definition 9.6 ( $\text{tag}_\varphi$ )**

We define  $\text{tag}_\varphi(t, d)$  where  $\varphi$  is a list of substitutions by induction on  $2 \cdot |d| + \text{ismu}(t)$ .

$$\begin{aligned}
\text{tag}_\varphi(1, \bullet) &= \varepsilon \\
\text{tag}_\varphi(l\langle t_1 \rangle, \langle d_1 \rangle) &= \text{tag}(l\langle t_1[\varphi] \rangle, \langle d_1 \rangle) \\
\text{tag}_\varphi(t_1 t_2, d_1 d_2) &= \text{if } \text{tag}(t_1[\varphi], d_1) = \varepsilon \\
&\quad \text{then } \text{tag}_\varphi(t_2, d_2) \\
&\quad \text{else let } x_1 = \text{tag}_\varphi(t_1, d_1), x_2 = \text{tag}(t_2[\varphi], d_2) \text{ in } x_1 \ x_2 \text{ end} \\
\text{tag}_\varphi(t_1 + t_2, d_1 -) &= \text{tag}_\varphi(t_1, d_1) \\
\text{tag}_\varphi(t_1 + t_2, -d_2) &= \text{tag}_\varphi(t_2, d_2) \\
\text{tag}_\varphi(\mu X.t_1, d) &= \text{tag}_{\mu X.t_1/X::\varphi}(t_1, \text{simplify}_{\mu X.t_1[\varphi], d}(t_1[\mu X.t_1/X :: \varphi], d)) \\
\text{tag}_\varphi(-, -) &= \top
\end{aligned}$$

**Observation 9.7 ( $\text{tag}_\varphi$  is well-defined)**

If  $\vdash t \equiv t'$  and  $\vdash t[\varphi] \equiv t'[\varphi']$  then  $\text{tag}_\varphi(t, d) = \text{tag}_{\varphi'}(t', d)$ .

If we generalize the observation to accept renaming of unbound variables in  $t$  but not in  $t[\varphi]$  then this observation can be proved by induction on  $2 \cdot |d| + \text{ismu}(t)$ .

We start with an example that shows how  $\text{tag}_\varphi$  works.

**Example 9.8 (Evaluation of  $\text{tag}_\varphi$ )**

We consider the case where

$$t = \mu X.1 + (1 + l\langle 1 \rangle)X \quad \text{and} \quad d = \diamond((\bullet \diamond)(\diamond((\diamond \langle \diamond \rangle)(\bullet \diamond)))).$$

We will now evaluate  $\text{tag}_{\square}(t, d)$ .

$$\begin{aligned}
& \text{tag}_{\square}(t, d) \\
&= \text{tag}_{\square}(\mu X.1 + (1 + l(1))X, \diamond((\bullet \diamond)(\diamond(\diamond \diamond))(\bullet \diamond))) \\
&= \text{tag}_{[t/X]}(1 + (1 + l(1))X, \text{simplify}_{t,d}(1 + (1 + l(1))X[ t/X :: \square ], d)) \\
&= \text{tag}_{[t/X]}(1 + (1 + l(1))X, \text{simplify}_{t,d}(1 + (1 + l(1))t, d)) \\
&= \text{tag}_{[t/X]}(1 + (1 + l(1))X, \diamond(\diamond \diamond)(\bullet \diamond)) \\
&= \text{tag}_{[t/X]}((1 + l(1))X, (\diamond \diamond)(\bullet \diamond)) \\
(\text{tag}(t_1[\varphi], d_1) \neq \varepsilon) &= \text{let } x_1 = \text{tag}_{[t/X]}(1 + l(1), \diamond \diamond), x_2 = \text{tag}(X[ [t/X] ], \bullet \diamond) \text{ in } x_1 \ x_2 \ \text{end} \\
&= \text{let } x_1 = \text{tag}_{[t/X]}(1 + l(1), \diamond \diamond), x_2 = \varepsilon \text{ in } x_1 \ x_2 \ \text{end} \\
&= \text{let } x_1 = \text{tag}_{[t/X]}(l(1), \diamond \diamond), x_2 = \varepsilon \text{ in } x_1 \ x_2 \ \text{end} \\
&= \text{let } x_1 = \text{tag}(l(1[ [t/X] ]), \diamond \diamond), x_2 = \varepsilon \text{ in } x_1 \ x_2 \ \text{end} \\
&= \text{let } x_1 = \langle 1 \rangle \varepsilon \langle /1 \rangle \varepsilon, x_2 = \varepsilon \text{ in } x_1 \ x_2 \ \text{end} \\
&= \langle 1 \rangle \varepsilon \langle /1 \rangle \varepsilon
\end{aligned}$$

This should illustrate how  $\text{tag}_{\varphi}$  uses *simplify* to avoid following the redundant xml-data, but still reaches the same xml-sequence as *tag*.

We will use the rest of this section to prove that  $\text{tag}_{\square}(t, d) = \text{tag}(t, d)$  for all  $t$  and  $d$ . In order to prove this we will need the following lemmas.

**Lemma 9.9**

$$X \notin \text{fv}'(t) \Rightarrow \text{tag}_{\mu X.t_X/X :: \varphi}(t, d) = \text{tag}_{\varphi}(t[\mu X.t_X/X], d)$$

This is proved in Proof 19.13 on page 93.

Intuitively this lemma states that if a substitution only affects the content of trees, then it does not matter if the substitution is added to the list in  $\text{tag}_{\varphi}$  or applied directly to  $t$ .

**Lemma 9.10**

$$\text{fv}'(t) \cap \text{dom}(\varphi) = \{\} \Rightarrow \text{tag}(t, d) = \varepsilon \Leftrightarrow \text{tag}(t[\varphi], d) = \varepsilon$$

This is proved in Proof 19.14 on page 95.

Intuitively this lemma states that if a substitution only affects the content of trees, then it does not change the ways the empty xml-sequence can be represented.

**Lemma 9.11**

$$\begin{aligned}
& \text{tr}_{\text{dom}(\varphi) \cup \{X\}}(t) \wedge |d| < |d'| \wedge \text{simplify}_{\mu X.t_X[\varphi], d'}(t[\mu X.t_X/X :: \varphi], d) = d' \\
& \Rightarrow \text{tag}_{\varphi}(t[\mu X.t_X/X], d) = \text{tag}_{\mu X.t_X/X :: \varphi}(t, d)
\end{aligned}$$

This is proved in Proof 19.15 on page 97.

Intuitively this lemma states that if `simplify` accepts  $d$  then it does not matter if the substitution in question is applied directly to  $t$  or added to the list in  $\text{tag}_\varphi$ .

**Lemma 9.12**

If  $d_s = \text{simplify}_{\mu X.t_X,d}(t_X[\mu X.t_X/X], d)$  then  $\text{simplify}_{\mu X.t_X,d_s}(t_X[\mu X.t_X/X], d_s) = d_s$

Intuitively this lemma states that `simplify` always accepts its on results.

If  $\text{simplify}_{\mu X.t_X,d}(t_X[\mu X.t_X/X], d) = d$  then the lemma is fulfilled.

The only places where  $\text{simplify}_{\mu X.t_X,d'}(t, d)$  can return  $d'$  is returned.

Therefore we can look at the last place where  $d'$  is changed in the recursive call that is returned.

There is only one place where  $d'$  is not reused for the recursive call that is returned, and in this case  $\text{simplify}_{\mu X.t_X,d}(t_X[\mu X.t_X/X], d)$  is returned.

Since  $d_s = d$  then we know that  $\text{simplify}_{\mu X.t_X,d_s}(t_X[\mu X.t_X/X], d_s) = d_s$ . ■

**Corollary 9.13**

$$\begin{aligned} & \text{tag}_{\square}(t_1[\mu X.t_1/X], \text{simplify}_{\mu X.t_1,d}(t_1[\mu X.t_1/X], d)) \\ &= \text{tag}_{[\mu X.t_1/X]}(t_1, \text{simplify}_{\mu X.t_1,d}(t_1[\mu X.t_1/X], d)) \end{aligned}$$

If  $d_s = \text{simplify}_{\mu X.t_1,d}(t_1[\mu X.t_1/X], d)$  then Lemma 9.12 yields that

$\text{simplify}_{\mu X.t_1,d_s}(t_1[\mu X.t_1/X], d_s) = d_s$ .

Therefore Lemma 9.11 yields that  $\text{tag}_{\square}(t_1[\mu X.t_1/X], d_s) = \text{tag}_{[\mu X.t_1/X]}(t_1, d_s)$ .

We must note that  $d_s$  is not smaller than  $d$ , but since  $\mu X.t_1$  is contractive, one iteration of the induction in Lemma 9.11 yields a case where  $d < d_s$ .

Therefore the Corollary is fulfilled. ■

We are now ready to prove that  $\text{tag}_{\square}(t, d) = \text{tag}(t, d)$ .

**Lemma 9.14** ( $\text{tag}_\varphi$  is a generalization of  $\text{tag}$ )

$$\text{tag}_{\square}(t, d) = \text{tag}(t, d)$$

This is proved in Proof 19.16 on page 100.

We have now introduced the restricted tag-function, the data-simplification function, and we have proved the lemmas we need for the data-splitting section.

## 10 Data Splitting and Unsplitting

In this section we will introduce a way to translate xml-data for an xml-type into xml-data for the splitted list of xml-types that represents the same xml-sequence.

### Definition 10.1 (Translation of data to the splitted type)

We will now define  $\text{Trans}_\varphi(t, d)$  by structural induction on  $t$ .

$$\begin{aligned}
\text{Trans}_\varphi(l\langle t_1 \rangle, d) &= \{(0, d\bullet)\} \\
\text{Trans}_\varphi(t_1 t_2, d_1 d_2) &= \text{if } \text{tag}(t_1, d_1) = \varepsilon \\
&\quad \text{then } \{(|\text{Split}_\varphi(t_1)| + n_2, d) \mid (n_2, d) \in \text{Trans}_\varphi(t_2, d_2)\} \\
&\quad \text{else } \{(n, d_{11}(d_{12}d_2)) \mid (n, d_{11}d_{12}) \in \text{Trans}_\varphi(t_1, d_1)\} \\
\text{Trans}_\varphi(t_1 + t_2, d_1 \diamond) &= \text{Trans}_\varphi(t_1, d_1) \\
\text{Trans}_\varphi(t_1 + t_2, \diamond d_2) &= \{(|\text{Split}_\varphi(t_1)| + n, d) \mid (n, d) \in \text{Trans}_\varphi(t_2, d_2)\} \\
\text{Trans}_\varphi(\mu X.t_1, d) &= \text{Trans}_{\mu X.t_1/X::\varphi}(t_1, \text{simplify}_{\mu X.t_1[\varphi], d}(t_1[\mu X.t_1/X :: \varphi], d)) \\
\text{Trans}_\varphi(-, -) &= \{\}
\end{aligned}$$

### Observation 10.2 ( $\text{Trans}_\varphi$ is well-defined)

If  $\vdash t \equiv t'$  and  $\vdash t[\varphi] \equiv t'[\varphi]$  then  $\text{Trans}_\varphi(t, d) = \text{Trans}_\varphi(t', d)$ .

If this observation is generalized to accept renaming of unbound variables in  $t$  but not in  $t[\varphi]$  then it can be proved by structural induction on  $t$ .

### Definition 10.3 (Trans)

$$\text{Trans}(t, d) = \text{Trans}_\square(t, d)$$

Next we give an example to show how this translation works.

### Example 10.4

We consider  $t = l_1\langle 1 \rangle + l_2\langle 1 \rangle$  and  $d = \diamond\langle \bullet \rangle$ .

We can see that

$$\begin{aligned}
\text{Trans}(t, d) &= \text{Trans}_\square(t, d) \\
&= \text{Trans}_\square(l_1\langle 1 \rangle + l_2\langle 1 \rangle, \diamond\langle \bullet \rangle) \\
&= \{(|\text{Split}_\square(l_1\langle 1 \rangle)| + n, d) \mid (n, d) \in \text{Trans}_\square(l_2\langle 1 \rangle, \langle \bullet \rangle)\} \\
&= \{(|\text{Split}_\square(l_1\langle 1 \rangle)| + n, d) \mid (n, d) \in \{(0, \langle \bullet \rangle \bullet)\}\} \\
&= \{(1 + n, d) \mid (n, d) \in \{(0, \langle \bullet \rangle \bullet)\}\} \\
&= \{(1, \langle \bullet \rangle \bullet)\}.
\end{aligned}$$

Now we just need to check that

$$\begin{aligned}
\text{TAG}(\text{Split}(t), (1, \langle \bullet \rangle \bullet)) &= \text{TAG}([l_1\langle 1 \rangle 1, l_2\langle 1 \rangle 1], (1, \langle \bullet \rangle \bullet)) \\
&= \text{TAG}([l_2\langle 1 \rangle 1], (0, \langle \bullet \rangle \bullet)) \\
&= \text{tag}(l_2\langle 1 \rangle 1, \langle \bullet \rangle \bullet) \\
&= \langle l_2 \rangle \varepsilon \langle /l_2 \rangle \varepsilon \\
&= \text{tag}(t, d).
\end{aligned}$$

We can see by the definition of  $\text{Trans}_\varphi$  that

$$(n', d') \in \text{Trans}_\varphi(t, d) \Rightarrow n' < |\text{Split}_\varphi(t)|.$$

We will now prove that the transformation of xml-data in  $\text{Trans}$  is sound and complete.

**Lemma 10.5 (Soundness of Trans)**

$$\forall D \in \text{Trans}_\varphi(t, d). \text{tag}_\varphi(t, d) = \text{TAG}(\text{Split}_\varphi(t), D)$$

*This is proved in Proof 19.17 on page 102.*

**Corollary 10.6 (Soundness of Trans)**

$$\forall D \in \text{Trans}(t, d). \text{tag}(t, d) = \text{TAG}(\text{Split}(t), D)$$

*This follows from the above lemma, since*

$$\begin{aligned} & \text{tag}(t, d) \\ (\text{Lemma 9.14}) &= \text{tag}_\square(t, d) \\ (\text{Lemma 10.5}) &= \text{TAG}(\text{Split}_\square(t), D) \\ &= \text{TAG}(\text{Split}(t), D). \end{aligned}$$

■

**Lemma 10.7**

$$\text{tag}_\varphi(t, d) = \varepsilon \Rightarrow \text{tag}(t[\varphi], d) = \varepsilon$$

*This is proved in Proof 19.18 on page 104.*

**Lemma 10.8 (Completeness of Trans)**

$$\text{tr}_{\text{dom}(\varphi)}(t) \Rightarrow \text{Trans}_\varphi(t, d) = \{\} \Rightarrow \text{tag}_\varphi(t, d) \in \{\varepsilon, \top\}.$$

*This is proved in Proof 19.19 on page 106.*

**Corollary 10.9 (Completeness of Trans)**

$$\text{tag}(t, d) \notin \{\varepsilon, \top\} \Rightarrow \text{Trans}(t, d) \neq \{\}$$

*Assume that  $\text{tag}(t, d) \notin \{\varepsilon, \top\}$ .*

*Now Lemma 9.14 yields that  $\text{tag}_\square(t, d) \notin \{\varepsilon, \top\}$ .*

*Therefore Lemma 10.8 yields that  $\text{Trans}_\square(t, d) \neq \{\}$  and therefore the corollary is fulfilled. ■*

Now we have defined a way to translate xml-data from an xml-type  $t$  to  $\text{Split}(t)$ , and have proved it to be sound and complete.

Next we will introduce a way to translate xml-data from  $\text{Split}(t)$  to the original xml-type  $t$ .

**Definition 10.10 (Translation of data from the splitted type)**

We will now define  $\text{TransInv}_\varphi(t, D)$  where  $\varphi$  is a list of substitutions by structural induction on  $t$ .

$$\begin{aligned}
 \text{TransInv}_\varphi(l\langle t_1 \rangle, (0, \langle d_1 \rangle \bullet)) &= \{\langle d_1 \rangle\} \\
 \text{TransInv}_\varphi(t_1 t_2, (|\text{Split}_\varphi(t_1)| + n, d)) &= \{d_1 d_2 \mid d_1 \in \text{esp}(t_1) \wedge d_2 \in \text{TransInv}_\varphi(t_2, (n, d))\} \\
 \text{TransInv}_\varphi(t_1 t_2, (n, d_{11}(d_{12} d_2))) &= \{d_1 d_2 \mid d_1 \in \text{TransInv}_\varphi(t_1, (n, d_{11} d_{12}))\} \\
 \text{TransInv}_\varphi(t_1 + t_2, (|\text{Split}_\varphi(t_1)| + n, d)) &= \{\diamond d_2 \mid d_2 \in \text{TransInv}_\varphi(t_2, (n, d))\} \\
 \text{TransInv}_\varphi(t_1 + t_2, D) &= \{d_1 \diamond \mid d_1 \in \text{TransInv}_\varphi(t_1, D)\} \\
 \text{TransInv}_\varphi(\mu X.t_1, D) &= \text{TransInv}_{\mu X.t_X/X::\varphi}(t_1, D) \\
 \text{TransInv}_\varphi(-, -) &= \{\}
 \end{aligned}$$

**Observation 10.11 (TransInv is well-defined)**

If  $\vdash t \equiv t'$  and  $\vdash t_1[\varphi] \equiv t'[\varphi]$  then  $\text{TransInv}_\varphi(t, D) = \text{TransInv}_{\varphi'}(t', D)$ .

If we generalize this observation to allow renaming of unbound variables in  $t$  but not in  $t[\varphi]$  then it can be proved by structural induction on  $t$ .

We will start with an example to show how TransInv works.

**Example 10.12**

We consider  $t = l_1\langle 1 \rangle + l_2\langle 1 \rangle$  and  $D = (1, \langle \bullet \rangle \bullet)$ .

We will now evaluate

$$\begin{aligned}
 \text{TransInv}(t, D) &= \text{TransInv}_{\square}(l_1\langle 1 \rangle + l_2\langle 1 \rangle, (1, \langle \bullet \rangle \bullet)) \\
 &= \text{TransInv}_{\square}(l_1\langle 1 \rangle + l_2\langle 1 \rangle, (0 + |\text{Split}_{\square}(l_1\langle 1 \rangle)|, \langle \bullet \rangle \bullet)) \\
 &= \{\diamond d \mid d \in \text{TransInv}(l_2\langle 1 \rangle, (0, \langle \bullet \rangle \bullet))\} \\
 &= \{\diamond d \mid d \in \{\langle \bullet \rangle\}\} \\
 &= \{\diamond \langle \bullet \rangle\}.
 \end{aligned}$$

We will now prove that this translation of xml-data is sound and complete.

**Lemma 10.13 (Soundness of TransInv)**

$$\forall d \in \text{TransInv}_\varphi(t, D). \text{tag}(t[\varphi], d) = \text{TAG}(\text{Split}_\varphi(t), D).$$

This is proved in Proof 19.20 on page 108.

**Corollary 10.14 (Soundness of TransInv)**

$$\forall d \in \text{TransInv}(t, D). \text{tag}(t, d) = \text{TAG}(\text{Split}(t), D)$$

If  $d \in \text{TransInv}(t, D) = \text{TransInv}_{\square}(t, D)$  then we get that

$$\begin{aligned}
 &\text{TAG}(\text{Split}(t), D) \\
 &= \text{TAG}(\text{Split}_{\square}(t), D) \\
 \text{(Lemma 10.13)} &= \text{tag}(t[\square], d) \\
 &= \text{tag}(t, d).
 \end{aligned}$$

Therefore this corollary is fulfilled. ■

**Lemma 10.15 (Completeness of TransInv)**

$$\text{TAG}(\text{Split}_\varphi(t), D) \neq \top \Rightarrow \text{TransInv}_\varphi(t, D) \neq \{\}$$

This is proved in Proof 19.21 on page 110.

**Corollary 10.16 (Completeness of TransInv)**

$$\text{TAG}(\text{Split}(t), D) \neq \top \Rightarrow \text{TransInv}(t, D) \neq \{\}$$

If  $\top \neq \text{TAG}(\text{Split}(t), D) = \text{TAG}(\text{Split}_\square(t), D)$  then Lemma 10.15 yields that  $\{\} \neq \text{TransInv}_\square(t, D) = \text{TransInv}(t, D)$ .

Therefore this corollary is fulfilled. ■

We have now introduced **Trans** and **TransInv**, and proved that they are sound and complete translations between an xml-type and its splitted list of xml-types.

Therefore we are now ready to generalize them to work on lists of xml-types.

**Definition 10.17**

$$\begin{aligned} \text{TRANS}(\square, \_) &= \{\} \\ \text{TRANS}(t_1 :: T_1, (0, d)) &= \text{Trans}(t_1, d) \\ \text{TRANS}(t_1 :: T_1, (n + 1, d)) &= \{(n' + |\text{Split}(t_1)|, d') \mid (n', d') \in \text{TRANS}(T_1, (n, d))\} \end{aligned}$$

We have now defined **TRANS** and we will now prove that it is sound and complete.

**Lemma 10.18 (Soundness of TRANS)**

$$\forall D' \in \text{TRANS}(T, D). \text{TAG}(T, D) = \text{TAG}(\text{SPLIT}(T), D')$$

This is proved in Proof 19.22 on page 112.

**Lemma 10.19 (Completeness of TRANS)**

$$\text{TAG}(T, D) \notin \{\varepsilon, \top\} \Rightarrow \text{TRANS}(T, D) \neq \{\}$$

This is proved in Proof 19.23 on page 113.

We have now proved that **TRANS** is sound and complete, and we will now generalize **TransInv** to work on lists of xml-types.

**Definition 10.20**

$$\begin{aligned} \text{TRANSINV}(\square, \_) &= \{\} \\ \text{TRANSINV}(t_1 :: T_1, (n + |\text{Split}(t_1)|, d)) &= \{(n' + 1, d') \mid (n', d') \in \text{TRANSINV}(T_1, (n, d))\} \\ \text{TRANSINV}(t_1 :: T_1, (n, d)) &= \{(0, d') \mid d' \in \text{TransInv}(t_1, (n, d))\} \end{aligned}$$

We have now defined **TRANSINV**, and we will now prove that it is sound and complete.

**Lemma 10.21 (Soundness of TRANSINV)**

$$\text{TAG}(\text{SPLIT}(T), D') \neq \top \Rightarrow \forall D \in \text{TRANSINV}(T, D'). \text{TAG}(T, D) = \text{TAG}(\text{SPLIT}(T), D')$$

This is proved in Proof 19.24 on page 114.

The normal way to define completeness for TRANSINV would be that  $\text{TAG}(\text{SPLIT}(T), D') \neq \top \Rightarrow \text{TRANSINV}(T, D') \neq \{\}$ , but we will need a stronger completeness in the section about coercions, and we will now prove this stronger completeness.

The reason for this is that we need to identify what index will be used for the result xml-data, based in the index of the given xml-data.

This makes the lemma more complex to express, which means that the induction hypothesis is a bit complicated.

The main complication is however the induction step, which have to keep track of all the indices when the induction hypothesis is used.

**Lemma 10.22 (Completeness of TRANSINV)**

$$\begin{aligned} \forall T = [t_1, t_2, \dots, t_k] \in \mathbf{XMLTypes}. \forall D' = (n', d') \in \mathbf{XMLDatas}. \forall j \in \{1, 2, \dots, k\}. \\ \text{TAG}(\text{SPLIT}(T), D') \neq \top \wedge \sum_{i=1}^{j-1} |\text{Split}(t_i)| \leq n' < \sum_{i=1}^j |\text{Split}(t_i)| \Rightarrow \\ \{(n, d) \in \text{TRANSINV}(T, D') \mid n = j\} \neq \{\} \end{aligned}$$

This is proved in Proof 19.25 on page 115.

We have now defined TRANS and TRANSINV and proved that they are sound and complete.

We have now actually proved in a constructive way that SPLIT preserves the semantics except for the empty xml-sequence, and this is explained in the following corollary.

**Corollary 10.23 (Correctness of SPLIT)**

$$\text{IN}[\text{SPLIT}(T)] = \text{IN}[T] \setminus \{\varepsilon\}$$

We will prove each inclusion separately.

If  $x \in \text{IN}[T] \setminus \{\varepsilon\}$  then Corollary 6.10 yields that there is a  $D$  such that  $\text{TAG}(T, D) = x$ .

Lemma 10.19 yields that there is a  $D' \in \text{TRANS}(T, D')$  and Lemma 10.18 yields that  $\text{TAG}(\text{SPLIT}(T), D') = \text{TAG}(T, D) = x$ .

Therefore Corollary 6.10 yields that  $x \in \text{IN}[\text{SPLIT}(T)]$ .

If  $x \in \text{IN}[\text{SPLIT}(T)]$  then Corollary 6.10 yields that there is a  $D'$  such that  $\text{TAG}(\text{SPLIT}(T), D') = x$ .

Lemma 10.22 yields that there is a  $D \in \text{TRANSINV}(T, D')$  and Lemma 10.21 yields that  $\text{TAG}(T, D) = \text{TAG}(\text{SPLIT}(T), D') = x$ .

Therefore Corollary 6.10 yields that  $x \in \text{IN}[T]$  and since  $\varepsilon \notin \text{IN}[\text{SPLIT}(T)]$  we get that  $x \in \text{IN}[T] \setminus \{\varepsilon\}$ .

We have now proved both inclusions, and we can therefore conclude that

$$\text{IN}[\text{SPLIT}(T)] = \text{IN}[T] \setminus \{\varepsilon\}.$$

■

We have now defined **TRANS** and **TRANSINV** and proved in a constructive way that **SPLIT** preserves the semantics except for the empty sequence.

## 11 Derivatives

The aim of this section is to define derivatives based on the `Split` function defined in the previous section, and to prove that every xml-type only has a finite number of derivatives. We will end this section by generalizing this result to derivatives of lists of xml-types.

We start by defining the set of derivatives for an xml-type.

**Definition 11.1 (Derivatives of an XMLType)**

$$\begin{aligned} \text{Deriv}(S) &= \bigcup_{t \in S} \{t_j \mid l\langle t_1 \rangle t_2 \in \overline{\text{Split}(t)} \wedge j \in \{1, 2\}\} \\ \text{Derivs}(t) &= \bigcup_{i=0}^{\infty} \text{Deriv}^i(\{t\}) \end{aligned}$$

We will start by evaluating `Derivs` in an example.

**Example 11.2**

We will now consider

$$t = \mu X.1 + l\langle X \rangle X.$$

Remember that we calculated  $\text{Split}(t) = [l\langle t \rangle(1t)]$  in Example 8.5.

We will now calculate  $\text{Derivs}(t)$ .

$$\begin{aligned}
\text{Derivs}(t) &= \bigcup_{i=0}^{\infty} \text{Deriv}^i(\{t\}) \\
&= \{t\} \cup \bigcup_{i=1}^{\infty} \text{Deriv}^i(\{t\}) \\
&= \{t\} \cup \bigcup_{i=0}^{\infty} \text{Deriv}^i(\{t_j \mid l'\langle t_1 \rangle t_2 \in \overline{\text{Split}(t)} \wedge j \in \{1, 2\}\}) \\
&= \{t\} \cup \bigcup_{i=0}^{\infty} \text{Deriv}^i(\{t_j \mid l'\langle t_1 \rangle t_2 \in \overline{[l\langle t \rangle(1t)]} \wedge j \in \{1, 2\}\}) \\
&= \{t\} \cup \bigcup_{i=0}^{\infty} \text{Deriv}^i(\{t, 1t\}) \\
&= \{t, 1t\} \cup \bigcup_{i=1}^{\infty} \text{Deriv}^i(\{t, 1t\}) \\
&= \{t, 1t\} \cup \bigcup_{i=0}^{\infty} \text{Deriv}^i(\{t_j \mid l'\langle t_1 \rangle t_2 \in \overline{\text{Split}(t)} \wedge j \in \{1, 2\}\} \\
&\quad \cup \{t_j \mid l'\langle t_1 \rangle t_2 \in \overline{\text{Split}(1t)} \wedge j \in \{1, 2\}\}) \\
&= \{t, 1t\} \cup \bigcup_{i=0}^{\infty} \text{Deriv}^i(\{t, 1t\} \cup \{t_j \mid l'\langle t_1 \rangle t_2 \in \overline{\text{Split}(1t)} \wedge j \in \{1, 2\}\}) \\
&= \{t, 1t\} \cup \bigcup_{i=0}^{\infty} \text{Deriv}^i(\{t, 1t\} \cup \{t_j \mid l'\langle t_1 \rangle t_2 \in \overline{\text{Split}(t)} \wedge j \in \{1, 2\}\}) \\
&= \{t, 1t\} \cup \bigcup_{i=0}^{\infty} \text{Deriv}^i(\{t, 1t\} \cup \{t, 1t\}) \\
&= \{t, 1t\} \cup \bigcup_{i=0}^{\infty} \text{Deriv}^i(\{t, 1t\}) \\
&= \{t, 1t\} \cup \bigcup_{i=1}^{\infty} \text{Deriv}^i(\{t, 1t\}) \\
&= \{t, 1t\}
\end{aligned}$$

The final equality is true, since  $\text{Deriv}(\{t, 1t\}) = \{t, 1t\}$  and therefore for all  $i \in \mathbb{N}_0$  we have that  $\text{Deriv}^i(\{t, 1t\}) = \{t, 1t\}$ .

Now we will prove that this set always will be finite, but in order to do so, we need to prove some lemmas which defines upper limits to types of the forms  $\mu X.t_1$  and  $t_1 t_2$ .

**Lemma 11.3 (Substitution Lemma for esp)**

$$X \notin \text{fv}'(t) \Rightarrow \text{esp}(t) = \text{esp}(t[t'/X])$$

This is proved in Proof 19.26 on page 117.

Intuitively this lemma states that if a substitution only affects the content of trees, then it does not affect the result of *esp*.

**Lemma 11.4 (Substitution Lemma for Split)**

$$X \notin \text{fv}'(t) \Rightarrow \text{Split}_\varphi(t[t_X/X]) = [t'[t_X/X :: \varphi] \mid t' \in \text{Split}(t)]$$

This is proved in Proof 19.27 on page 118.

Intuitively this lemma states that if a substitution only affects the content of trees then it does not matter if the substitution is applied before or after the *xml*-type has been splitted.

**Lemma 11.5 (Substitution Lemma for Deriv)**

$$\begin{aligned} & \forall t', X, t_X, \varphi. \text{tr}_{\{X\}}(t') \wedge \text{tr}_{\{X\}}(t_X) \Rightarrow \\ & \{t_j \mid l(t_1)t_2 \in \overline{\text{Split}_\varphi(t'[\mu X.t_X/X])} \wedge j \in \{1, 2\}\} \\ & \subseteq \{t'_j[\mu X.t_X/X :: \varphi] \mid l(t'_1)t'_2 \in \overline{\text{Split}(t')} \wedge j \in \{1, 2\}\} \cup \{t''[\mu X.t_X/X :: \varphi] \mid t'' \in \text{Derivs}(t_X)\} \end{aligned}$$

This is proved in Proof 19.28 on page 121.

Intuitively this lemma states that if  $t'$  is tail-recursive with respect to  $X$ , then derivatives of  $t'[t_X/X]$  are either a derivative of  $t'$  with  $t_X$  substituted for  $X$ , or a derivative of  $t_X$ .

**Lemma 11.6 (Deriv boundary for  $\mu X.t$ )**

$$\begin{aligned} \forall \mu X.t_X, S & \subseteq \{t'[\mu X.t_X/X] \mid t' \in \text{Derivs}(t_X)\} \cup \{\mu X.t_X\}. \\ \text{Deriv}(S) & \subseteq \{t'[\mu X.t_X/X] \mid t' \in \text{Derivs}(t_X)\}. \end{aligned}$$

This is proved in Proof 19.29 on page 126.

We are now ready to prove the upper limit for terms of the form  $\mu X.t$ .

**Lemma 11.7 (Derivs limit for  $\mu X.t$ )**

$$\forall \mu X.t \in \mathbf{XMLType}. \text{Derivs}(\mu X.t) \subseteq \{t'[\mu X.t/X] \mid t' \in \text{Derivs}(t)\} \cup \{\mu X.t\}$$

Let  $\mu X.t \in \mathbf{XMLType}$  be chosen.

Since  $\text{Derivs}(\mu X.t) = \bigcup_{i=0}^{\infty} \text{Deriv}^i(\{\mu X.t\})$  we only need to prove that  $\forall i \in \mathbb{N}. \text{Deriv}^i(\{\mu X.t\}) \subseteq \{t'[\mu X.t/X] \mid t' \in \text{Derivs}(t)\} \cup \{\mu X.t\}$ .

We prove this by induction on  $i$ .

Start:  $i = 0$

In this case  $\text{Deriv}^0(\{\mu X.t\}) = \{\mu X.t\} \subseteq \{t'[\mu X.t/X] \mid t' \in \text{Derivs}(t)\} \cup \{\mu X.t\}$ .

Therefore the lemma is fulfilled in this case.

Induction Hypothesis:

In the case where  $i > 0$  we can assume that  $\text{Deriv}^{i-1}(\{\mu X.t\}) \subseteq \{t'[\mu X.t/X] \mid t' \in \text{Derivs}(t)\} \cup \{\mu X.t\}$ .

Step:  $i > 0$

In this case  $\text{Deriv}^i(\{\mu X.t\}) = \text{Deriv}(\text{Deriv}^{i-1}(\{\mu X.t\}))$ .

Now the induction hypothesis yields that

$\text{Deriv}^{i-1}(\{t_1 t_2\}) \subseteq \{t'[\mu X.t/X] \mid t' \in \text{Derivs}(t)\} \cup \{\mu X.t\}$ , so

Lemma 11.6 yields that  $\text{Deriv}(\text{Deriv}^{i-1}(\{\mu X.t\})) \subseteq \{t'[\mu X.t/X] \mid t' \in \text{Derivs}(t)\}$ , which proves this case.

Now we can conclude that  $\forall i \in \mathbb{N}. \text{Deriv}^i(\{\mu X.t\}) \subseteq \{t'[\mu X.t/X] \mid t' \in \text{Derivs}(t)\} \cup \{\mu X.t\}$ , and therefore

$\text{Derivs}(\mu X.t) = \bigcup_{i=0}^{\infty} \text{Deriv}^i(\{\mu X.t\}) \subseteq \{t'[\mu X.t/X] \mid t' \in \text{Derivs}(t)\} \cup \{\mu X.t\}$ . ■

**Lemma 11.8** (Deriv boundary for  $t_1 t_2$ )

$$\begin{aligned} \forall t_1, t_2 \in \mathbf{XMLType}. \forall S &\subseteq \{t' t_2 \mid t' \in \text{Derivs}(t_1)\} \cup \text{Derivs}(t_1) \cup \text{Derivs}(t_2) \cup \{t_1 t_2\}. \\ \text{Deriv}(S) &\subseteq \{t' t_2 \mid t' \in \text{Derivs}(t_1)\} \cup \text{Derivs}(t_1) \cup \text{Derivs}(t_2) \cup \{t_1 t_2\}. \end{aligned}$$

This is proved in Proof 19.30 on page 127.

We are now ready to prove the upper limit for terms of the form  $t_1 t_2$ .

**Lemma 11.9** (Derivs limit for  $t_1 t_2$ )

$$\forall t_1, t_2 \in \mathbf{XMLType}. \text{Derivs}(t_1 t_2) \subseteq \{t' t_2 \mid t' \in \text{Derivs}(t_1)\} \cup \text{Derivs}(t_1) \cup \text{Derivs}(t_2) \cup \{t_1 t_2\}$$

Let  $t_1, t_2 \in \mathbf{XMLType}$  be chosen.

Since  $\text{Derivs}(t_1 t_2) = \bigcup_{i=0}^{\infty} \text{Deriv}^i(\{t_1 t_2\})$  we only need to prove that

$$\forall i \in \mathbb{N}. \text{Deriv}^i(\{t_1 t_2\}) \subseteq \{t' t_2 \mid t' \in \text{Derivs}(t_1)\} \cup \text{Derivs}(t_1) \cup \text{Derivs}(t_2) \cup \{t_1 t_2\}.$$

We prove this by induction on  $i$ .

Start:  $i = 0$

In this case

$$\text{Deriv}^0(\{t_1 t_2\}) = \{t_1 t_2\} \subseteq \{t' t_2 \mid t' \in \text{Derivs}(t_1)\} \cup \text{Derivs}(t_1) \cup \text{Derivs}(t_2) \cup \{t_1 t_2\}.$$

Therefore the lemma is fulfilled in this case.

Induction Hypothesis:

In the case where  $i > 0$  we can assume that

$$\text{Deriv}^{i-1}(\{t_1 t_2\}) \subseteq \{t' t_2 \mid t' \in \text{Derivs}(t_1)\} \cup \text{Derivs}(t_1) \cup \text{Derivs}(t_2) \cup \{t_1 t_2\}.$$

Step:  $i > 0$

In this case  $\text{Deriv}^i(\{t_1 t_2\}) = \text{Deriv}(\text{Deriv}^{i-1}(\{t_1 t_2\}))$ .

Now the induction hypothesis yields that

$\text{Deriv}^{i-1}(\{t_1 t_2\}) \subseteq \{t' t_2 \mid t' \in \text{Derivs}(t_1)\} \cup \text{Derivs}(t_1) \cup \text{Derivs}(t_2) \cup \{t_1 t_2\}$ , so

Lemma 11.8 yields that

$\text{Deriv}(\text{Deriv}^{i-1}(\{t_1 t_2\})) \subseteq \{t' t_2 \mid t' \in \text{Derivs}(t_1)\} \cup \text{Derivs}(t_1) \cup \text{Derivs}(t_2) \cup \{t_1 t_2\}$ , which proves this case.

Now we can conclude that

$\forall i \in \mathbb{N}. \text{Deriv}^i(\{t_1 t_2\}) \subseteq \{t' t_2 \mid t' \in \text{Derivs}(t_1)\} \cup \text{Derivs}(t_1) \cup \text{Derivs}(t_2) \cup \{t_1 t_2\}$ , and therefore

$\text{Derivs}(t_1 t_2) = \bigcup_{i=0}^{\infty} \text{Deriv}^i(\{t_1 t_2\}) \subseteq \{t' t_2 \mid t' \in \text{Derivs}(t_1)\} \cup \text{Derivs}(t_1) \cup \text{Derivs}(t_2) \cup \{t_1 t_2\}$ . ■

We have now proved the upper limits for terms of the forms  $\mu X.t$  and  $t_1 t_2$ .

We are therefore ready to prove that the set of derivatives always is finite.

**Theorem 11.10 (Only finitely many derivatives)**

$\forall t \in \text{XMLType}. \text{Derivs}(t) \text{ is a finite set.}$

We prove this by structural induction on  $t$ .

*Induction Hypothesis:*

We can assume that if  $t'$  is a sub-term of  $t$  then  $\text{Derivs}(t')$  is finite.

Case:  $t = 0$

Since  $\overline{\text{Split}(0)} = \{\}$ , we get that  $\text{Derivs}(0) = \bigcup_{i=0}^{\infty} \text{Deriv}^i(\{0\}) = \{0\} \cup \bigcup_{i=0}^{\infty} \{\} = \{0\}$ .

So  $\text{Derivs}(t) = \{0\}$  is finite.

Case:  $t = 1$

Since  $\overline{\text{Split}(1)} = \{\}$ , we get that  $\text{Derivs}(1) = \bigcup_{i=0}^{\infty} \text{Deriv}^i(\{1\}) = \{1\} \cup \bigcup_{i=0}^{\infty} \{\} = \{1\}$ .

So  $\text{Derivs}(t) = \{1\}$  is finite.

Case:  $t = X$

Since  $\overline{\text{Split}(X)} = \{\}$ , we get that  $\text{Derivs}(X) = \bigcup_{i=0}^{\infty} \text{Deriv}^i(\{X\}) = \{X\} \cup \bigcup_{i=0}^{\infty} \{\} = \{X\}$ .

So  $\text{Derivs}(t) = \{X\}$  is finite.

Case:  $t = l\langle t_1 \rangle$

Since  $\overline{\text{Split}(l\langle t_1 \rangle)} = \{l\langle t_1 \rangle 1\}$  we get that

$$\begin{aligned}
\text{Derivs}(l\langle t_1 \rangle) &= \bigcup_{i=0}^{\infty} \text{Deriv}^i(\{l\langle t_1 \rangle\}) \\
&= \{l\langle t_1 \rangle\} \cup \bigcup_{i=1}^{\infty} \text{Deriv}^i(\{l\langle t_1 \rangle\}) \\
&= \{l\langle t_1 \rangle\} \cup \bigcup_{i=0}^{\infty} \text{Deriv}^i(\{t'_j \mid l'\langle t'_1 \rangle t'_2 \in \{l\langle t_1 \rangle 1\}\}) \\
&= \{l\langle t_1 \rangle\} \cup \bigcup_{i=0}^{\infty} \text{Deriv}^i(\{t_1, 1\}) \\
&= \{l\langle t_1 \rangle\} \cup \bigcup_{i=0}^{\infty} \text{Deriv}^i(\{t_1\}) \cup \bigcup_{i=0}^{\infty} \text{Deriv}^i(\{1\}) \\
&= \{l\langle t_1 \rangle\} \cup \bigcup_{i=0}^{\infty} \text{Deriv}^i(\{t_1\}) \cup \{1\} \\
&= \{l\langle t_1 \rangle\} \cup \{1\} \cup \text{Derivs}(t_1)
\end{aligned}$$

Now the induction hypothesis yields that  $\text{Derivs}(t_1)$  is finite and therefore  $\text{Derivs}(t) = \{l\langle t_1 \rangle\} \cup \{1\} \cup \text{Derivs}(t_1)$  is finite.

Case:  $t = t_1 t_2$

Lemma 11.9 yields that

$$\text{Derivs}(t_1 t_2) \subseteq \{t' t_2 \mid t' \in \text{Derivs}(t_1)\} \cup \text{Derivs}(t_1) \cup \text{Derivs}(t_2) \cup \{t_1 t_2\}.$$

The induction hypothesis yields that  $\text{Derivs}(t_1)$  is finite, and therefore

$\{t' t_2 \mid t' \in \text{Derivs}(t_1)\}$  is also finite.

The induction hypothesis also yields that  $\text{Derivs}(t_2)$  is finite, and therefore we can conclude that  $\text{Derivs}(t_1 t_2) \subseteq \{t' t_2 \mid t' \in \text{Derivs}(t_1)\} \cup \text{Derivs}(t_1) \cup \text{Derivs}(t_2) \cup \{t_1 t_2\}$  is finite.

Case:  $t = t_1 + t_2$

Since  $\overline{\text{Split}(t_1 + t_2)} = \overline{\text{Split}(t_1)} \cup \overline{\text{Split}(t_2)}$  we get that

$$\begin{aligned}
\text{Derivs}(t_1 + t_2) &= \bigcup_{i=0}^{\infty} \text{Deriv}^i(\{t_1 + t_2\}) \\
&= \{t_1 + t_2\} \cup \bigcup_{i=1}^{\infty} \text{Deriv}^i(\{t_1 + t_2\}) \\
&= \{t_1 + t_2\} \cup \bigcup_{i=0}^{\infty} \text{Deriv}^i(\{t'_j \mid l'(t'_1)t'_2 \in \overline{\text{Split}(t_1 + t_2)} \wedge j \in \{1, 2\}\}) \\
&= \{t_1 + t_2\} \cup \bigcup_{i=0}^{\infty} \text{Deriv}^i(\{t'_j \mid l'(t'_1)t'_2 \in \overline{\text{Split}(t_1)} @ \overline{\text{Split}(t_2)} \wedge j \in \{1, 2\}\}) \\
&= \{t_1 + t_2\} \cup \bigcup_{i=0}^{\infty} \text{Deriv}^i(\{t'_j \mid l'(t'_1)t'_2 \in \overline{\text{Split}(t_1)} \cup \overline{\text{Split}(t_2)} \wedge j \in \{1, 2\}\}) \\
&= \{t_1 + t_2\} \cup \bigcup_{i=0}^{\infty} \text{Deriv}^i(\{t'_j \mid l'(t'_1)t'_2 \in \overline{\text{Split}(t_1)} \wedge j \in \{1, 2\}\}) \\
&\quad \cup \bigcup_{i=0}^{\infty} \text{Deriv}^i(\{t'_j \mid l'(t'_1)t'_2 \in \overline{\text{Split}(t_2)} \wedge j \in \{1, 2\}\}) \\
&= \{t_1 + t_2\} \cup \bigcup_{i=0}^{\infty} \text{Deriv}^i(\text{Deriv}(\{t_1\})) \cup \bigcup_{i=0}^{\infty} \text{Deriv}^i(\text{Deriv}(\{t_2\})) \\
&\subseteq \{t_1 + t_2\} \cup \text{Derivs}(t_1) \cup \text{Derivs}(t_2)
\end{aligned}$$

Now the induction hypothesis yields that  $\text{Derivs}(t_1)$  and  $\text{Derivs}(t_2)$  are finite, and therefore  $\text{Derivs}(t_1 + t_2)$  is finite.

Case:  $t = \mu X.t_1$

Lemma 11.7 yields that  $\text{Derivs}(\mu X.t_1) \subseteq \{t'[\mu X.t_1/X] \mid t' \in \text{Derivs}(t_1)\} \cup \{\mu X.t_1\}$ .

The induction hypothesis yields that  $\text{Derivs}(t_1)$  is finite, and therefore

$\{t'[\mu X.t_1/X] \mid t' \in \text{Derivs}(t_1)\}$  is also finite.

Therefore we can conclude that  $\text{Derivs}(t) = \text{Derivs}(\mu X.t_1) \subseteq \{t'[\mu X.t_1/X] \mid t' \in \text{Derivs}(t_1)\} \cup \{\mu X.t_1\}$  is finite.

This covers all the cases, and we can therefore conclude that

$$\forall t \in \text{XMLType}. \text{Derivs}(t) \text{ is a finite set.}$$

■

Now we are ready to generalize the notion of derivatives to lists of xml-types.

**Definition 11.11 (Derivatives of XMLTypes)**

$$\begin{aligned}
\text{DERIVS}(\square) &= \{\} \\
\text{DERIVS}(t_1 :: T_1) &= \text{Derivs}(t_1) \cup \text{DERIVS}(T_1)
\end{aligned}$$

We will now prove that the set of derivatives of lists of xml-types always will be finite.

**Corollary 11.12 (Only finitely many derivatives)**

$\forall T \in \mathbf{XMLTypes}. \mathbf{DERIVS}(T)$  is a finite set.

We prove this by induction on the length of  $T$ .

Start:  $T = []$

If this case  $\mathbf{DERIVS}(T_1) = \mathbf{DERIVS}([]) = \{\}$  which is a finite set.

Induction Hypothesis:

If  $T = t_1 :: T_1$  then we can assume that  $\mathbf{DERIVS}(T_1)$  is finite.

Step:  $T = t_1 :: T_1$

The induction hypothesis yields that  $\mathbf{DERIVS}(T_1)$  is finite.

Theorem 11.10 yields that  $\mathbf{Derivs}(t_1)$  is finite, and therefore

$\mathbf{DERIVS}(T) = \mathbf{DERIVS}(t_1 :: T_1) = \mathbf{Derivs}(t_1) \cup \mathbf{DERIVS}(T_1)$  is the union of two finite sets, which is finite. ■

We have now defined derivatives for both xml-types and lists of xml-types, and proved that the set of derivatives for any xml-type or list of xml-types is finite.

## 12 Simulations

In this section we will define simulations, and prove that  $\models T_1 < T_2$  if and only if there is a finite simulation that relates  $T_1$  and  $T_2$ .

We start by defining the requirements of a simulation.

### Definition 12.1

A set of pairs of lists of xml-types  $\mathcal{R}$  is a simulation if and only if

$$\begin{aligned} T_1 \mathcal{R} T_2 \Rightarrow & (\text{ESP}(T_1) \neq \{\} \Rightarrow \text{ESP}(T_2) \neq \{\}) \quad \wedge \\ & \forall l \langle t_{11} \rangle t_{12} \in \overline{\text{SPLIT}(T_1)}. \forall S \subseteq \overline{\text{SPLIT}(T_2)}. \\ & (\exists T'_2. [t_{11}] \mathcal{R} T'_2 \wedge \overline{T'_2} = \{t_{21} \mid l \langle t_{21} \rangle t_{22} \in S\}) \quad \vee \\ & (\exists T'_2. [t_{12}] \mathcal{R} T'_2 \wedge \overline{T'_2} = \{t_{22} \mid l \langle t_{21} \rangle t_{22} \in \overline{\text{SPLIT}(T_2)} \setminus S\}) \end{aligned}$$

Before we prove soundness and completeness of finite simulations, we will give an example of a simulation to show how the elements in a simulation interact.

### Example 12.2 (A simple simulation)

In this example we will consider

$$t_1 = \mu X. 1 + l_1 \langle 1 \rangle (l_2 \langle 1 \rangle X) \quad \text{and} \quad t_2 = \mu X. 1 + (l_1 \langle 1 \rangle X + l_2 \langle 1 \rangle X).$$

We will now show that the set

$$R = \{([t_1], [t_2]), ([1(l_2 \langle 1 \rangle t_1)], [1t_2]), ([1t_1], [1t_2]), ([1], [1])\}$$

is a simulation.

We consider the pair  $([t_1], [t_2])$  in  $R$ .

We find that

$$\text{SPLIT}([t_1]) = [l_1 \langle 1 \rangle (l_2 \langle 1 \rangle t_1)] \quad \text{and}$$

$$\text{SPLIT}([t_2]) = [l_1 \langle 1 \rangle (1t_2), l_2 \langle 1 \rangle (1t_2)].$$

Since there is only one element in  $\text{SPLIT}([t_1])$  we need to find a pair in  $R$  for for each  $S \subseteq \overline{\text{SPLIT}([t_2])} = \{l_1 \langle 1 \rangle (1t_2), l_2 \langle 1 \rangle (1t_2)\}$ .

If  $S = \{\}$  there are two possibilities.

The first possibility is that  $([1], T_{21}) \in R$  such that  $\overline{T_{21}} = \{t_{21} \mid l_1 \langle t_{21} \rangle t_{22} \in \{\}\} = \{\}$ , but since  $([1], []) \notin S$  this is not fulfilled. The second possibility is that  $([1(l_2 \langle 1 \rangle t_1)], T_{22}) \in R$  such that  $\overline{T_{22}} = \{t_{22} \mid l_1 \langle t_{21} \rangle t_{22} \in \overline{\text{SPLIT}(T_2)} \setminus S\} = \{1t_2\}$ . Since  $([1(l_2 \langle 1 \rangle t_1)], [1t_2]) \in R$  this property is fulfilled.

We also need to show one of the two properties for  $S = \{l_1 \langle 1 \rangle (1t_2)\}$ ,  $S = \{l_2 \langle 1 \rangle (1t_2)\}$  and  $S = \{l_1 \langle 1 \rangle (1t_2), l_2 \langle 1 \rangle (1t_2)\}$ .

We will only consider  $S = \{l_1 \langle 1 \rangle (1t_2), l_2 \langle 1 \rangle (1t_2)\}$ .

There are two possibilities.

The first possibility is that  $([1], T_{21}) \in R$  such that  $\overline{T_{21}} = \{t_{21} \mid l_1 \langle t_{21} \rangle t_{22} \in \{l_1 \langle 1 \rangle (1t_2), l_2 \langle 1 \rangle (1t_2)\}\} = \{1\}$ .

Since  $([1], [1]) \in R$  this property is fulfilled.

Assuming that one of the two properties is fulfilled for  $S = \{l_1 \langle 1 \rangle (1t_2)\}$  and  $S = \{l_2 \langle 1 \rangle (1t_2)\}$

we have now proved the necessary properties for the pair  $([t_1], [t_2])$ .

We need to consider the other pairs in  $R$  also, but this is omitted.

We can now illustrate the pairs in  $R$ , and how they use each other to fulfill the properties of a simulation.

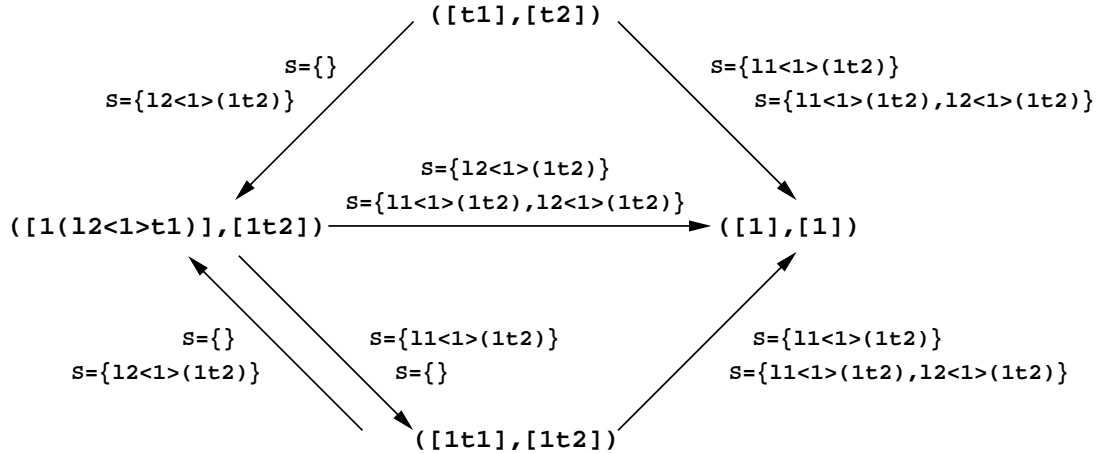


Figure 2: A graphical illustration of the simulation from this example.

Now we will prove that if  $(T_1, T_2)$  is in a simulation then  $\models T_1 < : T_2$ .

**Theorem 12.3 (Soundness of simulations)**

For all simulations  $\mathcal{R}$  the following is fulfilled

$$T_1 \mathcal{R} T_2 \Rightarrow \models T_1 < : T_2$$

This is proved in Proof 19.31 on page 128.

We will now prove the completeness of simulations, but first we need to prove a simple lemma.

**Lemma 12.4**

$$\models T_1 < : T_2 \Rightarrow \forall l \langle t_{11} \rangle t_{12} \in \overline{\text{SPLIT}(T_1)}. \text{in}[[l \langle t_{11} \rangle t_{12}]] \subseteq \text{IN}[\text{SPLIT}(T_2)]$$

This is proved by the following inclusions

$$\begin{aligned} \text{in}[[l \langle t_{11} \rangle t_{12}]] &\subseteq \bigcup_{t' \in \text{SPLIT}(T_1)} \text{in}[[t']] \\ (\text{Lemma 4.9}) &= \text{IN}[\text{SPLIT}(T_1)] \\ (\text{Corollary 10.23}) &= \text{IN}[[T_1]] \setminus \{\varepsilon\} \\ &\subseteq \text{IN}[[T_2]] \setminus \{\varepsilon\} \\ (\text{Corollary 10.23}) &= \text{IN}[\text{SPLIT}(T_2)] \end{aligned}$$

■

**Definition 12.5 (The subtyping simulation)**

We define  $\mathcal{R}_{<}$ : as the subtyping relation.

$$\mathcal{R}_{<} = \{(T_1, T_2) \mid \models T_1 < T_2\}$$

In order to prove completeness of simulations in the sense that if  $\models T_1 < T_2$  then there is a simulation that relates  $T_1$  to  $T_2$ , it is sufficient to prove that  $\mathcal{R}_{<}$  is a simulation, and we will do that now.

**Lemma 12.6 (Completeness of simulations)**

$\mathcal{R}_{<}$  is a simulation

This is proved in Proof 19.32 on page 129.

We have now proved the completeness of simulations, but we want to prove the completeness of finite simulations. This is not as difficult as we might fear, because we can use the concept of derivatives, and the results we have proved for them.

We will now define a family of sets, and then prove completeness of finite simulations.

**Definition 12.7**

We define  $\mathcal{R}_{T_1, T_2}$  for all  $(T_1, T_2) \in \mathcal{R}_{<}$ : in the following way.

$$\begin{aligned} \mathcal{R}_{T_1, T_2} = & \{(T'_1, T'_2) \in \mathcal{R}_{<} \mid \overline{T'_1} \subseteq \text{DERIVS}(T_1) \wedge \overline{T'_2} \subseteq \text{DERIVS}(T_2) \wedge \\ & T'_1, T'_2 \text{ have no multiple occurrences}\} \\ & \cup \{(T_1, T_2)\} \end{aligned}$$

**Theorem 12.8 (Completeness of finite simulations)**

$\forall T_1, T_2 \in \mathbf{XMLTypes}. \models T_1 < T_2 \Rightarrow \mathcal{R}_{T_1, T_2}$  is a finite simulation such that  $T_1 \mathcal{R} T_2$ .

This is proved in Proof 19.33 on page 130.

We have now defined simulations, and proved that  $\models T_1 < T_2$  if and only if  $\mathcal{R}_{T_1, T_2}$  is a finite simulation that relates  $T_1$  to  $T_2$ .

## 13 Axiomatization

In this section we will define a coinductive axiomatization for subtyping of lists of xml-types, and we will prove that this axiomatization is sound and complete.

We want the axiomatization to be deterministic, and therefore we need to introduce the following definitions in order to construct the derived types in a deterministic way.

### Definition 13.1 (Projections)

We define  $\Pi_0(T)$  by induction on the length of  $T$ .

$$\begin{aligned}\Pi_0(\[]) &= [] \\ \Pi_0((l\langle t_1 \rangle t_2) :: T_1) &= l :: \Pi_0(T_1).\end{aligned}$$

We define  $\Pi_1(T)$  by induction on the length of  $T$ .

$$\begin{aligned}\Pi_1(\[]) &= [] \\ \Pi_1((l\langle t_1 \rangle t_2) :: T_1) &= t_1 :: \Pi_1(T_1).\end{aligned}$$

We define  $\Pi_2(T)$  by induction on the length of  $T$ .

$$\begin{aligned}\Pi_2(\[]) &= [] \\ \Pi_2((l\langle t_1 \rangle t_2) :: T_1) &= t_2 :: \Pi_2(T_1).\end{aligned}$$

We will only use these projections on lists of xml-types of the form  $l\langle t_1 \rangle t_2$ , and therefore we only define them for those.

### Definition 13.2 (label)

We define  $label_1(T)$  by induction on the length of  $T$ .

$$\begin{aligned}label_1(\[]) &= [] \\ label_1((l\langle t_1 \rangle t_2) :: T_1) &= (l\langle t_1 \rangle t_2) :: label_1(T_1) \\ label_1(t_1 :: T_1) &= label_1(T_1).\end{aligned}$$

### Definition 13.3 (filter)

We define  $filter_S(T)$  by induction on the length of  $T$ .

$$\begin{aligned}filter_S(\[]) &= [] \\ filter_S(t_1 :: T_1) &= \begin{cases} t_1 :: filter_S(T_1) & \text{if } t_1 \in S \\ filter_S(T_1) & \text{otherwise} \end{cases}\end{aligned}$$

We are now ready to define the axiomatization.

### Definition 13.4 (Axiomatization)

The judgment  $\Gamma \vdash T_1 <: T_2$  where  $\Gamma \subseteq \mathbf{XMLTypes} \times \mathbf{XMLTypes}$  is a finite set, is given by the following inference rules.

$$\text{sub-diff} \frac{\begin{array}{c} ESP(T_1) \neq \{\} \Rightarrow ESP(T_2) \neq \{\} \quad \wedge \\ \forall l\langle t_{11} \rangle t_{12} \in \overline{SPLIT(T_1)}. \forall S \subseteq \overline{SPLIT(T_2)}. \\ (\Gamma \cup \{(T_1, T_2)\} \vdash [t_{11}] <: \Pi_1(filter_S(label_1(SPLIT(T_2)))))) \vee \\ (\Gamma \cup \{(T_1, T_2)\} \vdash [t_{12}] <: \Pi_2(filter_{\overline{SPLIT(T_2)} \setminus S}(label_1(SPLIT(T_2)))))) \end{array}}{\Gamma \vdash T_1 <: T_2}$$

$$\text{sub-hyp} \frac{\exists(T'_1, T'_2) \in \Gamma. \overline{T}_1 = \overline{T}'_1 \wedge \overline{T}_2 = \overline{T}'_2}{\Gamma \vdash T_1 < : T_2}$$

Now we will prove that the axiomatization is complete.

**Lemma 13.5**

If  $\mathcal{R}$  is a finite simulation such that  $T'_1 \mathcal{R} T'_2$ ,  $\overline{T}'_1 = \overline{T}_1$  and  $\overline{T}'_2 = \overline{T}_2$  then for all  $\Gamma$  there is a derivation of  $\Gamma \vdash T_1 < : T_2$ .

This is proved in Proof 19.34 on page 131.

**Corollary 13.6 (Completeness of axiomatization)**

$$\vDash T_1 < : T_2 \Rightarrow \{\} \vdash T_1 < : T_2$$

If  $\vDash T_1 < : T_2$  then Theorem 12.8 yields that  $\mathcal{R}_{T_1, T_2}$  is a finite simulation such that  $T_1 \mathcal{R}_{T_1, T_2} T_2$ . Therefore Lemma 13.5 yields that  $\forall \Gamma. \Gamma \vdash T_1 < : T_2$ , and therefore we have that

$$\{\} \vdash T_1 < : T_2.$$

■

Next we will define language approximation and size-stratified interpretation of subtyping, and use this to prove that the axiomatization is sound.

**Definition 13.7 (Language approximation)**

For all  $n \in \mathbb{N}$  we define the map  $\cdot|_n : \mathcal{P}(\mathbf{XMLSeq}) \rightarrow \mathcal{P}(\mathbf{XMLSeq})$  in the following way.

$$\chi|_n = \{x \in \chi \mid |x| \leq n\}.$$

**Lemma 13.8 (Correctness of Language approximation)**

$$(\forall n \in \mathbb{N}_0. \chi|_n \subseteq \chi'|_n) \Leftrightarrow \chi \subseteq \chi'$$

We prove each implication individually.

If  $\chi \subseteq \chi'$  then  $\forall n \in \mathbb{N}_0. \chi|_n \subseteq \chi'|_n$  because

$$\chi|_n = \{x \in \chi \mid |x| \leq n\} \subseteq \{x \in \chi' \mid |x| \leq n\} = \chi'|_n$$

We can therefore conclude that

$$(\forall n \in \mathbb{N}_0. \chi|_n \subseteq \chi'|_n) \Leftarrow \chi \subseteq \chi'.$$

If  $\forall n \in \mathbb{N}_0. \chi|_n \subseteq \chi'|_n$  and  $x \in \chi$  then  $x \in \chi|_{|x|}$  and therefore  $x \in \chi'|_{|x|}$  which means that  $x \in \chi'$ .

We can therefore conclude that  $\chi \subseteq \chi'$ , and this means that

$$(\forall n \in \mathbb{N}_0. \chi|_n \subseteq \chi'|_n) \Rightarrow \chi \subseteq \chi'$$

We have now proved both inclusions, and can therefore conclude that

$$(\forall n \in \mathbb{N}_0. \chi|_n \subseteq \chi'|_n) \Leftrightarrow \chi \subseteq \chi'.$$

■

**Definition 13.9 (Size stratified interpretation)**

We now introduce the following four definitions, where the last one defines the size stratified interpretation of subtyping.

$$\begin{aligned} \vDash_n T_1 < : T_2 &\Leftrightarrow \text{IM}[[T_1]]|_n \subseteq \text{IM}[[T_2]]|_n \\ \vDash_n \Gamma &\Leftrightarrow \forall (T_1, T_2) \in \Gamma. \vDash_n T_1 < : T_2 \\ \Gamma \vDash_n T_1 < : T_2 &\Leftrightarrow \vDash_n \Gamma \Rightarrow \vDash_n T_1 < : T_2 \\ \Gamma \vDash^s T_1 < : T_2 &\Leftrightarrow \forall n \in \mathbb{N}_0. \Gamma \vDash_n T_1 < : T_2 \end{aligned}$$

The intuitive understanding of  $\Gamma \vDash^s T_1 < : T_2$  is that for all approximation levels  $n$ , if the assumptions in  $\Gamma$  holds for that approximation then that approximation of  $\text{IM}[[T_1]]$  is a subset of the approximation of  $\text{IM}[[T_2]]$ .

Now we can prove that the axiomatization is sound with respect to the size stratified interpretation of subtyping.

**Lemma 13.10 (Size stratified soundness of axiomatization)**

$$\Gamma \vdash T_1 < : T_2 \Rightarrow \Gamma \vDash^s T_1 < : T_2$$

This is proved in Proof 19.35 on page 133.

Now we can use the size stratified soundness to conclude that the axiomatization is sound with respect to the semantical subtyping interpretation.

**Corollary 13.11 (Soundness of axiomatization)**

$$\{\} \vdash T_1 < : T_2 \Rightarrow \vDash T_1 < : T_2$$

If  $\{\} \vdash T_1 < : T_2$  then Lemma 13.10 yields that  $\{\} \vDash^s T_1 < : T_2$  and by definition this means that  $\forall n \in \mathbb{N}_0. \vDash_n \{\} \Rightarrow \vDash_n T_1 < : T_2$ . Since  $\vDash_n \{\}$  is vacantly fulfilled by definition, we have that  $\forall n \in \mathbb{N}_0. \vDash_n T_1 < : T_2$ .

By definition this means that  $\forall n \in \mathbb{N}_0. \text{IM}[[T_1]]|_n \subseteq \text{IM}[[T_2]]|_n$ .

Now Lemma 13.8 yields that  $\text{IM}[[T_1]] \subseteq \text{IM}[[T_2]]$ .

We can therefore conclude that

$$\forall T_1, T_2. \{\} \vdash T_1 < : T_2 \Rightarrow \vDash T_1 < : T_2.$$

■

We have now defined the axiomatization, and proved it to be sound and complete.

## 14 Coercions

We have now introduced a sound and complete axiomatization of semantical subtyping. We will in this section define a way to use a derivation of  $\Gamma \vdash T_1 <: T_2$  to construct a function that converts xml-data for  $T_1$  to xml-data for  $T_2$  that represents the same xml-sequence. We will then prove that this method is sound and complete.

Before we can do this, we must prove that **Trans** does not increase the size of the data.

**Lemma 14.1 (Trans does not increase the data-size)**

$$\forall(n, \langle d_1 \rangle d_2) \in \mathbf{Trans}_\varphi(t, d). |d_1| < |d| \wedge |d_2| < |d|$$

*This is proved in Proof 19.36 on page 136.*

We will now generalize the above definition and result to xml-data for lists of xml-types.

**Definition 14.2 (SIZE)**

$$\mathbf{SIZE}((n, d)) = |d|$$

**Corollary 14.3 (TRANS does not increase the data-size)**

$$\forall(n, \langle d_1 \rangle d_2) \in \mathbf{TRANS}(T, D). |d_1| < \mathbf{SIZE}(D) \wedge |d_2| < \mathbf{SIZE}(D)$$

*We prove this by induction on the length of the list  $T$ .*

*Start:  $T = []$*

*In this case  $\mathbf{TRANS}(T, D) = \{\}$  so the corollary is vacantly fulfilled.*

*Induction Hypothesis:*

*If  $T = t_1 :: T_1$ , and  $(n', \langle d'_1 \rangle d'_2) \in \mathbf{TRANS}(T_1, D)$  then  $|d'_1| < \mathbf{SIZE}(D)$  and  $|d'_2| < \mathbf{SIZE}(D)$ .*

*Step:  $T = t_1 :: T_1$*

*There are two sub-cases.*

*$\mathbf{TRANS}(T, (0, d)) = \mathbf{Trans}(t_1, d)$ : In this case  $\mathbf{SIZE}(D) = |d|$ , and therefore Lemma 14.1 yields that the corollary is true in this case.*

*$\mathbf{TRANS}(T, (n + 1, d)) = \{(n' + |\mathbf{Split}(t_1)|, d') \mid (n', d') \in \mathbf{TRANS}(T_1, (n, d))\}$ :*

*In this case  $\mathbf{SIZE}(D) = \mathbf{SIZE}((n, d))$ , and therefore the induction hypothesis yields that the corollary is true in this case. ■*

We will now define the function that given an index and a list returns the element the list has at the given index.

**Definition 14.4 (nth)**

$$\mathbf{nth}_n([t_0, t_1, t_2, \dots, t_k]) = t_n.$$

*We will only use  $\mathbf{nth}$  in places where  $0 \leq n \leq k$ .*

Each element in  $\text{label}_1(T)$  comes from an index in  $T$ .

We will now define the function that translates an index from  $\text{label}_1(T)$  to the original index in  $T$ .

**Definition 14.5**

$$\begin{aligned} \text{unlabel}(n, l, []) &= 0 \\ \text{unlabel}(0, l, l\langle t_1 \rangle t_2 :: T_1) &= 0 \\ \text{unlabel}(n+1, l, l\langle t_1 \rangle t_2 :: T_1) &= 1 + \text{unlabel}(n, l, T_1) \\ \text{unlabel}(n, l, t_1 :: T_1) &= 1 + \text{unlabel}(n, l, T_1) \end{aligned}$$

We are now ready to define a way to use a derivation of  $\Gamma \vdash T_1 <: T_2$  to construct a function that converts xml-data for  $T_1$  into xml-data for  $T_2$  that represents the same xml-sequence.

**Definition 14.6**

We now define  $\mathcal{C}[\cdot]$  which takes a derivation of  $\Gamma \vdash T_1 <: T_2$  and returns a function which converts xml-data for  $T_1$  to xml-data for  $T_2$ .

We define  $\mathcal{C}[\cdot]$  by induction on the derivation of  $\Gamma \vdash T_1 <: T_2$ .

$$\mathcal{C}[\text{sub-hyp} \frac{(T'_1, T'_2) \in \Gamma}{\Gamma \vdash T_1 <: T_2}] = \lambda(n_1, d_1).$$

$$\begin{aligned} &\bigcup_{n'_1 \in \{n \mid \text{nth}_n(T'_1) = \text{nth}_{n_1}(T_1)\}} \bigcup_{(n'_2, d_2) \in c_{T'_1, T'_2}(n'_1, d_1)} \\ &\bigcup_{n_2 \in \{n \mid \text{nth}_n(T_2) = \text{nth}_{n'_2}(T'_2)\}} \{(n_2, d_2)\} \end{aligned}$$

$$\mathcal{C}[\text{sub-diff} \frac{\dots}{\Gamma \vdash T_1 <: T_2}] = \text{fix } c_{T_1, T_2} \cdot \lambda D_1.$$

if  $\text{TAG}(T_1, D_1) = \top$

then  $\{\}$

else if  $\text{TAG}(T_1, D_1) = \varepsilon$

then  $\text{ESP}(T_2)$

else

$$\begin{aligned} &\bigcup_{(n_1, \langle d_{11} \rangle d_{12}) \in \text{TRANS}(T_1, D_1)} \\ &\text{let } l\langle t_{11} \rangle t_{12} = \text{nth}_{n_1}(\text{SPLIT}(T_1)) \end{aligned}$$

in

$$(n_2, d_{21}) \in \mathcal{C}[\Gamma \cup \{(T_1, T_2)\} \vdash [t_{11}] <: \Pi_1(\text{label}_1(\text{SPLIT}(T_2)))](0, d_{11})$$

$$(n_2, d_{22}) \in \mathcal{C}[\Gamma \cup \{(T_1, T_2)\} \vdash [t_{12}] <: \Pi_2(\text{label}_1(\text{SPLIT}(T_2)))](0, d_{12})$$

$$\text{TRANSINV}(T_2, (\text{unlabel}(n_2, l, \text{SPLIT}(T_2)), \langle d_{21} \rangle d_{22}))$$

end

Before we start, there are some things we have to consider.  
This is done in the following observations.

**Observation 14.7**

When  $\mathcal{C}[\Gamma \vdash T_1 < : T_2]$  calls  $\mathcal{C}[\Gamma' \vdash T'_1 < : T'_2]$  then  $\Gamma' \vdash T'_1 < : T'_2$  is a subderivation of  $\Gamma \vdash T_1 < : T_2$ .

The recursive calls to  $\mathcal{C}[\cdot]$  are of the forms  
 $\mathcal{C}[\Gamma \cup \{(T_1, T_2)\} \vdash [t_{11}] < : \Pi_1(\text{label}_1(\text{SPLIT}(T_2)))]$  and  
 $\mathcal{C}[\Gamma \cup \{(T_1, T_2)\} \vdash [t_{12}] < : \Pi_2(\text{label}_1(\text{SPLIT}(T_2)))]$ .

If we consider  $l(t_{11})t_{12} \in \overline{\text{SPLIT}(T_1)}$  and  $S = \overline{\text{SPLIT}(T_2)}$  then either

$\Gamma \cup \{(T_1, T_2)\} \vdash [t_{11}] < : \Pi_1(\text{label}_1(\text{SPLIT}(T_2)))$  or  
 $\Gamma \cup \{(T_1, T_2)\} \vdash [t_{12}] < : []$  is a subderivation.

If  $\Gamma \cup \{(T_1, T_2)\} \vdash [t_{12}] < : []$  is a subderivation then  $t_{12}$  must be the uninhabited type, and therefore we can just return the empty set in stead of making the recursive call.

The call to  $\mathcal{C}[\Gamma \cup \{(T_1, T_2)\} \vdash [t_{12}] < : \Pi_2(\text{label}_1(\text{SPLIT}(T_2)))]$  can be shown symmetrically for  $S = \{\}$ .

Therefore the calls to  $\mathcal{C}[\cdot]$  follows the structure of the derivation of  $\Gamma \vdash T_1 < : T_2$ .

**Observation 14.8**

When  $\mathcal{C}[\Gamma \vdash T_1 < : T_2]$  calls  $c_{T'_1, T'_2}$  then it has already been defined.

This is fulfilled, since  $c_{T'_1, T'_2}$  is only called when  $(T'_1, T'_2) \in \Gamma$ , and **fix**  $c_{T_1, T_2}$  is declared before  $(T'_1, T'_2)$  is added to  $\Gamma$ .

**Observation 14.9**

$\mathcal{C}[\Gamma \vdash T_1 < : T_2](D_1)$  terminates.

First of all  $\mathcal{C}[\Gamma \vdash T_1 < : T_2]$  terminates, since it is declared by induction on the derivation of  $\Gamma \vdash T_1 < : T_2$ .

The runtime analysis in Section 15 yields an upper limit to the runningtime of  $\mathcal{C}[\Gamma \vdash T_1 < : T_2](D_1)$  and therefore it terminates.

We are now ready to prove soundness of  $\mathcal{C}[\cdot]$ .

**Lemma 14.10 (Soundness of  $\mathcal{C}[\cdot]$ )**

$$\begin{aligned} (\forall (T'_1, T'_2) \in \Gamma. \forall D'_1. \text{SIZE}(D'_1) \leq \text{SIZE}(D_1) \Rightarrow \forall D'_2 \in c_{T'_1, T'_2}(D'_1). \text{TAG}(T'_2, D'_2) = \text{TAG}(T'_1, D'_1)) \\ \Rightarrow \forall D_2 \in \mathcal{C}[\Gamma \vdash T_1 < : T_2]. \text{TAG}(T_2, D_2) = \text{TAG}(T_1, D_1) \end{aligned}$$

This is proved in Proof 19.37 on page 137.

**Corollary 14.11 (Soundness of  $\mathcal{C}[\cdot]$ )**

$$\forall D_2 \in \mathcal{C}[\{\} \vdash T_1 < : T_2](D_1). \text{TAG}(T_2, D_2) = \text{TAG}(T_1, D_1)$$

In the case where  $\Gamma = \{\}$  the extra assumptions in Lemma 14.10 are vacantly fulfilled and therefore Lemma 14.10 yields that  $\forall D_2 \in \mathcal{C}[\{\} \vdash T_1 < : T_2](D_1). \text{TAG}(T_2, D_2) = \text{TAG}(T_1, D_1)$ . ■

We are now ready to prove completeness of  $\mathcal{C}[\cdot]$ .

Like for ESP we will prove a more strict completeness, but this time it is because it does not seem possible to prove the normal completeness directly.

**Lemma 14.12 (Completeness of  $\mathcal{C}[\cdot]$ )**

$$\begin{aligned} \forall D_1 \ . (\forall (T'_1, T'_2) \in \Gamma. \forall D'_1. \text{SIZE}(D'_1) \leq \text{SIZE}(D_1) \Rightarrow \forall j. \text{TAG}(T'_1, D'_1) \in \text{in}[\text{nth}_j(T'_2)]) \\ \Rightarrow \{(n'_2, d'_2) \in \mathcal{C}_{T'_1, T'_2}(D'_1) \mid n'_2 = j\} \neq \{\}) \\ \Rightarrow \forall j. \text{TAG}(T_1, D_1) \in \text{IN}[\text{nth}_j(T_2)] \Rightarrow \{(n_2, d_2) \in \mathcal{C}[\Gamma \vdash T_1 < : T_2](D_1) \mid n_2 = j\} \neq \{\} \end{aligned}$$

This is proved in Proof 19.38 on page 139.

**Corollary 14.13 (Completeness of  $\mathcal{C}[\cdot]$ )**

$$\text{TAG}(T_1, D_1) \neq \top \Rightarrow \mathcal{C}[\{\} \vdash T_1 < : T_2](D_1) \neq \{\}$$

If  $\text{TAG}(T_1, D_1) \neq \top$  then Corollary 6.10 yields that  $\text{TAG}(T_1, D_1) \in \text{IN}[T_1]$ .

Since  $\{\} \vdash T_1 < : T_2$  Corollary 13.6 yields that  $\vDash T_1 < : T_2$  which means that  $\text{IN}[T_1] \subseteq \text{IN}[T_2]$  and therefore  $\text{TAG}(T_1, D_1) \in \text{IN}[T_2]$ .

Now Lemma 4.9 yields that there is a  $j$  such that

$$\text{TAG}(T_1, D_1) \in \text{in}[\text{nth}_j(T_2)].$$

In the case where  $\Gamma = \{\}$  the extra assumptions in Lemma 14.12 are vacantly fulfilled and therefore Lemma 14.12 yields that  $\{(n_2, d_2) \in \mathcal{C}[\{\} \vdash T_1 < : T_2](D_1) \mid n_2 = j\} \neq \{\}$  and therefore  $\mathcal{C}[\{\} \vdash T_1 < : T_2](D_1) \neq \{\}$ . ■

We have now defined a way to use a derivation of  $\{\} \vdash T_1 < : T_2$  to construct a sound and complete function that converts xml-data for  $T_1$  to xml-data for  $T_2$  that represents the same xml-sequence.

## 15 Runtime Analysis

In this section we will find an upper limit to the running time of the coercions as a function of the size of the data.

We will not consider the running time of the finding a derivation of the subtyping, or using the derivation of subtyping to generate the coercions. We focus on the running time of the coercions because this is what effects the efficiency of the compiled programs.

There have only been time to prove a rough upper limit to the running time of the coercions.

There is plenty of room for improvement, but some of them will require a more thorough analysis, and some of them will require the functions to be redefined.

The functions have been defined to make the necessary lemmas as easy to prove as possible.

This has been at the cost of efficient running times.

We will use the notation from [CLRS, Section 3] to describe the upper limits of the running-times. The only exception is that we will write  $f(n) \in O(g(n))$  in stead of  $f(n) = O(g(n))$  because the notation in [CLRS] is mathematically misleading.

We will let  $\text{rt}_f(a)$  denote the runningtime of the function call  $f(a)$ .

We start by proving that

$$\text{rt}_{\text{tag}}(t, d) \leq f_1(t) \cdot |d|^2$$

for some function  $f_1$ .

We prove this by induction on  $|d|$ .

We have to prove this for all the cases in the definition of  $\text{tag}$ , but we focus on the case where  $t = t_1 t_2$  and  $d = d_1 d_2$  because this is the most complicated case.

In this case  $\text{tag}(t, d) = \text{tag}(t_1 t_2, d_1 d_2) = \text{let } x_1 = \text{tag}(t_1, d_1), x_1 = \text{tag}(t_2, d_2) \text{ in } x_1 x_2 \text{ end}$ . This means that  $\text{rt}_{\text{tag}}(t_1 t_2, d_1 d_2) = \text{rt}_{\text{tag}}(t_1, d_1) + \text{rt}_{\text{tag}}(t_2, d_2) + k_1 \cdot |d|$ . The last part of the equation is obtained, because the sequencing of xml-sequences is linear in  $|x_1|$  and this is at most  $|d|$ . This means that

$$\begin{aligned} & \text{rt}_{\text{tag}}(t_1 t_2, d_1 d_2) \\ &= \text{rt}_{\text{tag}}(t_1, d_1) + \text{rt}_{\text{tag}}(t_2, d_2) + k_1 \cdot |d| \\ \text{(IH)} \quad & \leq f_1(t_1) \cdot |d_1|^2 + f_1(t_2) \cdot |d_2|^2 + k_1 \cdot |d| \\ (|d| = |d_1| + |d_2| + 1) \quad & \leq \max(f(t_1), f(t_2)) \cdot (|d|^2 - |d|) + |d| \\ (f(t) \geq f(t_1), f(t_2), k_1) \quad & \leq f(t) \cdot |d|^2 \end{aligned}$$

This proves that the upper limit is true for this case, under the condition that  $f_1$  fulfills that  $f_1(t_1 t_2) \geq f_1(t_1)$ ,  $f_1(t_1 t_2) \geq f_1(t_2)$  and  $f_1(t_1 t_2) \geq k_1$ .

We can prove the upper limit for the other cases as well.

It is important to note that in the  $\mu$ -case we will require that  $f(\mu X.t_1) \geq f(t_1[\mu X.t_1/X])$ .

This makes  $f(t)$  exponential in  $|t|$  but it is possible to construct a function with this property, because the xml-types are tail-recursive.

We can therefore conclude that  $\mathbf{rt}_{\text{tag}}(t, d) \in O(f_1(t) \cdot |d|^2)$ .

It is now simple to see that

$$\mathbf{rt}_{\text{TAG}}(T, (n, d)) \in O(f_2(T) \cdot |d|^2)$$

for some function  $f_2$ , because  $T$  always has a limited length.

now we can prove that

$$\mathbf{rt}_{\text{simplify}}(\mu X.t', d', t, d) \leq f_3(\mu X.t', t) \cdot |d|^3$$

for some function  $f_3$ .

We prove this by induction on  $\text{ismu}(t) + 2 \cdot |d|$ .

Again we only consider the most complicated case, and this is the case where  $t = \mu Y.t_1$ . In this case

$$\begin{aligned} \text{simplify}_{\mu X.t', d'}(\mu Y.t_1, d) &= \text{if } \mu Y.t_1 = \mu X.t' \\ &\quad \text{then } \text{simplify}_{\mu Y.t_1, d}(t_1[\mu Y.t_1/Y], d) \\ &\quad \text{else } \text{simplify}_{\mu X.t', d'}(t_1[\mu Y.t_1/Y], d) \end{aligned}$$

This means that

$$\begin{aligned} &\mathbf{rt}_{\text{simplify}}(\mu X.t', d', \mu Y.t_1, d) \\ &\leq f'_3(\mu X.t', \mu Y.t_1) + \max(\mathbf{rt}_{\text{simplify}}(\mu X.t', d, t_1[\mu Y.t_1/Y], d), \mathbf{rt}_{\text{simplify}}(\mu X.t', d', t_1[\mu Y.t_1/Y], d)) \\ \text{(IH)} &\leq f'_3(\mu X.t', \mu Y.t_1) + \max(f_3(\mu X.t', t_1[\mu Y.t_1/Y]) \cdot |d|^3, f_3(\mu X.t', t_1[\mu Y.t_1/Y]) \cdot |d|^3) \\ &= f'_3(\mu X.t', \mu Y.t_1) + f_3(\mu X.t', t_1[\mu Y.t_1/Y]) \cdot |d|^3 \\ &\leq f_3(\mu X.t', \mu Y.t_1) \cdot |d|^3. \end{aligned}$$

Therefore the upper limit is fulfilled in this case.

We can therefore conclude that  $\mathbf{rt}_{\text{simplify}}(\mu X.t', d', t, d) \in O(f_3(\mu X.t', t) \cdot |d|^3)$ .

Now we can prove that

$$\mathbf{rt}_{\text{Trans}}(\varphi, t, d) \leq f_4(\varphi, t) \cdot |d|^3$$

for some function  $f_4$ .

We prove this by induction on  $t$ .

Again we only consider the most complicated case, and this is the case where  $t = \mu X.t_1$ .

In this case  $\text{Trans}_{\varphi}(t, d) = \text{Trans}_{\mu X.t_1/X::\varphi}(t_1, \text{simplify}_{\mu X.t_1[\varphi], d}(t_1[\mu X.t_1/X :: \varphi], d))$ .

This means that

$$\begin{aligned}
& \mathbf{rt}_{\text{Trans}}(\varphi, \mu X.t_1, d) \\
= & f'_4(\varphi, \mu X.t_1) + \mathbf{rt}_{\text{simplify}}(\mu X.t_1[\varphi], d, t_1[\mu X.t_1/X :: \varphi], d) \\
& + \mathbf{rt}_{\text{Trans}}(\mu X.t_1/X :: \varphi, t_1, \mathbf{simplify}_{\mu X.t_1[\varphi], d}(t_1[\mu X.t_1/X :: \varphi], d)) \\
\leq & f'_4(\varphi, \mu X.t_1) + f_3(\mu X.t_1[\varphi], t_1[\mu X.t_1/X :: \varphi]) \cdot |d|^3 \\
& + \mathbf{rt}_{\text{Trans}}(\mu X.t_1/X :: \varphi, t_1, \mathbf{simplify}_{\mu X.t_1[\varphi], d}(t_1[\mu X.t_1/X :: \varphi], d)) \\
\text{(IH)} \leq & f'_4(\varphi, \mu X.t_1) + f_3(\mu X.t_1[\varphi], t_1[\mu X.t_1/X :: \varphi]) \cdot |d|^3 \\
& + f_4(\mu X.t_1/X :: \varphi, t_1) \cdot |\mathbf{simplify}_{\mu X.t_1[\varphi], d}(t_1[\mu X.t_1/X :: \varphi], d)|^3 \\
\leq & f'_4(\varphi, \mu X.t_1) + f_3(\mu X.t_1[\varphi], t_1[\mu X.t_1/X :: \varphi]) \cdot |d|^3 + f_4(\mu X.t_1/X :: \varphi, t_1) \cdot |d|^3 \\
\leq & f_4(\varphi, \mu Y.t_1) \cdot |d|^3
\end{aligned}$$

where the last inequality requires that

$$f_4(\varphi, \mu X.t_1) > f_4(\mu X.t_1/X :: \varphi, t_1) + f_3(\mu X.t_1[\varphi], t_1[\mu X.t_1/X :: \varphi]) + f'_4(\varphi, \mu X.t_1).$$

Therefore the upper limit is fulfilled in this case.

We can therefore conclude that

$$\mathbf{rt}_{\text{Trans}}(\varphi, t, d) \in O(f_4(\varphi, t) \cdot |d|^3).$$

We can now see that

$$\mathbf{rt}_{\text{TRANS}}(T, (n, d)) \in O(f_5(T) \cdot |d|^3)$$

for some function  $f_5$ , because  $T$  always has a limited length.

We can see that

$$\mathbf{rt}_{\text{TransInv}}(t, D) \in O(f_6(t))$$

for some function  $f_6$  because  $D$  is not used for any of the calculations.

We can now see that

$$\mathbf{rt}_{\text{TRANSINV}}(T, (n, d)) \in O(f_7(T))$$

for some function  $f_7$  because  $D$  is not used for any of the calculations.

We are now finally ready to find an upper limit to  $\mathbf{rt}_{\mathbf{c}[\Gamma \vdash T_1 <: T_2]}(D)$ .

We will show that

$$\mathbf{rt}_{\mathbf{c}[\Gamma \vdash T_1 <: T_2]}((n, d)) \leq f_8(\Gamma, T_1, T_2) \cdot |d|^3 \cdot |\mathbf{TAG}(T_1, (n, d))|$$

for some function  $f_8$  by induction on  $|\mathbf{TAG}(T_1, (n, d))|$ .

There are two cases to consider.

We first consider the case generated by the rule **sub-diff**.

In this case we get that

$$\begin{aligned}
& \mathbf{rt}_{\mathbf{c}_{[\Gamma \vdash T_1 < T_2]}}((n, d)) \\
\leq & f'_8(T_1, T_2) + \mathbf{rt}_{\mathbf{TAG}}(T_1, (n, d)) + \mathbf{rt}_{\mathbf{TRANS}}(T_1, (n, d)) + \mathbf{length}(T_2) \cdot \mathbf{rt}_{\mathbf{TRANSINV}}(T_2, (n'_2, \langle d_{21} \rangle d_{22})) \\
& + \mathbf{rt}_{\mathbf{c}_{[\Gamma \vdash [t_{11}] < T'_2]}}((0, d_{11})) + \mathbf{rt}_{\mathbf{c}_{[\Gamma \vdash [t_{12}] < T''_2]}}((0, d_{12})) \\
\leq & f'_8(T_1, T_2) + f_2(T_1) \cdot |d^2| + f_5(T_1) \cdot |d^3| + \mathbf{length}(T_2) \cdot f_7(T_1) \\
& + \mathbf{rt}_{\mathbf{c}_{[\Gamma \vdash [t_{11}] < T'_2]}}((0, d_{11})) + \mathbf{rt}_{\mathbf{c}_{[\Gamma \vdash [t_{12}] < T''_2]}}((0, d_{12})) \\
\text{(IH)} \leq & f'_8(T_1, T_2) + f_2(T_1) \cdot |d^2| + f_5(T_1) \cdot |d^3| + \mathbf{length}(T_2) \cdot f_7(T_1) \\
& + f_8(\Gamma \cup \{(T_1, T_2)\}, [t_{11}], T'_2) \cdot |d_{11}|^3 \cdot |\mathbf{TAG}([t_{11}], (0, d_{11}))| \\
& + f_8(\Gamma \cup \{(T_1, T_2)\}, [t_{12}], T''_2) \cdot |d_{12}|^3 \cdot |\mathbf{TAG}([t_{12}], (0, d_{12}))| \\
\leq & f'_8(T_1, T_2) + f_2(T_1) \cdot |d^2| + f_5(T_1) \cdot |d^3| + \mathbf{length}(T_2) \cdot f_7(T_1) \\
& + \max(f_8(\Gamma \cup \{(T_1, T_2)\}, [t_{11}], T'_2), f_8(\Gamma \cup \{(T_1, T_2)\}, [t_{12}], T''_2)) \cdot |d|^3 \cdot (|\mathbf{TAG}(T_1, (n, d))| - 1) \\
\leq & f_8(\Gamma, T_1, T_2) \cdot |d|^3 \cdot |\mathbf{TAG}(T_1, (n, d))|
\end{aligned}$$

Therefore the upper limit is fulfilled in this case.

We now consider the case generated by the rule **sub-hyp**.

In this case a recursive call  $\mathbf{c}_{T'_1, T'_2}((n', d))$  is made, but we know that the calls to  $\mathbf{c}_{T'_1, T'_2}$  will always use the case produced by the **sub-diff** rule.

Therefore only every second node in the calltree can be to at case generated by the **sub-hyp** rule, and since the time used in these nodes is limited by a function on  $T_1$  and  $T_2$  the limit is fulfilled in this case, simply by raising  $f_8$  by the time used in these nodes.

Therefore the limit is fulfilled in both cases, and we can conclude that

$$\mathbf{rt}_{\mathbf{c}_{[\{\} \vdash T_1 < T_2]}}((n, d)) \in O(f_8(\{\}, T_1, T_2) \cdot |d|^3 \cdot |\mathbf{TAG}(T_1, (n, d))|) \subseteq O(\{\}, f_8(T_1, T_2) \cdot |d|^4)$$

The factor  $f_8(T_1, T_2)$  is probably very fast growing (at least exponentially).

Since  $T_1$  and  $T_2$  are fixed, this is almost an acceptable running time of the conversion of xml-data, but it is possible that this upper limit can be improved by a more thorough analysis, and by optimizing the functions.

Possible optimizations are discussed in Section 18.2 about future work.

## 16 Implementation

We have implemented the subtyping algorithm in [SML].

The code can be found in Appendix 19.2.

The output from interpreting the code with [MOSML] can be found in Appendix 19.3.

As the output shows, the little testing we have done was successful.

One thing that is emphasized in the code is that the case where  $\Gamma \not\vdash T_1 <: T_2$  the algorithm can actually return a witness which proves that the subtyping is not fulfilled.

This is done by finding an xml-sequence that is in  $\text{IN}[[T_1]]$  but not in  $\text{IN}[[T_2]]$ .

This witness generation follows the idea in the completeness proof of simulations, in Lemma 12.6.

The case where  $\text{ESP}(T_1) \neq \{\}$  and  $\text{ESP}(T_2) = \{\}$  this evidence is simply the empty xml-sequence  $\varepsilon$ .

In the case where there is a  $l\langle t_{11} \rangle t_{12}$  in  $\text{SPLIT}(T_1)$  and a  $S \subseteq \overline{\text{SPLIT}(T_2)}$  such that

$\Gamma' \not\vdash [t_{11}] <: \Pi_1(\text{filter}_S(\text{label}_1(\text{SPLIT}(T_2))))$  and

$\Gamma' \not\vdash [t_{12}] <: \Pi_2(\text{filter}_{\overline{\text{SPLIT}(T_2)} \setminus S}(\text{label}_1(\text{SPLIT}(T_2))))$

the algorithm yields two xml-sequences  $x_1, x_2$  such that

$x_1 \in \text{IN}[[t_{11}]]$ ,  $x_1 \notin \text{IN}[[\Pi_1(\text{filter}_S(\text{label}_1(\text{SPLIT}(T_2))))]]$ ,

$x_2 \in \text{IN}[[t_{12}]]$  and  $x_2 \notin \text{IN}[[\Pi_2(\text{filter}_{\overline{\text{SPLIT}(T_2)} \setminus S}(\text{label}_1(\text{SPLIT}(T_2))))]]$ .

Therefore we get, just as in the completeness proof of simulations, that  $\langle 1 \rangle x_1 \langle /1 \rangle x_2 \in \text{in}[[l\langle t_{11} \rangle t_{12}]] \subseteq \text{IN}[[T_1]]$  and  $\langle 1 \rangle x_1 \langle /1 \rangle x_2 \notin \text{IN}[[T_2]]$ .

Therefore if the algorithm does not return a proof of  $\Gamma \vdash T_1 <: T_2$  then it returns an exception containing a witness that  $T_1$  is *not* a subtype of  $T_2$ .

One note about the implementation is that the coercions has not been implemented the same way as we have defined them.

The reason for this is that we have defined the coercions using **fix** expressions, and it is easier to write an implementation that avoids the **fix** expressions.

We therefore use that the given data is finite to avoid the **fix** expressions, but this means that we cannot specialize the coercions with respect to the given types.

The result is that the coercion functions cannot be optimized as much as they can the way we have defined them, but we can accept this as the implementation is just meant for testing.

## 17 Applications

In this section, we will suggest a series of applications of the theory presented in this project.

The main reason for considering this type system is to create languages that use and manipulate xml-sequences in an intuitive, native, efficient, and safe way.

There are many ways to obtain such a language, and the projects mentioned in Section 18.1 about related work implements some of them.

The main difference between previous projects on this type system and this project is that this project uses type-specific representations of xml-sequences where as other projects use a uniform representation of xml-sequences.

Type specific representations can be used to make the programs more memory efficient, because there is a lot of information in the types, that we do not need to store in the data.

To illustrate this, we have found a sample xml-sequence from [W3SCHOOLS.COM] for which we have constructed a serialized version of the xml-data for a fitting xml-type, so we can compare the sizes they will use in memory.

The xml-sequence, the serialized xml-data, and the xml-type can be found in Appendix 19.4 on page 152.

This yields the following sizes in bytes.

	xml-sequence	xml-data
Uncompressed	4095	1689
Compressed	906	764

We see that in this example the type-specific representation uses about 42% of the space the entire xml-sequence uses.

We also see that even though we compress the data, there is still a difference in the size of the xml-sequence and the type-specific representation.

In this example the compressed version of the type-specific representation used about 85% of the space used by the compressed version of the xml-sequence.

In this example we have used [BZIP2] with default parameters for compression.

This comparison is just an example. The gain of using type-specific representations can be made arbitrarily large or small by selecting the ratio between the size of the tag-names and the size of the real data.

The general idea is that we can use the information in the xml-type to make the representation of xml-sequences as compact as possible. Since uniform representations cannot use this information they will in general not be as compact as type-specific representations. The type-specific representation can be made more compact than xml-data, but this is discussed in Section 18.2 about future work.

Another reason to use type-specific representations is to make the data-manipulation and pattern-matching efficient.

When unique data representations are used, it is necessary to parse the data every time the data is used for tasks like pattern matching.

One example of this is if you have data for the xml-type  $t_1t_2$  and you want to split the data into the data that belongs to  $t_1$  and the data that belongs to  $t_2$ .

With our type-specific representation of data this is a simple task, because the data is of the form  $d_1d_2$  where  $d_1$  belongs to  $t_1$  and  $d_2$  belongs to  $t_2$ .

If we use xml-sequences as data, then we will have to parse the xml-sequence in order to split it into the two parts. Another question arises when we do this, because there can be more than one way to split the data, and what way do we select, when it is not given by the representation of the data.

The downside to using type-specific representations of data is that we need to convert the representation when the data is copied to a more general type as it often is before a function call.

This is why it is important that the coercions are efficient.

## 18 Summing Up

### 18.1 Related Work

[HU79] is referenced by [HVP] to have proved that the general problem of context free language inclusion which is equivalent to subtyping of unrestricted xml-types is undecidable. This is the reason why we only consider tail-recursive xml-types.

Many of the methods used here are introduced in [BH98], which presents a sound and complete axiomatization of equality for a recursive first order type-language. Some of the methods are the stratified interpretation of subtyping used for the soundness proof and the way assumptions are used in the axiomatization.

Most of the basic theory we use, such as the  $\mu$ -calculus and semantic subtyping is introduced in [TAPL].

The rest of the projects we will mention in this section can be divided into two types. The first type does the same thing as this project. They construct an axiomatization of subtyping, and uses this axiomatization to construct functions that convert the type-specific representation of data from one type to another. Some of them uses these results to integrate their type system in the type system in an existing language. All the references we have found of this type use a type system that is simpler than the one considered in this project.

The second type uses an equivalent or generalized type system, but they do not consider type-specific representations of xml-sequences. This means that they use a uniform representation of data, often the data is simply the xml-sequence. This means that they do not have to consider conversion of the type-specific representation, but this also means that the memory is not used efficiently, and they will not be able to use efficient pattern matching and data manipulation because of the extra need for xml-parsing.

We will now describe the related project we have found.

[HVP] introduces a coinductive axiomatization for an equivalent definition of tail recursive xml-types. There are many similarities between the methods used in [HVP] and here. The central similarity is the way subtyping of splitted xml-types is expressed using every subset of the list on the right hand side, which in this case is used to define the second property of simulations.

Even though there are many similarities, there are also many differences. First of all the article in [HVP] does not consider translation of the type-specific representations. In their type system the xml-data is just the actual xml-sequence. Therefore they only consider the problem of deciding if the semantic subtyping is fulfilled, and not of translating data from one type to another. Another difference is that they start by translating the xml-types to a simplified internal type language and then work on the internal type language. It turns out that this trick saves them

a lot of work in the proofs. All the theory about data simplification can probably be avoided this way.

The last difference we will mention here is that the types in [HVP] are represented by an environment of declarations of the form  $name = type$ , and the recursion occurs because all the types can refer to each other by name. We represent them using a variation of the  $\mu$ -calculus. Both choices have advantages. Some advantages of their representation are that this representation allows them to do the trick with the internal type language, and that this representation is closer to the way it probably will be implemented. Some advantages of our representation are that properties such as the tail-recursiveness requirement can be expressed simpler and there is no need for a type environment. The work done in [HVP] results in a language called **XDuce** that has a typing system equivalent to our tail recursive xml-types.

[LS05] integrates a part of **XDuce** into ML.

They do this by translating some types from **XDuce** into ML types, translating xml-sequences into the data-structure matching the found ML types and translates **XDuce** code into ML code.

The last step includes translating the ML data when a call to a function that takes a more general datatype is made. They make this translation of ML data using coercions based on the subtyping derivation of the **XDuce** types.

They also use the coercions to implement pattern matching, by using so called downcast translation functions.

Even though [LS05] claims to have done this for the entire **XDuce** system, their type translation yields that they do not consider the entire type system.

The types considered in [LS05] are of the form

$$R ::= ()|R * |l|(R_1|R_2)|(R_1, R_2).$$

This means that the only form of recursion allowed is  $R*$ , which is equivalent to the xml-type  $\mu X.1 + tX$  where  $t$  is a type equivalent to  $R$ .

Therefore they have no way to represent types where the recursion occurs inside a tag. An example of an xml-type they cannot represent is the xml-type

$$t = \mu X.1 + l\langle X \rangle.$$

To summarize the results of [LS05] they have developed an integration of **XDuce** into ML, and therefore extended the type system from [HVP] with type-specific representations, but they only consider types with a flat recursion which are equivalent to regular expressions.

The two projects [F06] and [BCF03] add parametric polymorphism to the **XDuce** system.

[F06] combine the type system with the type system from OCaml. This yields a language with two sets of separated type systems.

[BCF03] extends the type system from **XDuce** with higher order types and parametric polymorphism.

[CA-REG] builds a sound and complete axiomatization for equality of regular expressions by using derivatives. The method we use to create a sound and complete axiomatization for tail-recursive xml-types is based on the same principle of derivatives.

Also our implementation has been based on the implementation from [CA-REG].

## 18.2 Future Work

There are still a lot of loose ends in this project. We will in this section mention a few of them.

First of all, the xml-types should be extended to include simple types as `string`, `int` and so on.

This should be fairly trivial, although it does require adding extra cases to all the proofs, and extending the implementation.

Although we have implemented the proof-searching and conversion of xml-data, there are still a number of implementation tasks to be performed.

The code needs to be optimized, and there may be a few bugs left that was not found by the testing.

The system must be combined with a compiler in order to be useful. This has already been done for the [HVP] project, but using this system would allow the programs to be more memory efficient and perform efficient pattern matching by using type-specific representations of xml-sequences.

The runtime analysis needs to be improved, and this can include redefining some functions to be more efficient and reimplementing them.

One optimization could be to improve the way the xml-data simplification is done. Currently the optimization is done in every step of the coercions, but it would be more efficient to perform the simplification before the conversion.

The [HVP] project uses an internal type language, and if this theory was applied to that language instead of the general xml-type language, the need for data simplification could probably be removed entirely.

One optimization that can improve the runningtime to be in  $O(|d|^3)$  is simply to use a representation of lists that allows appending in constant time.

The memory efficiency could still be improved.

There is much redundancy in the current xml-data grammar.

The first thing is that the  $\langle d \rangle$  constructor is not necessary. There is only one constructor that matches types of the form  $l\langle t \rangle$ , so in stead of using the xml-data  $\langle d \rangle$  we could just use  $d$ .

The only information we currently get from the xml-data is a selection of the elements in a sum-type.

Using this view of the data, we could simply let the type-specific representation be a sequence of 0's and 1's, where the 0's selected the first element in a sum-type, and the 1's selected the second element in a sum-type.

This would yield a much more compact representation of xml-sequences, but it would be at the cost of efficient pattern matching.

Finally the memory efficiency could be improved by compressing the data. This is of course at the cost of runtime efficiency, but if a compression that respects the structure of the xml-sequence like [XCOMPACT] is used, the runtime cost could be minimized.

## 19 Appendix

### 19.1 Technical Proofs

#### Proof 19.1 (Substitution is well-defined)

We will now prove Lemma 3.3 which states that

$$\vdash t \equiv t_\alpha \wedge \vdash t' \equiv t'_\alpha \Rightarrow \vdash t[t'/X] \equiv t_\alpha[t'_\alpha/X].$$

We will prove this by induction on  $|t|$ .

*Induction Hypothesis:*

If  $|t_1| < |t|$  and  $\vdash t_1 \equiv t_{1\alpha}$  and  $\vdash t'_1 \equiv t'_{1\alpha}$  then we can assume that  $\vdash t_1[t'_1/X] \equiv t_{1\alpha}[t'_{1\alpha}/X]$ .

We will consider each form of  $t$  separately.

Case:  $t = 0$

In this case  $t_\alpha = 0$ .

Since  $t[t'/X] = 0[t'/X] = 0$  and  $t_\alpha[t'_\alpha/X] = 0[t'_\alpha/X] = 0$ , we only need to prove that  $\vdash 0 \equiv 0$  and this is done using the **equiv-0** rule.

Case:  $t = 1$

In this case  $t_\alpha = 1$ .

Since  $t[t'/X] = 1[t'/X] = 1$  and  $t_\alpha[t'_\alpha/X] = 1[t'_\alpha/X] = 1$ , we only need to prove that  $\vdash 1 \equiv 1$  and this is done using the **equiv-1** rule.

Case:  $t = l\langle t_1 \rangle$

In this case  $t_\alpha = l\langle t_{1\alpha} \rangle$  where  $\vdash t_1 \equiv t_{1\alpha}$ .

The induction hypothesis yields that  $\vdash t_1[t'/X] \equiv t_{1\alpha}[t'_\alpha/X]$ .

Since  $t[t'/X] = l\langle t_1 \rangle[t'/X] = l\langle t_1[t'/X] \rangle$  and  $t_\alpha[t'_\alpha/X] = l\langle t_{1\alpha} \rangle[t'_\alpha/X] = l\langle t_{1\alpha}[t'_\alpha/X] \rangle$ , we only need to prove that  $\vdash l\langle t_1[t'/X] \rangle \equiv l\langle t_{1\alpha}[t'_\alpha/X] \rangle$  and this is done using the **equiv-tree** rule.

Case:  $t = t_1 t_2$

In this case  $t_\alpha = t_{1\alpha} t_{2\alpha}$  where  $\vdash t_1 \equiv t_{1\alpha}$  and  $\vdash t_2 \equiv t_{2\alpha}$ .

The induction hypothesis yields that  $\vdash t_1[t'/X] \equiv t_{1\alpha}[t'_\alpha/X]$  and  $\vdash t_2[t'/X] \equiv t_{2\alpha}[t'_\alpha/X]$ .

Since  $t[t'/X] = t_1 t_2[t'/X] = (t_1[t'/X])(t_2[t'/X])$  and  $t_\alpha[t'_\alpha/X] = t_{1\alpha} t_{2\alpha}[t'_\alpha/X] = (t_{1\alpha}[t'_\alpha/X])(t_{2\alpha}[t'_\alpha/X])$ , we only need to prove that  $\vdash (t_1[t'/X])(t_2[t'/X]) \equiv (t_{1\alpha}[t'_\alpha/X])(t_{2\alpha}[t'_\alpha/X])$  and this is done using the **equiv-seq** rule.

Case:  $t = t_1 + t_2$

In this case  $t_\alpha = t_{1\alpha} + t_{2\alpha}$  where  $\vdash t_1 \equiv t_{1\alpha}$  and  $\vdash t_2 \equiv t_{2\alpha}$ .

The induction hypothesis yields that  $\vdash t_1[t'/X] \equiv t_{1\alpha}[t'_\alpha/X]$  and  $\vdash t_2[t'/X] \equiv t_{2\alpha}[t'_\alpha/X]$ .

Since  $t[t'/X] = t_1 + t_2[t'/X] = (t_1[t'/X]) + (t_2[t'/X])$  and  $t_\alpha[t'_\alpha/X] = t_{1\alpha} + t_{2\alpha}[t'_\alpha/X] = (t_{1\alpha}[t'_\alpha/X]) + (t_{2\alpha}[t'_\alpha/X])$ , we only need to prove that  $\vdash (t_1[t'/X]) + (t_2[t'/X]) \equiv (t_{1\alpha}[t'_\alpha/X]) + (t_{2\alpha}[t'_\alpha/X])$  and this is done using the **equiv-sum** rule.

Case:  $t = \mu Y. t_1$

In this case  $t_\alpha = \mu Z.t_{1\alpha}$  where  $\vdash t_1[X'/Y] \equiv t_{1\alpha}[X'/Z]$  where  $X' \notin \text{fv}(\mu Y.t_1) \cup \text{fv}(\mu Z.t_{1\alpha})$ . There are two subcases.

If  $X = X'$  then  $X \notin \text{fv}(\mu Y.t_1) \cup \text{fv}(\mu Z.t_{1\alpha})$ .

This means that  $\vdash \mu Y.t_1[t'/X] \equiv \mu Y.t_1$  and  $\vdash \mu Z.t_{1\alpha}[t'_\alpha/X] \equiv \mu Z.t_{1\alpha}$ .

Since we have that  $\vdash \mu Y.t_1 \equiv \mu Z.t_{1\alpha}$  it follows from two applications of the transitivity of  $\equiv$  that  $\vdash \mu Y.t_1[t'/X] \equiv \mu Z.t_{1\alpha}[t'_\alpha/X]$ .

If  $X \neq X'$  then we can use the induction hypothesis three times to obtain that

$\vdash t_1[X'/Y][Y'/X'] [Z'/Y'] [X'/Z'] \equiv t_{1\alpha}[X'/Z] [Y'/X'] [Z'/Y'] [X'/Z']$  where  $Y', Z' \notin \text{fv}(t_1) \cup \text{fv}(t_{1\alpha}) \cup \text{fv}(t') \cup \{X, X', Y, Z\}$ .

It is clear that

$\vdash t_1[X'/Y][Y'/X'] [Z'/Y'] [X'/Z'] \equiv t_1[Y'/Y][X'/Y']$  and

$\vdash t_{1\alpha}[X'/Z][Y'/X'] [Z'/Y'] [X'/Z'] \equiv t_{1\alpha}[Z'/Z][X'/Z']$ .

Therefore the transitivity of  $\equiv$  yields that  $\vdash t_1[Y'/Y][X'/Y'] \equiv t_{1\alpha}[Z'/Z][X'/Z']$ .

Now the induction hypothesis yields that  $\vdash t_1[Y'/Y][X'/Y'] [t'/X] \equiv t_{1\alpha}[Z'/Z][X'/Z'] [t'_\alpha/X]$ .

We can now see that

$\vdash t_1[Y'/Y][X'/Y'] [t'/X] \equiv t_1[Y'/Y][t'/X][X'/Y']$  and

$\vdash t_{1\alpha}[Z'/Z][X'/Z'] [t'_\alpha/X] \equiv t_{1\alpha}[Z'/Z][t'_\alpha/X][X'/Z']$ , and therefore rule **equiv-mu** yields that  $\vdash \mu Y'.(t_1[Y'/Y][t'/X]) \equiv \mu Z'.(t_{1\alpha}[Z'/Z][t'_\alpha/X])$ .

Therefore the lemma is fulfilled in this case.

Case:  $t = Y$

In this case  $t_\alpha = Y$ .

If  $X = Y$  then  $t[t'/X] = t'$  and  $t_\alpha[t'_\alpha/X] = t'_\alpha$ .

Therefore the lemma is fulfilled in this case.

If  $X \neq Y$  then  $t[t'/X] = Y$  and  $t_\alpha[t'_\alpha/X] = Y$  and therefore rule **equiv-var** yields that the lemma is fulfilled in this case.

We have now proved the lemma for all the cases and can conclude that

$$\vdash t \equiv t_\alpha \wedge \vdash t' \equiv t'_\alpha \Rightarrow \vdash t[t'/X] \equiv t_\alpha[t'_\alpha/X].$$

■

### Proof 19.2 (Commutation of substitutions)

We will now prove Lemma 3.4 which states that

$$X \neq Y \wedge X \notin \text{fv}(t_Y) \Rightarrow t[t_X/X][t_Y/Y] = t[t_Y/Y][t_X[t_Y/Y]/X]$$

We will prove this by induction on  $|t|$ .

*Induction Hypothesis:*

If  $|t_1| < |t|$  then we can assume that  $t_1[t_X/X][t_Y/Y] = t_1[t_Y/Y][t_X[t_Y/Y]/X]$ .

Case:  $t = 0$

In this case

$$\begin{aligned}
& 0[t_X/X][t_Y/Y] \\
&= 0[t_Y/Y] \\
&= 0 \\
&= 0[t_Y/Y] \\
&= 0[t_Y/Y][t_X[t_Y/Y]/X]
\end{aligned}$$

Therefore the lemma is fulfilled in this case.

Case:  $t = 1$

In this case

$$\begin{aligned}
& 1[t_X/X][t_Y/Y] \\
&= 1[t_Y/Y] \\
&= 1 \\
&= 1[t_Y/Y] \\
&= 1[t_Y/Y][t_X[t_Y/Y]/X]
\end{aligned}$$

Therefore the lemma is fulfilled in this case.

Case:  $t = Z$

There are three sub-cases.

If  $Z = X$  then

$$\begin{aligned}
& X[t_X/X][t_Y/Y] \\
&= t_X[t_Y/Y] \\
&= X[t_X[t_Y/Y]/X] \\
(X \neq Y) &= X[t_Y/Y][t_X[t_Y/Y]/X]
\end{aligned}$$

If  $Z = Y$  then

$$\begin{aligned}
& Y[t_X/X][t_Y/Y] \\
(X \neq Y) &= Y[t_Y/Y] \\
&= t_Y \\
(X \notin \text{fv}(t_Y)) &= t_Y[t_X[t_Y/Y]/X] \\
&= Y[t_Y/Y][t_X[t_Y/Y]/X]
\end{aligned}$$

If  $Z \notin \{X, Y\}$  then

$$\begin{aligned}
& t[t_X/X][t_Y/Y] \\
&= Z[t_X/X][t_Y/Y] \\
(Z \neq X) &= Z[t_Y/Y] \\
(Z \neq Y) &= Z \\
(Z \neq Y) &= Z[t_Y/Y] \\
(Z[t_Y/Y] = Z \neq X) &= Z[t_Y/Y][t_X[t_Y/Y]/X] \\
&= t[t_Y/Y][t_X[t_Y/Y]/X]
\end{aligned}$$

Therefore the lemma is fulfilled in this case.

Case:  $t = l\langle t_1 \rangle$

In this case

$$\begin{aligned}
& l\langle t_1 \rangle[t_X/X][t_Y/Y] \\
&= l\langle t_1[t_X/X][t_Y/Y] \rangle \\
(\mathbf{IH}) &= l\langle t_1[t_Y/Y][t_X[t_Y/Y]/X] \rangle \\
&= l\langle t_1 \rangle[t_Y/Y][t_X[t_Y/Y]/X]
\end{aligned}$$

Therefore the lemma is fulfilled in this case.

Case:  $t = t_1 t_2$

In this case

$$\begin{aligned}
& t_1 t_2[t_X/X][t_Y/Y] \\
&= (t_1[t_X/X][t_Y/Y])(t_2[t_X/X][t_Y/Y]) \\
(2 \times \mathbf{IH}) &= (t_1[t_Y/Y][t_X[t_Y/Y]/X])(t_2[t_Y/Y][t_X[t_Y/Y]/X]) \\
&= t_1 t_2[t_Y/Y][t_X[t_Y/Y]/X]
\end{aligned}$$

Therefore the lemma is fulfilled in this case.

Case:  $t = t_1 + t_2$

In this case

$$\begin{aligned}
& t_1 + t_2[t_X/X][t_Y/Y] \\
&= (t_1[t_X/X][t_Y/Y]) + (t_2[t_X/X][t_Y/Y]) \\
(2 \times \mathbf{IH}) &= (t_1[t_Y/Y][t_X[t_Y/Y]/X]) + (t_2[t_Y/Y][t_X[t_Y/Y]/X]) \\
&= t_1 + t_2[t_Y/Y][t_X[t_Y/Y]/X]
\end{aligned}$$

Therefore the lemma is fulfilled in this case.

Case:  $t = \mu Z.t_1$

In this case

$$\begin{aligned}
& \mu Z.t_1[t_X/X][t_Y/Y] \\
&= \mu Z'.(t_1[Z'/Z][t_X/X][t_Y/Y]) \quad \text{where } Z' \notin \text{fv}(t_X) \cup \text{fv}(\mu Z.t_1) \cup \{X\} \\
&= \mu Z''.(t_1[Z'/Z][t_X/X][Z''/Z'][t_Y/Y]) \quad \text{where } Z'' \notin \dots \\
&= \mu Z'''.(t_1[Z'/Z][t_X/X][Z''/Z'][t_Y/Y][Z'''/Z'']) \quad \text{where } Z''' \notin \dots \\
(Z''' \notin \{X, Y\}) &= \mu Z'''.(t_1[Z'''/Z][t_X/X][t_Y/Y]) \\
(\mathbf{IH}) &= \mu Z'''.(t_1[Z'''/Z][t_Y/Y][t_X[t_Y/Y]/X]) \\
&= \mu Z'''.(t_1[Z'''/Z][t_Y/Y][Z'/Z''] [t_X[t_Y/Y]/X][Z'''/Z']) \\
&= \mu Z'.(t_1[Z''/Z][t_Y/Y][Z'/Z''] [t_X[t_Y/Y]/X]) \\
(Z'' \notin \text{fv}(t_X)) &= \mu Z''.(t_1[Z''/Z][t_Y/Y][t_X[t_Y/Y]/X]) \\
(Z' \notin \text{fv}(t_Y)) &= \mu Z.t_1[t_Y/Y][t_X[t_Y/Y]/X]
\end{aligned}$$

Therefore the lemma is fulfilled in this case.

We have now proved the lemma for all the cases, and we can therefore conclude that

$$X \neq Y \wedge X \notin \text{fv}(t_Y) \Rightarrow t[t_X/X][t_Y/Y] = t[t_Y/Y][t_X[t_Y/Y]/X].$$

■

### Proof 19.3 (Soundness of tag)

We will now prove Theorem 6.6 which states that

$$\text{tag}(t, d) \neq \top \Rightarrow \vdash \text{tag}(t, d) \text{ inhabits } t$$

by induction on the calltree of  $\text{tag}(t, d)$ .

Induction Hypothesis:

If  $\text{tag}(t, d)$  calls  $\text{tag}(t', d')$  and  $\text{tag}(t', d') \neq \top$  then we can assume that  $\vdash \text{tag}(t', d') \text{ inhabits } t'$ .

Case:  $\text{tag}(1, \bullet) = \varepsilon$

In this case we need to prove that  $\vdash \varepsilon \text{ inhabits } 1$ , but this follows from the rule *in-1*.

Case:  $\text{tag}(l\langle t_1 \rangle, \langle d_1 \rangle) = \text{let } x_1 = \text{tag}(t_1, d_1) \text{ in } \langle l \rangle x_1 \langle /l \rangle \varepsilon$

In this case  $\text{tag}(t, d)$  calls  $\text{tag}(t_1, d_1)$ .

If  $\text{tag}(t_1, d_1) = \top$  then  $\text{tag}(l\langle t_1 \rangle, \langle d_1 \rangle) = \top$ , and the theorem is trivially fulfilled.

If  $\text{tag}(t_1, d_1) \neq \top$  then the induction hypothesis yields that  $\vdash \text{tag}(t_1, d_1) \text{ inhabits } t_1$  and therefore rule *in-tree* yields that  $\vdash \langle l \rangle \text{tag}(t_1, d_1) \langle /l \rangle \varepsilon \text{ inhabits } l\langle t_1 \rangle$  and therefore the theorem is fulfilled in this case.

Case:  $\text{tag}(t_1 t_2, d_1 d_2) = \text{let } x_1 = \text{tag}(t_1, d_1), x_2 = \text{tag}(t_2, d_2) \text{ in } x_1 x_2 \text{ end}$

In this case  $\text{tag}(t, d)$  calls  $\text{tag}(t_1, d_1)$  and  $\text{tag}(t_2, d_2)$ .

If  $\text{tag}(t_1, d_1) = \top$  or  $\text{tag}(t_2, d_2) = \top$  then  $\text{tag}(t_1 t_2, d_1 d_2) = \top$ , and the theorem is trivially fulfilled.

If  $\text{tag}(t_1, d_1) \neq \top$  and  $\text{tag}(t_2, d_2) \neq \top$  then the induction hypothesis yields that  $\vdash \text{tag}(t_1, d_1) \text{ inhabits } t_1$  and  $\vdash \text{tag}(t_2, d_2) \text{ inhabits } t_2$  and therefore rule *in-seq* yields that

$\vdash \mathbf{tag}(t_1, d_1) \mathbf{tag}(t_2, d_2) \mathbf{inhabits} t_1 t_2$ , and therefore the theorem is fulfilled in this case.

Case:  $\mathbf{tag}(t_1 + t_2, d_1 \diamond) = \mathbf{tag}(t_1, d_1)$

In this case  $\mathbf{tag}(t, d)$  calls  $\mathbf{tag}(t_1, d_1)$ .

If  $\mathbf{tag}(t_1, d_1) = \top$  then  $\mathbf{tag}(t_1 + t_2, d_1 \diamond) = \top$ , and the theorem is trivially fulfilled.

If  $\mathbf{tag}(t_1, d_1) \neq \top$  then the induction hypothesis yields that  $\vdash \mathbf{tag}(t_1, d_1) \mathbf{inhabits} t_1$  and therefore rule *in-sum1* yields that  $\vdash \mathbf{tag}(t_1, d_1) \mathbf{inhabits} t_1 + t_2$  and therefore the theorem is fulfilled in this case.

Case:  $\mathbf{tag}(t_1 + t_2, \diamond d_2) = \mathbf{tag}(t_2, d_2)$

In this case  $\mathbf{tag}(t, d)$  calls  $\mathbf{tag}(t_2, d_2)$ .

If  $\mathbf{tag}(t_2, d_2) = \top$  then  $\mathbf{tag}(t_1 + t_2, \diamond d_2) = \top$ , and the theorem is trivially fulfilled.

If  $\mathbf{tag}(t_2, d_2) \neq \top$  then the induction hypothesis yields that  $\vdash \mathbf{tag}(t_2, d_2) \mathbf{inhabits} t_2$  and therefore rule *in-sum2* yields that  $\vdash \mathbf{tag}(t_2, d_2) \mathbf{inhabits} t_1 + t_2$  and therefore the theorem is fulfilled in this case.

Case:  $\mathbf{tag}(\mu X.t_1, d) = \mathbf{tag}(t_1[\mu X.t_1/X], d)$

In this case  $\mathbf{tag}(t, d)$  calls  $\mathbf{tag}(t_1[\mu X.t_1/X], d)$ .

If  $\mathbf{tag}(t_1[\mu X.t_1/X], d) = \top$  then  $\mathbf{tag}(\mu X.t_1, d) = \top$ , and the theorem is trivially fulfilled.

If  $\mathbf{tag}(t_1[\mu X.t_1/X], d) \neq \top$  then the induction hypothesis yields that

$\vdash \mathbf{tag}(t_1[\mu X.t_1/X], d) \mathbf{inhabits} t_1[\mu X.t_1/X]$  and therefore rule *in-mu* yields that

$\vdash \mathbf{tag}(t_1[\mu X.t_1/X], d) \mathbf{inhabits} \mu X.t_1$  and therefore the theorem is fulfilled in this case.

Case:  $\mathbf{tag}(-, -) = \top$

in this case  $\mathbf{tag}(t, d) = \top$  and therefore the theorem is trivially fulfilled in this case.

We have now proved all the cases, and we can therefore conclude that

$$\mathbf{tag}(t, d) \neq \top \Rightarrow \vdash \mathbf{tag}(t, d) \mathbf{inhabits} t.$$

■

#### Proof 19.4 (Completeness of $\mathbf{tag}$ )

We will now prove Theorem 6.7 which states that

$$\vdash x \mathbf{inhabits} t \Rightarrow \exists d. \mathbf{tag}(t, d) = x$$

by induction on the derivation of  $\vdash x \mathbf{inhabits} t$ .

*Induction Hypothesis:*

If the derivation of  $\vdash x \mathbf{inhabits} t$  has a true sub-derivation of  $\vdash x' \mathbf{inhabits} t'$  we can assume that there is a  $d'$  such that  $\mathbf{tag}(t', d') = x'$ .

We will now prove the theorem for all derivations, by dividing them by the rule used in the root-node (rule induction).

Case: *in-1*  $\frac{}{\vdash \varepsilon \mathbf{inhabits} 1}$

In this case we can use  $d = e$  since  $\text{tag}(1, \bullet) = \varepsilon$ .

$$\text{Case: in-tree} \frac{\vdash x_1 \text{ inhabits } t_1}{\vdash \langle l \rangle x_1 \langle /l \rangle \varepsilon \text{ inhabits } l \langle t_1 \rangle}$$

In this case, the induction hypothesis yields that there is a  $d_1$  such that  $\text{tag}(t_1, d_1) = x_1$ .

We can now use  $d = \langle d_1 \rangle$  since

$\text{tag}(l \langle t_1 \rangle, \langle d_1 \rangle) = \text{let } x_1 = \text{tag}(t_1, d_1) \text{ in } \langle l \rangle x_1 \langle /l \rangle \varepsilon \text{ end} = \langle l \rangle x_1 \langle /l \rangle \varepsilon$ , and therefore the theorem is fulfilled in this case.

$$\text{Case: in-seq} \frac{\vdash x_1 \text{ inhabits } t_1 \quad \vdash x_2 \text{ inhabits } t_2}{\vdash x_1 x_2 \text{ inhabits } t_1 t_2}$$

In this case the induction hypothesis yields that there is  $d_1$  and  $d_2$  such that  $\text{tag}(t_1, d_1) = x_1$  and  $\text{tag}(t_2, d_2) = x_2$ .

We can now use  $d = d_1 d_2$  since

$\text{tag}(t_1 t_2, d_1 d_2) = \text{let } x_1 = \text{tag}(t_1, d_1), x_2 = \text{tag}(t_2, d_2) \text{ in } x_1 x_2 \text{ end} = x_1 x_2$ , and therefore the theorem is fulfilled in this case.

$$\text{Case: in-sum1} \frac{\vdash x_1 \text{ inhabits } t_1}{\vdash x_1 \text{ inhabits } t_1 + t_2}$$

In this case the induction hypothesis yields that there is a  $d_1$  such that  $\text{tag}(t_1, d_1) = x_1$ .

We can now use  $d = d_1 \diamond$  since  $\text{tag}(t_1 + t_2, d_1 \diamond) = \text{tag}(t_1, d_1) = x_1$ , and therefore the theorem is fulfilled in this case.

$$\text{Case: in-sum2} \frac{\vdash x_2 \text{ inhabits } t_1}{\vdash x_2 \text{ inhabits } t_1 + t_2}$$

In this case the induction hypothesis yields that there is a  $d_2$  such that  $\text{tag}(t_2, d_2) = x_2$ .

We can now use  $d = \diamond d_2$  since  $\text{tag}(t_1 + t_2, \diamond d_2) = \text{tag}(t_2, d_2) = x_2$ , and therefore the theorem is fulfilled in this case.

$$\text{Case: in-mu} \frac{\vdash x \text{ inhabits } t_1[\mu X.t_1/X]}{\vdash x \text{ inhabits } \mu X.t_1}$$

In this case the induction hypothesis yields that there is a  $d_1$  such that  $\text{tag}(t_1[\mu X.t_1/X], d_1) = x$ .

We can now use  $d = d_1$  since  $\text{tag}(\mu X.t_1, d_1) = \text{tag}(t_1[\mu X.t_1/X], d_1) = x$ , and therefore the theorem is fulfilled in this case.

We have now proved the theorem for all the cases, and we can therefore conclude that

$$\vdash x \text{ inhabits } t \Rightarrow \exists d. \text{tag}(t, d) = x.$$

■

### Proof 19.5

We will now prove Lemma 7.4 which states that

$$\text{tag}(t, d) \neq \top \Rightarrow \text{tag}(t, d) = \text{tag}(t[t'/X], d)$$

by induction on the calltree of  $\text{tag}(t, d)$ .

*Induction Hypothesis:*

If  $\mathbf{tag}(t, d)$  calls  $\mathbf{tag}(t_1, d_1)$  and  $\mathbf{tag}(t_1, d_1) \neq \top$  then we can assume that  $\mathbf{tag}(t_1, d_1) = \mathbf{tag}(t_1[t'/X], d_1)$ .

Case:  $\mathbf{tag}(1, \bullet) = \varepsilon$

In this case  $t[t'/X] = 1[t'/X] = 1 = t$  and therefore we have that  $\mathbf{tag}(t, d) = \mathbf{tag}(t[t'/X], d)$  since  $\mathbf{tag}$  is a function.

Case:  $\mathbf{tag}(l\langle t_1 \rangle, \langle d_1 \rangle) = \mathbf{let } x_1 = \mathbf{tag}(t_1, d_1) \mathbf{ in } \langle l \rangle x_1 \langle /l \rangle \varepsilon \mathbf{ end}$ .

In this case the induction hypothesis yields that  $\mathbf{tag}(t_1, d_1) = \mathbf{tag}(t_1[t'/X], d_1)$  and therefore

$$\begin{aligned} \mathbf{tag}(l\langle t_1 \rangle, \langle d_1 \rangle) &= \mathbf{let } x_1 = \mathbf{tag}(t_1, d_1) \mathbf{ in } \langle l \rangle x_1 \langle /l \rangle \varepsilon \mathbf{ end} \\ (\mathbf{IH}) &= \mathbf{let } x_1 = \mathbf{tag}(t_1[t'/X], d_1) \mathbf{ in } \langle l \rangle x_1 \langle /l \rangle \varepsilon \mathbf{ end} \\ &= \mathbf{tag}(l\langle t_1[t'/X] \rangle, \langle d_1 \rangle) \\ &= \mathbf{tag}(l\langle t_1 \rangle[t'/X], \langle d_1 \rangle) \\ &= \mathbf{tag}(t[t'/X], \langle d_1 \rangle). \end{aligned}$$

Therefore the lemma is fulfilled in this case.

Case:  $\mathbf{tag}(t_1 t_2, d_1 d_2) = \mathbf{let } x_1 = \mathbf{tag}(t_1, d_1), x_2 = \mathbf{tag}(t_2, d_2) \mathbf{ in } x_1 x_2 \mathbf{ end}$

In this case the induction hypothesis yields that

$\mathbf{tag}(t_1, d_1) = \mathbf{tag}(t_1[t'/X], d_1)$  and

$\mathbf{tag}(t_2, d_2) = \mathbf{tag}(t_2[t'/X], d_2)$  and therefore

$$\begin{aligned} \mathbf{tag}(t_1 t_2, d_1 d_2) &= \mathbf{let } x_1 = \mathbf{tag}(t_1, d_1), x_2 = \mathbf{tag}(t_2, d_2) \mathbf{ in } x_1 x_2 \mathbf{ end} \\ (2 \times \mathbf{IH}) &= \mathbf{let } x_1 = \mathbf{tag}(t_1[t'/X], d_1), x_2 = \mathbf{tag}(t_2[t'/X], d_2) \mathbf{ in } x_1 x_2 \mathbf{ end} \\ &= \mathbf{tag}((t_1[t'/X])(t_2[t'/X]), d_1 d_2) \\ &= \mathbf{tag}(t_1 t_2[t'/X], d_1 d_2) \\ &= \mathbf{tag}(t[t'/X], d_1 d_2). \end{aligned}$$

Therefore the lemma is fulfilled in this case.

Case:  $\mathbf{tag}(t_1 + t_2, d_1 \diamond) = \mathbf{tag}(t_1, d_1)$

In this case the induction hypothesis yields that

$\mathbf{tag}(t_1, d_1) = \mathbf{tag}(t_1[t'/X], d_1)$  and therefore

$$\begin{aligned} \mathbf{tag}(t_1 + t_2, d_1 \diamond) &= \mathbf{tag}(t_1, d_1) \\ (\mathbf{IH}) &= \mathbf{tag}(t_1[t'/X], d_1) \\ &= \mathbf{tag}((t_1[t'/X]) + (t_2[t'/X]), d_1 \diamond) \\ &= \mathbf{tag}(t_1 + t_2[t'/X], d_1 \diamond) \\ &= \mathbf{tag}(t[t'/X], d_1 \diamond). \end{aligned}$$

Therefore the lemma is fulfilled in this case.

Case:  $\mathbf{tag}(t_1 + t_2, \diamond d_2) = \mathbf{tag}(t_2, d_2)$

In this case the induction hypothesis yields that  $\mathbf{tag}(t_2, d_2) = \mathbf{tag}(t_2[t'/X], d_2)$  and therefore

$$\begin{aligned}
\mathbf{tag}(t_1 + t_2, \diamond d_1) &= \mathbf{tag}(t_2, d_2) \\
(\mathbf{IH}) &= \mathbf{tag}(t_2[t'/X], d_2) \\
&= \mathbf{tag}((t_1[t'/X]) + (t_2[t'/X]), \diamond d_2) \\
&= \mathbf{tag}(t_1 + t_2[t'/X], \diamond d_2) \\
&= \mathbf{tag}(t[t'/X], \diamond d_2).
\end{aligned}$$

Therefore the lemma is fulfilled in this case.

Case:  $\mathbf{tag}(\mu Y.t_1, d) = \mathbf{tag}(t_1[\mu Y.t_1/Y], d)$

In this case the induction hypothesis yields that  $\mathbf{tag}(t_1[\mu Y.t_1/Y], d) = \mathbf{tag}(t_1[\mu Y.t_1/Y][t'/X], d)$  and therefore

$$\begin{aligned}
&\mathbf{tag}(\mu Y.t_1[t'/X], d) \\
&= \mathbf{tag}(\mu Z.(t_1[Z/Y][t'/X]), d) \\
&= \mathbf{tag}(t_1[Z/Y][t'/X][\mu Z.(t_1[Z/Y][t'/X])/Z]) \\
&= \mathbf{tag}(t_1[Z/Y][t'/X][\mu Y.t_1[t'/X]/Z]) \\
(\mathbf{Lemma\ 3.4}) &= \mathbf{tag}(t_1[Z/Y][\mu Y.t_1/Z][t'/X], d) \\
&= \mathbf{tag}(t_1[\mu Y.t_1/Y][t'/X], d) \\
(\mathbf{IH}) &= \mathbf{tag}(t_1[\mu Y.t_1/Y], d) \\
&= \mathbf{tag}(\mu Y.t_1, d).
\end{aligned}$$

Therefore the lemma is fulfilled in this case.

Case:  $\mathbf{tag}(-, -) = \top$

In this case  $\mathbf{tag}(t, d) = \top$  and therefore the lemma is trivially fulfilled in this case.

We have now proved the lemma for all the cases and we can therefore conclude that

$$\mathbf{tag}(t, d) \neq \top \Rightarrow \mathbf{tag}(t, d) = \mathbf{tag}(t[t'/X], d).$$

■

### Proof 19.6 (Soundness of $\mathbf{esp}$ )

We will now prove Lemma 7.5 which states that

$$\forall t \in \mathbf{XMLType}. \forall d \in \mathbf{esp}(t). \mathbf{tag}(t, d) = \varepsilon$$

by structural induction on  $t$ .

*Induction Hypothesis:*

If  $t'$  is a sub-term of  $t$  then we can assume that  $\forall d' \in \mathbf{esp}(t'). \mathbf{tag}(t', d') = \varepsilon$ .

Case:  $t = 0$

In this case  $\mathbf{esp}(t) = \{\}$  so the lemma is vacantly fulfilled in this case.

Case:  $t = 1$

In this case  $\mathbf{esp}(t) = \{\bullet\}$ , so we only need to prove that  $\mathbf{tag}(1, \bullet) = \varepsilon$ .

This is however the first case in the definition of  $\mathbf{tag}$ , so the lemma is fulfilled in this case.

Case:  $t = l\langle t_1 \rangle$

In this case  $\mathbf{esp}(t) = \{\}$  so the lemma is vacantly fulfilled in this case.

Case:  $t = t_1 t_2$

In this case  $\mathbf{esp}(t) = \{d_1 d_2 \mid d_1 \in \mathbf{esp}(t_1) \wedge d_2 \in \mathbf{esp}(t_2)\}$ .

Let  $d_1 \in \mathbf{esp}(t_1)$  and  $d_2 \in \mathbf{esp}(t_2)$  be chosen.

The induction hypothesis yields that  $\mathbf{tag}(t_1, d_1) = \varepsilon$  and  $\mathbf{tag}(t_2, d_2) = \varepsilon$ , and this means that  $\mathbf{tag}(t_1 t_2, d_1 d_2) = \varepsilon \varepsilon = \varepsilon$ .

We can therefore conclude that

$\forall d \in \{d_1 d_2 \mid d_1 \in \mathbf{esp}(t_1) \wedge d_2 \in \mathbf{esp}(t_2)\} = \mathbf{esp}(t_1 t_2). \mathbf{tag}(t_1 t_2, d) = \varepsilon$ , so the lemma is fulfilled in this case.

Case:  $t = t_1 + t_2$

In this case  $\mathbf{esp}(t) \subseteq \{d_1 \diamond \mid d_1 \in \mathbf{esp}(t_1)\} \cup \{\diamond d_2 \mid d_2 \in \mathbf{esp}(t_2)\}$ .

If  $d \in \mathbf{esp}(t)$  there are two cases.

If  $d \in \{d_1 \diamond \mid d_1 \in \mathbf{esp}(t_1)\}$  then we need to prove that  $\mathbf{tag}(t_1 + t_2, d_1 \diamond) = \varepsilon$ , but since  $\mathbf{tag}(t_1 + t_2, d_1 \diamond) = \mathbf{tag}(t_1, d_1)$  this follows from the induction hypothesis.

If  $d \in \{\diamond d_2 \mid d_2 \in \mathbf{esp}(t_2)\}$  then we need to prove that  $\mathbf{tag}(t_1 + t_2, \diamond d_2) = \varepsilon$ , but since  $\mathbf{tag}(t_1 + t_2, \diamond d_2) = \mathbf{tag}(t_2, d_2)$  this follows from the induction hypothesis.

Therefore the lemma is fulfilled in this case.

Case:  $t = \mu X.t_1$

In this case  $\mathbf{esp}(t) = \mathbf{esp}(t_1)$ .

The induction hypothesis yields that  $\forall d \in \mathbf{esp}(t_1). \mathbf{tag}(t_1, d) = \varepsilon$ . Since  $\mathbf{tag}(t_1, d) = \varepsilon \neq \top$  Lemma 7.4 yields that  $\forall d \in \mathbf{esp}(t). \mathbf{tag}(t_1[\mu X.t_1/X], d) = \varepsilon$  and therefore

$\forall d \in \mathbf{esp}(t). \mathbf{tag}(t, d) = \varepsilon$ .

Case:  $t = X$

In this case  $\mathbf{esp}(t) = \{\}$ , so the lemma is vacantly fulfilled.

This covers all the cases, and we can therefore conclude that

$$\forall t \in \mathbf{XMLType}. \forall d \in \mathbf{esp}(t). \mathbf{tag}(t, d) = \varepsilon$$

■

### Proof 19.7

We will now prove Lemma 7.6 which states that

$$\mathbf{tag}(t[t'/X], d) = \varepsilon \Rightarrow \mathbf{tag}(t, d) = \varepsilon \vee \exists d'. \mathbf{tag}(t', d') = \varepsilon \wedge |d'| \leq |d|$$

by induction on the calltree of  $\text{tag}(t[t'/X], d)$ .

*Induction Hypothesis:*

If  $\text{tag}(t[t'/X], d)$  calls  $\text{tag}(t_1[t'/X], d_1)$  and  $\text{tag}(t_1[t'/X], d_1) = \varepsilon$  then we can assume that  $\text{tag}(t_1, d_1) = \varepsilon$  or  $\exists d'. \text{tag}(t', d') = \varepsilon$  and  $|d'| \leq |d_1|$ .

Case:  $t = 0$

In this case  $\text{tag}(t[t'/X], d) = \text{tag}(0[t'/X], d) = \text{tag}(0, d) = \top$ .

Therefore the lemma is trivially fulfilled in this case.

Case:  $t = 1$

If  $d \neq \bullet$  then  $\text{tag}(t[t'/X], d) = \text{tag}(1[t'/X], d) = \text{tag}(1, d) = \top$  and therefore the lemma is trivially fulfilled.

If  $d = \bullet$  then  $\text{tag}(t[t'/X], d) = \text{tag}(1[t'/X], \bullet) = \text{tag}(1, \varepsilon) = \varepsilon$ .

Since  $\text{tag}(t, d) = \text{tag}(1, \bullet) = \varepsilon$  the lemma is fulfilled in this case.

Case:  $t = l\langle t_1 \rangle$

If  $d \neq \langle d_1 \rangle$  then  $\text{tag}(t[t'/X], d) = \text{tag}(l\langle t_1 \rangle[t'/X], d) = \text{tag}(l\langle t_1[t'/X] \rangle, d) = \top$  and therefore the lemma is trivially fulfilled.

If  $d = \langle d_1 \rangle$  then

$$\begin{aligned} & \text{tag}(t[t'/X], d) \\ &= \text{tag}(l\langle t_1 \rangle[t'/X], \langle d_1 \rangle) \\ &= \text{tag}(l\langle t_1[t'/X] \rangle, \langle d_1 \rangle) \\ &= \text{let } x_1 = \text{tag}(t_1[t'/X], d_1) \text{ in } \langle 1 \rangle x_1 \langle /1 \rangle \varepsilon \text{ end} \\ &\neq \varepsilon. \end{aligned}$$

Therefore the lemma is trivially fulfilled in this case.

Case:  $t = t_1 t_2$

If  $d \neq d_1 d_2$  then  $\text{tag}(t[t'/X], d) = \text{tag}(t_1 t_2[t'/X], d) = \top$  and therefore the lemma is trivially fulfilled.

If  $d = d_1 d_2$  then

$\text{tag}(t[t'/X], d) = \text{tag}(t_1 t_2[t'/X], d_1 d_2) = \text{let } x_1 = \text{tag}(t_1[t'/X], d_1), x_2 = \text{tag}(t_2[t'/X], d_2) \text{ in } x_1 x_2 \text{ end}$ .

If  $x_1 x_2 = \varepsilon$  then  $x_1 = x_2 = \varepsilon$  and therefore the induction hypothesis yields that

$\text{tag}(t_1, d_1) = \varepsilon \vee \exists d'. \text{tag}(t', d') = \varepsilon \wedge |d'| \leq |d_1|$  and

$\text{tag}(t_2, d_2) = \varepsilon \vee \exists d'. \text{tag}(t', d') = \varepsilon \wedge |d'| \leq |d_2|$ .

Therefore if there is not a  $d'. \text{tag}(t', d') = \varepsilon \wedge |d'| \leq |d|$  then  $\text{tag}(t_1, d_1) = \text{tag}(t_2, d_2) = \varepsilon$ .

This means that

$$\begin{aligned}
& \mathbf{tag}(t_1 t_2, d_1 d_2) \\
&= \mathbf{let } x_1 = \mathbf{tag}(t_1, d_1), x_2 = \mathbf{tag}(t_2, d_2) \mathbf{ in } x_1 x_2 \mathbf{ end} \\
&= \mathbf{let } x_1 = \varepsilon, x_2 = \varepsilon \mathbf{ in } x_1 x_2 \mathbf{ end} \\
&= \varepsilon.
\end{aligned}$$

Therefore the lemma is fulfilled in this case.

Case:  $t = t_1 + t_2$

If  $d \neq d_1 \diamond$  and  $d \neq \diamond d_2$  then  $\mathbf{tag}(t[t'/X], d) = \mathbf{tag}(t_1 + t_2[t'/X], d) = \top$ , and therefore the lemma is trivially fulfilled.

If  $d = d_1 \diamond$  then  $\mathbf{tag}(t[t'/X], d) = \mathbf{tag}(t_1 + t_2[t'/X], d_1 \diamond) = \mathbf{tag}(t_1[t'/X], d_1)$ .

Now the induction hypothesis yields that  $\mathbf{tag}(t_1, d_1) = \varepsilon \vee \exists d'. \mathbf{tag}(t', d') = \varepsilon \wedge |d'| \leq |d_1|$ .

If  $\mathbf{tag}(t_1, d_1) = \varepsilon$  then  $\mathbf{tag}(t, d) = \mathbf{tag}(t_1 + t_2, d_1 \diamond) = \mathbf{tag}(t_1, d_1) = \varepsilon$ .

Therefore the lemma is fulfilled in this case.

The case where  $d = \diamond d_2$  follows by symmetry.

Case:  $t = \mu Y. t_1$

Since

$$\begin{aligned}
& \mathbf{tag}(t[t'/X], d) \\
&= \mathbf{tag}(\mu Y. t_1[t'/X], d) \\
&= \mathbf{tag}(\mu Z'. (t_1[Z'/Y][t'/X]), d) \\
&= \mathbf{tag}(t_1[Z'/Y][t'/X][\mu Z'. (t_1[Z'/Y][t'/X])/Z'], d) \\
&= \mathbf{tag}(t_1[Z'/Y][t'/X][\mu Y. t_1[t'/X]/Z'], d) \\
&\quad (\mathbf{Lemma 3.4}) = \mathbf{tag}(t_1[Z'/Y][\mu Y. t_1/Z'] [t'/X]) \\
&= \mathbf{tag}(t_1[\mu Y. t_1/Y][t'/X])
\end{aligned}$$

Now the induction hypothesis yields that

$\mathbf{tag}(t_1[\mu Y. t_1/Y], d) = \varepsilon \vee \exists d'. \mathbf{tag}(t', d') = \varepsilon \wedge |d'| \leq |d|$ .

If  $\mathbf{tag}(t_1[\mu Y. t_1/Y], d) = \varepsilon$  then

$\mathbf{tag}(\mu Y. t_1, d) = \mathbf{tag}(t_1[\mu Y. t_1/Y], d) = \varepsilon$ .

Therefore the lemma is fulfilled in this case.

Case:  $t = Y$

If  $X \neq Y$  then  $\mathbf{tag}(t[t'/X], d) = \mathbf{tag}(Y[t'/X], d) = \mathbf{tag}(Y, d) = \top$  and therefore the lemma is trivially fulfilled.

If  $X = Y$  then  $\varepsilon = \mathbf{tag}(t[t'/X], d) = \mathbf{tag}(X[t'/X], d) = \mathbf{tag}(t', d)$  and therefore the lemma is fulfilled with  $d' = d$ .

This covers all the cases and we can therefore conclude that

$$\mathbf{tag}(t[t'/X], d) = \varepsilon \Rightarrow \mathbf{tag}(t, d) = \varepsilon \vee \exists d'. \mathbf{tag}(t', d') = \varepsilon \wedge |d'| \leq |d|.$$

■

**Proof 19.8**

We will now prove Lemma 7.7 which states that

$$\mathbf{tag}(\mu X.t_1, d) = \varepsilon \Rightarrow \exists d'. \mathbf{tag}(t_1, d') = \varepsilon.$$

Assume that  $\mathbf{tag}(\mu X.t_1, d) = \varepsilon$ .

We can then select  $d'$  such that  $\mathbf{tag}(\mu X.t_1, d') = \varepsilon$  and  $|d'|$  is minimal.

Since  $\mathbf{tag}(\mu X.t_1, d') = \mathbf{tag}(t_1[\mu X.t_1/X], d')$ .

Now Lemma 7.6 yields that  $\mathbf{tag}(t_1, d') = \varepsilon$  or  $\exists d''. \mathbf{tag}(\mu X.t_1, d'') = \varepsilon \wedge |d''| \leq |d'|$ .

Assume that there is a  $d''$  such that  $\mathbf{tag}(\mu X.t_1, d'') = \varepsilon \wedge |d''| \leq |d'|$ .

Since  $\mu X.t_1$  is contractive,  $t_1$  is of the form  $t'_1 t'_2$ ,  $t'_1 + t'_2$  or  $l\langle t'_1 \rangle$ .

Therefore one iteration of the induction in Lemma 7.6 yields that  $|d''| < |d'|$ .

This means that  $|d''| < |d'|$  and  $\mathbf{tag}(\mu X.t_1, d'') = \mathbf{tag}(t_1[\mu X.t_1/X], d'') = \varepsilon$ , but this is a contradiction since  $|d'|$  was minimal.

We can therefore conclude that  $\mathbf{tag}(t_1, d') = \varepsilon$ . ■

**Proof 19.9 (Completeness of esp)**

We will now prove Lemma 7.8 which states that

$$\forall t \in \mathbf{XMLType}. \varepsilon \in \mathbf{in}[t] \Rightarrow \mathbf{esp}(t) \neq \{\}$$

by structural induction on  $t$ .

*Induction Hypothesis:*

If  $t'$  is a sub-term of  $t$  then we can assume that

$$\varepsilon \in \mathbf{in}[t'] \Rightarrow \mathbf{esp}(t') \neq \{\}.$$

*Case:  $t = 0$*

Since  $\varepsilon \notin \{\} = \mathbf{in}[0]$ , the lemma is trivially fulfilled in this case.

*Case:  $t = 1$*

In this case  $\mathbf{esp}(t) = \{\bullet\} \neq \{\}$ , so the lemma is fulfilled in this case.

*Case:  $t = l\langle t_1 \rangle$*

In this case  $\mathbf{in}[t] = \{\langle l \rangle x_1 \langle /l \rangle \varepsilon \mid x_1 \in \mathbf{in}[t_1]\}$ . This means that  $\varepsilon \notin \mathbf{in}[t]$ , so the lemma is trivially fulfilled in this case.

*Case:  $t = t_1 t_2$*

Since  $\mathbf{in}[t_1 t_2] = \{x_1 x_2 \mid x_1 \in \mathbf{in}[t_1] \wedge x_2 \in \mathbf{in}[t_2]\}$ , we know that

$$\varepsilon \in \mathbf{in}[t_1 t_2] \Rightarrow \varepsilon \in \mathbf{in}[t_1] \wedge \varepsilon \in \mathbf{in}[t_2].$$

Now the induction hypothesis yields that  $\varepsilon \in \mathbf{in}[t_1 t_2] \Rightarrow \mathbf{esp}(t_1) \neq \{\} \wedge \mathbf{esp}(t_2) \neq \{\}$ .

Therefore the definition of  $\mathbf{esp}(t_1 t_2)$  yields that

$$\varepsilon \in \mathbf{in}[t_1 t_2] \Rightarrow \mathbf{esp}(t_1 t_2) = \{d_1 d_2 \mid d_1 \in \mathbf{esp}(t_1) \wedge d_2 \in \mathbf{esp}(t_2)\} \neq \{\}, \text{ so the lemma is fulfilled}$$

in this case.

Case:  $t = t_1 + t_2$

Since  $\mathbf{in}[[t_1 + t_2]] = \mathbf{in}[[t_1]] \cup \mathbf{in}[[t_2]]$  we know that  $\varepsilon \in \mathbf{in}[[t_1 + t_2]] \Rightarrow \varepsilon \in \mathbf{in}[[t_1]] \vee \varepsilon \in \mathbf{in}[[t_2]]$ .

Now the induction hypothesis yields that

$\varepsilon \in \mathbf{in}[[t_1 + t_2]] \Rightarrow \mathbf{esp}(t_1) \neq \{\} \vee \mathbf{esp}(t_2) \neq \{\}$ .

Therefore the definition of  $\mathbf{esp}(t_1 + t_2)$  yields that  $\varepsilon \in \mathbf{in}[[t_1 + t_2]] \Rightarrow \mathbf{esp}(t_1 + t_2) \neq \{\}$ .

Therefore the lemma is fulfilled in this case.

Case:  $t = \mu X.t_1$

Assume that  $\varepsilon \in \mathbf{in}[[\mu X.t_1]]$ .

Lemma 6.7 yields that there is a  $d$  such that  $\mathbf{tag}(\mu X.t_1, d) = \varepsilon$ .

Now Lemma 7.7 yields that there is a  $d'$  such that  $\mathbf{tag}(t_1, d') = \varepsilon$ .

Now Lemma 6.6 yields that  $\varepsilon \in \mathbf{in}[[t_1]]$ .

Now the induction hypothesis yields that  $\mathbf{esp}(t_1) \neq \{\}$  and therefore  $\mathbf{esp}(\mu X.t_1) = \mathbf{esp}(t_1) \neq \{\}$ .

Therefore the lemma is fulfilled in this case.

Case:  $t = X$

In this case  $\mathbf{in}[[t]] = \{\}$  so the lemma is trivially fulfilled in this case.

Now we have proved all the cases, so we can conclude that

$$\forall t \in \mathbf{XMLType}. \varepsilon \in \mathbf{in}[[t]] \Rightarrow \mathbf{esp}(t) \neq \{\}$$

■

### Proof 19.10 (Completeness of ESP)

We will now prove Lemma 7.11 which states that

$$\forall T = [t_0, t_1, \dots, t_k] \in \mathbf{XMLTypes}. \forall i \in \{0, 1, \dots, k\}. \varepsilon \in \mathbf{in}[[t_i]] \Rightarrow \{(n, d) \in \mathbf{ESP}(T) \mid n = i\} \neq \{\}$$

by induction on  $i$ .

Start:  $i = 0$

In this case

$$\begin{aligned} \mathbf{ESP}(T) &= \mathbf{ESP}([t_0, t_1, \dots, t_k]) \\ &= \{(0, d) \mid d \in \mathbf{esp}(t_0)\} \cup \{(n+1, d) \mid (n, d) \in \mathbf{ESP}([t_1, t_2, \dots, t_k])\}. \end{aligned}$$

The definition of  $\mathbf{ESP}$  yields that

$$\begin{aligned} &\{(n, d) \in \mathbf{ESP}(T) \mid n = i\} \\ &= \{(n, d) \in \{(0, d') \mid d' \in \mathbf{esp}(t_0)\} \cup \{(n'+1, d') \mid (n', d') \in \mathbf{ESP}([t_1, t_2, \dots, t_k])\} \mid n = 0\} \\ &\subseteq \{(0, d) \mid d \in \mathbf{esp}(t_0)\} \end{aligned}$$

Now Lemma 7.8 yields that  $\varepsilon \in \mathbf{in}[[t_0]] \Rightarrow \mathbf{esp}(t_0) \neq \{\}$  and therefore we get that  $\varepsilon \in \mathbf{in}[[t_0]] \Rightarrow \{(n, d) \in \mathbf{ESP}(T) \mid n = 0\} \neq \{\}$ .

*Induction Hypothesis:*

If  $T = [t_0, t_1, \dots, t_k]$  and  $\varepsilon \in \mathbf{in}[t_i]$  for some  $i \in \{1, 2, \dots, k\}$  then we can assume that  $\{(n, d) \in \mathbf{ESP}([t_1, t_2, \dots, t_k]) \mid n = i - 1\} \neq \{\}$ .

It is important to notice that the first element of the list has been removed, so the index  $i - 1$  points to  $t_i$ .

Step:  $i > 0$

In this case,  $\mathbf{ESP}(T) = \mathbf{ESP}([t_0, t_1, \dots, t_k]) = \{(0, d) \mid d \in \mathbf{esp}(t_0)\} \cup \{(n + 1, d) \mid (n, d) \in \mathbf{ESP}([t_1, t_2, \dots, t_k])\}$ .

The induction hypothesis yields that

$\varepsilon \in \mathbf{in}[t_i] \Rightarrow \{(n, d) \in \mathbf{ESP}([t_1, t_2, \dots, t_k]) \mid n = i - 1\} \neq \{\}$  and therefore

$\varepsilon \in \mathbf{in}[t_i] \Rightarrow \{(n - 1, d) \in \mathbf{ESP}([t_1, t_2, \dots, t_k]) \mid n = i\} \neq \{\}$ .

If we now assume that  $\varepsilon \in \mathbf{in}[t_i]$  we get that

$$\begin{aligned}
& \{(n, d) \in \mathbf{ESP}(T) \mid n = i\} \\
&= \{(n, d) \in \{(0, d') \mid d' \in \mathbf{esp}(t_0)\} \cup \{(n' + 1, d') \mid (n', d') \in \mathbf{ESP}([t_1, t_2, \dots, t_k])\} \mid n = i\} \\
&\supseteq \{(n, d) \in \{(n' + 1, d') \mid (n', d') \in \mathbf{ESP}([t_1, t_2, \dots, t_k])\} \mid n = i\} \\
&= \{(n, d) \mid (n, d) \in \{(n' + 1, d') \mid (n', d') \in \mathbf{ESP}([t_1, t_2, \dots, t_k])\} \wedge n = i\} \\
&= \{(n, d) \mid (n, d) \in \{(n', d') \mid (n' - 1, d') \in \mathbf{ESP}([t_1, t_2, \dots, t_k])\} \wedge n = i\} \\
&= \{(n, d) \mid (n - 1, d) \in \mathbf{ESP}([t_1, t_2, \dots, t_k]) \wedge n = i\} \\
&\neq \{\}
\end{aligned}$$

We can therefore conclude that  $\varepsilon \in \mathbf{in}[t_i] \Rightarrow \{(n, d) \in \mathbf{ESP}(T) \mid n = i\} \neq \{\}$ .

We can now conclude that

$\forall T = [t_0, t_1, \dots, t_k] \in \mathbf{XMLTypes}. \forall i \in \{0, 1, \dots, k\}. \varepsilon \in \mathbf{in}[t_i] \Rightarrow \{(n, d) \in \mathbf{ESP}(T) \mid n = i\} \neq \{\}$ .

■

### Proof 19.11 (Split is well-formed)

We will now prove Lemma 8.6 which states that

$$\forall t' \in \overline{\mathbf{Split}_\varphi(t)}. t' = l' \langle t'_1 \rangle t'_2 \text{ for some } l', t'_1 \text{ and } t'_2$$

by induction on the calltree of  $\mathbf{Split}_\varphi(t)$ .

We will consider each case in  $\mathbf{Split}_\varphi$  separately.

Case:  $\mathbf{Split}_\varphi(0) = []$

Since  $\mathbf{Split}_\varphi(t) = []$ , the lemma is vacantly fulfilled in this case.

Case:  $\mathbf{Split}_\varphi(1) = []$

Since  $\mathbf{Split}_\varphi(t) = []$ , the lemma is vacantly fulfilled in this case.

Case:  $\mathbf{Split}_\varphi(l \langle t_1 \rangle) = [l \langle t_1[\varphi] \rangle 1]$

Since  $\mathbf{Split}_\varphi(t) = [l \langle t_1[\varphi] \rangle 1]$ , we have that  $\overline{\mathbf{Split}(t)} = \{l \langle t_1 \rangle 1\}$ .

Therefore the lemma is fulfilled in this case because  $l\langle t_1 \rangle 1$  is of the desired form.

Case:  $\text{Split}_\varphi(t_1 t_2) = \text{if } \text{esp}(t_1) \neq \{\} \{$   
 $\text{then } [l\langle t_{11} \rangle(t_{12}(t_2[\varphi])) \mid l\langle t_{11} \rangle t_{12} \in \text{Split}_\varphi(t_1)] @ \text{Split}_\varphi(t_2)$   
 $\text{else } [l\langle t_{11} \rangle(t_{12}(t_2[\varphi])) \mid l\langle t_{11} \rangle t_{12} \in \text{Split}_\varphi(t_1)]$   
 $\text{If } \text{esp}(t_1) \neq \{\} \text{ then } \text{Split}_\varphi(t_1 t_2) = [l\langle t_{11} \rangle(t_{12}(t_2[\varphi])) \mid l\langle t_{11} \rangle t_{12} \in \text{Split}_\varphi(t_1)] @ \text{Split}_\varphi(t_2),$   
 $\text{so } \overline{\text{Split}_\varphi(t_1 t_2)} = \{l\langle t_{11} \rangle(t_{12}(t_2[\varphi])) \mid l\langle t_{11} \rangle t_{12} \in \overline{\text{Split}_\varphi(t_1)}\} \cup \overline{\text{Split}_\varphi(t_2)}.$   
 $\text{Now the induction hypothesis yields that } t' \in \overline{\text{Split}_\varphi(t_2)} \Rightarrow t' = l'\langle t'_1 \rangle t'_2 \text{ for some } l', t'_1 \text{ and } t'_2$   
 $\text{the lemma is fulfilled in this case.}$   
 $\text{If } \text{esp}(t_1) = \{\} \text{ then } \text{Split}_\varphi(t_1 t_2) = [l\langle t_{11} \rangle(t_{12}(t_2[\varphi])) \mid l\langle t_{11} \rangle t_{12} \in \text{Split}_\varphi(t_1)], \text{ so}$   
 $\overline{\text{Split}_\varphi(t_1 t_2)} = \{l\langle t_{11} \rangle(t_{12}(t_2[\varphi])) \mid l\langle t_{11} \rangle t_{12} \in \overline{\text{Split}_\varphi(t_1)}\}.$   
 $\text{Therefore the lemma is fulfilled in this case.}$

Case:  $\text{Split}_\varphi(t_1 + t_2) = \text{Split}_\varphi(t_1) @ \text{Split}_\varphi(t_2)$   
 $\text{In this case the induction hypothesis yields that } t'_1 \in \overline{\text{Split}_\varphi(t_1)} \Rightarrow t'_1 = l'_1 \langle t'_{11} \rangle t'_{12} \text{ for some}$   
 $l'_1, t'_{11} \text{ and } t'_{12}, \text{ and } t'_2 \in \overline{\text{Split}_\varphi(t_2)} \Rightarrow t'_2 = l'_2 \langle t'_{21} \rangle t'_{22} \text{ for some } l'_2, t'_{21} \text{ and } t'_{22}.$   
 $\text{Since } \overline{\text{Split}_\varphi(t_1 + t_2)} = \overline{\text{Split}_\varphi(t_1)} \cup \overline{\text{Split}_\varphi(t_2)}, \text{ the lemma is fulfilled in this case.}$

Case:  $\text{Split}_\varphi(\mu X.t_1) = \text{Split}_{\mu X.t_1/X::\varphi}(t_1)$   
 $\text{The induction hypothesis yields that if } t' \in \text{Split}_{\mu X.t_1/X::\varphi}(t_1) \text{ then } t' = l'\langle t'_1 \rangle t'_2 \text{ for some}$   
 $l', t'_1 \text{ and } t'_2.$

$\text{If } t' \in \text{Split}_\varphi(\mu X.t_1) \text{ then } t' \in \text{Split}_{\mu X.t_1/X::\varphi}(t_1) \text{ and therefore } t' = l'\langle t'_1 \rangle t'_2 \text{ for some } l', t'_1$   
 $\text{and } t'_2.$

Therefore the lemma is fulfilled in this case.

Case:  $\text{Split}_\varphi(X) = []$   
 $\text{Since } \text{Split}_\varphi(t) = [], \text{ the lemma is vacantly fulfilled in this case.}$

We have now proved that the lemma is fulfilled in all the cases, and we can therefore conclude that

$$\forall t' \in \overline{\text{Split}_\varphi(t)}. t' = l'\langle t'_1 \rangle t'_2 \text{ for some } l', t'_1 \text{ and } t'_2.$$

■

### Proof 19.12 (Soundness of simplify)

We will now prove Lemma 9.5 which states that

$$\text{tag}(t, d) = \text{tag}(\mu X.t', d') \Rightarrow \text{tag}(\mu X.t', \text{simplify}_{\mu X.t', d'}(t, d)) = \text{tag}(t, d)$$

by induction on  $2 \cdot |d| + \text{ismu}(t)$ .

Induction Hypothesis:

$\text{If } \text{tag}(t_1, d_1) = \text{tag}(\mu X.t', d'') \text{ and } 2 \cdot |d_1| + \text{ismu}(t_1) < 2 \cdot |d| + \text{ismu}(t)$   
 $\text{then we can assume that } \text{tag}(\mu X.t', \text{simplify}_{\mu X.t', d''}(t_1, d_1)) = \text{tag}(t_1, d_1).$

We consider each form of  $t$  separately.

Case:  $t = 0$ ,  $t = 1$ ,  $t = X$  or  $t = l\langle t_1 \rangle$

In these cases  $\text{simplify}_{\mu X.t',d'}(t, d) = d'$  and therefore

$\text{tag}(\mu X.t', \text{simplify}_{\mu X.t',d'}(t, d)) = \text{tag}(\mu X.t', d') = \text{tag}(t, d)$ .

Therefore the lemma is fulfilled in this case.

Case:  $t = t_1 t_2$

If  $d \neq d_1 d_2$  then  $\text{simplify}_{\mu X.t',d'}(t, d) = d'$ , and therefore the lemma is fulfilled in the same way as the first case.

We can therefore assume that  $d = d_1 d_2$ .

If  $\text{tag}(t_1, d_1) \neq \varepsilon$  then  $\text{simplify}_{\mu X.t',d'}(t, d) = d'$  and therefore the lemma is fulfilled in the same way as the first case.

If  $\text{tag}(t_1, d_1) = \varepsilon$  then  $\text{simplify}_{\mu X.t',d'}(t, d) = \text{simplify}_{\mu X.t',d'}(t_2, d_2)$ .

Since  $\text{tag}(\mu X.t', d') = \text{tag}(t_1 t_2, d_1 d_2) = \text{let } x_1 = \text{tag}(t_1, d_1), x_2 = \text{tag}(t_2, d_2) \text{ in } x_1 \ x_2 \text{ end} = \text{tag}(t_2, d_2)$ , the induction hypothesis yields that  $\text{tag}(\mu X.t', \text{simplify}_{\mu X.t',d'}(t_2, d_2)) = \text{tag}(t_2, d_2)$ .

Therefore we get that

$$\begin{aligned}
 & \text{tag}(t, d) \\
 &= \text{tag}(t_1 t_2, d_1 d_2) \\
 &= \text{let } x_1 = \text{tag}(t_1, d_1), x_2 = \text{tag}(t_2, d_2) \text{ in } x_1 \ x_2 \text{ end} \\
 (\text{tag}(t_1, d_1) = \varepsilon) &= \text{tag}(t_2, d_2) \\
 \text{(IH)} &= \text{tag}(\mu X.t', \text{simplify}_{\mu X.t',d'}(t_2, d_2)) \\
 &= \text{tag}(\mu X.t', \text{simplify}_{\mu X.t',d'}(t_1 t_2, d_1 d_2)).
 \end{aligned}$$

Therefore the lemma is fulfilled in this case.

Case:  $t = t_1 + t_2$

If  $d = d_1 \diamond$  then

$\text{simplify}_{\mu X.t',d'}(t_1 + t_2, d_1 \diamond) = \text{simplify}_{\mu X.t',d'}(t_1, d_1)$ .

Since  $\text{tag}(\mu X.t', d') = \text{tag}(t_1 + t_2, d_1 \diamond) = \text{tag}(t_1, d_1)$  the induction hypothesis yields that  $\text{tag}(\mu X.t', \text{simplify}_{\mu X.t',d'}(t_1, d_1)) = \text{tag}(t_1, d_1)$ .

Therefore we get that

$$\begin{aligned}
 & \text{tag}(t, d) \\
 &= \text{tag}(t_1 + t_2, d_1 \diamond) \\
 &= \text{tag}(t_1, d_1) \\
 \text{(IH)} &= \text{tag}(\mu X.t', \text{simplify}_{\mu X.t',d'}(t_1, d_1)) \\
 &= \text{tag}(\mu X.t', \text{simplify}_{\mu X.t',d'}(t_1 + t_2, d_1 \diamond)).
 \end{aligned}$$

Therefore the lemma is fulfilled in this case.

If  $d = \diamond d_2$  then the lemma is fulfilled because it is symmetric to the previous case.

If  $d$  is not of the above forms then  $\text{simplify}_{\mu X.t',d'}(t_1 + t_2, d) = d'$  and therefore the lemma is fulfilled in the same way as the first case.

Case:  $t = \mu Y.t_1$ .

In this case

$$\begin{aligned} \text{simplify}_{\mu X.t',d'}(\mu Y.t_1, d) &= \text{if } \mu Y.t_1 = \mu X.t' \\ &\quad \text{then } \text{simplify}_{\mu Y.t_1,d}(t_1[\mu Y.t_1/Y], d) \\ &\quad \text{else } \text{simplify}_{\mu X.t',d'}(t_1[\mu Y.t_1/Y], d). \end{aligned}$$

If  $\mu Y.t_1 = \mu X.t'$  then  $\text{tag}(t_1[\mu Y.t_1/Y], d) = \text{tag}(\mu Y.t_1, d)$  and therefore the induction hypothesis yields that  $\text{tag}(\mu Y.t_1, \text{simplify}_{\mu Y.t_1,d}(t_1[\mu Y.t_1/Y], d)) = \text{tag}(t_1[\mu Y.t_1/Y], d)$ .

Therefore we have that

$$\begin{aligned} &\text{tag}(\mu Y.t_1, d) \\ &= \text{tag}(t_1[\mu Y.t_1/Y], d) \\ \text{(IH)} &= \text{tag}(\mu Y.t_1, \text{simplify}_{\mu Y.t_1,d}(t_1[\mu Y.t_1/Y], d)) \\ &= \text{tag}(\mu X.t', \text{simplify}_{\mu Y.t_1,d}(t_1[\mu Y.t_1/Y], d)) \\ &= \text{tag}(\mu X.t', \text{simplify}_{\mu X.t',d'}(\mu Y.t_1, d)). \end{aligned}$$

Therefore the lemma is fulfilled in this case.

If  $\mu Y.t_1 \neq \mu X.t'$  then we have that  $\text{tag}(\mu X.t', d') = \text{tag}(\mu Y.t_1, d) = \text{tag}(t_1[\mu Y.t_1/Y], d)$  and therefore the induction hypothesis yields that

$$\text{tag}(\mu Y.t_1, \text{simplify}_{\mu X.t',d'}(t_1[\mu X.t'/Y], d)) = \text{tag}(t_1[\mu Y.t_1/Y], d).$$

Therefore we get that

$$\begin{aligned} &\text{tag}(\mu Y.t_1, d) \\ &= \text{tag}(t_1[\mu Y.t_1/Y], d) \\ \text{(IH)} &= \text{tag}(\mu X.t', \text{simplify}_{\mu X.t',d'}(t_1[\mu Y.t_1/Y], d)) \\ &= \text{tag}(\mu X.t', \text{simplify}_{\mu X.t',d'}(\mu Y.t_1, d)). \end{aligned}$$

Therefore the lemma is fulfilled in this case.

We have now proved the lemma for all the cases and we can therefore conclude that

$$\text{tag}(t, d) = \text{tag}(\mu X.t', d') \Rightarrow \text{tag}(\mu X.t', \text{simplify}_{\mu X.t',d'}(t, d)) = \text{tag}(t, d).$$

■

### Proof 19.13

We will now prove Lemma 9.9 which states that

$$X \notin \text{fv}'(t) \Rightarrow \text{tag}_{\mu X.t_X/X::\varphi}(t, d) = \text{tag}_{\varphi}(t[\mu X.t_X/X], d)$$

by induction on  $|t|$ .

Induction Hypothesis:

If  $|t'| < |t|$  and  $X \notin \text{fv}'(t')$  then we can assume that  $\text{tag}_{\mu X.t_X/X::\varphi'}(t', d') = \text{tag}_{\varphi'}(t'[\mu X.t_X/X], d')$ .

We will consider each case in  $\text{tag}_{\varphi}(t, d)$  separately.

Case:  $\text{tag}_{\mu X.t_X/X::\varphi}(1, \bullet) = \varepsilon$

In this case  $\text{tag}_{\mu X.t_X/X::\varphi}(1, \bullet) = \varepsilon = \text{tag}_{\varphi}(1[\mu X.t_X/X], \bullet)$ .

Therefore the lemma is fulfilled in this case.

Case:  $\text{tag}_{\mu X.t_X/X::\varphi}(l\langle t_1 \rangle, \langle d_1 \rangle) = \text{tag}(l\langle t_1[\mu X.t_X/X] :: \varphi \rangle, \langle d_1 \rangle)$

In this case

$$\begin{aligned} & \text{tag}_{\mu X.t_X/X::\varphi}(l\langle t_1 \rangle, \langle d_1 \rangle) \\ &= \text{tag}(l\langle t_1[\mu X.t_X/X] :: \varphi \rangle, \langle d_1 \rangle) \\ &= \text{tag}_{\varphi}(l\langle t_1[\mu X.t_X/X] \rangle, \langle d_1 \rangle) \\ &= \text{tag}_{\varphi}(l\langle t_1 \rangle[\mu X.t_X/X], \langle d_1 \rangle). \end{aligned}$$

Therefore the lemma is fulfilled in this case.

Case:

$$\begin{aligned} \text{tag}_{\mu X.t_X/X::\varphi}(t_1 t_2, d_1 d_2) &= \text{if } \text{tag}(t_1[\mu X.t_X/X] :: \varphi, d_1) = \varepsilon \\ &\quad \text{then } \text{tag}_{\mu X.t_X/X::\varphi}(t_2, d_2) \\ &\quad \text{else let } x_1 = \text{tag}_{\mu X.t_X/X::\varphi}(t_1, d_1), x_2 = \text{tag}(t_2[\varphi], d_2) \text{ in } x_1 x_2 \text{ end} \end{aligned}$$

There are two cases.

If  $\text{tag}(t_1[\mu X.t_X/X] :: \varphi, d_1) = \varepsilon$  then

$$\begin{aligned} & \text{tag}_{\varphi}(t_1 t_2[\mu X.t_X/X], d_1 d_2) \\ &= \text{tag}_{\varphi}(t_2[\mu X.t_X/X], d_2) \\ \text{(IH)} &= \text{tag}_{\mu X.t_X/X::\varphi}(t_2, d_2) \\ &= \text{tag}_{\mu X.t_X/X::\varphi}(t_1 t_2, d_1 d_2) \end{aligned}$$

Therefore the lemma is fulfilled in this case.

If  $\text{tag}(t_1[\mu X.t_X/X] :: \varphi, d_1) \neq \varepsilon$  then

$$\begin{aligned} & \text{tag}_{\varphi}(t_1 t_2[\mu X.t_X/X], d_1 d_2) \\ &= \text{let } x_1 = \text{tag}_{\varphi}(t_1[\mu X.t_X/X], d_1), x_2 = \text{tag}(t_2[\mu X.t_X/X] :: \varphi, d_2) \text{ in } x_1 x_2 \text{ end} \\ \text{(IH)} &= \text{let } x_1 = \text{tag}_{\mu X.t_X/X::\varphi}(t_1, d_1), x_2 = \text{tag}(t_2[\mu X.t_X/X] :: \varphi, d_2) \text{ in } x_1 x_2 \text{ end.} \\ &= \text{tag}_{\mu X.t_X/X::\varphi}(t_1 t_2, d_1 d_2). \end{aligned}$$

Therefore the lemma is fulfilled in this case.

Case:  $\mathbf{tag}_{\mu X.t_X/X::\varphi}(t_1 + t_2, d_1 \diamond) = \mathbf{tag}_{\mu X.t_X/X::\varphi}(t_1, d_1)$

In this case, the desired follows directly by the induction hypothesis.

Case:  $\mathbf{tag}_{\mu X.t_X/X::\varphi}(t_1 + t_2, \diamond d_2) = \mathbf{tag}_{\mu X.t_X/X::\varphi}(t_2, d_2)$

In this case, the desired follows directly by the induction hypothesis.

Case:  $\mathbf{tag}_{\mu X.t_X/X::\varphi}(\mu Y.t_1, d) = \mathbf{tag}_{\mu Y.t_1/Y::\mu X.t_X/X::\varphi}(t_1, d)$

In this case

$$\begin{aligned}
& \mathbf{tag}_{\varphi}(\mu Y.t_1[\mu X.t_X/X], d) \\
&= \mathbf{tag}_{\varphi}(\mu Z.(t_1[Z/Y][\mu X.t_X/X]), d) \quad \mathbf{where} \ Z \notin \mathbf{fv}(t_X) \cup \mathbf{fv}(\mu Y.t_1) \cup \{X\} \\
&= \mathbf{tag}_{\mu Z.(t_1[Z/Y][\mu X.t_X/X])/Z::\varphi}(t_1[Z/Y][\mu X.t_X/X], d) \\
\text{(IH)} \quad &= \mathbf{tag}_{\mu X.t_X/X::\mu Z.(t_1[Z/Y][\mu X.t_X/X])/Z::\varphi}(t_1[Z/Y], d) \\
&= \mathbf{tag}_{\mu X.t_X/X::\mu Z.(t_1[Z/Y][\mu X.t_X/X])/Z::\varphi}(t_1[Z/Y], d) \\
\text{(Lemma 3.4)} \quad &= \mathbf{tag}_{\mu Z.(t_1[Z/Y])/Z::\mu X.t_X/X::\varphi}(t_1[Z/Y], d) \\
&= \mathbf{tag}_{\mu X.t_X/X::\varphi}(\mu Z.(t_1[Z/Y]), d) \\
&= \mathbf{tag}_{\mu X.t_X/X::\varphi}(\mu Y.t_1, d)
\end{aligned}$$

Therefore the lemma is fulfilled in this case.

Case:  $\mathbf{tag}_{\mu X.t_X/X::\varphi}(-, -) = \top$

There are two cases.

If  $t = Y$  then the assumptions yield that  $X \neq Y$  and therefore  $t[\mu X.t_X/X] = Y[\mu X.t_X/X] = Y$  and therefore  $\mathbf{tag}_{\varphi}(t[\mu X.t_X/X], d) = \mathbf{tag}_{\varphi}(Y, d) = \top$ .

If  $t \neq Y$  then the last case in  $\mathbf{tag}_{\varphi}$  is used and therefore  $\mathbf{tag}_{\varphi}(t[\mu X.t_X/X], d) = \top$ .

We have now proved the lemma for all the cases, and we can therefore conclude that

$$X \notin \mathbf{fv}'(t) \Rightarrow \mathbf{tag}_{\mu X.t_X/X::\varphi}(t, d) = \mathbf{tag}_{\varphi}(t[\mu X.t_X/X], d)$$

■

### Proof 19.14

We will now prove Lemma 9.10 which states that

$$\mathbf{fv}'(t) \cap \mathbf{dom}(\varphi) = \{\} \Rightarrow \mathbf{tag}(t, d) = \varepsilon \Leftrightarrow \mathbf{tag}(t[\varphi], d) = \varepsilon.$$

We will prove each implication separately.

If we assume that  $\mathbf{tag}(t, d) = \varepsilon \neq \top$  then Lemma 7.4 yields that  $\mathbf{tag}(t[\varphi], d) = \mathbf{tag}(t, d) = \varepsilon$ . Therefore the first implication is fulfilled.

We will now prove that

$$\mathbf{fv}'(t) \cap \mathbf{dom}(\varphi) = \{\} \Rightarrow \mathbf{tag}(t[\varphi], d) = \varepsilon \Rightarrow \mathbf{tag}(t, d) = \varepsilon$$

by induction on  $2 \cdot |d| + \text{ismu}(t)$ .

Induction hypothesis:

If  $2 \cdot |d'| + \text{ismu}(t') < 2 \cdot |d| + \text{ismu}(t)$ ,  $\text{fv}'(t') \cap \text{dom}(\varphi) = \{\}$  and  $\text{tag}(t'[\varphi], d') = \varepsilon$  then we can assume that  $\text{tag}(t', d') = \varepsilon$ .

We consider each case in  $\text{tag}(t, d)$  separately.

Case:  $\text{tag}(1, \bullet) = \varepsilon$

In this case  $\text{tag}(t, d) = \varepsilon$  and therefore the lemma is fulfilled.

Case:  $\text{tag}(l\langle t_1 \rangle, \langle d_1 \rangle) = \text{let } x_1 = \text{tag}(t_1, d_1) \text{ in } \langle l \rangle x_1 \langle /l \rangle \varepsilon \text{ end}$

In this case  $\text{tag}(l\langle t_1 \rangle[\varphi], \langle d_1 \rangle) = \text{let } x_1 = \text{tag}(t_1[\varphi], d_1) \text{ in } \langle l \rangle x_1 \langle /l \rangle \varepsilon \text{ end} \neq \varepsilon$ .

Therefore the lemma is fulfilled in this case.

Case:  $\text{tag}(t_1 t_2, d_1 d_2) = \text{let } x_1 = \text{tag}(t_1, d_1), x_2 = \text{tag}(t_2, d_2) \text{ in } x_1 x_2 \text{ end}$

In this case  $\text{tag}(t_1 t_2[\varphi], d_1 d_2) = \text{let } x_1 = \text{tag}(t_1[\varphi], d_1), x_2 = \text{tag}(t_2[\varphi], d_2) \text{ in } x_1 x_2 \text{ end}$ .

Therefore, if  $\text{tag}(t_1 t_2[\varphi], d_1 d_2) = \varepsilon$  then  $\text{tag}(t_1[\varphi], d_1) = \varepsilon$  and  $\text{tag}(t_2[\varphi], d_2) = \varepsilon$ .

Therefore the induction hypothesis yields that  $\text{tag}(t_1, d_1) = \varepsilon$  and  $\text{tag}(t_2, d_2) = \varepsilon$ , and therefore  $\text{tag}(t_1 t_2, d_1 d_2) = \varepsilon$ .

Therefore the lemma is fulfilled in this case.

Case:  $\text{tag}(t_1 + t_2, d_1 \diamond) = \text{tag}(t_1, d_1)$

In this case  $\text{tag}(t_1 + t_2[\varphi], d_1 \diamond) = \text{tag}(t_1[\varphi], d_1)$ .

The induction hypothesis yields that  $\text{tag}(t_1[\varphi], d_1) = \varepsilon \Rightarrow \text{tag}(t_1, d_1) = \varepsilon$ .

Therefore the lemma is fulfilled in this case.

Case:  $\text{tag}(t_1 + t_2, \diamond d_2) = \text{tag}(t_2, d_2)$

This follows by symmetry of the previous case.

Case:  $\text{tag}(\mu X.t_1, d) = \text{tag}(t_1[\mu X.t_1/X], d)$

In this case

$$\begin{aligned}
 & \text{tag}(\mu X.t_1[\varphi], d) \\
 = & \text{tag}(\mu Y.(t_1[Y/X][\varphi]), d) \quad \text{where } Y \notin \dots \\
 = & \text{tag}(t_1[Y/X][\varphi][\mu Y.(t_1[Y/X][\varphi])/Y], d) \\
 = & \text{tag}(t_1[Y/X][\varphi][\mu X.t_1[\varphi]/Y], d) \\
 \text{(Lemma 3.4)} = & \text{tag}(t_1[Y/X][\mu X.t_1/Y][\varphi], d) \\
 = & \text{tag}(t_1[\mu X.t_1/X][\varphi], d).
 \end{aligned}$$

The induction hypothesis yields that if  $\text{tag}(t_1[\mu X.t_1/X][\varphi], d) = \varepsilon$  then  $\text{tag}(t_1[\mu X.t_1/X], d) = \varepsilon$ .

Therefore the lemma is fulfilled in this case.

Case:  $\text{tag}(t, d) = \top$

There are two cases.

If  $t = X$  then  $\text{fv}'(t) = \{X\}$  and therefore the assumption yields that  $X \notin \text{dom}(\varphi)$ .  
 This means that  $\text{tag}(t[\varphi], d) = \text{tag}(X[\varphi], d) = \text{tag}(X, d) = \top$ .  
 Therefore the lemma is fulfilled in this case.

If  $t \neq X$  then  $\text{tag}(t[\varphi], d) = \top$ .  
 Therefore the lemma is fulfilled in this case.

We have now proved the lemma for all the cases, and we can therefore conclude that

$$\text{fv}'(t) \cap \text{dom}(\varphi) = \{\} \Rightarrow \text{tag}(t[\varphi], d) = \varepsilon \Rightarrow \text{tag}(t, d) = \varepsilon.$$

■

### Proof 19.15

We will now prove Lemma 9.11 which states that

$$\begin{aligned} \text{tr}_{\text{dom}(\varphi) \cup \{X\}}(t) \wedge |d| < |d'| \wedge \text{simplify}_{\mu X.t_X[\varphi], d'}(t[\mu X.t_X/X :: \varphi], d) &= d' \\ \Rightarrow \text{tag}_{\varphi}(t[\mu X.t_X/X], d) &= \text{tag}_{\mu X.t_X/X :: \varphi}(t, d) \end{aligned}$$

by induction on the calltree of  $\text{simplify}_{\mu X.t_X[\varphi], d'}(t[\mu X.t_X/X :: \varphi], d)$ .

*Induction Hypothesis:*

If  $\text{simplify}_{\mu X.t_X[\varphi], d'}(t[\mu X.t_X/X :: \varphi], d)$  returns  $\text{simplify}_{\mu X.t_X[\varphi'], d'}(t_1[\mu X.t_X/X :: \varphi'], d_1)$  and  $\text{tr}_{\text{dom}(\varphi') \cup \{X\}}(t_1)$  then we can assume that  $\text{tag}_{\varphi'}(t_1[\mu X.t_X/X], d_1) = \text{tag}_{\mu X.t_X/X :: \varphi'}(t_1, d_1)$ .

We consider each form of  $t$  separately.

Case:  $t = 1$

In this case  $\text{simplify}_{\mu X.t_X[\varphi], d'}(1[\mu X.t_X/X :: \varphi], d) = d'$

If  $d = \bullet$  then  $\text{tag}_{\mu X.t_X/X :: \varphi}(t, d) = \varepsilon = \text{tag}_{\varphi}(t[\mu X.t_X/X], d)$ .

If  $d \neq \bullet$  then  $\text{tag}_{\mu X.t_X/X :: \varphi}(t, d) = \top = \text{tag}_{\varphi}(t[\mu X.t_X/X], d)$ .

Therefore the lemma is fulfilled in this case.

Case:  $t = l\langle t_1 \rangle$

In this case  $\text{tag}_{\mu X.t_X/X :: \varphi}(t, d) = \text{tag}(l\langle t_1[\mu X.t_X/X :: \varphi] \rangle, d) = \text{tag}_{\varphi}(t[\mu X.t_X/X], d)$ .

Therefore the lemma is fulfilled in this case.

Case:  $t = t_1 t_2$

In this case

$\text{simplify}_{\mu X.t_X[\varphi], d'}(t_1 t_2[\mu X.t_X/X :: \varphi], d) =$

if  $\text{tag}(t_1[\mu X.t_X/X :: \varphi], d_1) = \varepsilon$

then  $\text{simplify}_{\mu X.t_X[\varphi], d'}(t_2, d_2)$

else  $d'$

Since  $\text{tr}_{\text{dom}(\varphi) \cup \{X\}}(t_1 t_2)$  Lemma 9.10 yields that

$$\text{tag}(t_1, d_1) = \varepsilon \Leftrightarrow \text{tag}(t_1[\mu X.t_X/X], d_1) = \varepsilon \Leftrightarrow \text{tag}(t_1[\mu X.t_X/X :: \varphi], d_1) = \varepsilon.$$

If  $\mathbf{tag}(t_1, d_1) = \varepsilon$  then

$\mathbf{tag}_\varphi(t_1 t_2 [\mu X.t_X/X], d_1 d_2) = \mathbf{tag}_\varphi(t_2 [\mu X.t_X/X], d_2)$  and

$\mathbf{tag}_{\mu X.t_X/X::\varphi}(t_1 t_2, d_1 d_2) = \mathbf{tag}_{\mu X.t_X/X::\varphi}(t_2, d_2)$ .

The induction hypothesis yields that  $\mathbf{tag}_\varphi(t_2 [\mu X.t_X/X], d_2) = \mathbf{tag}_{\mu X.t_X/X::\varphi}(t_2, d_2)$

This means that

$$\begin{aligned} & \mathbf{tag}_\varphi(t_1 t_2 [\mu X.t_X/X], d_1 d_2) \\ &= \mathbf{tag}_\varphi(t_2 [\mu X.t_X/X], d_2) \\ \text{(IH)} &= \mathbf{tag}_{\mu X.t_X/X::\varphi}(t_2, d_2) \\ &= \mathbf{tag}_{\mu X.t_X/X::\varphi}(t_1 t_2, d_1 d_2). \end{aligned}$$

Therefore the lemma is fulfilled in this case.

If  $\mathbf{tag}(t_1 [\mu X.t_X/X :: \varphi], d_1) \neq \varepsilon$  then

$\mathbf{tag}_\varphi(t_1 t_2 [\mu X.t_X/X], d_1 d_2) =$

let  $x_1 = \mathbf{tag}_\varphi(t_1 [\mu X.t_X/X], d_1)$ ,  $x_2 = \mathbf{tag}(t_2 [\mu X.t_X/X :: \varphi], d_2)$

in  $x_1 x_2$  end and

$\mathbf{tag}_{\mu X.t_X/X::\varphi}(t_1 t_2, d_1 d_2) = \text{let } x_1 = \mathbf{tag}_{\mu X.t_X/X::\varphi}(t_1, d_1)$ ,  $x_2 = \mathbf{tag}(t_2 [\mu X.t_X/X :: \varphi], d_2)$  in  $x_1 x_2$  end.

Since  $\mathbf{tr}_{\text{dom}(\varphi) \cup \{X\}}(t_1 t_2)$  we know that  $\mathbf{fv}'(t_1) \cap (\text{dom}(\varphi) \cup \{X\}) = \{\}$  and therefore Lemma

9.9 yields that  $\mathbf{tag}_\varphi(t_1 [\mu X.t_X/X], d_1) = \mathbf{tag}_{\mu X.t_X/X::\varphi}(t_1, d_1)$ .

This means that

$$\begin{aligned} & \mathbf{tag}_\varphi(t_1 t_2 [\mu X.t_X/X], d_1 d_2) \\ &= \text{let } x_1 = \mathbf{tag}_\varphi(t_1 [\mu X.t_X/X], d_1)$$
,  $x_2 = \mathbf{tag}(t_2 [\mu X.t_X/X :: \varphi], d_2)$  in  $x_1 x_2$  end \\ \text{(Lemma 9.9)} &= \text{let } x\_1 = \mathbf{tag}\_{\mu X.t\_X/X::\varphi}(t\_1, d\_1),  $x_2 = \mathbf{tag}(t_2 [\mu X.t_X/X :: \varphi], d_2)$  in  $x_1 x_2$  end \\ &= \mathbf{tag}\_{\mu X.t\_X/X::\varphi}(t\_1 t\_2, d\_1 d\_2). \end{aligned}

Therefore the lemma is fulfilled in this case.

If  $d \neq d_1 d_2$  then  $\mathbf{tag}_\varphi(t_1 t_2 [\mu X.t_X/X], d) = \top = \mathbf{tag}_{\mu X.t_X/X::\varphi}(t_1 t_2, d)$ .

Therefore the lemma is fulfilled in this case.

Case:  $t = t_1 + t_2$

If  $d = d_1 \diamond$  then

$\mathbf{simplify}_{\mu X.t_X[\varphi], d'}(t_1 + t_2 [\mu X.t_X/X :: \varphi], d_1 \diamond) = \mathbf{simplify}_{\mu X.t_X[\varphi], d'}(t_1 [\mu X.t_X/X :: \varphi], d_1)$

The induction hypothesis yields that  $\mathbf{tag}_\varphi(t_1 [\mu X.t_X/X], d_1) = \mathbf{tag}_{\mu X.t_X/X::\varphi}(t_1, d_1)$ .

This means that

$$\begin{aligned} & \mathbf{tag}_\varphi(t_1 + t_2 [\mu X.t_X/X], d_1 \diamond) \\ &= \mathbf{tag}_\varphi(t_1 [\mu X.t_X/X], d_1) \\ \text{(IH)} &= \mathbf{tag}_{\mu X.t_X/X::\varphi}(t_1, d_1) \\ &= \mathbf{tag}_{\mu X.t_X/X::\varphi}(t_1 + t_2, d_1 \diamond). \end{aligned}$$

Therefore the lemma is fulfilled in this case.

If  $d = \diamond d_2$  then

$\mathbf{simplify}_{\mu X.t_X[\varphi],d'}(t_1 + t_2[\mu X.t_X/X :: \varphi], \diamond d_2) = \mathbf{simplify}_{\mu X.t_X[\varphi],d'}(t_2[\mu X.t_X/X :: \varphi], d_2)$   
 This case follows by symmetry of the previous case.

If  $d \neq d_1 \diamond$  and  $d \neq \diamond d_2$  then  $\mathbf{tag}_{\varphi}(t_1 + t_2[\mu X.t_X/X], d) = \top = \mathbf{tag}_{\mu X.t_X/X :: \varphi}(t_1 + t_2, d)$ .  
 Therefore the lemma is fulfilled in this case.

Case:  $t = \mu Y.t_1$

In this case  $t[\mu X.t_X/X :: \varphi] = \mu Y.t_1[\mu X.t_X/X :: \varphi] = \mu Z.(t_1[Z/Y][\mu X.t_X/X :: \varphi])$  where  $Z$  is not in  $\text{dom}(\varphi) \cup \{X\}$  or in the free variables of  $\mu Y.t_1$  or any of the terms in the substitution, including  $\mu X.t_X$ .

This means that

$$\begin{aligned} & \mathbf{simplify}_{\mu X.t_X[\varphi],d'}(\mu Y.t_1[\mu X.t_X/X :: \varphi], d) \\ = & \mathbf{simplify}_{\mu X.t_X[\varphi],d'}(\mu Z.(t_1[Z/Y][\mu X.t_X/X :: \varphi]), d) \\ = & \mathbf{if} \ \mu X.t_X[\varphi] = \mu Z.(t_1[Z/Y][\mu X.t_X/X :: \varphi]) \\ & \mathbf{then} \ \mathbf{simplify}_{\mu Z.(t_1[Z/Y][\mu X.t_X/X :: \varphi]),d}(t_1[Z/Y][\mu X.t_X/X :: \varphi][\mu Z.(t_1[Z/Y][\mu X.t_X/X :: \varphi])/Z], d) \\ & \mathbf{else} \ \mathbf{simplify}_{\mu X.t_X[\varphi],d'}(t_1[Z/Y][\mu X.t_X/X :: \varphi][\mu Z.(t_1[Z/Y][\mu X.t_X/X :: \varphi])/Z], d) \end{aligned}$$

If  $\mu X.t_X[\varphi] = \mu Y.t_1[\mu X.t_X/X :: \varphi]$  then

$\mathbf{simplify}_{\mu X.t_X[\varphi],d}(t_1[Z/Y][\mu X.t_X/X :: \varphi][\mu Z.(t_1[Z/Y][\mu X.t_X/X :: \varphi])/Z], d)$  is returned.

This means that  $\mathbf{simplify}_{\mu X.t_X[\varphi],d'}(t[\mu X.t_X/X :: \varphi], d) \neq d'$  and therefore the lemma is trivially fulfilled in this case.

Otherwise  $\mathbf{simplify}_{\mu X.t_X[\varphi],d'}(t_1[Z/Y][\mu X.t_X/X :: \varphi][\mu Z.(t_1[Z/Y][\mu X.t_X/X :: \varphi])/Z], d)$  is returned.

We now have that

$$\begin{aligned} & \mu X.t_X[\mu Z.(t_1[Z/Y][\mu X.t_X/X])/Z :: \varphi] \\ (Z \notin \text{fv}(\mu X.t_X)) & = \mu X.t_X[\varphi] \end{aligned}$$

and

$$\begin{aligned} & t_1[Z/Y][\mu X.t_X/X :: \varphi][\mu Z.(t_1[Z/Y][\mu X.t_X/X :: \varphi])/Z] \\ = & t_1[Z/Y][\mu X.t_X/X :: \varphi][\mu Z.(t_1[Z/Y][\mu X.t_X/X])/Z] \\ (\mathbf{Lemma 3.4}) & = t_1[Z/Y][\mu X.t_X/X :: \mu Z.(t_1[Z/Y][\mu X.t_X/X])/Z :: \varphi]. \end{aligned}$$

Therefore, if we set  $\varphi' = [\mu Z.(t_1[Z/Y][\mu X.t_X/X])/Z :: \varphi]$  we get that

$$\begin{aligned} & \mathbf{simplify}_{\mu X.t_X[\varphi],d'}(t_1[Z/Y][\mu X.t_X/X :: \varphi][\mu Z.(t_1[Z/Y][\mu X.t_X/X :: \varphi])/Z], d) \\ = & \mathbf{simplify}_{\mu X.t_X[\varphi'],d'}(t_1[Z/Y][\mu X.t_X/X :: \varphi'], d). \end{aligned}$$

Therefore the induction hypothesis yields that

$$\mathbf{tag}_{\varphi'}(t_1[Z/Y][\mu X.t_X/X], d) = \mathbf{tag}_{\mu X.t_X/X :: \varphi'}(t_1[Z/Y], d).$$

This means that

$$\begin{aligned}
& \text{tag}_\varphi(\mu Y.t_1[\mu X.t_X/X], d) \\
&= \text{tag}_\varphi(\mu Z.(t_1[Z/Y][\mu X.t_X/X]), d) \\
&= \text{tag}'_\varphi(t_1[Z/Y][\mu X.t_X/X], d) \\
\text{(IH)} &= \text{tag}_{\mu X.t_X/X::\varphi'}(t_1[Z/Y], d) \\
&= \text{tag}_{\mu X.t_X/X::\mu Z.(t_1[Z/Y][\mu X.t_X/X])/Z::\varphi}(t_1[Z/Y], d) \\
&= \text{tag}_{\mu X.t_X/X::\mu Z.(t_1[Z/Y][\mu X.t_X/X])/Z::\varphi}(t_1[Z/Y], d) \\
\text{(Lemma 3.4)} &= \text{tag}_{\mu Z.(t_1[Z/Y])/Z::\mu X.t_X/X::\varphi}(t_1[Z/Y], d) \\
&= \text{tag}_{\mu X.t_X/X::\varphi}(\mu Z.(t_1[Z/Y]), d) \\
&= \text{tag}_{\mu X.t_X/X::\varphi}(\mu Y.t_1, d).
\end{aligned}$$

Therefore the lemma is fulfilled in this case.

Case  $t = Y$

If  $X = Y$  then

$$\begin{aligned}
& \text{simplify}_{\mu X.t_X[\varphi], d'}(Y[\mu X.t_X/X :: \varphi], d) \\
&= \text{simplify}_{\mu X.t_X[\varphi], d'}(\mu X.t_X[\varphi], d) \\
&= \text{simplify}_{\mu X.t_X[\varphi], d}(t_X[\mu X.t_X/X :: \varphi], d) \neq d'.
\end{aligned}$$

Therefore the lemma is trivially fulfilled in this case.

If  $X \neq Y$  then  $\text{tag}_\varphi(Y[\mu X.t_X/X], d) = \text{tag}_\varphi(Y, d) = \top = \text{tag}_{\mu X.t_X/X::\varphi}(Y, d)$ .

Therefore the lemma is fulfilled in this case.

We have now proved the lemma for all possible cases and can therefore conclude that

$$\begin{aligned}
& \text{tr}_{\text{dom}(\varphi) \cup \{X\}}(t) \wedge |d| < |d'| \wedge \text{simplify}_{\mu X.t_X[\varphi], d'}(t[\mu X.t_X/X :: \varphi], d) = d' \\
& \Rightarrow \text{tag}_\varphi(t[\mu X.t_X/X], d) = \text{tag}_{\mu X.t_X/X::\varphi}(t, d).
\end{aligned}$$

■

**Proof 19.16** ( $\text{tag}_\varphi$  is a generalization of  $\text{tag}$ )

We will now prove Lemma 9.14 which states that

$$\text{tag}_\square(t, d) = \text{tag}(t, d)$$

by induction on  $2 \cdot |d| + \text{ismu}(t)$ .

*Induction Hypothesis:*

If  $2 \cdot |d_1| + \text{ismu}(t_1) < 2 \cdot |d| + \text{ismu}(t)$  then we can assume that  $\text{tag}_\square(t_1, d_1) = \text{tag}(t_1, d_1)$ .

We will consider each case in  $\text{tag}(t, d)$  separately.

Case:  $\text{tag}(1, e) = \varepsilon$

Since  $\mathbf{tag}_{\square}(1, e) = \varepsilon$  the lemma is fulfilled in this case.

Case:  $\mathbf{tag}(l\langle t_1 \rangle, \langle d_1 \rangle) = \mathbf{let } x_1 = \mathbf{tag}(t_1, d_1) \mathbf{ in } \langle l \rangle x_1 \langle /l \rangle \varepsilon \mathbf{ end}$

In this case  $\mathbf{tag}_{\square}(l\langle t_1 \rangle, \langle d_1 \rangle) = \mathbf{tag}(l\langle t_1 \rangle, \langle d_1 \rangle)$ .

Therefore the lemma is fulfilled in this case.

Case:  $\mathbf{tag}(t_1 t_2, d_1 d_2) = \mathbf{let } x_1 = \mathbf{tag}(t_1, d_1), x_2 = \mathbf{tag}(t_2, d_2) \mathbf{ in } x_1 x_2 \mathbf{ end}$

There are two cases.

If  $\mathbf{tag}(t_1, d_1) = \varepsilon$  then  $\mathbf{tag}_{\square}(t_1 t_2, d_1 d_2) = \mathbf{tag}_{\square}(t_2, d_2)$ .

The induction hypothesis yields that  $\mathbf{tag}(t_2, d_2) = \mathbf{tag}_{\square}(t_2, d_2)$  and therefore

$$\begin{aligned}
 & \mathbf{tag}(t_1 t_2, d_1 d_2) \\
 &= \mathbf{let } x_1 = \mathbf{tag}(t_1, d_1), x_2 = \mathbf{tag}(t_2, d_2) \mathbf{ in } x_1 x_2 \mathbf{ end} \\
 &= \mathbf{tag}(t_2, d_2) \\
 \text{(IH)} \quad &= \mathbf{tag}_{\square}(t_2, d_2) \\
 &= \mathbf{tag}_{\square}(t_1 t_2, d_1 d_2).
 \end{aligned}$$

Therefore the lemma is fulfilled in this case.

If  $\mathbf{tag}(t_1, d_1) \neq \varepsilon$  then  $\mathbf{tag}_{\square}(t_1 t_2, d_1 d_2) = \mathbf{let } x_1 = \mathbf{tag}_{\square}(t_1, d_1), x_2 = \mathbf{tag}(t_2, d_2) \mathbf{ in } x_1 x_2 \mathbf{ end}$ .

The induction hypothesis yields that  $\mathbf{tag}(t_1, d_1) = \mathbf{tag}_{\square}(t_1, d_1)$  and therefore

$$\begin{aligned}
 & \mathbf{tag}(t_1 t_2, d_1 d_2) \\
 &= \mathbf{let } x_1 = \mathbf{tag}(t_1, d_1), x_2 = \mathbf{tag}(t_2, d_2) \mathbf{ in } x_1 x_2 \mathbf{ end} \\
 \text{(IH)} \quad &= \mathbf{let } x_1 = \mathbf{tag}_{\square}(t_1, d_1), x_2 = \mathbf{tag}(t_2, d_2) \mathbf{ in } x_1 x_2 \mathbf{ end} \\
 &= \mathbf{tag}_{\square}(t_1 t_2, d_1 d_2).
 \end{aligned}$$

Therefore the induction hypothesis is fulfilled in this case.

Case:  $\mathbf{tag}(t_1 + t_2, d_1 \diamond) = \mathbf{tag}(t_1, d_1)$

In this case  $\mathbf{tag}_{\square}(t_1 + t_2, d_1 \diamond) = \mathbf{tag}_{\square}(t_1, d_1)$  and therefore the desired follows from the induction hypothesis.

Case:  $\mathbf{tag}(t_1 + t_2, \diamond d_2) = \mathbf{tag}(t_2, d_2)$

In this case  $\mathbf{tag}_{\square}(t_1 + t_2, \diamond d_2) = \mathbf{tag}_{\square}(t_2, d_2)$  and therefore the desired follows from the induction hypothesis.

Case:  $\mathbf{tag}(\mu X.t_1, d) = \mathbf{tag}(t_1[\mu X.t_1/X], d)$

In this case the induction hypothesis yields that

$$\begin{aligned}
 & \mathbf{tag}_{\square}(t_1[\mu X.t_1/X], \mathbf{simplify}_{\mu X.t_1, d}(t_1[\mu X.t_1/X], d)) = \\
 & \mathbf{tag}(t_1[\mu X.t_1/X], \mathbf{simplify}_{\mu X.t_1, d}(t_1[\mu X.t_1/X], d)).
 \end{aligned}$$

This means that

$$\begin{aligned}
& \mathbf{tag}(\mu X.t_1, d) \\
&= \mathbf{tag}(t_1[\mu X.t_1/X], d) \\
\text{(Lemma 9.5)} &= \mathbf{tag}(t_1[\mu X.t_1/X], \mathbf{simplify}_{\mu X.t_1, d}(t_1[\mu X.t_1/X], d)) \\
\text{(IH)} &= \mathbf{tag}_{\square}(t_1[\mu X.t_1/X], \mathbf{simplify}_{\mu X.t_1, d}(t_1[\mu X.t_1/X], d)) \\
\text{(Corollary 9.13)} &= \mathbf{tag}_{[\mu X.t_X/X]}(t_1, \mathbf{simplify}_{\mu X.t_1, d}(t_1[\mu X.t_1/X], d)) \\
&= \mathbf{tag}_{\square}(\mu X.t_1, d).
\end{aligned}$$

Therefore the lemma is fulfilled in this case.

Case:  $\mathbf{tag}(-, -) = \top$

If none of the above cases are fulfilled then  $\mathbf{tag}_{\square}(t, d) = \top = \mathbf{tag}(t, d)$ .

Therefore the lemma is fulfilled in this case.

We have now proved the lemma in all the cases and we can therefore conclude that

$$\mathbf{tag}_{\square}(t, d) = \mathbf{tag}(t, d)$$

■

### Proof 19.17 (Soundness of Trans)

We will now prove Lemma 10.5 which states that

$$\forall D \in \mathbf{Trans}_{\varphi}(t, d). \mathbf{tag}_{\varphi}(t, d) = \mathbf{TAG}(\mathbf{Split}_{\varphi}(t), D)$$

by induction on the calltree of  $\mathbf{Trans}_{\varphi}(t, d)$ .

Induction Hypothesis:

If  $\mathbf{Trans}_{\varphi}(t, d)$  calls  $\mathbf{Trans}_{\varphi'}(t', d')$  then we can assume that

$$\forall D' \in \mathbf{Trans}_{\varphi'}(t', d'). \mathbf{tag}_{\varphi'}(t', d') = \mathbf{TAG}(\mathbf{Split}_{\varphi'}(t'), D').$$

Case:  $\mathbf{Trans}_{\varphi}(l\langle t_1 \rangle, d) = \{(0, d\bullet)\}$

Since  $\mathbf{Split}_{\varphi}(l\langle t_1 \rangle) = [l\langle t_1[\varphi] \rangle 1]$  we only need to check that

$$\begin{aligned}
& \mathbf{TAG}([l\langle t_1[\varphi] \rangle 1], (0, d\bullet)) \\
&= \mathbf{tag}_{\varphi}(l\langle t_1[\varphi] \rangle 1, d\bullet) \\
&= \mathbf{let } x_1 = \mathbf{tag}_{\varphi}(l\langle t_1[\varphi] \rangle, d), x_2 = \mathbf{tag}(1, \bullet) \mathbf{ in } x_1 \ x_2 \ \mathbf{end} \\
&= \mathbf{let } x_1 = \mathbf{tag}_{\varphi}(l\langle t_1[\varphi] \rangle, d) \mathbf{ in } x_1 \ \varepsilon \ \mathbf{end} \\
&= \mathbf{tag}_{\varphi}(l\langle t_1 \rangle, d).
\end{aligned}$$

Therefore the lemma is fulfilled in this case.

Case:

$$\begin{aligned}
\mathbf{Trans}_{\varphi}(t_1 t_2, d_1 d_2) &= \mathbf{if } \mathbf{tag}(t_1, d_1) = \varepsilon \\
&\quad \mathbf{then } \{[\mathbf{Split}_{\varphi}(t_1)] + n, d \mid (n, d) \in \mathbf{Trans}_{\varphi}(t_2, d_2)\} \\
&\quad \mathbf{else } \{(n, d_{11}(d_{12}d_2)) \mid (n, d_{11}d_{12}) \in \mathbf{Trans}_{\varphi}(t_1, d_1)\}
\end{aligned}$$

There are two cases.

If  $\text{tag}(t_1, d_1) = \varepsilon$  then  $\text{Trans}_\varphi(t_1 t_2, d_1 d_2) = \{(|\text{Split}_\varphi(t_1)| + n, d) \mid (n, d) \in \text{Trans}_\varphi(t_2, d_2)\}$ .  
 The induction hypothesis yields that  $\forall D_2 \in \text{Trans}_\varphi(t_2, d_2). \text{TAG}(\text{Split}_\varphi(t_2), D_2) = \text{tag}_\varphi(t_2, d_2)$ .  
 Since  $\text{tag}(t_1, d_1) = \varepsilon$  Lemma 7.8 yields that  $\text{esp}(t_1) \neq \{\}$  and therefore  $\text{Split}_\varphi(t_1 t_2) = [l\langle t_{11} \rangle(t_{12}(t_2)) \mid l\langle t_{11} \rangle t_{12} \in \text{Split}_\varphi(t_1)] @ \text{Split}_\varphi(t_2)$ .

Now

$$\begin{aligned} \text{TAG}(\text{Split}_\varphi(t_1 t_2), D) &= \text{TAG}([l\langle t_{11} \rangle(t_{12}(t_2)) \mid l\langle t_{11} \rangle t_{12} \in \text{Split}_\varphi(t_1)] @ \text{Split}_\varphi(t_2), (|\text{Split}_\varphi(t_1)| + n, d)) \\ &= \text{TAG}(\text{Split}_\varphi(t_2), (n, d)) \\ \text{(IH)} &= \text{tag}_\varphi(t_2, d_2) \\ &= \text{tag}_\varphi(t_1 t_2, d_1 d_2). \end{aligned}$$

Therefore the lemma is fulfilled in this case.

If  $\text{tag}(t_1, d_1) \neq \varepsilon$  then  $\text{Trans}_\varphi(t_1 t_2, d_1 d_2) = \{(n, d_{11}(d_{12}d_2)) \mid d_{11}d_{12} \in \text{Trans}_\varphi(t_1, d_1)\}$ .  
 In this case the induction hypothesis yields that  $\forall D_1 \in \text{Trans}_\varphi(t_1, d_1). \text{tag}_\varphi(t_1, d_1) = \text{TAG}(\text{Split}_\varphi(t_1), D_1)$ .  
 Since  $D_1$  is of the form  $(n, \langle d_{11} \rangle d_{12})$ , and  
 $\text{Split}_\varphi(t_1 t_2) = [l\langle t_{11} \rangle(t_{12}(t_2[\varphi])) \mid l\langle t_{11} \rangle t_{12} \in \text{Split}_\varphi(t_1)] @ T$ , we can obtain the following.

$$\begin{aligned} \text{tag}_\varphi(t_1 t_2, d_1 d_2) &= \text{let } x_1 = \text{tag}_\varphi(t_1, d_1), x_2 = \text{tag}(t_2[\varphi], d_2) \text{ in } x_1 x_2 \text{ end} \\ \text{(IH)} &= \text{let } x_1 = \text{TAG}(\text{Split}_\varphi(t_1), (n, \langle d_{11} \rangle d_{12})), x_2 = \text{tag}(t_2[\varphi], d_2) \text{ in } x_1 x_2 \text{ end} \\ &= \text{TAG}([l\langle t_{11} \rangle(t_{12}(t_2[\varphi])) \mid l\langle t_{11} \rangle t_{12} \in \text{Split}_\varphi(t_1)], (n, \langle d_{11} \rangle (d_{12}d_2))) \\ &= \text{TAG}(\text{Split}_\varphi(t_1 t_2), (n, \langle d_{11} \rangle (d_{12}d_2))) \end{aligned}$$

We can therefore conclude that

$$(n, \langle d_{11} \rangle d_{12}) \in \text{Trans}_\varphi(t_1, d_1) \Rightarrow \text{tag}_\varphi(t_1 t_2, d_1 d_2) = \text{TAG}(\text{Split}_\varphi(t_1 t_2), (n, \langle d_{11} \rangle (d_{12}d_2))).$$

Therefore the lemma is fulfilled in this case.

Case:  $\text{Trans}_\varphi(t_1 + t_2, d_1 \diamond) = \text{Trans}_\varphi(t_1, d_1)$

In this case  $\text{Split}_\varphi(t_1 + t_2) = \text{Split}_\varphi(t_1) @ \text{Split}_\varphi(t_2)$ .

The induction hypothesis yields that  $\forall D_1 \in \text{Trans}_\varphi(t_1, d_1). \text{tag}_\varphi(t_1, d_1) = \text{TAG}(\text{Split}_\varphi(t_1), D_1)$ .

Now we can conclude the following, if  $D \in \text{Trans}_\varphi(t_1, d_1) = \text{Trans}_\varphi(t_1 + t_2, d_1 \diamond)$ .

$$\begin{aligned} \text{tag}_\varphi(t_1 + t_2, d_1 \diamond) &= \text{tag}_\varphi(t_1, d_1) \\ \text{(IH)} &= \text{TAG}(\text{Split}_\varphi(t_1), D) \\ &= \text{TAG}(\text{Split}_\varphi(t_1) @ \text{Split}_\varphi(t_2), D) \\ &= \text{TAG}(\text{Split}_\varphi(t_1 + t_2), D). \end{aligned}$$

Therefore the lemma is fulfilled in this case.

Case:  $\text{Trans}_\varphi(t_1 + t_2, \diamond d_2) = \{(n' + |\text{Split}_\varphi(t_1)|, d') \mid (n', d') \in \text{Trans}_\varphi(t_1, d_1)\}$

In this case  $\text{Split}_\varphi(t_1 + t_2) = \text{Split}_\varphi(t_1) @ \text{Split}_\varphi(t_2)$ .

Therefore the induction hypothesis yields that  $\forall D_2 \in \text{Trans}_\varphi(t_2, d_2). \text{tag}_\varphi(t_2, d_2) = \text{TAG}(\text{Split}_\varphi(t_2), D_2)$ .

Now we can conclude the following, if  $(n' + |\mathbf{Split}_\varphi(t_1)|, d') \in \mathbf{Trans}_\varphi(t_1 + t_2, d_2)$ .

$$\begin{aligned}
& \mathbf{TAG}(\mathbf{Split}_\varphi(t_1 + t_2), (n' + |\mathbf{Split}_\varphi(t_1)|, d')) \\
&= \mathbf{TAG}(\mathbf{Split}_\varphi(t_1) @ \mathbf{Split}_\varphi(t_2), (n' + |\mathbf{Split}_\varphi(t_1)|, d')) \\
&= \mathbf{TAG}(\mathbf{Split}_\varphi(t_2), (n', d')) \\
\text{(IH)} \quad &= \mathbf{tag}_\varphi(t_2, d_2) \\
&= \mathbf{tag}_\varphi(t_1 + t_2, \diamond d_2)
\end{aligned}$$

Therefore the lemma is fulfilled in this case.

Case:  $\mathbf{Trans}_\varphi(\mu X.t_1, d) = \mathbf{Trans}_{\mu X.t_1/X::\varphi}(t_1, \mathbf{simplify}_{\mu X.t_1[\varphi], d}(t_1[\mu X.t_1/X :: \varphi], d))$

In this case the induction hypothesis yields that

$\forall D \in \mathbf{Trans}_{\mu X.t_1/X::\varphi}(t_1, \mathbf{simplify}_{\mu X.t_1[\varphi], d}(t_1[\mu X.t_1/X :: \varphi], d))$ .

$\mathbf{tag}_{\mu X.t_1/X::\varphi}(t_1, \mathbf{simplify}_{\mu X.t_1[\varphi], d}(t_1[\mu X.t_1/X :: \varphi], d)) = \mathbf{TAG}(\mathbf{Split}_{\mu X.t_1/X::\varphi}(t_1), D)$ .

Therefore

$$\begin{aligned}
& \mathbf{TAG}(\mathbf{Split}_\varphi(\mu X.t_1), D) \\
&= \mathbf{TAG}(\mathbf{Split}_{\mu X.t_1/X::\varphi}(t_1), D) \\
\text{(IH)} \quad &= \mathbf{tag}_{\mu X.t_1/X::\varphi}(t_1, \mathbf{simplify}_{\mu X.t_1[\varphi], d}(t_1[\mu X.t_1/X :: \varphi], d)) \\
&= \mathbf{tag}_\varphi(\mu X.t_1, d)
\end{aligned}$$

Therefore the lemma is fulfilled in this case.

Case:  $\mathbf{Trans}_\varphi(-, -) = \{\}$

The lemma is vacantly fulfilled in this case.

We have now proved the lemma for all cases, and can therefore conclude that

$$\forall D \in \mathbf{Trans}_\varphi(t, d). \mathbf{tag}_\varphi(t, d) = \mathbf{TAG}(\mathbf{Split}_\varphi(t), D)$$

■

### Proof 19.18

We will now prove Lemma 10.7 which states that

$$\mathbf{tag}_\varphi(t, d) = \varepsilon \Rightarrow \mathbf{tag}(t[\varphi], d) = \varepsilon$$

by structural induction on  $t$ .

Induction Hypothesis:

If  $t'$  is a subterm of  $t$  and  $\mathbf{tag}_{\varphi'}(t_1, d_1) = \varepsilon$  then  $\mathbf{tag}(t_1[\varphi'], d_1) = \varepsilon$ .

We consider each case in  $\mathbf{tag}_\varphi(t, d)$  separately.

Case:  $\mathbf{tag}_\varphi(1, \bullet) = \varepsilon$

Since  $\mathbf{tag}(1[\varphi], \bullet) = \mathbf{tag}(1, \bullet) = \varepsilon$  the lemma is fulfilled in this case.

Case:  $\mathbf{tag}_\varphi(l\langle t_1 \rangle t_2, \langle d_1 \rangle) = \mathbf{tag}(l\langle t_1[\varphi] \rangle, \langle d_1 \rangle)$

In this case  $\mathbf{tag}(t[\varphi], d) = \mathbf{tag}(l\langle t_1[\varphi] \rangle, \langle d_1 \rangle)$ .

Therefore the lemma is fulfilled in this case.

Case:

$$\begin{aligned} & \mathbf{tag}_\varphi(t_1 t_2, d_1 d_2) \\ = & \text{if } \mathbf{tag}(t_1[\varphi], d_1) = \varepsilon \\ & \text{then } \mathbf{tag}_\varphi(t_2, d_2) \\ & \text{else let } x_1 = \mathbf{tag}_\varphi(t_1, d_1), x_2 = \mathbf{tag}(t_2[\varphi], d_2) \text{ in } x_1 \ x_2 \text{ end} \end{aligned}$$

If  $\mathbf{tag}(t_1[\varphi], d_1) = \varepsilon$  then the induction hypothesis yields that  $\mathbf{tag}(t_2[\varphi], d_2) = \varepsilon$  and therefore  $\mathbf{tag}(t_1 t_2[\varphi], d_1 d_2) = \varepsilon$ .

Therefore the lemma is fulfilled in this case.

If  $\mathbf{tag}(t_1[\varphi], d_1) \neq \varepsilon$  then

$\mathbf{tag}_\varphi(t_1 t_2, d_1 d_2) = \text{let } x_1 = \mathbf{tag}_\varphi(t_1, d_1), x_2 = \mathbf{tag}(t_2[\varphi], d_2) \text{ in } x_1 \ x_2 \text{ end.}$

This means that  $\mathbf{tag}_\varphi(t_1, d_1) = \varepsilon$  and  $\mathbf{tag}(t_2[\varphi], d_2) = \varepsilon$  and therefore the induction hypothesis yields that  $\mathbf{tag}(t_1[\varphi], d_1) = \varepsilon$ .

This means that  $\mathbf{tag}(t_1 t_2[\varphi], d_1 d_2) = \text{let } x_1 = \mathbf{tag}(t_1[\varphi], d_1), x_2 = \mathbf{tag}(t_2[\varphi], d_2) \text{ in } x_1 \ x_2 \text{ end} = \varepsilon$ .

Therefore the lemma is fulfilled in this case.

Case:  $\mathbf{tag}_\varphi(t_1 + t_2, d_1 \diamond) = \mathbf{tag}(t_1, d_1)$

In this case the induction hypothesis yields that  $\mathbf{tag}(t_1[\varphi], d_1) = \varepsilon$ .

Therefore the lemma is fulfilled in this case.

Case:  $\mathbf{tag}_\varphi(t_1 + t_2, \diamond d_2) = \mathbf{tag}(t_2, d_2)$

In this case the induction hypothesis yields that  $\mathbf{tag}(t_2[\varphi], d_2) = \varepsilon$ .

Therefore the lemma is fulfilled in this case.

Case:  $\mathbf{tag}_\varphi(\mu X.t_1, d) = \mathbf{tag}_{\mu X.t_1/X::\varphi}(t_1, \mathbf{simplify}_{\mu X.t_1[\varphi], d}(t_1[\mu X.t_1/X :: \varphi], d))$

Now the induction hypothesis yields that

$\text{tag}(t_1[\mu X.t_1/X :: \varphi], \text{simplify}_{\mu X.t_1[\varphi], d}(t_1[\mu X.t_1/X :: \varphi], d)) = \varepsilon$  and therefore

$$\begin{aligned}
& \text{tag}(\mu X.t_1[\varphi], d) \\
&= \text{tag}(\mu Y.(t_1[Y/X][\varphi]), d) \\
&= \text{tag}(t_1[Y/X][\varphi][\mu Y.(t_1[Y/X][\varphi])/Y], d) \\
&= \text{tag}(t_1[Y/X][\varphi][\mu X.t_1[\varphi]/Y], d) \\
(\text{Lemma 3.4}) \quad &= \text{tag}(t_1[Y/X][\mu X.t_1/Y][\varphi], d) \\
&= \text{tag}(t_1[\mu X.t_1/X][\varphi], d) \\
(\text{Lemma 9.5}) \quad &= \text{tag}(\mu X.t_1[\varphi], \text{simplify}_{\mu X.t_1[\varphi], d}(t_1[\mu X.t_1/X :: \varphi], d)) \\
(\text{Lemma 3.4}) \quad &= \text{tag}(t_1[\mu X.t_1/X][\varphi], \text{simplify}_{\mu X.t_1[\varphi], d}(t_1[\mu X.t_1/X :: \varphi], d)) \\
&= \text{tag}(t_1[\mu X.t_1/X :: \varphi], \text{simplify}_{\mu X.t_1[\varphi], d}(t_1[\mu X.t_1/X :: \varphi], d)) \\
&= \varepsilon
\end{aligned}$$

Therefore the lemma is fulfilled in this case.

Case:  $\text{tag}_\varphi(t, d) = \top$

Since  $\text{tag}_\varphi(t, d) \neq \varepsilon$  the lemma is fulfilled in this case.

We have now proved the lemma for all the cases and we can therefore conclude that

$$\text{tag}_\varphi(t, d) = \varepsilon \Rightarrow \text{tag}(t[\varphi], d) = \varepsilon$$

■

### Proof 19.19 (Completeness of Trans)

We will now prove Lemma 10.8 which states that

$$\text{tr}_{\text{dom}(\varphi)}(t) \Rightarrow \text{Trans}_\varphi(t, d) = \{\} \Rightarrow \text{tag}_\varphi(t, d) \in \{\varepsilon, \top\}$$

by structural induction on  $t$ .

Induction Hypothesis:

If  $t'$  is a subterm of  $t$ ,  $\text{tr}_{\text{dom}(\varphi')}(t')$  and  $\text{Trans}_{\varphi'}(t', d') = \{\}$

then we can assume that  $\text{tag}_{\varphi'}(t', d') \in \{\varepsilon, \top\}$ .

We consider each form of  $t$  separately.

Case:  $t = 0, t = X, t = 1$

In this case  $\text{tag}(t, d) \in \{\varepsilon, \top\}$  and therefore the lemma is fulfilled in this case.

Case:  $t = l\langle t_1 \rangle$

In this case  $\text{Trans}_\varphi(t, d) = \{(0, d\bullet)\}$ .

Therefore the lemma is fulfilled in this case.

Case:  $t = t_1 t_2$

If  $d \neq d_1 d_2$  then  $\text{tag}_\varphi(t, d) = \top$ .

We can therefore assume that  $d = d_1d_2$ .

This means that

$$\begin{aligned}
& \text{Trans}_\varphi(t, d) \\
&= \text{Trans}_\varphi(t_1t_2, d_1d_2) \\
&= \text{if } \text{tag}(t_1, d_1) = \varepsilon \\
&\quad \text{then } \{(|\text{Split}_\varphi(t_1)| + n', d') \mid (n', d') \in \text{Trans}_\varphi(t_2, d_2)\} \\
&\quad \text{else } \{(n', d_{11}(d_{12}d_2)) \mid (n', d_{11}d_{12}) \in \text{Trans}_\varphi(t_1, d_1)\}
\end{aligned}$$

If  $\text{tag}(t_1, d_1) = \varepsilon$  then the induction hypothesis yields that  $\text{tag}_\varphi(t_2, d_2) \in \{\varepsilon, \top\}$ .

Lemma 7.4 yields that  $\text{tag}(t_1[\varphi], d_1) = \varepsilon$  and therefore  $\text{tag}_\varphi(t_1t_2, d_1d_2) = \text{tag}_\varphi(t_2, d_2) = \varepsilon$ .

If  $\text{tag}(t_1, d_1) \neq \varepsilon$  then the induction hypothesis yields that  $\text{tag}_\varphi(t_1, d_1) \in \{\varepsilon, \top\}$ .

Lemma 9.10 yields that  $\text{tag}(t_1[\varphi], d_1) \neq \varepsilon$ , and therefore Lemma 10.7 yields that  $\text{tag}_\varphi(t_1, d_1) \neq \varepsilon$ .

This means that  $\text{tag}_\varphi(t_1, d_1) = \top$  and therefore

$\text{tag}_\varphi(t_1t_2, d_1d_2) = \text{let } x_1 = \text{tag}_\varphi(t_1, d_1), x_2 = \text{tag}(t_2[\varphi], d_2) \text{ in } x_1 \ x_2 \ \text{end} = \top$ .

Therefore the lemma is fulfilled in this case.

Case:  $t = t_1 + t_2$

If  $d \neq d_1 \diamond$  and  $d \neq \diamond d_2$  then  $\text{tag}_\varphi(t, d) = \top$ .

We can therefore assume that  $d = d_1 \diamond$  or  $d = \diamond d_2$ .

If  $d = d_1 \diamond$  then  $\text{Trans}_\varphi(t_1 + t_2, d_1 \diamond) = \text{Trans}_\varphi(t_1, d_1)$

In this case the induction hypothesis yields that  $\text{tag}_\varphi(t_1, d_1) \in \{\varepsilon, \top\}$  and therefore the lemma is fulfilled in this case.

If  $d = \diamond d_2$  then  $\text{Trans}_\varphi(t_1 + t_2, \diamond d_2) = \text{Trans}_\varphi(t_2, d_2)$

In this case the induction hypothesis yields that  $\text{tag}_\varphi(t_2, d_2) \in \{\varepsilon, \top\}$  and therefore the lemma is fulfilled in this case.

Case:  $t = \mu X.t_1$

In this case  $\text{Trans}_\varphi(\mu X.t_1, d) = \text{Trans}_{\mu X.t_1/X::\varphi}(t_1, \text{simplify}_{\mu X.t_1[\varphi], d}(t_1[\mu X.t_1/X::\varphi], d))$ .

Now the induction hypothesis yields that

$\text{tag}_{\mu X.t_1/X::\varphi}(t_1, \text{simplify}_{\mu X.t_1[\varphi], d}(t_1[\mu X.t_1/X::\varphi], d)) \in \{\varepsilon, \top\}$  and therefore

$$\begin{aligned}
& \text{tag}_\varphi(\mu X.t_1, d) \\
&= \text{tag}_{\mu X.t_1/X::\varphi}(t_1, \text{simplify}_{\mu X.t_1[\varphi], d}(t_1[\mu X.t_1/X::\varphi], d)) \\
&\in \{\varepsilon, \top\}
\end{aligned}$$

Therefore the lemma is fulfilled in this case.

We have now proved the lemma for all the cases, and we can therefore conclude that

$$\text{tr}_{\text{dom}(\varphi)}(t) \Rightarrow \text{Trans}_\varphi(t, d) = \{\} \Rightarrow \text{tag}_\varphi(t, d) \in \{\varepsilon, \top\}.$$

■

**Proof 19.20 (Soundness of TransInv)**

We will now prove Lemma 10.13 which states that

$$\forall d \in \text{TransInv}_\varphi(t, D). \text{tag}(t[\varphi], d) = \text{TAG}(\text{Split}_\varphi(t), D)$$

by structural induction on  $t$ .

*Induction Hypothesis:*

If  $t'$  is a subterm of  $t$  then we can assume that

$$\forall d' \in \text{TransInv}_{\varphi'}(t', D'). \text{tag}(t'[\varphi']d') = \text{TAG}(\text{Split}_{\varphi'}(t'), D').$$

We consider each case in  $\text{TransInv}_\varphi(t, D)$  separately.

*Case:  $\text{TransInv}_\varphi(l\langle t_1 \rangle, (0, \langle d_1 \rangle \bullet)) = \{\langle d_1 \rangle\}$*

*In this case we only need to prove the following*

$$\begin{aligned} & \text{TAG}(\text{Split}_\varphi(l\langle t_1 \rangle), (0, \langle d_1 \rangle \bullet)) \\ = & \text{TAG}([l\langle t_1[\varphi] \rangle]1, (0, \langle d_1 \rangle \bullet)) \\ = & \text{tag}(l\langle t_1[\varphi] \rangle]1, \langle d_1 \rangle \bullet) \\ = & \text{let } x_1 = \text{tag}(l\langle t_1[\varphi] \rangle, \langle d_1 \rangle), x_2 = \text{tag}(1, \bullet) \text{ in } x_1 \ x_2 \text{end} \\ = & \text{let } x_1 = \text{tag}(l\langle t_1[\varphi] \rangle, \langle d_1 \rangle), x_2 = \varepsilon \text{ in } x_1 \ x_2 \text{end} \\ = & \text{tag}(l\langle t_1[\varphi] \rangle, \langle d_1 \rangle) \\ = & \text{tag}(l\langle t_1 \rangle[\varphi], \langle d_1 \rangle). \end{aligned}$$

Therefore the lemma is fulfilled in this case.

*Case:  $\text{TransInv}_\varphi(t_1 t_2, (|\text{Split}_\varphi(t_1)| + n, d)) = \{d_1 d_2 \mid d_1 \in \text{esp}(t_1) \wedge d_2 \in \text{TransInv}_\varphi(t_2, (n, d))\}$*

*If we assume that  $\text{esp}(t_1) = \{\}$  then  $\text{TransInv}(t_1 t_2, (|\text{Split}_\varphi(t_1)| + n, d)) = \{\}$  and then the lemma is vacantly fulfilled.*

*We can therefore assume that  $\text{esp}(t_1) \neq \{\}$ .*

*This means that  $\text{Split}_\varphi(t_1 t_2) = [l\langle t_{11} \rangle(t_{12}(t_2[\varphi])) \mid l\langle t_{11} \rangle t_{12} \in \text{Split}_\varphi(t_1)] @ \text{Split}_\varphi(t_2)$ .*

*Lemma 7.5 yields that  $\forall d_1 \in \text{esp}(t_1). \text{tag}(t_1, d_1) = \varepsilon$ , and therefore Lemma 7.4 yields that  $\forall d_1 \in \text{esp}(t_1). \text{tag}(t_1[\varphi], d_1) = \varepsilon$ .*

*The induction hypothesis yields that  $\forall d_2 \in \text{TransInv}_\varphi(t_2, (n, d)). \text{tag}(t_2[\varphi], d_2) = \text{TAG}(\text{Split}_\varphi(t_2), (n, d))$ .*

Now we have that

$$\begin{aligned} & \text{TAG}(\text{Split}_\varphi(t_1 t_2), (|\text{Split}(t_1)| + n, d)) \\ = & \text{TAG}([l\langle t_{11} \rangle(t_{12}(t_2[\varphi])) \mid l\langle t_{11} \rangle t_{12} \in \text{Split}_\varphi(t_1)] @ \text{Split}_\varphi(t_2), (|\text{Split}_\varphi(t_1)| + n, d)) \\ = & \text{TAG}(\text{Split}_\varphi(t_2), (n, d)) \\ \text{(IH)} = & \text{tag}(t_2[\varphi], d_2) \\ (\text{tag}(t_1[\varphi], d_1) = \varepsilon) = & \text{let } x_1 = \text{tag}(t_1[\varphi], d_1), x_2 = \text{tag}(t_2[\varphi], d_2) \text{ in } x_1 \ x_2 \text{end} \\ = & \text{tag}(t_1 t_2[\varphi], d_1 d_2). \end{aligned}$$

Therefore the lemma is fulfilled in this case.

Case:  $\text{TransInv}_\varphi(t_1 t_2, (n, d_{11}(d_{12}d_2))) = \{d_1 d_2 \mid d_1 \in \text{TransInv}_\varphi(t_1, (n, d_{11}d_{12}))\}$

In this case the induction hypothesis yields that  $\forall d_1 \in \text{TransInv}(t_1, (n, d_{11}d_{12})). \text{tag}(t_1[\varphi], d_1) = \text{TAG}(\text{Split}_\varphi(t_1), (n, d_{11}d_{12}))$ .

Therefore we get that

$$\begin{aligned} & \text{TAG}(\text{Split}_\varphi(t_1 t_2), (n, d_{11}(d_{12}d_2))) \\ & \text{TAG}([l\langle t_{11} \rangle t_1 2(t_1[\varphi]) \mid l\langle t_{11} \rangle t_1 2 \in \text{Split}_\varphi(t_1)]@T, (n, d_{11}(d_{12}d_2))) \\ (n < |\text{Split}_\varphi(t_1)|) &= \text{let } x_1 = \text{TAG}(\text{Split}_\varphi(t_1), (n, d_{11}d_{12})), x_2 = \text{tag}(t_2[\varphi], d_2) \text{ in } x_1 \ x_2 \ \text{end} \\ \text{(IH)} &= \text{let } x_1 = \text{tag}(t_1[\varphi], d_1), x_2 = \text{tag}(t_2[\varphi], d_2) \text{ in } x_1 \ x_2 \ \text{end} \\ (\text{tag}(t_1[\varphi], d_1) \neq \varepsilon) &= \text{tag}(t_1 t_2, d_1 d_2). \end{aligned}$$

Therefore the lemma is fulfilled in this case.

Case:  $\text{TransInv}_\varphi(t_1 + t_2, (|\text{Split}_\varphi(t_1)| + n, d)) = \{\diamond d_2 \mid d_2 \in \text{TransInv}_\varphi(t_2, (n, d))\}$

In this case the induction hypothesis yields that

$\forall d_2 \in \text{TransInv}_\varphi(t_2, (n, d)). \text{tag}(t_2[\varphi], d_2) = \text{TAG}(\text{Split}_\varphi(t_2), (n, d))$ .

Therefore we have that

$$\begin{aligned} & \text{TAG}(\text{Split}_\varphi(t_1 + t_2), (|\text{Split}_\varphi(t_1)| + n, d)) \\ &= \text{TAG}(\text{Split}_\varphi(t_2), (n, d)) \\ \text{(IH)} &= \text{tag}_\varphi(t_2, d_2) \\ &= \text{tag}_\varphi(t_1 + t_2, d_2 \diamond). \end{aligned}$$

Therefore the lemma is fulfilled in this case.

Case:  $\text{TransInv}_\varphi(t_1 + t_2, D) = \{d_1 \diamond \mid d_1 \in \text{TransInv}_\varphi(t_1, D)\}$

Since the previous case was not used, we know that  $D = (n_0, d_0)$  where  $n_0 < |\text{Split}(t_1)|$ .

The induction hypothesis yields that

$\forall d_1 \in \text{TransInv}(t_1, D). \text{tag}(t_1[\varphi], d_1) = \text{TAG}(\text{Split}_\varphi(t_1), D)$ .

Therefore we get that

$$\begin{aligned} & \text{TAG}(\text{Split}_\varphi(t_1 + t_2, D)) \\ &= \text{TAG}(\text{Split}_\varphi(t_1), D) \\ \text{(IH)} &= \text{tag}(t_1[\varphi], d_1) \\ &= \text{tag}(t_1 + t_2[\varphi], d_1 \diamond). \end{aligned}$$

Therefore the lemma is fulfilled in this case.

Case:  $\text{TransInv}_\varphi(\mu X.t_1, D) = \text{TransInv}_{\mu X.t_1/X::\varphi}(t_1, D)$

In this case the induction hypothesis yields that

$\forall d \in \text{TransInv}_{\mu X.t_1/X::\varphi}(t_1, D). \text{tag}(t_1[\mu X.t_1/X :: \varphi], d) = \text{TAG}(\text{Split}_{\mu X.t_1/X::\varphi}(t_1), D)$ .

Therefore we have that

$$\begin{aligned}
& \text{tag}(\mu X.t_1[\varphi], d) \\
&= \text{tag}(\mu Y.(t_1[Y/X][\varphi]), d) \quad \text{where } Y \notin \text{dom}(\varphi) \cup \text{fv}(\mu X.t_1) \text{ and } t'/Z \in \varphi \Rightarrow Y \notin \text{fv}(t') \\
&= \text{tag}(t_1[Y/X][\varphi][\mu Y.(t_1[Y/X][\varphi])/Y], d) \\
&= \text{tag}(t_1[Y/X][\varphi][\mu X.t_1[\varphi]/Y], d) \\
&= \text{tag}(t_1[Y/X][\mu X.t_1/Y][\varphi], d) \\
&= \text{tag}(t_1[Y/X][\mu X.t_1/Y :: \varphi], d) \\
(\text{IH}) &= \text{TAG}(\text{Split}_{\mu X.t_1/X :: \varphi}(t_1, D)) \\
&= \text{TAG}(\text{Split}_{\varphi}(\mu X.t_1), D).
\end{aligned}$$

Therefore the lemma is fulfilled in this case.

Case:  $\text{TransInv}_{\varphi}(-, -) = \{\}$

In this case  $\text{TransInv}_{\varphi}(t, D) = \{\}$  and therefore the lemma is vacantly fulfilled.

We have now proved the lemma in all the cases, and we can therefore conclude that

$$\forall d \in \text{TransInv}_{\varphi}(t, D). \text{tag}(t[\varphi], d) = \text{TAG}(\text{Split}_{\varphi}(t), D).$$

■

### Proof 19.21 (Completeness of TransInv)

We will now prove Lemma 10.15 which states that

$$\text{TAG}(\text{Split}_{\varphi}(t), D) \neq \top \Rightarrow \text{TransInv}_{\varphi}(t, D) \neq \{\}$$

by structural induction on  $t$ .

Induction Hypothesis:

If  $t'$  is a subterm of  $t$  and  $\text{TAG}(\text{Split}_{\varphi'}(t'), D') \neq \top$  then we can assume that  $\text{TransInv}_{\varphi'}(t', D') \neq \{\}$

We will handle each case in  $\text{TransInv}_{\varphi}(t, d)$  separately.

Case:  $\text{TransInv}_{\varphi}(l\langle t_1 \rangle, (0, \langle d_1 \rangle \bullet)) = \{\langle d_1 \rangle\}$

In this case  $\text{TransInv}_{\varphi}(t, D) = \{\langle d_1 \rangle\} \neq \{\}$ , so the lemma is fulfilled.

Case:  $\text{TransInv}_{\varphi}(t_1 t_2, (|\text{Split}_{\varphi}(t_1)| + n, d)) = \{d_1 d_2 \mid d_1 \in \text{esp}(t_1) \wedge d_2 \in \text{TransInv}_{\varphi}(t_2, (n, d))\}$

If  $\text{esp}(t_2) = \{\}$  then  $\text{Split}_{\varphi}(t_1 t_2) = [l\langle t_{11} \rangle(t_{12}(t_2[\varphi])) \mid l\langle t_{11} \rangle t_{12} \in \text{Split}_{\varphi}(t_1)]$  and therefore  $\text{TAG}(\text{Split}_{\varphi}(t_1 t_2), (|\text{Split}_{\varphi}(t_1)| + n, d)) = \top$ .

Therefore we can assume that  $\text{esp}(t_2) \neq \{\}$ , which means that

$$\text{Split}(t_1 t_2) = [l\langle t_{11} \rangle(t_{12}(t_2[\varphi])) \mid l\langle t_{11} \rangle t_{12} \in \text{Split}_{\varphi}(t_1)] @ \text{Split}_{\varphi}(t_2).$$

We now have that

$$\begin{aligned}
\top &\neq \text{TAG}(\text{Split}_{\varphi}(t_1 t_2), (|\text{Split}_{\varphi}(t_1)| + n, d)) \\
&= \text{TAG}([l\langle t_{11} \rangle(t_{12}(t_2[\varphi])) \mid l\langle t_{11} \rangle t_{12} \in \text{Split}_{\varphi}(t_1)] @ \text{Split}_{\varphi}(t_2), (|\text{Split}_{\varphi}(t_1)| + n, d)) \\
&= \text{TAG}(\text{Split}_{\varphi}(t_2), (n, d)).
\end{aligned}$$

Therefore the induction hypothesis yields that  $\text{TransInv}_\varphi(t_2, (n, d)) \neq \{\}$ .

We can therefore conclude that

$$\begin{aligned} & \text{TransInv}_\varphi(t_1 t_2, (|\text{Split}_\varphi(t_1)| + n, d)) \\ &= \{d_1 d_2 \mid d_1 \in \text{esp}(t_1) \wedge d_2 \in \text{TransInv}_\varphi(t_2, (n, d))\} \\ (\text{esp}(t_1) \neq \{\} \wedge \text{TransInv}(t_2, (n, d)) \neq \{\}) &\neq \{\} \end{aligned}$$

and therefore the lemma is fulfilled in this case.

Case:  $\text{TransInv}_\varphi(t_1 t_2, (n, d_{11}(d_{12}d_2))) = \{d_1 d_2 \mid d_1 \in \text{TransInv}(t_1, (n, d_{11}d_{12}))\}$

Since

$$\begin{aligned} \top &\neq \text{TAG}(\text{Split}_\varphi(t_1 t_2), (n, d_{11}(d_{12}d_2))) \\ (n < |\text{Split}_\varphi(t_1)|) &= \text{TAG}([l\langle t_{11} \rangle(t_{12}(t_2[\varphi])) \mid l\langle t_{11} \rangle t_{12} \in \text{Split}_\varphi(t_1)], (n, d_{11}(d_{12}d_2))) \\ &= \text{let } x_1 = \text{TAG}(\text{Split}_\varphi(t_1), (n, d_{11}d_{12})), x_2 = \text{tag}(t_2[\varphi], d_2) \text{ in } x_1 \ x_2 \text{end} \end{aligned}$$

we have that  $\text{TAG}(\text{Split}(t_1), (n, d_{11}d_{12})) \neq \top$ .

Now the induction hypothesis yields that  $\text{TransInv}_\varphi(t_1, (n, d_{11}d_{12})) \neq \{\}$  and therefore

$$\begin{aligned} & \text{TransInv}_\varphi(t_1 t_2, (n, d_{11}(d_{12}d_2))) \\ &= \{d_1 d_2 \mid d_1 \in \text{TransInv}(t_1, (n, d_{11}d_{12}))\} \\ &\neq \{\}. \end{aligned}$$

Therefore the lemma is fulfilled in this case.

Case:  $\text{TransInv}_\varphi(t_1 + t_2, (|\text{Split}_\varphi(t_1)| + n, d)) = \{\diamond d_2 \mid d_2 \in \text{TransInv}_\varphi(t_2, (n, d))\}$

Since

$$\begin{aligned} \top &\neq \text{TAG}(\text{Split}_\varphi(t_1 + t_2), (|\text{Split}_\varphi(t_1)| + n, d)) \\ &= \text{TAG}(\text{Split}_\varphi(t_1) @ \text{Split}_\varphi(t_2), (|\text{Split}(t_1)| + n, d)) \\ &= \text{TAG}(\text{Split}_\varphi(t_2), (n, d)) \end{aligned}$$

the induction hypothesis yields that  $\text{TransInv}_\varphi(t_2, (n, d)) \neq \{\}$ .

This means that

$$\begin{aligned} & \text{TransInv}_\varphi(t_1 + t_2, (|\text{Split}_\varphi(t_1)| + n, d)) \\ &= \{\diamond d_2 \mid d_2 \in \text{TransInv}_\varphi(t_2, (n, d))\} \\ &\neq \{\} \end{aligned}$$

and therefore the lemma is fulfilled in this case.

Case:  $\text{TransInv}_\varphi(t_1 + t_2, D) = \{d_1 \diamond \mid d_1 \in \text{TransInv}_\varphi(t_1, D)\}$

Since

$$\begin{aligned} \top &\neq \text{TAG}(\text{Split}_\varphi(t_1 + t_2), D) \\ &= \text{TAG}(\text{Split}_\varphi(t_1) @ \text{Split}_\varphi(t_2), D) \\ (D = (n, d) \text{ where } n < |\text{Split}_\varphi(t_1)|) &= \text{TAG}(\text{Split}_\varphi(t_1), D) \end{aligned}$$

the induction hypothesis yields that  $\text{TransInv}_\varphi(t_1, D) \neq \{\}$ .  
This means that

$$\begin{aligned} & \text{TransInv}(t_1 + t_2, D) \\ &= \{d_1 \diamond \mid d_1 \in \text{TransInv}_\varphi(t_1, D)\} \\ &\neq \{\} \end{aligned}$$

and therefore the lemma is fulfilled in this case.

Case:  $\text{TransInv}_\varphi(\mu X.t_1, D) = \text{TransInv}_{\mu X.t_1/X::\varphi}(t_1, D)$

Since

$$\begin{aligned} \top &\neq \text{TAG}(\text{Split}_\varphi(\mu X.t_1), D) \\ &= \text{TAG}(\text{Split}_{\mu X.t_1/X::\varphi}(t_1), D) \end{aligned}$$

the induction hypothesis yields that  $\text{TransInv}_\varphi(\mu X.t_1, D) = \text{TransInv}_{\mu X.t_1/X::\varphi}(t_1, D) \neq \{\}$ ,  
and therefore the lemma is fulfilled in this case.

Case:  $\text{TransInv}_\varphi(-, -) = \{\}$

In none of the above cases match then  $\text{TAG}(\text{Split}_\varphi(t), D) = \top$ , and therefore the lemma is trivially fulfilled in this case.

Here are the possibilities that have not yet been matched

$t = 0$  or  $t = 1$  or  $t = X$ : Since  $\text{Split}_\varphi(t) = []$  we have that  $\text{TAG}(\text{Split}_\varphi(t), D) = \text{TAG}([], D) = \top$ .

$t = l\langle t_1 \rangle$  and  $D = (1 + n, d')$ : Since  $\text{Split}_\varphi(l\langle t_1 \rangle) = [l\langle t_1 \rangle 1]$  we have that  $\text{TAG}(\text{Split}_\varphi(t), D) = \text{TAG}([l\langle t_1 \rangle 1], (1 + n, d')) = \text{TAG}([], (n, d')) = \top$ .

$t = l\langle t_1 \rangle$  and  $D = (0, d)$  where  $d \neq \langle d_1 \rangle \bullet$ :  $\text{TAG}(\text{Split}_\varphi(t), D) = \text{tag}(l\langle t_1 \rangle 1, d) = \top$ .

$t = t_1 + t_2$  and  $D = (n, d)$  where  $n < |\text{Split}(t_1)|$  and  $d \neq d_{11}(d_{12}d_2)$ :  $\text{TAG}(\text{Split}(t), D) = \text{TAG}([t_{11}(t_{12}t_2) \mid t_{11}t_{12} \in \text{Split}_\varphi(t_1)], (n, d)) = \top$ .

We have now proved the lemma in all the cases, and we can therefore conclude that

$$\text{TAG}(\text{Split}_\varphi(t), D) \neq \top \Rightarrow \text{TransInv}_\varphi(t, D) \neq \{\}.$$

■

### Proof 19.22 (Soundness of TRANS)

We will now prove Lemma 10.18 which states that

$$\forall D' \in \text{TRANS}(T, D). \text{TAG}(T, D) = \text{TAG}(\text{SPLIT}(T), D')$$

by induction on the calltree of  $\text{TRANS}(T, D)$ .

*Induction Hypothesis:*

If  $\text{TRANS}(T, D)$  calls  $\text{TRANS}(T_1, D_1)$ , we can assume that  
 $\forall D'_1 \in \text{TRANS}(T_1, D_1). \text{TAG}(T_1, D_1) = \text{TAG}(\text{SPLIT}(T_1), D'_1)$ .

Case:  $\text{TRANS}([], \_) = \{\}$

Since  $\text{TRANS}(T, D) = \{\}$ , the lemma is trivially fulfilled in this case.

Case:  $\text{TRANS}(t_1 :: T_1, (0, d)) = \text{Trans}(t_1, d)$

In this case Lemma 10.5 yields that  $\forall D' \in \text{Trans}(t_1, d). \text{TAG}(\text{Split}(t_1), D') = \text{tag}(t_1, d)$ .

Since

$$\begin{aligned} \text{TAG}(t_1 :: T_1, (0, d)) &= \text{tag}(t_1, d) \\ &= \text{TAG}(\text{Split}(t_1), D') \\ &= \text{TAG}(\text{Split}(t_1) @ \text{SPLIT}(T_1), D') \\ &= \text{TAG}(\text{SPLIT}(t_1 :: T_1), D') \end{aligned}$$

the lemma is fulfilled in this case.

Case:  $\text{TRANS}(t_1 :: T_1, (n+1, d)) = \{(n' + |\text{Split}(t_1)|, d') \mid (n', d') \in \text{TRANS}(T_1, (n, d))\}$

In this case  $\text{TRANS}(T, D)$  calls  $\text{TRANS}(T_1, (n, d))$  and therefore the induction hypothesis yields that  $\forall (n'_1, d'_1) \in \text{TRANS}(T_1, (n, d)). \text{TAG}(\text{SPLIT}(T_1), (n'_1, d'_1)) = \text{TAG}(T_1, (n, d))$ .

Since

$$\begin{aligned} &\text{TAG}(t_1 :: T_1, (n+1, d)) \\ &= \text{TAG}(T_1, (n, d)) \\ \text{(IH)} &= \text{TAG}(\text{SPLIT}(T_1), (n'_1, d'_1)) \\ &= \text{TAG}(\text{Split}(t_1) @ \text{SPLIT}(T_1), (n'_1 + |\text{Split}(t_1)|, d'_1)) \\ &= \text{TAG}(\text{SPLIT}(t_1 :: T_1), (n'_1 + |\text{Split}(t_1)|, d'_1)) \end{aligned}$$

the lemma is fulfilled in this case.

We have now proved the lemma for all the cases, and we can therefore conclude that

$$\forall D' \in \text{TRANS}(T, D). \text{TAG}(T, D) = \text{TAG}(\text{SPLIT}(T), D').$$

■

### **Proof 19.23 (Completeness of TRANS)**

We will now prove Lemma 10.19 which states that

$$\text{TAG}(T, D) \notin \{\varepsilon, \top\} \Rightarrow \text{TRANS}(T, D) \neq \{\}$$

by induction on the calltree of  $\text{TRANS}(T, D)$ .

*Induction Hypothesis:*

If  $\text{TRANS}(T, D)$  calls  $\text{TRANS}(T', D')$  and  $\text{TAG}(T', D') \notin \{\varepsilon, \top\}$

then we can assume that  $\text{TRANS}(T, D) \neq \{\}$ .

We will handle each case in *TRANS* separately.

Case:  $\text{TRANS}(\square, \_) = \{\}$

In this case  $\text{TAG}(T, D) = \top$  and therefore the lemma is trivially fulfilled in this case.

Case:  $\text{TRANS}(t_1 :: T_1, (0, d)) = \text{Trans}(t_1, d)$

In this case  $\text{TAG}(T, D) = \text{TAG}(t_1 :: T_1, (0, d)) = \text{tag}(t_1, d)$ .

We can therefore assume that  $\text{tag}(t_1, d) \notin \{\varepsilon, \top\}$ .

Now Lemma 10.8 yields that  $\text{Trans}(t_1, d) \neq \{\}$ .

Therefore the lemma is fulfilled in this case.

Case:  $\text{TRANS}(t_1 :: T_1, (n+1, d)) = \{(n' + |\text{Split}(t_1)|, d') \mid (n', d') \in \text{TRANS}(T_1, (n, d))\}$

In this case  $\text{TAG}(T, D) = \text{TAG}(t_1 :: T_1, ((n+1), d)) = \text{TAG}(T_1, (n, d))$ .

We can therefore assume that  $\text{TAG}(T_1, (n, d)) \neq \{\varepsilon, \top\}$ .

Now the induction hypothesis yields that  $\text{TRANS}(T_1, (n, d)) \neq \{\}$ .

Therefore the lemma is fulfilled in this case.

We have now proved the lemma for all the cases, and we can therefore conclude that

$$\text{TAG}(T, D) \notin \{\varepsilon, \top\} \Rightarrow \text{TRANS}(T, D) \neq \{\}.$$

■

### Proof 19.24 (Soundness of *TRANSINV*)

We will now prove Lemma 10.21 which states that

$$\text{TAG}(\text{SPLIT}(T), D') \neq \top \Rightarrow \forall D \in \text{TRANSINV}(T, D'). \text{TAG}(T, D) = \text{TAG}(\text{SPLIT}(T), D')$$

by induction on the calltree of *TRANSINV*( $T, D'$ ).

*Induction Hypothesis:*

If  $\text{TRANSINV}(T, D')$  calls  $\text{TRANSINV}(T_1, D'_1)$  and  $\text{TAG}(\text{SPLIT}(T_1), D'_1) \neq \top$  then we can assume that  $\forall D_1 \in \text{TRANSINV}(T_1, D'_1)$  we have that  $\text{TAG}(\text{SPLIT}(T_1), D'_1) = \text{TAG}(T_1, D_1)$ .

We will consider each case in  $\text{TRANSINV}(T, D')$  separately.

Case:  $\text{TRANSINV}(\square, \_) = \{\}$

Since  $\text{TRANSINV}(T, D') = \{\}$  the lemma is trivially fulfilled in this case.

Case:  $\text{TRANSINV}(t_1 :: T_1, (n' + |\text{Split}(t_1)|, d')) = \{(n+1, d) \mid (n, d) \in \text{TRANSINV}(T_1, (n', d'))\}$

In this case the induction hypothesis yields that

$$\forall D_1 \in \text{TRANSINV}(T_1, (n', d')). \text{TAG}(\text{SPLIT}(T_1), (n', d')) = \text{TAG}(T_1, D_1).$$

If  $D \in \text{TRANSINV}(T, D') = \text{TRANSINV}(t_1 :: T_1, (n' + |\text{Split}(t_1)|, d'))$  then  $D = (n+1, d)$  where  $(n, d) \in \text{TRANSINV}(T_1, (n', d'))$ .

Therefore we have that

$$\begin{aligned}
& \text{TAG}(t_1 :: T_1, D) \\
&= \text{TAG}(t_1 :: T_1, (n+1, d)) \\
&= \text{TAG}(T_1, (n, d)) \\
(\mathbf{IH}) \quad &= \text{TAG}(\text{SPLIT}(T_1), (n', d')) \\
&= \text{TAG}(\text{Split}(t_1) @ \text{SPLIT}(T_1), (n' + |\text{Split}(t_1)|, d')) \\
&= \text{TAG}(\text{SPLIT}(T), D').
\end{aligned}$$

Therefore the lemma is fulfilled in this case.

Case:  $\text{TRANSINV}(t_1 :: T_1, (n', d')) = \{(0, d) \mid d \in \text{TransInv}(t_1, (n', d'))\}$   
Assume that  $(n, d) \in \text{TRANSINV}(T, D')$ , then  $n = 0$  and  $d \in \text{TransInv}(t_1, (n', d'))$ .  
Therefore we get that

$$\begin{aligned}
& \text{TAG}(\text{SPLIT}(T), D') \\
&= \text{TAG}(\text{SPLIT}(t_1 :: T_2), (n', d')) \\
&= \text{TAG}(\text{Split}(t_1) @ \text{SPLIT}(T_1), (n', d')) \\
(n' < |\text{Split}(t_1)|) \quad &= \text{TAG}(\text{Split}(t_1), (n', d')) \\
(\mathbf{Lemma 10.14}) \quad &= \text{tag}(t_1, d) \\
&= \text{TAG}(t_1 :: T_1, (0, d)) \\
&= \text{TAG}(T, D).
\end{aligned}$$

therefore the lemma is fulfilled in this case.

We have now proved the lemma in all the cases, and we can therefore conclude that

$$\text{TAG}(\text{SPLIT}(T), D') \neq \top \Rightarrow \forall D \in \text{TransInv}(T, D'). \text{TAG}(T, D) = \text{TAG}(\text{SPLIT}(T), D').$$

■

### Proof 19.25 (Completeness of TRANSINV)

We will now prove Lemma 10.22 which states that

$$\forall T = [t_1, t_2, \dots, t_k] \in \mathbf{XMLTypes}. \forall D' = (n', d') \in \mathbf{XMLDatas}. \forall j \in \{1, 2, \dots, k\}.$$

$$\begin{aligned}
& \text{TAG}(\text{SPLIT}(T), D') \neq \top \wedge \sum_{i=1}^{j-1} |\text{Split}(t_i)| \leq n' < \sum_{i=1}^j |\text{Split}(t_i)| \Rightarrow \\
& \{(n, d) \in \text{TRANSINV}(T, D') \mid n = j\} \neq \{\}
\end{aligned}$$

by induction on  $|T|$ .

*Induction Hypothesis:*

If  $T = t_1 :: T_1$ , then we can assume that

$$\begin{aligned}
& \forall D'_1 = (n'_1, d'_1) \in \mathbf{XMLDatas}. \forall j_1 \in \{1, 2, \dots, k-1\}. \\
& \text{TAG}(\text{SPLIT}(T_1), D'_1) \neq \top \wedge \sum_{i=1}^{j_1-1} |\text{Split}(t_{i+1})| \leq n'_1 < \sum_{i=1}^{j_1} |\text{Split}(t_{i+1})| \Rightarrow
\end{aligned}$$

$$\{(n_1, d_1) \in \mathbf{TransInv}(T_1, D'_1) \mid n_1 = j_1\} \neq \{\}$$

We will consider each case in  $\mathbf{TRANSINV}(T, D')$  independently.

Case:  $\mathbf{TRANSINV}(\[], \_) = \{\}$

In this case  $\mathbf{TAG}(\mathbf{SPLIT}(\[]), D') = \top$  and therefore the lemma is trivially fulfilled in this case.

Case:  $\mathbf{TRANSINV}(t_1 :: T_1, (n' + |\mathbf{Split}(t_1)|, d')) = \{(n+1), d \mid (n', d') \in \mathbf{TRANSINV}(T_1, (n', d'))\}$

In this case

$$\begin{aligned} & \mathbf{TAG}(\mathbf{SPLIT}(T), D') \\ &= \mathbf{TAG}(\mathbf{SPLIT}(t_1 :: T_1), (n' + |\mathbf{Split}(t_1)|, d')) \\ &= \mathbf{TAG}(\mathbf{Split}(t_1)@\mathbf{SPLIT}(T_1), (n' + |\mathbf{Split}(t_1)|, d')) \\ &= \mathbf{TAG}(\mathbf{SPLIT}(T_1), (n', d')). \end{aligned}$$

Therefore we can assume that  $\mathbf{TAG}(\mathbf{SPLIT}(T_1), (n', d')) \neq \top$ .

Now the induction hypothesis yields that

$$\forall j_1 \in \{1, 2, \dots, k-1\}.$$

$$\begin{aligned} \Sigma_{i=1}^{j_1-1} |\mathbf{Split}(t_{i+1})| \leq n' < \Sigma_{i=1}^{j_1} |\mathbf{Split}(t_{i+1})| \Rightarrow \\ \{(n_1, d_1) \in \mathbf{TRANSINV}(T_1, D'_1) \mid n_1 = j_1\} \neq \{\} \end{aligned}$$

Assume that  $\Sigma_{i=1}^{j-1} |\mathbf{Split}(t_{i+1})| \leq n' + |\mathbf{Split}(t_1)| < \Sigma_{i=1}^j |\mathbf{Split}(t_i)|$  then

$\Sigma_{i=2}^{j-1} |\mathbf{Split}(t_i)| \leq n' < \Sigma_{i=2}^j |\mathbf{Split}(t_i)|$  and therefore

$$\Sigma_{i=1}^{j-1} |\mathbf{Split}(t_{i+1})| \leq n' < \Sigma_{i=1}^j |\mathbf{Split}(t_{i+1})|.$$

This means that

$\{(n_1, d_1) \in \mathbf{TRANSINV}(T_1, (n', d')) \mid n_1 = j-1\} \neq \{\}$  and therefore

$\{(n, d) \in \mathbf{TRANSINV}(t_1 :: T_1, (n' + |\mathbf{Split}(t_1)|, d')) \mid n = j\} \neq \{\}$ .

Therefore the lemma is fulfilled in this case.

Case:  $\mathbf{TRANSINV}(t_1 :: T_1, (n', d')) = \{(0, d') \mid d' \in \mathbf{TransInv}(t_1, (n, d))\}$

In this case  $n' < |\mathbf{Split}(t_1)|$  and therefore  $n' < \Sigma_{i=1}^j |\mathbf{Split}(t_i)|$  is only fulfilled for  $j = 0$ .

Since

$$\begin{aligned} & \mathbf{TAG}(\mathbf{SPLIT}(T), D') \\ &= \mathbf{TAG}(\mathbf{SPLIT}(t_1 :: T_1), (n', d')) \\ &= \mathbf{TAG}(\mathbf{Split}(t_1)@\mathbf{SPLIT}(T_1), (n', d')) \\ &= \mathbf{TAG}(\mathbf{Split}(t_1), (n', d')) \end{aligned}$$

Lemma 10.15 yields that  $\mathbf{TransInv}(t_1, (n', d')) \neq \{\}$  and therefore

$\{(n, d) \in \mathbf{TRANSINV}(t_1 :: T_1, (n', d')) \mid n = 0\} \neq \{\}$ .

Therefore the lemma is fulfilled in this case.

We have now proved the lemma for all cases and we can therefore conclude that

$$\forall T = [t_1, t_2, \dots, t_k] \in \mathbf{XMLTypes}. \forall D' = (n', d') \in \mathbf{XMLDat}. \forall j \in \{1, 2, \dots, k\}.$$

$$\begin{aligned} \text{TAG}(\text{SPLIT}(T), D') \neq \top \wedge \sum_{i=1}^{j-1} |\text{Split}(t_i)| \leq n' < \sum_{i=1}^j |\text{Split}(t_i)| \Rightarrow \\ \{(n, d) \in \text{TRANSINV}(T, D') \mid n = j\} \neq \{ \}. \end{aligned}$$

■

**Proof 19.26 (Substitution Lemma for esp)**

We will now prove Lemma 11.3 which states that

$$X \notin \text{fv}'(t) \Rightarrow \text{esp}(t) = \text{esp}(t[t'/X])$$

by induction on  $|t|$ .

*Induction Hypothesis:*

If  $|t_1| < |t|$  and  $X \notin \text{fv}'(t_1)$  then we can assume that  $\forall t''. \text{esp}(t_1[t''/X]) = \text{esp}(t_1)$ .

Case:  $\text{esp}(0) = \{ \}$

Since  $t[t'/X] = t$  the lemma is fulfilled in this case, because  $\text{esp}$  is a function.

Case:  $\text{esp}(1) = \{ \bullet \}$

Since  $t[t'/X] = t$  the lemma is fulfilled in this case, because  $\text{esp}$  is a function.

Case:  $\text{esp}(l\langle t_1 \rangle) = \{ \}$

In this case  $\text{esp}(t[t'/X]) = \text{esp}(l\langle t_1 \rangle[t'/X]) = \text{esp}(l\langle t_1[t'/X] \rangle) = \{ \}$ .

Therefore the lemma is fulfilled in this case.

Case:  $\text{esp}(t_1 + t_2) = \text{if } \text{esp}(t_1) \neq \{ \} \text{ then } \text{esp}(t_1) \text{ else } \text{esp}(t_2)$

Since  $\text{fv}'(t_1 + t_2) = \text{fv}'(t_1) \cup \text{fv}'(t_2)$  the induction hypothesis yields that

$\text{esp}(t_1) = \text{esp}(t_1[t'/X])$  and

$\text{esp}(t_2) = \text{esp}(t_2[t'/X])$  and therefore we have that

$$\begin{aligned} \text{esp}(t_1 + t_2) &= \text{if } \text{esp}(t_1) \neq \{ \} \text{ then } \text{esp}(t_1) \text{ else } \text{esp}(t_2) \\ (2 \times \mathbf{IH}) &= \text{if } \text{esp}(t_1[t'/X]) \neq \{ \} \text{ then } \text{esp}(t_1[t'/X]) \text{ else } \text{esp}(t_2[t'/X]) \\ &= \text{esp}((t_1[t'/X]) + (t_2[t'/X])) \\ &= \text{esp}(t_1 + t_2[t'/X]). \end{aligned}$$

Therefore the lemma is fulfilled in this case.

Case:  $\text{esp}(t_1 t_2) = \{ d_1 d_2 \mid d_1 \in \text{esp}(t_1) \wedge d_2 \in \text{esp}(t_2) \}$

Since  $\text{fv}'(t_1 t_2) = \text{fv}'(t_1) \cup \text{fv}'(t_2)$  the induction hypothesis yields that

$\text{esp}(t_1) = \text{esp}(t_1[t'/X])$  and

$\text{esp}(t_2) = \text{esp}(t_2[t'/X])$  and therefore we have that

$$\begin{aligned} \text{esp}(t_1 t_2) &= \{ d_1 d_2 \mid d_1 \in \text{esp}(t_1) \wedge d_2 \in \text{esp}(t_2) \} \\ (2 \times \mathbf{IH}) &= \{ d_1 d_2 \mid d_1 \in \text{esp}(t_1[t'/X]) \wedge d_2 \in \text{esp}(t_2[t'/X]) \} \\ &= \text{esp}((t_1[t'/X])(t_2[t'/X])) \\ &= \text{esp}(t_1 t_2[t'/X]). \end{aligned}$$

Therefore the lemma is fulfilled in this case.

Case:  $\mathbf{esp}(\mu Y.t_1) = \mathbf{esp}(t_1)$

Since  $\mathbf{fv}'(t) = \mathbf{fv}'(\mu Y.t_1) = \mathbf{fv}'(t_1) \setminus \{Y\}$  we have that if  $X \notin \mathbf{fv}'(t)$  then  $X \notin \mathbf{fv}'(t_1[Z/Y])$  if  $Z \neq X$ .

Now the induction hypothesis yields that  $\mathbf{esp}(t_1[Z/Y]) = \mathbf{esp}(t_1[Z/Y][t'/X])$ , and therefore we have that

$$\begin{aligned}
 & \mathbf{esp}(\mu Y.t_1[t'/X]) \\
 &= \mathbf{esp}(\mu Y.(t_1[Z/Y][t'/X])) \\
 &= \mathbf{esp}(t_1[Z/Y][t'/X]) \\
 (\mathbf{IH}) \quad &= \mathbf{esp}(t_1[Z/Y]) \\
 &= \mathbf{esp}(\mu Z.(t_1[Z/Y])) \\
 &= \mathbf{esp}(\mu Y.t_1).
 \end{aligned}$$

Therefore the lemma is fulfilled in this case.

Case:  $\mathbf{esp}(Y) = \{\}$

Since  $\mathbf{fv}'(Y) = \{Y\}$  we know that  $X \neq Y$ .

Therefore we know that  $t[t'/X] = Y[t'/X] = Y = t$ , and therefore the lemma is fulfilled in this case, since  $\mathbf{esp}$  is a function.

We have now proved the lemma for all cases, and can conclude that

$$X \notin \mathbf{fv}'(t) \Rightarrow \mathbf{esp}(t) = \mathbf{esp}(t[t'/X]).$$

■

### Proof 19.27 (Substitution Lemma for Split)

We will now prove Lemma 11.4 which states that

$$X \notin \mathbf{fv}'(t) \Rightarrow \mathbf{Split}_\varphi(t[t_X/X]) = [t'[t_X/X :: \varphi] \mid t' \in \mathbf{Split}(t)]$$

by induction on  $|t|$ .

*Induction Hypothesis:*

If  $|t_1| < |t|$  and  $X \notin \mathbf{fv}'(t')$  then we can assume that

$\mathbf{Split}_\varphi(t_1[t''/X]) = [t'[t''/X :: \varphi'] \mid t' \in \mathbf{Split}(t_1)]$  for all  $t''$ .

We consider each form of  $t$  separately.

Case:  $t = 0$

In this case

$\mathbf{Split}(t) = \mathbf{Split}(0) = []$ , and

$\mathbf{Split}_\varphi(t[t_X/X]) = \mathbf{Split}_\varphi(0[t_X/X]) = \mathbf{Split}_\varphi(0) = []$ .

Therefore the lemma is fulfilled in this case.

Case:  $t = 1$

In this case

$\mathbf{Split}(t) = \mathbf{Split}(1) = []$ , and

$\mathbf{Split}_\varphi(t[t_X/X]) = \mathbf{Split}_\varphi(1[t_X/X]) = \mathbf{Split}_\varphi(1) = []$ .

Therefore the lemma is fulfilled in this case.

Case:  $t = l\langle t_1 \rangle$

In this case  $\mathbf{Split}(l\langle t_1 \rangle) = [l\langle t_1 \rangle 1]$ .

This means that

$$\begin{aligned}
& \mathbf{Split}_\varphi(t[t_X/X]) \\
&= \mathbf{Split}_\varphi(l\langle t_1 \rangle[t_X/X]) \\
&= \mathbf{Split}_\varphi(l\langle t_1[t_X/X] \rangle) \\
&= [l\langle t_1[t_X/X :: \varphi] \rangle 1] \\
&= [l\langle t_1 \rangle 1[t_X/X :: \varphi]] \\
&= [t'[t_X/X :: \varphi] \mid t' \in [l\langle t_1 \rangle 1]] \\
&= [t'[t_X/X :: \varphi] \mid t' \in \mathbf{Split}(t)]
\end{aligned}$$

Therefore the lemma is fulfilled in this case.

Case:  $t = t_1 t_2$

In this case

$$\begin{aligned}
\mathbf{Split}(t_1 t_2) &= \text{if } \mathbf{esp}(t_1) \neq \{\} \\
&\quad \text{then } [l\langle t_{11} \rangle(t_{12} t_2) \mid l\langle t_{11} \rangle t_{12} \in \mathbf{Split}(t_1)] @ \mathbf{Split}(t_2) \\
&\quad \text{else } [l\langle t_{11} \rangle(t_{12} t_2) \mid l\langle t_{11} \rangle t_{12} \in \mathbf{Split}(t_1)].
\end{aligned}$$

Since  $X \notin \mathbf{fv}'(t)$  Lemma 11.3 yields that  $\mathbf{esp}(t) = \mathbf{esp}(t[t_X/X])$ .

Therefore we know that  $\mathbf{esp}(t) = \{\}$  if and only if  $\mathbf{esp}(t[t_X/X]) = \{\}$ .

If  $\mathbf{esp}(t) = \{\}$  then

$\mathbf{Split}(t) = [l\langle t_{11} \rangle(t_{12} t_2) \mid l\langle t_{11} \rangle t_{12} \in \mathbf{Split}(t_1)]$  and

$\mathbf{Split}_\varphi(t[t_X/X]) = [l\langle t_{11} \rangle(t_{12}(t_2[t_X/X :: \varphi])) \mid l\langle t_{11} \rangle t_{12} \in \mathbf{Split}_\varphi(t_1[t_X/X])]$ .

In this case the induction hypothesis yields that  $\mathbf{Split}_\varphi(t_1[t_X/X]) = [t'[t_X/X :: \varphi] \mid t' \in \mathbf{Split}(t_1)]$ .

Therefore

$$\begin{aligned}
& \mathbf{Split}_\varphi(t_1 t_2[t_X/X]) \\
&= [l\langle t_{11} \rangle(t_{12}(t_2[t_X/X :: \varphi])) \mid l\langle t_{11} \rangle t_{12} \in \mathbf{Split}_\varphi(t_1[t_X/X])] \\
\text{(IH)} \quad &= [l\langle t_{11} \rangle(t_{12}(t_2[t_X/X :: \varphi])) \mid l\langle t_{11} \rangle t_{12} \in [t'[t_X/X :: \varphi] \mid t' \in \mathbf{Split}(t_1)]] \\
&= [l\langle t_{11} \rangle(t_2[t_X/X :: \varphi]) \mid l\langle t_{11} \rangle t_{12} \in \mathbf{Split}(t_1)] \\
&= [l\langle t_{11} \rangle(t_{12} t_2)[t_X/X :: \varphi] \mid l\langle t_{11} \rangle t_{12} \in \mathbf{Split}(t_1)] \\
&= [t'[t_X/X :: \varphi] \mid t' \in [l\langle t_{11} \rangle(t_{12} t_2) \mid l\langle t_{11} \rangle t_{12} \in \mathbf{Split}(t_1)]] \\
&= [t'[t_X/X :: \varphi] \mid t' \in \mathbf{Split}(t_1 t_2)].
\end{aligned}$$

Therefore the lemma is fulfilled in this case.

If  $\text{esp}(t) \neq \{\}$  then

$\text{Split}(t) = [l\langle t_{11} \rangle(t_{12}t_2) \mid l\langle t_{11} \rangle t_{12} \in \text{Split}(t_1)] @ \text{Split}(t_2)$  and

$\text{Split}_\varphi(t[t_X/X]) = [l\langle t_{11} \rangle(t_{12}(t_2[t_X/X :: \varphi])) \mid l\langle t_{11} \rangle t_{12} \in \text{Split}_\varphi(t_1[t_X/X])] @ \text{Split}_\varphi(t_2[t_X/X])$ .

We get in the same way as above, that

$$[l\langle t_{11} \rangle(t_{12}(t_2[t_X/X :: \varphi])) \mid l\langle t_{11} \rangle t_{12} \in \text{Split}_\varphi(t_1[t_X/X])] = [t'[t_X/X :: \varphi] \mid t' \in [l\langle t_{11} \rangle(t_{12}t_2) \mid l\langle t_{11} \rangle t_{12} \in \text{Split}(t_1)]]$$

The induction hypothesis yields that  $\text{Split}_\varphi(t_2[t_X/X]) = [t'[t_X/X :: \varphi] \mid t' \in \text{Split}(t_2)]$ .

Therefore we have that

$$\begin{aligned} & \text{Split}_\varphi(t_1t_2[t_X/X]) \\ &= [l\langle t_{11} \rangle(t_{12}(t_2[t_X/X :: \varphi])) \mid l\langle t_{11} \rangle t_{12} \in \text{Split}_\varphi(t_1[t_X/X])] @ \text{Split}_\varphi(t_2[t_X/X]) \\ &= [t'[t_X/X :: \varphi] \mid t' \in [l\langle t_{11} \rangle(t_{12}t_2) \mid l\langle t_{11} \rangle t_{12} \in \text{Split}(t_1)]] @ \text{Split}_\varphi(t_2[t_X/X]) \\ \text{(IH)} &= [t'[t_X/X :: \varphi] \mid t' \in [l\langle t_{11} \rangle(t_{12}t_2) \mid l\langle t_{11} \rangle t_{12} \in \text{Split}(t_1)]] @ [t'[t_X/X :: \varphi] \mid t' \in \text{Split}(t_2)] \\ &= [t'[t_X/X :: \varphi] \mid t' \in [l\langle t_{11} \rangle(t_{12}t_2) \mid l\langle t_{11} \rangle t_{12} \in \text{Split}(t_1)]] @ \text{Split}(t_2) \\ &= [t'[t_X/X :: \varphi] \mid t' \in \text{Split}(t_1t_2)]. \end{aligned}$$

Therefore the lemma is fulfilled in this case.

Case:  $t = t_1 + t_2$

In this case  $\text{Split}(t_1 + t_2) = \text{Split}(t_1) @ \text{Split}(t_2)$ .

Since  $\text{fv}'(t_1 + t_2) = \text{fv}'(t_1) \cup \text{fv}'(t_2)$  the induction hypothesis yields that

$\text{Split}_\varphi(t_1[t_X/X]) = \{t'[t_X/X :: \varphi] \mid t' \in \text{Split}(t_1)\}$  and

$\text{Split}_\varphi(t_2[t_X/X]) = \{t'[t_X/X :: \varphi] \mid t' \in \text{Split}(t_2)\}$ .

Therefore

$$\begin{aligned} & \text{Split}_\varphi(t_1 + t_2[t_X/X]) \\ &= \text{Split}_\varphi(t_1[t_X/X]) @ \text{Split}_\varphi(t_2[t_X/X]) \\ (2 \times \text{IH}) &= [t'[t_X/X :: \varphi] \mid t' \in \text{Split}(t_1)] @ [t'[t_X/X :: \varphi] \mid t' \in \text{Split}(t_2)] \\ &= [t'[t_X/X :: \varphi] \mid t' \in \text{Split}(t_1) @ \text{Split}(t_2)] \\ &= [t'[t_X/X :: \varphi] \mid t' \in \text{Split}(t_1 + t_2)]. \end{aligned}$$

Therefore the lemma is fulfilled in this case.

Case:  $t = \mu Y.t_1$

In this case  $\text{Split}(\mu Y.t_1) = \text{Split}_{[\mu Y.t_1/Y]}(t_1)$ .

If we choose  $Z \notin \text{fv}(t_1)$  then the induction hypothesis yields that

$\text{Split}_{[\mu Y.t_1/Y]}(t_1[Z/Z]) = [t'[Z/Z][\mu Y.t_1/Y] \mid t' \in \text{Split}(t_1)]$ , and therefore

$\text{Split}_{[\mu Y.t_1/Y]}(t_1) = [t'[\mu Y.t_1/Y] \mid t' \in \text{Split}(t_1)]$ .

We get that  $\mu Y.t_1[t_X/X] = \mu Z.(t_1[Z/Y][t_X/X])$  where  $Z \notin \text{fv}(t_X) \cup \{X\} \cup \text{fv}(\mu Y.t_1)$ .

Since we have assumed that  $X \notin \text{fv}'(\mu Y.t_1) = \text{fv}'(t_1) \setminus \{Y\}$ , we know that  $X \notin \text{fv}'(t_1[Z/Y])$ .

Therefore the induction hypothesis yields that

$\text{Split}_\varphi(t_1[Z/Y][t_X/X]) = [t'[t_X/X :: \varphi'] \mid t' \in \text{Split}(t_1[Z/Y])]$ .

Therefore

$$\begin{aligned}
& \mathbf{Split}_\varphi(\mu Y.t_1[t_X/X]) \\
&= \mathbf{Split}_\varphi(\mu Z.(t_1[Z/Y][t_X/X])) \\
&= \mathbf{Split}_{\mu Z.(t_1[Z/Y][t_X/X])/Z::\varphi}(t_1[Z/Y][t_X/X]) \\
(\mathbf{IH}) \quad &= [t'[t_X/X :: \mu Z.(t_1[Z/Y][t_X/X])/Z :: \varphi] \mid t' \in \mathbf{Split}(t_1[Z/Y])] \\
&= [t'[t_X/X :: \mu Y.t_1[t_X/X]/Z :: \varphi] \mid t' \in \mathbf{Split}(t_1[Z/Y])] \\
(\mathbf{Lemma 3.4}) \quad &= [t'[\mu Y.t_1/Z :: t_X/X :: \varphi] \mid t' \in \mathbf{Split}(t_1[Z/Y])] \\
&= [t'[\mu Y.t_1/Y :: t_X/X :: \varphi] \mid t' \in \mathbf{Split}(t_1)] \\
&= [t'[t_X/X :: \varphi] \mid t' \in [t''[\mu Y.t_1/Y] \mid t'' \in \mathbf{Split}(t_1)]] \\
(\mathbf{IH}) \quad &= [t'[t_X/X :: \varphi] \mid t' \in \mathbf{Split}_{[\mu Y.t_1/Y]}(t_1)] \\
&= [t'[t_X/X :: \varphi] \mid t' \in \mathbf{Split}(\mu Y.t_1)].
\end{aligned}$$

Therefore the lemma is fulfilled in this case.

Case:  $t = Y$

In this case  $\mathbf{Split}(Y) = []$

Since  $\mathbf{fv}'(Y) = \{Y\}$  and  $X \notin \mathbf{fv}'(t)$  we know that  $Y \neq X$ .

This means that  $\mathbf{Split}_\varphi(t[t_X/X]) = \mathbf{Split}_\varphi(Y[t_X/X]) = \mathbf{Split}_\varphi(Y) = []$  and therefore the lemma is fulfilled in this case.

We have now proved the lemma for all the cases, and can conclude that

$$X \notin \mathbf{fv}'(t) \Rightarrow \mathbf{Split}_\varphi(t[t_X/X]) = [t'[t_X/X :: \varphi] \mid t' \in \mathbf{Split}(t)].$$

■

### Proof 19.28 (Substitution Lemma for Deriv)

We will now prove Lemma 11.5 which states that

$$\begin{aligned}
& \forall t', X, t_X, \varphi. \mathbf{tr}_{\{X\}}(t') \wedge \mathbf{tr}_{\{X\}}(t_X) \Rightarrow \\
& \{t_j \mid l(t_1)t_2 \in \overline{\mathbf{Split}_\varphi(t'[\mu X.t_X/X])} \wedge j \in \{1, 2\}\} \\
& \subseteq \{t'_j[\mu X.t_X/X :: \varphi] \mid l(t'_1)t'_2 \in \overline{\mathbf{Split}(t')} \wedge j \in \{1, 2\}\} \cup \{t''[\mu X.t_X/X :: \varphi] \mid t'' \in \mathbf{Derivs}(t_X)\}
\end{aligned}$$

by induction on  $|t'|$ .

*Induction Hypothesis:*

If  $|t''| < |t'|$  and  $\mathbf{tr}_{\{X\}}(t'')$

then we can assume that

$$\begin{aligned}
& \{t_j \mid l(t_1)t_2 \in \overline{\mathbf{Split}_\varphi(t''[\mu X.t_X])} \wedge j \in \{1, 2\}\} \\
& \subseteq \{t'_j[\mu X.t_X/X :: \varphi'] \mid l(t'_1)t'_2 \in \overline{\mathbf{Split}(t'')} \wedge j \in \{1, 2\}\} \cup \{t'_1[\mu X.t_X/X :: \varphi'] \mid t'_1 \in \mathbf{Derivs}(t_X)\}.
\end{aligned}$$

We handle the each form of  $t'$  independently.

Case:  $t' = 0$

In this case  $\mathbf{Split}_\varphi(t'[\mu X.t_X/X]) = \mathbf{Split}_\varphi(0) = []$  and therefore the lemma is vacantly fulfilled in this case.

Case:  $t' = 1$

In this case  $\mathbf{Split}_\varphi(t'[\mu X.t_X/X]) = \mathbf{Split}_\varphi(1) = []$  and therefore the lemma is vacantly fulfilled in this case.

Case:  $t' = l\langle t'_1 \rangle$

In this case

$$\begin{aligned}
& \mathbf{Split}_\varphi(t'[\mu X.t_X/X]) \\
&= \mathbf{Split}_\varphi(l\langle t'_1 \rangle[\mu X.t_X/X]) \\
&= \mathbf{Split}_\varphi(l\langle t'_1 \rangle[\mu X.t_X/X]) \\
&= [l\langle t'_1 \rangle[\mu X.t_X/X :: \varphi]1] \\
&= [l\langle t'_1 \rangle[\mu X.t_X/X :: \varphi](1[\mu X.t_X/X :: \varphi])].
\end{aligned}$$

Therefore it is sufficient to show that  $1, t'_1 \in \{t'_j \mid l\langle t'_1 \rangle t'_2 \in \overline{\mathbf{Split}(t')} \wedge j \in \{1, 2\}\}$ .

Since  $\mathbf{Split}(t') = \mathbf{Split}(l\langle t'_1 \rangle) = [l\langle t'_1 \rangle 1]$  this is true.

Therefore the lemma is fulfilled in this case.

Case:  $t' = t'_1 t'_2$

In this case

$$\begin{aligned}
\mathbf{Split}(t'_1 t'_2) &= \text{if } \mathbf{esp}(t'_1) = \{\} \\
&\quad \text{then } [l\langle t'_{11} \rangle(t'_{12} t'_2) \mid l\langle t'_{11} \rangle t'_{12} \in \mathbf{Split}(t'_1)] \\
&\quad \text{else } [l\langle t'_{11} \rangle(t'_{12} t'_2) \mid l\langle t'_{11} \rangle t'_{12} \in \mathbf{Split}(t'_1)] @ \mathbf{Split}(t'_2)
\end{aligned}$$

and

$$\begin{aligned}
\mathbf{Split}_\varphi(t'_1 t'_2[\mu X.t_X/X]) &= \text{if } \mathbf{esp}(t'_1[\mu X.t_X/X]) = \{\} \\
&\quad \text{then } [l\langle t'_{11} \rangle(t'_{12}(t'_2[\mu X.t_X/X :: \varphi])) \mid l\langle t'_{11} \rangle t'_{12} \in \mathbf{Split}_\varphi(t'_1[\mu X.t_X/X])] \\
&\quad \text{else } [l\langle t'_{11} \rangle(t'_{12}(t'_2[\mu X.t_X/X :: \varphi])) \mid l\langle t'_{11} \rangle t'_{12} \in \mathbf{Split}_\varphi(t'_1[\mu X.t_X/X])] @ \\
&\quad \mathbf{Split}_\varphi(t'_2[\mu X.t_X/X]).
\end{aligned}$$

Since we know that  $\mathbf{tr}_{\{X\}}(t'_1 t'_2)$ , we know that  $X \notin \mathbf{fv}'(t'_1)$ .

If  $\mathbf{esp}(t'_1) = \{\}$  then Lemma 11.3 yields that  $\mathbf{esp}(t'_1[\mu X.t_X/X]) = \mathbf{esp}(t'_1) = \{\}$ .

This means that

$$\begin{aligned}
& \mathbf{Split}_\varphi(t'_1 t'_2[\mu X.t_X/X]) \\
&= [l\langle t'_{11} \rangle(t'_{12}(t'_2[\mu X.t_X/X :: \varphi])) \mid l\langle t'_{11} \rangle t'_{12} \in \mathbf{Split}_\varphi(t'_1[\mu X.t_X/X])] \\
(\mathbf{Lemma 11.4}) &= [l\langle t'_{11} \rangle(t'_{12}(t'_2[\mu X.t_X/X :: \varphi])) \mid l\langle t'_{11} \rangle t'_{12} \in [t''_1[\mu X.t_X/X :: \varphi] \mid t''_1 \in \mathbf{Split}(t'_1)]] \\
&= [l\langle t'_{11} \rangle[\mu X.t_X/X :: \varphi](t'_{12}[\mu X.t_X/X :: \varphi](t'_2[\mu X.t_X/X :: \varphi])) \mid l\langle t'_{11} \rangle t'_{12} \in \mathbf{Split}(t'_1)] \\
&= [l\langle t'_{11} \rangle(t'_{12} t'_2)[\mu X.t_X/X :: \varphi] \mid l\langle t'_{11} \rangle t'_{12} \in \mathbf{Split}(t'_1)] \\
&= [t''[\mu X.t_X/X :: \varphi] \mid t'' \in \mathbf{Split}(t'_1 t'_2)].
\end{aligned}$$

This means that

$$\begin{aligned}
& \{t_j \mid l\langle t_1 \rangle t_2 \in \overline{\text{Split}_\varphi(t'_1 t'_2 [\mu X.t_X/X])} \wedge j \in \{1, 2\}\} \\
&= \{t_j \mid l\langle t_1 \rangle t_2 \in \{\overline{l''\langle t'_1 \rangle [\mu X.t_X/X :: \varphi]} \mid l''\langle t'_1 \rangle t'_2 \in \overline{\text{Split}(t'_1 t'_2)}\} \wedge j \in \{1, 2\}\} \\
&= \{t'_j [\mu X.t_X/X :: \varphi] \mid l''\langle t'_1 \rangle t'_2 \in \overline{\text{Split}(t'_1 t'_2)} \wedge j \in \{1, 2\}\}.
\end{aligned}$$

Therefore the lemma is fulfilled in this case.

If  $\text{esp}(t'_1) \neq \{\}$  then Lemma 11.3 yields that  $\text{esp}(t'_1 [\mu X.t_X/X]) = \text{esp}(t'_1) \neq \{\}$ .

This means that

$$\begin{aligned}
& \text{Split}_\varphi(t'_1 t'_2 [\mu X.t_X/X]) \\
&= [l\langle t'_{11} \rangle (t'_{12} (t'_2 [\mu X.t_X/X :: \varphi])) \mid l\langle t'_{11} \rangle t'_{12} \in \text{Split}_\varphi(t'_1 [\mu X.t_X/X])]@ \\
& \quad \text{Split}_\varphi(t'_2 [\mu X.t_X/X]) \\
\text{(Lemma 11.4)} &= [l\langle t'_{11} \rangle (t'_{12} (t'_2 [\mu X.t_X/X :: \varphi])) \mid l\langle t'_{11} \rangle t'_{12} \in [t'_1 [\mu X.t_X/X :: \varphi] \mid t'_1 \in \text{Split}(t'_1)]]@ \\
& \quad \text{Split}_\varphi(t'_2 [\mu X.t_X/X]) \\
&= [l\langle t'_{11} \rangle (t'_{12} t'_2) [\mu X.t_X/X :: \varphi] \mid l\langle t'_{11} \rangle t'_{12} \in \text{Split}(t'_1)]@ \text{Split}_\varphi(t'_2 [\mu X.t_X/X])
\end{aligned}$$

This means that

$$\begin{aligned}
& \{t_j \mid l\langle t_1 \rangle t_2 \in \overline{\text{Split}_\varphi(t'_1 t'_2 [\mu X.t_X/X])} \wedge j \in \{1, 2\}\} \\
&= \{t'_j [\mu X.t_X/X :: \varphi] \mid l'\langle t'_1 \rangle t'_2 \in \overline{[l\langle t'_{11} \rangle (t'_{12} t'_2) \mid l\langle t'_{11} \rangle t'_{12} \in \text{Split}(t'_1)]} \wedge j \in \{1, 2\}\} \cup \\
& \quad \{t_j \mid l\langle t_1 \rangle t_2 \in \overline{\text{Split}_\varphi(t'_2 [\mu X.t_X/X])} \wedge j \in \{1, 2\}\} \\
\text{(IH)} &= \{t'_j [\mu X.t_X/X :: \varphi] \mid l'\langle t'_1 \rangle t'_2 \in \overline{[l\langle t'_{11} \rangle (t'_{12} t'_2) \mid l\langle t'_{11} \rangle t'_{12} \in \text{Split}(t'_1)]} \wedge j \in \{1, 2\}\} \cup \\
& \quad \{t'_j [\mu X.t_X/X :: \varphi] \mid l'\langle t'_1 \rangle t'_2 \in \overline{\text{Split}(t'_2)} \wedge j \in \{1, 2\}\} \cup \\
& \quad \{t'' [\mu X.t_X/X :: \varphi] \mid t'' \in \text{Derivs}(t_X)\} \\
&= \{t'_j [\mu X.t_X/X :: \varphi] \mid l'\langle t'_1 \rangle t'_2 \in \overline{[l\langle t'_{11} \rangle (t'_{12} t'_2) \mid l\langle t'_{11} \rangle t'_{12} \in \text{Split}(t'_1)]} \cup \overline{\text{Split}(t'_2)} \wedge j \in \{1, 2\}\} \cup \\
& \quad \{t'' [\mu X.t_X/X :: \varphi] \mid t'' \in \text{Derivs}(t_X)\} \\
&= \{t'_j [\mu X.t_X/X :: \varphi] \mid l'\langle t'_1 \rangle t'_2 \in \overline{[l\langle t'_{11} \rangle (t'_{12} t'_2) \mid l\langle t'_{11} \rangle t'_{12} \in \text{Split}(t'_1)]@ \text{Split}(t'_2)} \wedge j \in \{1, 2\}\} \cup \\
& \quad \{t'' [\mu X.t_X/X :: \varphi] \mid t'' \in \text{Derivs}(t_X)\} \\
&= \{t'_j [\mu X.t_X/X :: \varphi] \mid l'\langle t'_1 \rangle t'_2 \in \overline{\text{Split}(t'_1 t'_2)} \wedge j \in \{1, 2\}\} \cup \\
& \quad \{t'' [\mu X.t_X/X :: \varphi] \mid t'' \in \text{Derivs}(t_X)\}.
\end{aligned}$$

Therefore the lemma is fulfilled in this case.

Case:  $t' = t'_1 + t'_2$

In this case

$$\begin{aligned}
& \text{Split}_\varphi(t' [\mu X.t_X/X]) \\
&= \text{Split}_\varphi(t'_1 + t'_2 [\mu X.t_X/X]) \\
&= \text{Split}_\varphi(t'_1 [\mu X.t_X/X])@ \text{Split}_\varphi(t'_2 [\mu X.t_X/X]).
\end{aligned}$$

Therefore the induction hypothesis yields that

$$\begin{aligned} & \{u_j \mid l\langle u_1 \rangle u_2 \in \overline{\mathbf{Split}_\varphi(t'_1[\mu X.t_X/X])} \wedge j \in \{1, 2\}\} \\ \subseteq & \{u'_j[\mu X.t_X/X :: \varphi] \mid l\langle u'_1 \rangle u'_2 \in \overline{\mathbf{Split}(t'_1)} \wedge j \in \{1, 2\}\} \cup \{u'[\mu X.t_X/X :: \varphi] \mid u' \in \mathbf{Derivs}(t_X)\} \end{aligned}$$

and

$$\begin{aligned} & \{u_j \mid l\langle u_1 \rangle u_2 \in \overline{\mathbf{Split}_\varphi(t'_2[\mu X.t_X/X])} \wedge j \in \{1, 2\}\} \\ \subseteq & \{u'_j[\mu X.t_X/X :: \varphi] \mid l\langle u'_1 \rangle u'_2 \in \overline{\mathbf{Split}(t'_2)} \wedge j \in \{1, 2\}\} \cup \{u'[\mu X.t_X/X :: \varphi] \mid u' \in \mathbf{Derivs}(t_X)\}. \end{aligned}$$

Since  $\mathbf{Split}(t'_1 + t'_2) = \mathbf{Split}(t'_1) @ \mathbf{Split}(t'_2)$  we get that

$$\begin{aligned} & \{u_j \mid l\langle u_1 \rangle u_2 \in \overline{\mathbf{Split}_\varphi(t'_1 + t'_2[\mu X.t_X/X])} \wedge j \in \{1, 2\}\} \\ \subseteq & \{u'_j[\mu X.t_X/X :: \varphi] \mid l\langle u'_1 \rangle u'_2 \in \overline{\mathbf{Split}(t'_1 + t'_2)} \wedge j \in \{1, 2\}\} \cup \\ & \{u'[\mu X.t_X/X :: \varphi] \mid u' \in \mathbf{Derivs}(t_X)\}. \end{aligned}$$

Therefore the lemma is fulfilled in this case.

Case:  $t' = \mu Y.t'_1$

If we chose  $Z \notin \mathbf{fv}(t_1) \cup \mathbf{fv}(t_X) \cup \{X\}$  then

$$\begin{aligned} & \mathbf{Split}(\mu Y.t_1) \\ = & \mathbf{Split}_{[\mu Y.t_1/Y]}(t_1) \\ = & \mathbf{Split}_{[\mu Y.t_1/Y]}(t_1[Z/Z]) \\ \text{(Lemma 11.4)} = & [t''[Z/Z :: \mu Y.t_1/Y] \mid t'' \in \mathbf{Split}(t_1)] \\ = & [t''[\mu Y.t_1/Y] \mid t'' \in \mathbf{Split}(t_1)] \end{aligned}$$

Therefore we get that

$$\begin{aligned}
& \{u_j \mid l\langle u_1 \rangle u_2 \in \overline{\text{Split}_\varphi(t'[\mu X.t_X/X])} \wedge j \in \{1, 2\}\} \\
&= \{u_j \mid l\langle u_1 \rangle u_2 \in \overline{\text{Split}_\varphi(\mu Y.t_1[\mu X.t_X/X])} \wedge j \in \{1, 2\}\} \\
&= \{u_j \mid l\langle u_1 \rangle u_2 \in \overline{\text{Split}_\varphi(\mu Z.(t_1[Z/Y][\mu X.t_X/X])} \wedge j \in \{1, 2\}\} \\
&= \{u_j \mid l\langle u_1 \rangle u_2 \in \overline{\text{Split}_{\mu Z.(t_1[Z/Y][\mu X.t_X/X])/Z::\varphi}(t_1[Z/Y][\mu X.t_X/X])} \wedge j \in \{1, 2\}\} \\
\text{(IH)} &= \{u'_j[\mu X.t_X/X :: \mu Z.(t_1[Z/Y][\mu X.t_X/X])/Z :: \varphi] \mid l'\langle u'_1 \rangle u'_2 \in \overline{\text{Split}(t_1[Z/Y])} \wedge j \in \{1, 2\}\} \cup \\
&\quad \{u'[\mu X.t_X/X :: \mu Z.(t_1[Z/Y][\mu X.t_X/X])/Z :: \varphi] \mid u' \in \text{Derivs}(t_X)\} \\
&= \{u'_j[\mu X.t_X/X :: \mu Y.t_1[\mu X.t_X/X]/Z :: \varphi] \mid l'\langle u'_1 \rangle u'_2 \in \overline{\text{Split}(t_1[Z/Y])} \wedge j \in \{1, 2\}\} \cup \\
&\quad \{u'[\mu X.t_X/X :: \mu Y.t_1[\mu X.t_X/X])/Z :: \varphi] \mid u' \in \text{Derivs}(t_X)\} \\
\text{(Lemma 3.4)} &= \{u'_j[\mu Y.t_1/Z :: \mu X.t_X/X :: \varphi] \mid l'\langle u'_1 \rangle u'_2 \in \overline{\text{Split}(t_1[Z/Y])} \wedge j \in \{1, 2\}\} \cup \\
&\quad \{u'[\mu Y.t_1/Z :: \mu X.t_X/X :: \varphi] \mid u' \in \text{Derivs}(t_X)\} \\
&= \{u'_j[\mu Y.t_1/Y :: \mu X.t_X/X :: \varphi] \mid l'\langle u'_1 \rangle u'_2 \in \overline{\text{Split}(t_1)} \wedge j \in \{1, 2\}\} \cup \\
&\quad \{u'[\mu Y.t_1/Z :: \mu X.t_X/X :: \varphi] \mid u' \in \text{Derivs}(t_X)\} \\
&= \{u'_j[\mu X.t_X/X :: \varphi] \mid l'\langle u'_1 \rangle u'_2 \in \overline{\text{Split}(\mu Y.t_1)} \wedge j \in \{1, 2\}\} \cup \\
&\quad \{u'[\mu Y.t_1/Z :: \mu X.t_X/X :: \varphi] \mid u' \in \text{Derivs}(t_X)\} \\
&= \{u'_j[\mu X.t_X/X :: \varphi] \mid l'\langle u'_1 \rangle u'_2 \in \overline{\text{Split}(\mu Y.t_1)} \wedge j \in \{1, 2\}\} \cup \\
&\quad \{u'[\mu X.t_X/X :: \varphi] \mid u' \in \text{Derivs}(t_X)\}.
\end{aligned}$$

Where the last equality is fulfilled because  $Z \notin \text{fv}(t_X)$  and therefore not in  $u' \in \text{Derivs}(t_X)$ .

Therefore the lemma is fulfilled in this case.

Case:  $t' = Y$

If  $X = Y$  then

$$\begin{aligned}
& \text{Split}_\varphi(t'[\mu X.t_X/X]) \\
&= \text{Split}_\varphi(\mu X.t_X) \\
&= \text{Split}_{\mu X.t_X/X::\varphi}(t_X) \\
&= \text{Split}_{\mu X.t_X/X::\varphi}(t_X[Z/Z]) \quad \text{where } Z \notin \text{fv}(t_X) \\
&= [t''[Z/Z :: \mu X.t_X/X :: \varphi] \mid t'' \in \text{Split}(t_X)] \\
&= [t''[\mu X.t_X/X :: \varphi] \mid t'' \in \text{Split}(t_X)].
\end{aligned}$$

Therefore we have that

$$\begin{aligned}
& \{u_j \mid l\langle u_1 \rangle u_2 \in \overline{\text{Split}_\varphi(t'[\mu X.t_X/X])} \wedge j \in \{1, 2\}\} \\
&= \{u_j \mid l\langle u_1 \rangle u_2 \in \overline{[t''[\mu X.t_X/X :: \varphi] \mid t'' \in \text{Split}(t_X)]} \wedge j \in \{1, 2\}\} \\
&= \{u'_j[\mu X.t_X/X :: \varphi] \mid l\langle u_1 \rangle u_2 \in \text{Split}(t_X) \wedge j \in \{1, 2\}\} \\
&\subseteq \{u'[\mu X.t_X/X] \mid u' \in \text{Derivs}(t_X)\}.
\end{aligned}$$

Therefore the lemma is fulfilled in this case.

If  $X \neq Y$  then  $\text{Split}_\varphi(t'[\mu X.t_X/X]) = \text{Split}_\varphi(Y[\mu X.t_X/X]) = \text{Split}_\varphi(Y) = []$  and therefore the lemma is vacantly fulfilled in this case.

We have now proved the lemma for all the cases and we can therefore conclude that

$$\begin{aligned} & \{t_j \mid l\langle t_1 \rangle t_2 \in \overline{\text{Split}_\varphi(t'[\mu X.t_X])} \wedge j \in \{1, 2\}\} \\ \subseteq & \{t'_j[\mu X.t_X/X :: \varphi] \mid l\langle t'_1 \rangle t'_2 \in \overline{\text{Split}(t')} \wedge j \in \{1, 2\}\} \cup \{t'_1[\mu X.t_X/X :: \varphi] \mid t'_1 \in \text{Derivs}(t_X)\}. \end{aligned}$$

■

**Proof 19.29 (Deriv boundary for  $\mu X.t$ )**

We will now prove Lemma 11.6 which states that

$$\begin{aligned} \forall \mu X.t_X, S & \subseteq \{t'[\mu X.t_X/X] \mid t' \in \text{Derivs}(t_X)\} \cup \{\mu X.t_X\}. \\ \text{Deriv}(S) & \subseteq \{t'[\mu X.t_X/X] \mid t' \in \text{Derivs}(t_X)\}. \end{aligned}$$

This follows from the following inclusions.

$$\begin{aligned} \text{Deriv}(S) & = \bigcup_{t \in S} \{t_j \mid l\langle t_1 \rangle t_2 \in \overline{\text{Split}(t)} \wedge j \in \{1, 2\}\} \\ & \subseteq \bigcup_{t \in \{t'[\mu X.t_X/X] \mid t' \in \text{Derivs}(t_X)\}} \{t_j \mid l\langle t_1 \rangle t_2 \in \overline{\text{Split}(t)} \wedge j \in \{1, 2\}\} \\ & \quad \cup \{t_j \mid l\langle t_1 \rangle t_2 \in \overline{\text{Split}(\mu X.t_X)} \wedge j \in \{1, 2\}\} \\ & = \bigcup_{t' \in \text{Derivs}(t_X)} \{t_j \mid l\langle t_1 \rangle t_2 \in \overline{\text{Split}(t'[\mu X.t_X/X])} \wedge j \in \{1, 2\}\} \\ & \quad \cup \{t_j \mid l\langle t_1 \rangle t_2 \in \overline{\text{Split}_{[\mu X.t_X/X]}(t_X)} \wedge j \in \{1, 2\}\} \\ \text{(Lemma 11.4)} & = \bigcup_{t' \in \text{Derivs}(t_X)} \{t_j \mid l\langle t_1 \rangle t_2 \in \overline{\text{Split}(t'[\mu X.t_X/X])} \wedge j \in \{1, 2\}\} \\ & \quad \cup \{t_j[\mu X.t_X/X] \mid l\langle t_1 \rangle t_2 \in \overline{\text{Split}(t_X)} \wedge j \in \{1, 2\}\} \\ \text{(Lemma 11.5)} & \subseteq \bigcup_{t' \in \text{Derivs}(t_X)} (\{t_j[\mu X.t_X/X] \mid l\langle t_1 \rangle t_2 \in \overline{\text{Split}(t')} \wedge j \in \{1, 2\}\} \\ & \quad \cup \{t''[\mu X.t_X/X] \mid t'' \in \text{Derivs}(t_X)\}) \\ & \quad \cup \{t_j[\mu X.t_X/X] \mid l\langle t_1 \rangle t_2 \in \overline{\text{Split}(t_X)} \wedge j \in \{1, 2\}\} \\ (t' \in \text{Derivs}(t_X)) & = \bigcup_{t'' \in \text{Derivs}(t_X)} \{t''[\mu X.t_X/X] \mid t'' \in \text{Derivs}(t_X)\} \\ & \quad \cup \{t_j[\mu X.t_X/X] \mid l\langle t_1 \rangle t_2 \in \overline{\text{Split}(t_X)} \wedge j \in \{1, 2\}\} \\ & \subseteq \{t''[\mu X.t_X/X] \mid t'' \in \text{Derivs}(t_X)\} \\ & \quad \cup \{t_j[\mu X.t_X/X] \mid l\langle t_1 \rangle t_2 \in \overline{\text{Split}(t_X)} \wedge j \in \{1, 2\}\} \\ (t_X \in \text{Derivs}(t_X)) & \subseteq \{t''[\mu X.t_X/X] \mid t'' \in \text{Derivs}(t_X)\} \cup \{t''[\mu X.t_X/X] \mid t'' \in \text{Derivs}(t_X)\} \\ & = \{t''[\mu X.t_X/X] \mid t'' \in \text{Derivs}(t_X)\}. \end{aligned}$$

■

**Proof 19.30** (Deriv boundary for  $t_1t_2$ )

We will now prove Lemma 11.8 which states that

$$\begin{aligned} \forall t_1, t_2 \in \mathbf{XMLType}. \forall S &\subseteq \{t't_2 \mid t' \in \mathit{Derivs}(t_1)\} \cup \mathit{Derivs}(t_1) \cup \mathit{Derivs}(t_2) \cup \{t_1t_2\}. \\ \mathit{Deriv}(S) &\subseteq \{t't_2 \mid t' \in \mathit{Derivs}(t_1)\} \cup \mathit{Derivs}(t_1) \cup \mathit{Derivs}(t_2) \cup \{t_1t_2\}. \end{aligned}$$

Since  $\mathit{Deriv}(S) = \bigcup_{t' \in S} \{t_j \mid l\langle t_1 \rangle t_2 \in \mathit{Split}(t') \wedge j \in \{1, 2\}\}$ , we let  $t'' \in S \subseteq \{t't_2 \mid t' \in \mathit{Derivs}(t_1)\} \cup \mathit{Derivs}(t_1) \cup \mathit{Derivs}(t_2) \cup \{t_1t_2\}$  be chosen.

Assume that  $t'' \in \{t't_2 \mid t' \in \mathit{Derivs}(t_1)\}$ , so  $t'' = t't_2$  where  $t' \in \mathit{Derivs}(t_1)$ .

Then  $\overline{\mathit{Split}(t'')} = \overline{\mathit{Split}(t't_2)} \subseteq \{l\langle t_{11} \rangle (t_{12}t_2) \mid l\langle t_{11} \rangle t_{12} \in \overline{\mathit{Split}(t_1)}\} \cup \overline{\mathit{Split}(t_2)}$ .

Therefore if  $l\langle t'_1 \rangle t'_2 \in \overline{\mathit{Split}(t'')}$  then  $t'_1 \in \mathit{Derivs}(t_1) \cup \mathit{Derivs}(t_2)$  and

$t'_2 \in \{t_{12}t_2 \mid t_{12} \in \overline{\mathit{Derivs}(t_1)}\} \cup \mathit{Derivs}(t_2)$  so we can conclude that

$$\{t_j \mid l\langle t_1 \rangle t_2 \in \overline{\mathit{Split}(t)} \wedge j \in \{1, 2\}\} \subseteq \{t't_2 \mid t' \in \mathit{Derivs}(t_1)\} \cup \mathit{Derivs}(t_1) \cup \mathit{Derivs}(t_2) \cup \{t_1t_2\}.$$

Assume that  $t'' \in \mathit{Derivs}(t_j) = \bigcup_{i=0}^{\infty} \mathit{Deriv}^i(\{t_j\})$  where  $j \in \{1, 2\}$ .

This means that  $\exists i \in \mathbb{N}. t'' \in \mathit{Deriv}^i(\{t_j\})$ , so we can conclude that

$$\begin{aligned} &\{t'_k \mid l\langle t'_1 \rangle t'_2 \in \overline{\mathit{Split}(t'')} \wedge k \in \{1, 2\}\} \\ &\subseteq \bigcup_{t' \in \mathit{Deriv}^i(\{t_j\})} \{t'_k \mid l\langle t'_1 \rangle t'_2 \in \overline{\mathit{Split}(t')} \wedge k \in \{1, 2\}\} \\ &= \mathit{Deriv}(\mathit{Deriv}^i(\{t_j\})) \\ &= \mathit{Deriv}^{i+1}(\{t_j\}) \\ &\subseteq \mathit{Derivs}(t_j) \\ &\subseteq \{t't_2 \mid t' \in \mathit{Derivs}(t_1)\} \cup \mathit{Derivs}(t_1) \cup \mathit{Derivs}(t_2) \cup \{t_1t_2\}. \end{aligned}$$

Finally assume that  $t'' = t_1t_2$ .

There are two cases for  $\mathit{Split}(t_1t_2)$ , but both satisfy the following.

$\overline{\mathit{Split}(t'')} = \overline{\mathit{Split}(t_1t_2)} \subseteq \{l\langle t_{11} \rangle (t_{12}t_2) \mid l\langle t_{11} \rangle t_{12} \in \overline{\mathit{Split}(t_1)}\} \cup \overline{\mathit{Split}(t_2)}$ , so if

$l\langle t'_1 \rangle t'_2 \in \overline{\mathit{Split}(t'')}$  then  $t'_1 \in \mathit{Derivs}(t_1) \cup \mathit{Derivs}(t_2)$  and

$t'_2 \in \{t't_2 \mid t' \in \mathit{Derivs}(t_1)\} \cup \mathit{Derivs}(t_2)$  so we can conclude that

$$\{t_j \mid l\langle t_1 \rangle t_2 \in \overline{\mathit{Split}(t)} \wedge j \in \{1, 2\}\} \subseteq \{t't_2 \mid t' \in \mathit{Derivs}(t_1)\} \cup \mathit{Derivs}(t_1) \cup \mathit{Derivs}(t_2) \cup \{t_1t_2\}.$$

We can now conclude that

$$\begin{aligned} \mathit{Deriv}(S) &= \bigcup_{t'' \in S} \{t_j \mid l\langle t_1 \rangle t_2 \in \overline{\mathit{Split}(t'')} \wedge j \in \{1, 2\}\} \\ &\subseteq \bigcup_{t'' \in \{t't_2 \mid t' \in \mathit{Derivs}(t_1)\} \cup \mathit{Derivs}(t_1) \cup \mathit{Derivs}(t_2) \cup \{t_1t_2\}} \{t_j \mid l\langle t_1 \rangle t_2 \in \overline{\mathit{Split}(t'')} \wedge j \in \{1, 2\}\} \\ &\subseteq \{t't_2 \mid t' \in \mathit{Derivs}(t_1)\} \cup \mathit{Derivs}(t_1) \cup \mathit{Derivs}(t_2) \cup \{t_1t_2\} \end{aligned}$$

■

**Proof 19.31 (Soundness of simulations)**

We will now prove Theorem 12.3 which states for all simulations  $\mathcal{R}$  that

$$T_1 \mathcal{R} T_2 \Rightarrow \models T_1 <: T_2.$$

We assume that  $\mathcal{R}, T_1, T_2$  are chosen such that  $T_1 \mathcal{R} T_2$ . Assume that  $x \in \text{IN}[[T_1]]$ .

We will now prove that  $x \in \text{IN}[[T_2]]$  by structural induction on  $x$ .

*Induction Hypothesis:*

If  $x'$  is a sub-term of  $x$ ,  $x' \in \text{IN}[[T'_1]]$  and  $T'_1 \mathcal{R} T'_2$  then we can assume that  $x' \in \text{IN}[[T'_2]]$ .

*Case:  $x = \varepsilon$*

In this case  $\varepsilon \in \text{IN}[[T_1]]$ . Corollary 7.10 yields that  $\text{ESP}(T_1) \neq \{\}$ . Since  $T_1 \mathcal{R} T_2$  the definition of simulations yield that  $\text{ESP}(T_2) \neq \{\}$ , and now Corollary 7.10 yields that  $x = \varepsilon \in \text{IN}[[T_2]]$ , which concludes this case.

*Case:  $x = \langle l \rangle x_1 \langle /l \rangle x_2$*

Since  $\langle l \rangle x_1 \langle /l \rangle x_2 \in \text{IN}[[T_1]]$ , then Corollary 10.23 yields that  $\langle l \rangle x_1 \langle /l \rangle x_2 \in \text{IN}[[\text{SPLIT}(T_1)]]$  and Lemma 4.9 yields that  $\exists l \langle t_{11} \rangle t_{12}. \langle l \rangle x_1 \langle /l \rangle x_2 \in \text{in}[[l \langle t_{11} \rangle t_{12}]]$ . The definition of  $\text{in}[[\cdot]]$  now yields that  $x_1 \in \text{in}[[t_{11}]]$  and  $x_2 \in \text{in}[[t_{12}]]$ .

We now define

$$S = \{l' \langle t_{21} \rangle t_{22} \in \overline{\text{SPLIT}(T_2)} \mid x_2 \in \text{in}[[t_{22}]]\}.$$

Since  $\mathcal{R}$  is a simulation, one of the following properties must be fulfilled

$$\begin{aligned} \exists T_{21}. [t_{11}] \mathcal{R} T_{21} \wedge \overline{T_{21}} &= \{t_{21} \mid l \langle t_{21} \rangle t_{22} \in S\} \\ \exists T_{22}. [t_{12}] \mathcal{R} T_{22} \wedge \overline{T_{22}} &= \{t_{22} \mid l \langle t_{21} \rangle t_{22} \in \overline{\text{SPLIT}(T_2)} \setminus S\} \end{aligned}$$

Assume that the second property is fulfilled. Then  $[t_{12}] \mathcal{R} T_{22}$ .

Since  $x_2 \in \text{in}[[t_{12}]] = \text{IN}[[[t_{12}]]]$  the induction hypothesis yields that  $x_2 \in \text{IN}[[T_{22}]]$ .

Lemma 4.9 yields that  $x_2 \in \bigcup_{t_{22} \in \overline{T_{22}}} \text{in}[[t_{22}]] = \bigcup_{l' \langle t_{21} \rangle t_{22} \in \overline{\text{SPLIT}(T_2)} \setminus S} \text{in}[[t_{22}]]$ .

This means that there must be  $l' \langle t_{21} \rangle t_{22} \in \overline{\text{SPLIT}(T_2)} \setminus S. x_2 \in \text{in}[[t_{22}]]$ .

This is a contradiction, since we defined  $S = \{l' \langle t_{21} \rangle t_{22} \in \overline{\text{SPLIT}(T_2)} \mid x_2 \in \text{in}[[t_{22}]]\}$ .

We can therefore conclude that the second property can not be fulfilled. Therefore the first property must be fulfilled.

This means that  $[t_{12}] \mathcal{R} T_{21}$ .

Since  $x_1 \in \text{in}[[t_{11}]] = \text{IN}[[[t_{11}]]]$  the induction hypothesis yields that  $x_1 \in \text{IN}[[T_{21}]]$ .

Now Lemma 4.9 yields that  $x_1 \in \bigcup_{t_{21} \in \overline{T_{21}}} \text{in}[[t_{21}]] = \bigcup_{l \langle t_{21} \rangle t_{22} \in S} \text{in}[[t_{21}]]$ .

This means that there must be  $l \langle t_{21} \rangle t_{22} \in S$  such that  $x_1 \in \text{in}[[t_{21}]]$ .

Because of the way we have defined  $S$  we know that  $x_2 \in \text{in}[[t_{22}]]$  and now the definition of  $\text{in}[[\cdot]]$  yields that  $\langle l \rangle x_1 \langle /l \rangle x_2 \in \text{in}[[l \langle t_{21} \rangle t_{22}]] \subseteq \bigcup_{t_2 \in \overline{\text{SPLIT}(T_2)}} \text{in}[[t_2]]$ .

Now Lemma 4.9 yields that  $\langle l \rangle x_1 \langle /l \rangle x_2 \in \text{IN}[[\text{SPLIT}(T_2)]]$  and Corollary 10.23 yields that  $\langle l \rangle x_1 \langle /l \rangle x_2 \in \text{IN}[[T_2]]$ , which concludes the induction step.

We can therefore conclude that  $\forall x \in \text{IN}[[T_1]]. x \in \text{IN}[[T_2]]$ , which means that  $\text{IN}[[T_1]] \subseteq \text{IN}[[T_2]]$ , and we can therefore conclude that  $\models T_1 <: T_2$ . ■

**Proof 19.32 (Completeness of simulations)**

We will now prove Lemma 12.6 which states that

$\mathcal{R}_{<}$  is a simulation.

We will prove that the the properties of a simulation is fulfilled by  $\mathcal{R}_{<}$ .

$T_1 \mathcal{R}_{<} T_2 \Rightarrow ESP(T_1) \neq \{\} \Rightarrow ESP(T_2) \neq \{\}$ :

If  $ESP(T_1) \neq \{\}$  then Corollary 7.10 yields that  $\varepsilon \in IN[[T_1]]$ . Since  $T_1 \mathcal{R}_{<} T_2$  then  $IN[[T_1]] \subseteq IN[[T_2]]$  and therefore  $\varepsilon \in IN[[T_2]]$ . Now Corollary 7.10 yields that  $ESP(T_2) \neq \{\}$ , so this property is fulfilled.

$T_1 \mathcal{R}_{<} T_2 \Rightarrow \forall l \langle t_{11} \rangle t_{12} \in \overline{SPLIT(T_1)}. \forall S \subseteq \overline{SPLIT(T_2)}$ .

$(\exists T_{21}. [t_{11}] \mathcal{R}_{<} T_{21} \wedge \overline{T_{21}} = \{t_{21} \mid l \langle t_{21} \rangle t_{22} \in S\}) \vee$

$(\exists T_{22}. [t_{12}] \mathcal{R}_{<} T_{22} \wedge \overline{T_{22}} = \{t_{22} \mid l \langle t_{21} \rangle t_{22} \in \overline{SPLIT(T_2)} \setminus S\})$ :

Assume there is a  $T_1, T_2, l \langle t_{11} \rangle t_{12} \in \overline{SPLIT(T_1)}, S \subseteq \overline{SPLIT(T_2)}$  such that the above is not fulfilled, that is

$(\forall T_{21}. [t_{11}] \mathcal{R}_{<} T_{21} \vee \overline{T_{21}} \neq \{t_{21} \mid l \langle t_{21} \rangle t_{22} \in S\}) \wedge$

$(\forall T_{22}. [t_{12}] \mathcal{R}_{<} T_{22} \vee \overline{T_{22}} \neq \{t_{22} \mid l \langle t_{21} \rangle t_{22} \in \overline{SPLIT(T_2)} \setminus S\})$ .

Now select  $T_{21}, T_{22}$  such that

$\overline{T_{21}} = \{t_{21} \mid l \langle t_{21} \rangle t_{22} \in S\}$  and  $\overline{T_{22}} = \{t_{22} \mid l \langle t_{21} \rangle t_{22} \in \overline{SPLIT(T_2)} \setminus S\}$ .

We can obtain  $T_{21}$  and  $T_{22}$  from  $[t_{21} \mid l \langle t_{21} \rangle t_{22} \in \overline{SPLIT(T_2)}]$  and  $[t_{22} \mid l \langle t_{21} \rangle t_{22} \in \overline{SPLIT(T_2)}]$  by removing the exceeding elements.

Since  $\overline{T_{21}} = \{t_{21} \mid l \langle t_{21} \rangle t_{22} \in S\}$  the assumed  $T_1, T_2, l \langle t_{11} \rangle t_{12}$  and  $S$  now yields that  $[t_{11}] \mathcal{R}_{<} T_{21}$ , which means that  $in[[t_{11}]] = IN[[[t_{11}]]] \not\subseteq IN[[T_{21}]]$ .

This means that there is an  $x_1 \in in[[t_{11}]]$  such that  $x_1 \notin IN[[T_{21}]]$ .

Since  $\overline{T_{22}} = \{t_{22} \mid l \langle t_{21} \rangle t_{22} \in \overline{SPLIT(T_2)} \setminus S\}$  the assumed  $T_1, T_2, l \langle t_{11} \rangle t_{12}$  and  $S$  now yields that  $[t_{12}] \mathcal{R}_{<} T_{22}$ , which means that  $in[[t_{12}]] = IN[[[t_{12}]]] \not\subseteq IN[[T_{22}]]$ .

This means that there is an  $x_2 \in in[[t_{12}]]$  such that  $x_2 \notin IN[[T_{22}]]$ .

Since there is an  $x_1 \in in[[t_{11}]]$  such that  $x_1 \notin IN[[T_{21}]]$  Lemma 4.9 yields that

$x_1 \notin \bigcup_{t_{21} \in \overline{T_{21}}} in[[t_{21}]] = \bigcup_{l \langle t_{21} \rangle t_{22} \in S} in[[t_{21}]]$ ,

and the result of this is that  $\langle l \rangle x_1 \langle / \rangle x_2 \notin \bigcup_{l' \langle t_{21} \rangle t_{22} \in S} in[[l' \langle t_{21} \rangle t_{22}]]$ .

Since there is an  $x_2 \in in[[t_{12}]]$  such that  $x_2 \notin IN[[T_{22}]]$  Lemma 4.9 yields that

$x_2 \notin \bigcup_{t_{22} \in \overline{T_{22}}} in[[t_{22}]] = \bigcup_{l \langle t_{21} \rangle t_{22} \in \overline{SPLIT(T_2)} \setminus S} in[[t_{22}]]$ ,

and the result of this is that  $\langle l \rangle x_1 \langle / \rangle x_2 \notin \bigcup_{l' \langle t_{21} \rangle t_{22} \in \overline{SPLIT(T_2)} \setminus S} in[[l' \langle t_{21} \rangle t_{22}]]$ .

Combining this, we get that  $\langle l \rangle x_1 \langle / \rangle x_2 \notin \bigcup_{l' \langle t_{21} \rangle t_{22} \in \overline{SPLIT(T_2)}} in[[l' \langle t_{21} \rangle t_{22}]]$ , and Lemma 4.9 now yields that  $\langle l \rangle x_1 \langle / \rangle x_2 \notin IN[[\overline{SPLIT(T_2)}]]$ .

Finally Corollary 10.23 yields that

$\langle l \rangle x_1 \langle / \rangle x_2 \notin IN[[T_2]]$ . Since  $\langle l \rangle x_1 \langle / \rangle x_2 \in in[[l \langle t_{11} \rangle t_{12}]] \subseteq IN[[\overline{SPLIT(T_1)}]] \subseteq IN[[T_1]]$  we can conclude that  $IN[[T_1]] \not\subseteq IN[[T_2]]$ .

We can now use contraposition to conclude that

$$\begin{aligned}
T_1 \mathcal{R}_{<} T_2 &\Rightarrow \forall l \langle t_{11} \rangle t_{12} \in \overline{\text{SPLIT}(T_1)}. \forall S \subseteq \overline{\text{SPLIT}(T_2)}. \\
(\exists T_{21}. [t_{11}] \mathcal{R}_{<} T_{21} \wedge \overline{T_{21}} &= \{t_{21} \mid l \langle t_{21} \rangle t_{22} \in S\}) \vee \\
(\exists T_{22}. [t_{12}] \mathcal{R}_{<} T_{22} \wedge \overline{T_{22}} &= \{t_{22} \mid l \langle t_{21} \rangle t_{22} \in \overline{\text{SPLIT}(T_2)} \setminus S\}).
\end{aligned}$$

This proves the second property, and we can conclude that  $\mathcal{R}_{<}$  is a simulation. ■

### Proof 19.33 (Completeness of finite simulations)

We will now prove Theorem 12.8 which states that

$$\forall T_1, T_2 \in \mathbf{XMLTypes}. \models T_1 < T_2 \Rightarrow \mathcal{R}_{T_1, T_2} \text{ is a finite simulation such that } T_1 \mathcal{R} T_2.$$

Select  $T_1, T_2 \in \mathbf{XMLTypes}$  such that  $\models T_1 < T_2$ .

Now we only need to prove the following properties for  $\mathcal{R}_{T_1, T_2}$ .

- $\mathcal{R}_{T_1, T_2}$  is a simulation
- $\mathcal{R}_{T_1, T_2}$  is finite
- $T_1 \mathcal{R}_{T_1, T_2} T_2$

We will now prove them in order.

$\mathcal{R}_{T_1, T_2}$  is a simulation.

First we need to prove that  $T'_1 \mathcal{R}_{T_1, T_2} T'_2 \Rightarrow \text{ESP}(T'_1) \neq \{\} \Rightarrow \text{ESP}(T'_2) \neq \{\}$ .

This is trivial, since  $T'_1 \mathcal{R}_{T_1, T_2} T'_2 \Rightarrow T'_1 \mathcal{R}_{<} T'_2$  and since  $\mathcal{R}_{<}$  is a simulation we have that  $\text{ESP}(T'_1) \neq \{\} \Rightarrow \text{ESP}(T'_2) \neq \{\}$ .

Now assume that  $T'_1 \mathcal{R}_{T_1, T_2} T'_2$ .

Then we know that  $T'_1 \mathcal{R}_{<} T'_2$ , that  $\overline{T'_1} \subseteq \overline{\text{DERIVS}(T_1)}$  and that  $\overline{T'_2} \subseteq \overline{\text{DERIVS}(T_2)}$ .

Now select  $l \langle t_{11} \rangle t_{12} \in \overline{\text{SPLIT}(T_1)}$ , and  $S \subseteq \overline{\text{SPLIT}(T_2)}$ .

Since  $T'_1 \mathcal{R}_{<} T'_2$ , and  $\mathcal{R}_{<}$  is a simulation, we know that

$$\begin{aligned}
(\exists T_{21}. [t_{11}] \mathcal{R}_{<} T_{21} \wedge \overline{T_{21}} &= \{t_{21} \mid l \langle t_{21} \rangle t_{22} \in S\}) \vee \\
(\exists T_{22}. [t_{12}] \mathcal{R}_{<} T_{22} \wedge \overline{T_{22}} &= \{t_{21} \mid l \langle t_{21} \rangle t_{22} \in \overline{\text{SPLIT}(T_2)} \setminus S\}).
\end{aligned}$$

If the first clause is fulfilled, then  $[t_{11}] \mathcal{R}_{<} T_{21}$ .

This does not necessarily mean, that  $[t_{11}] \mathcal{R}_{T_1, T_2} T_{21}$ , but if we define  $T'_{21} = T_{21}$  with multiple occurrences removed, then we have that  $\overline{T'_{21}} = \overline{T_{21}}$ , and therefore  $\text{IN}[\overline{T'_{21}}] = \bigcup_{t \in \overline{T'_{21}}} \text{in}[t] = \bigcup_{t \in \overline{T_{21}}} \text{in}[t] = \text{IN}[\overline{T_{21}}]$ . Since  $[t_{11}] \mathcal{R}_{<} T_{21}$  means that  $\text{IN}[[t_{11}]] \subseteq \text{IN}[\overline{T_{21}}] = \text{IN}[\overline{T'_{21}}]$  we have that  $[t_{11}] \mathcal{R}_{<} T'_{21}$  and since  $\overline{T'_{21}} = \overline{T_{21}} = \{t_{21} \mid l \langle t_{21} \rangle t_{22} \in S\}$  and  $T'_{21}$  was defined to have no multiple occurrences, we have that

$$\begin{aligned}
[t_{11}] \mathcal{R}_{T_1, T_2} T'_{21} \wedge \overline{T'_{21}} &= \{t_{21} \mid l \langle t_{21} \rangle t_{22} \in S\}, \text{ since} \\
\overline{T'_{21}} \subseteq \{t_{21} \mid l \langle t_{21} \rangle t_{22} \in \overline{\text{SPLIT}(T'_2)}\} &\subseteq \text{Deriv}(\overline{T'_2}) \subseteq \text{Deriv}(\overline{\text{DERIVS}(T_2)}) \subseteq \overline{\text{DERIVS}(T_2)}.
\end{aligned}$$

If the second clause is fulfilled, then  $[t_{12}] \mathcal{R}_{<} T_{22}$ .

This does not necessarily mean, that  $[t_{12}] \mathcal{R}_{T_1, T_2} T_{22}$ , but if we define  $T'_{22} = T_{22}$  with multiple occurrences removed, then we have that  $\overline{T'_{22}} = \overline{T_{22}}$ , and therefore

$$\text{IN}[\overline{T'_{22}}] = \bigcup_{t \in \overline{T'_{22}}} \text{in}[t] = \bigcup_{t \in \overline{T_{22}}} \text{in}[t] = \text{IN}[\overline{T_{22}}].$$

Since  $[t_{12}] \mathcal{R}_{<} T_{22}$  means that  $\text{IN}[[t_{12}]] \subseteq \text{IN}[\overline{T_{22}}] = \text{IN}[\overline{T'_{22}}]$  we have that  $[t_{12}] \mathcal{R}_{<} T'_{22}$  and

since  $\overline{T'_{22}} = \overline{T_{22}} = \{t_{22} \mid l\langle t_{21} \rangle t_{22} \in \overline{\text{SPLIT}(T_2)} \setminus S\}$  and  $T'_{21}$  was defined to have no multiple occurrences, we have that

$$\begin{aligned} [t_{12}] \mathcal{R}_{T_1, T_2} T'_{22} \wedge \overline{T'_{22}} &= \{t_{22} \mid l\langle t_{21} \rangle t_{22} \in \overline{\text{SPLIT}(T_2)} \setminus S\}, \text{ since} \\ \overline{T'_{22}} &\subseteq \{t_{22} \mid l\langle t_{21} \rangle t_{22} \in \overline{\text{SPLIT}(T'_2)}\} \subseteq \text{Deriv}(T'_2) \subseteq \text{Deriv}(\text{DERIVS}(T_2)) \subseteq \text{DERIVS}(T_2). \end{aligned}$$

We can therefore conclude that  $\forall l\langle t_{11} \rangle t_{12} \in \overline{\text{SPLIT}(T_1)}. \forall S \subseteq \overline{\text{SPLIT}(T_2)}$ .

$$\begin{aligned} (\exists T_{21}. [t_{11}] \mathcal{R}_{<}. T_{21} \wedge \overline{T_{21}} &= \{t_{21} \mid l\langle t_{21} \rangle t_{22} \in S\}) \vee \\ (\exists T_{22}. [t_{12}] \mathcal{R}_{<}. T_{22} \wedge \overline{T_{22}} &= \{t_{21} \mid l\langle t_{21} \rangle t_{22} \in \overline{\text{SPLIT}(T_2)} \setminus S\}). \end{aligned}$$

We have now proved that  $\mathcal{R}_{T_1, T_2}$  fulfills both properties of a simulation, so  $\mathcal{R}_{T_1, T_2}$  is a simulation.

$\overline{\mathcal{R}_{T_1, T_2}}$  is finite.

Corollary 11.12 yields that  $\text{DERIVS}(T_1)$  and  $\text{DERIVS}(T_2)$  are finite sets, and therefore all  $(T'_1, T'_2) \in \mathcal{R}_{T_1, T_2} \setminus \{(T_1, T_2)\}$  must fulfill that  $T'_1$  and  $T'_2$  are permutations of elements from these finite sets. Because there are finitely many of these permutations  $\mathcal{R}_{T_1, T_2}$  is a finite set.

$\overline{T_1 \mathcal{R}_{T_1, T_2} T_2}$ .

Since  $(T_1, T_2)$  is explicitly added to  $\mathcal{R}_{T_1, T_2}$ , this property is fulfilled.

We have now proved all the desired properties for  $\mathcal{R}_{T_1, T_2}$ , and can therefore conclude that  $\mathcal{R}_{T_1, T_2}$  is a finite simulation such that  $T_1 \mathcal{R}_{T_1, T_2} T_2$ .  $\blacksquare$

### Proof 19.34

We will now prove Lemma 13.5 which states for all finite simulations  $\mathcal{R}$  that

$$T'_1 \mathcal{R} T'_2 \wedge \overline{T'_1} = \overline{T_1} \wedge \overline{T'_2} = \overline{T_2} \Rightarrow \forall \Gamma. \Gamma \vdash T_1 < : T_2$$

by induction on  $|\{(T'_1, T'_2) \in \mathcal{R} \mid \forall (T''_1, T''_2) \in \Gamma. \overline{T''_1} \neq \overline{T'_1} \vee \overline{T''_2} \neq \overline{T'_2}\}|$ .

Start:

If  $|\{(T'_1, T'_2) \in \mathcal{R} \mid \forall (T''_1, T''_2) \in \Gamma. \overline{T''_1} \neq \overline{T'_1} \vee \overline{T''_2} \neq \overline{T'_2}\}| = 0$  then

$$\{(T'_1, T'_2) \in \mathcal{R} \mid \forall (T''_1, T''_2) \in \Gamma. \overline{T''_1} \neq \overline{T'_1} \vee \overline{T''_2} \neq \overline{T'_2}\} = \{\}.$$

Since  $\exists (T'_1, T'_2) \in \mathcal{R}. \overline{T_1} = \overline{T'_1} \wedge \overline{T_2} = \overline{T'_2}$  there must be  $(T''_1, T''_2) \in \Gamma$  such that  $\overline{T_1} = \overline{T''_1}$  and  $\overline{T_2} = \overline{T''_2}$ .

We can therefore use the rule **sub-hyp** to conclude that  $\Gamma \vdash T_1 < : T_2$ .

Induction Hypothesis:

If  $(U'_1, U'_2) \in \mathcal{R}$  such that  $\overline{U_1} = \overline{U'_1}$  and  $\overline{U_2} = \overline{U'_2}$  and

$$|\{(U'_1, U'_2) \in \mathcal{R} \mid \forall (U''_1, U''_2) \in \Gamma'. \overline{U''_1} \neq \overline{U'_1} \vee \overline{U''_2} \neq \overline{U'_2}\}| < |\{(T'_1, T'_2) \in \mathcal{R} \mid \forall (T''_1, T''_2) \in \Gamma. \overline{T''_1} \neq \overline{T'_1} \vee \overline{T''_2} \neq \overline{T'_2}\}|$$

then we can assume that  $\Gamma' \vdash U_1 < : U_2$ .

Step:  $|\{(T'_1, T'_2) \in \mathcal{R} \mid \forall (T''_1, T''_2) \in \Gamma. \overline{T''_1} \neq \overline{T'_1} \vee \overline{T''_2} \neq \overline{T'_2}\}| > 0$

If  $\exists (T''_1, T''_2) \in \Gamma. \overline{T_1} = \overline{T''_1} \wedge \overline{T_2} = \overline{T''_2}$  then we can prove the desired using rule **sub-hyp** as in the induction start.

Otherwise  $\forall (T_1'', T_2'') \in \Gamma. \overline{T_1} \neq \overline{T_1''} \vee \overline{T_2} \neq \overline{T_2''}$ .

This means that  $|\{(U_1', U_2') \in \mathcal{R} \mid \forall (U_1'', U_2'') \in \Gamma \cup \{(T_1, T_2)\}. \overline{U_1''} \neq \overline{U_1'} \vee \overline{U_2''} \neq \overline{U_2'}\}| < |\{(T_1', T_2') \in \mathcal{R} \mid \forall (T_1'', T_2'') \in \Gamma. \overline{T_1''} \neq \overline{T_1'} \vee \overline{T_2''} \neq \overline{T_2'}\}|$ .

We will now show that we can prove the desired using the *sub-diff* rule.

First we need to check that  $\mathbf{ESP}(T_1) \neq \{\} \Rightarrow \mathbf{ESP}(T_2) \neq \{\}$ .

This follows directly from the fact that  $(T_1, T_2) \in \mathcal{R}$  since it is one of the requirements of a simulation.

Now we need to prove that

$$\forall l \langle t_{11} \rangle t_{12} \in \overline{\mathbf{SPLIT}(T_1)}. \forall S \subseteq \overline{\mathbf{SPLIT}(T_2)}.$$

$$(\Gamma \cup \{(T_1, T_2)\} \vdash [t_{11}] \langle : \Pi_1(\mathbf{filter}_S(\mathbf{label}_1(\mathbf{SPLIT}(T_2)))) \rangle) \vee$$

$$(\Gamma \cup \{(T_1, T_2)\} \vdash [t_{12}] \langle : \Pi_2(\mathbf{filter}_{\overline{\mathbf{SPLIT}(T_2)} \setminus S}(\mathbf{label}_1(\mathbf{SPLIT}(T_2)))) \rangle)$$

We start by assuming that  $l \langle t_{11} \rangle t_{12} \in \overline{\mathbf{SPLIT}(T_1)}$ , and that  $S \subseteq \overline{\mathbf{SPLIT}(T_2)}$ .

Since  $\mathcal{R}$  is a simulation, we get that

$$(\exists T_{21}. [t_{11}] \mathcal{R} T_{21} \wedge \overline{T_{21}} = \{t_{21} \mid l \langle t_{21} \rangle t_{22} \in S\}) \vee$$

$$(\exists T_{22}. [t_{12}] \mathcal{R} T_{22} \wedge \overline{T_{22}} = \{t_{22} \mid l \langle t_{21} \rangle t_{22} \in \overline{\mathbf{SPLIT}(T_2)} \setminus S\}).$$

If the first possibility is fulfilled, then we choose a  $T_{21}$  such that it is fulfilled.

Since  $\overline{T_{21}} = \{t_{21} \mid l \langle t_{21} \rangle t_{22} \in S\} = \overline{\Pi_1(\mathbf{filter}_S(\mathbf{label}_1(\mathbf{SPLIT}(T_2))))}$ , the induction hypothesis then yields that

$$\Gamma \cup \{(T_1, T_2)\} \vdash [t_{11}] \langle : \Pi_1(\mathbf{filter}_S(\mathbf{label}_1(\mathbf{SPLIT}(T_2)))) \rangle.$$

If the second possibility is fulfilled, then we choose a  $T_{22}$  such that it is fulfilled.

Since  $\overline{T_{22}} = \{t_{22} \mid l \langle t_{21} \rangle t_{22} \in \overline{\mathbf{SPLIT}(T_2)} \setminus S\} = \overline{\Pi_2(\mathbf{filter}_{\overline{\mathbf{SPLIT}(T_2)} \setminus S}(\mathbf{label}_1(\mathbf{SPLIT}(T_2))))}$ ,

The induction hypothesis then yields that

$$\Gamma \cup \{(T_1, T_2)\} \vdash [t_{12}] \langle : \Pi_2(\mathbf{filter}_{\overline{\mathbf{SPLIT}(T_2)} \setminus S}(\mathbf{label}_1(\mathbf{SPLIT}(T_2)))) \rangle.$$

Since there are only those two possibilities, we can therefore conclude that

$$(\Gamma \cup \{(T_1, T_2)\} \vdash [t_{11}] \langle : \Pi_1(\mathbf{filter}_S(\mathbf{label}_1(\mathbf{SPLIT}(T_2)))) \rangle) \vee$$

$$(\Gamma \cup \{(T_1, T_2)\} \vdash [t_{12}] \langle : \Pi_2(\mathbf{filter}_{\overline{\mathbf{SPLIT}(T_2)} \setminus S}(\mathbf{label}_1(\mathbf{SPLIT}(T_2)))) \rangle).$$

Since we have had no constraints on  $l \langle t_{11} \rangle t_{12}$  and  $S$ , we can conclude that

$$\forall l \langle t_{11} \rangle t_{12} \in \overline{\mathbf{SPLIT}(T_1)}. \forall S \subseteq \overline{\mathbf{SPLIT}(T_2)}.$$

$$(\Gamma \cup \{(T_1, T_2)\} \vdash [t_{11}] \langle : \Pi_1(\mathbf{filter}_S(\mathbf{label}_1(\mathbf{SPLIT}(T_2)))) \rangle) \vee$$

$$(\Gamma \cup \{(T_1, T_2)\} \vdash [t_{12}] \langle : \Pi_2(\mathbf{filter}_{\overline{\mathbf{SPLIT}(T_2)} \setminus S}(\mathbf{label}_1(\mathbf{SPLIT}(T_2)))) \rangle).$$

We can therefore use rule *sub-diff* to conclude that  $\Gamma \vdash T_1 \langle : T_2$ . ■

**Proof 19.35 (Size stratified soundness of axiomatization)**

We will now prove Lemma 13.10 which states that

$$\forall T_1, T_2, \Gamma. \Gamma \vdash T_1 < : T_2 \Rightarrow \Gamma \vDash^s T_1 < : T_2$$

by rule induction.

*Induction Hypothesis:*

If the current rule has the requirement that  $\Gamma' \vdash T'_1 < : T'_2$  then we can assume that  $\Gamma' \vDash^s T'_1 < : T'_2$ .

*Case: sub-hyp*

Assume that  $\exists (T''_1, T''_2) \in \Gamma. \overline{T''_1} = \overline{T_1} \wedge \overline{T''_2} = \overline{T_2}$ .

We need to prove that  $\Gamma \vDash^s T_1 < : T_2$ .

By definition this means that we need to prove that

$$\forall n \in \mathbb{N}_0. \vDash_n \Gamma \Rightarrow \vDash_n T_1 < : T_2.$$

Assume that  $n \in \mathbb{N}_0$ , and then assume that  $\vDash_n \Gamma$ .

This means that  $\forall (T''_1, T''_2) \in \Gamma. \vDash_n T''_1 < : T''_2$ .

Since  $(T''_1, T''_2) \in \Gamma$  we get that  $\vDash_n T''_1 < : T''_2$  and therefore

$$\begin{aligned} & \vDash_n T''_1 < : T''_2 \\ \Rightarrow & \text{IN}[T''_1] \upharpoonright_n \subseteq \text{IN}[T''_2] \upharpoonright_n \\ \text{(Lemma 4.9)} \Rightarrow & \left( \bigcup_{t \in \overline{T''_1}} \text{in}[t] \right) \Big|_n \subseteq \left( \bigcup_{t \in \overline{T''_2}} \text{in}[t] \right) \Big|_n \\ \Rightarrow & \left( \bigcup_{t \in \overline{T_1}} \text{in}[t] \right) \Big|_n \subseteq \left( \bigcup_{t \in \overline{T_2}} \text{in}[t] \right) \Big|_n \\ \text{(Lemma 4.9)} \Rightarrow & \text{IN}[T_1] \upharpoonright_n \subseteq \text{IN}[T_2] \upharpoonright_n \\ \Rightarrow & \vDash_n T_1 < : T_2 \end{aligned}$$

We have therefore proved the desired in this case.

*Case: sub-diff*

In this case the induction hypothesis yields that

$$\forall \langle t_{11} \rangle t_{12} \in \overline{\text{SPLIT}(T_1)}, S \subseteq \overline{\text{SPLIT}(T_2)}.$$

$$(\Gamma \cup \{T_1, T_2\} \vDash^s [t_{11}] < : \Pi_1(\text{filters}(\text{label}_1(\text{SPLIT}(T_2)))) \vee$$

$$(\Gamma \cup \{T_1, T_2\} \vDash^s [t_{12}] < : \Pi_2(\text{filter}_{\overline{\text{SPLIT}(T_2)} \setminus S}(\text{label}_1(\text{SPLIT}(T_2))))).$$

We need to prove that  $\Gamma \vDash^s T_1 < : T_2$ .

By definition this mean that we need to prove that

$$\forall n \in \mathbb{N}_0. \vDash_n \Gamma \Rightarrow \vDash_n T_1 < : T_2.$$

We will prove this by induction on  $n$ .

Start:  $k = 0$

The right side of the implication we wish to prove is  $\models_0 T_1 < : T_2$  and this means that  $IN[[T_1]]|_0 \subseteq IN[[T_2]]|_0$ .

Since  $IN[[T_1]]|_0 = \{\}$  this is vacantly fulfilled.

We can therefore conclude that  $\models_0 T_1 < : T_2$ .

Induction Hypothesis 2:

We can in the induction step assume that  $\Gamma \models_{n-1} T_1 < : T_2$ .

Step:  $n > 0$

In this case the induction hypothesis 2 yields that  $\Gamma \models_{n-1} T_1 < : T_2$ .

Assume  $\models_n \Gamma$ .

Hence  $\models_{n-1} \Gamma$ .

Since we have that  $\Gamma \models_{n-1} T_1 < : T_2$  which means that  $\models_{n-1} \Gamma \Rightarrow \models_{n-1} T_1 < : T_2$  we get that  $\models_{n-1} T_1 < : T_2$ .

Therefore we have that  $\models_{n-1} \Gamma \cup \{T_1, T_2\}$ .

Now the first induction hypothesis yields that

$$\forall l \langle t_{11} \rangle t_{12} \in \overline{SPLIT(T_1)}, S \subseteq \overline{SPLIT(T_2)}.$$

$$(\Gamma \cup \{T_1, T_2\} \models^s [t_{11}] < : \Pi_1(\text{filters}(\text{label}_1(\overline{SPLIT(T_2)})))) \vee$$

$$(\Gamma \cup \{T_1, T_2\} \models^s [t_{12}] < : \Pi_2(\text{filter}_{\overline{SPLIT(T_2)} \setminus S}(\text{label}_1(\overline{SPLIT(T_2)})))).$$

Since  $\models_{n-1} \Gamma \cup \{T_1, T_2\}$  this yields that

$$\forall l \langle t_{11} \rangle t_{12} \in \overline{SPLIT(T_1)}, S \subseteq \overline{SPLIT(T_2)}.$$

$$(\models_{n-1} [t_{11}] < : \Pi_1(\text{filters}(\text{label}_1(\overline{SPLIT(T_2)})))) \vee$$

$$(\models_{n-1} [t_{12}] < : \Pi_2(\text{filter}_{\overline{SPLIT(T_2)} \setminus S}(\text{label}_1(\overline{SPLIT(T_2)})))).$$

Assume that  $\langle 1 \rangle x_1 \langle /1 \rangle x_2 \in IN[\overline{SPLIT(T_1)}]|_n$ .

This means that  $|\langle 1 \rangle x_1 \langle /1 \rangle x_2| \leq n$  and therefore  $|x_1| \leq n - 1$  and  $|x_2| \leq n - 1$ .

Now Lemma 4.9 yields that  $\exists l \langle t_{11} \rangle t_{12} \in \overline{SPLIT(T_1)}$  such that  $\langle 1 \rangle x_1 \langle /1 \rangle x_2 \in \text{in}[[l \langle t_{11} \rangle t_{12}]]$ .

The definition of  $\text{in}[[\cdot]]$  yields that  $x_1 \in \text{in}[[t_{11}]]$  and  $x_2 \in \text{in}[[t_{12}]]$ .

We now define

$$S = \{l \langle t_{21} \rangle t_{22} \in \overline{SPLIT(T_2)} \mid x_2 \in \text{in}[[t_{21}]]\}.$$

Assume that  $\models_{n-1} [t_{12}] < : \Pi_2(\text{filter}_{\overline{SPLIT(T_2)} \setminus S}(\text{label}_1(\overline{SPLIT(T_2)}))))$ .

Since  $x_2 \in \text{in}[[t_{12}]] = IN[[t_{12}]]$  and  $|x_2| \leq n - 1$  we get that

$x_2 \in \text{IN}[\Pi_2(\overline{\text{filter}_{\text{SPLIT}(T_2)} \setminus S}(\text{label}_1(\text{SPLIT}(T_2))))]$  and therefore

$$\begin{aligned}
x_2 &\in \text{IN}[\Pi_2(\overline{\text{filter}_{\text{SPLIT}(T_2)} \setminus S}(\text{label}_1(\text{SPLIT}(T_2))))] \\
(\text{Lemma 4.9}) &= \bigcup_{t \in \Pi_2(\overline{\text{filter}_{\text{SPLIT}(T_2)} \setminus S}(\text{label}_1(\text{SPLIT}(T_2))))} \text{in}[t] \\
&= \bigcup_{t \in \{t_{22} \mid l\langle t_{21} \rangle t_{22} \in \overline{\text{SPLIT}(T_2)} \setminus S\}} \text{in}[t] \\
&= \bigcup_{t \in \{t_{22} \mid l\langle t_{21} \rangle t_{22} \in \overline{\text{SPLIT}(T_2)} \setminus \{l\langle t'_{21} \rangle t'_{22} \in \overline{\text{SPLIT}(T_2)} \mid x_2 \in \text{in}[t'_{22}]\}} \text{in}[t] \\
&= \bigcup_{t \in \{t_{22} \mid l\langle t_{21} \rangle t_{22} \in \overline{\text{SPLIT}(T_2)} \wedge x_2 \notin \text{in}[t_{22}]\}} \text{in}[t]
\end{aligned}$$

This means that there must be a  $l\langle t_{21} \rangle t_{22} \in \overline{\text{SPLIT}(T_2)}$  where  $x_2 \notin \text{in}[t_{22}]$  such that  $x_2 \in \text{in}[t_{22}]$ .

This is a contradiction, so we can conclude that the assumption  $\models_{n-1} [t_{12}] <: \Pi_2(\overline{\text{filter}_{\text{SPLIT}(T_2)} \setminus S}(\text{label}_1(\text{SPLIT}(T_2))))$  is not true.

Since we have that

$$\begin{aligned}
&\forall l\langle t_{11} \rangle t_{12} \in \overline{\text{SPLIT}(T_1)}, S \subseteq \overline{\text{SPLIT}(T_2)}. \\
&(\models_{n-1} [t_{11}] <: \Pi_1(\overline{\text{filter}_S(\text{label}_1(\text{SPLIT}(T_2)))))) \vee \\
&(\models_{n-1} [t_{12}] <: \Pi_2(\overline{\text{filter}_{\text{SPLIT}(T_2)} \setminus S}(\text{label}_1(\text{SPLIT}(T_2))))))
\end{aligned}$$

we can conclude that

$$\models_{n-1} [t_{11}] <: \Pi_1(\overline{\text{filter}_S(\text{label}_1(\text{SPLIT}(T_2))))).$$

This means that  $x_1 \in \text{in}[t_{12}]|_{n-1} = \text{IN}[[t_{12}]]|_{n-1} \subseteq \text{IN}[\Pi_1(\overline{\text{filter}_S(\text{label}_1(\text{SPLIT}(T_2))))]|_{n-1}$  and since

$$\begin{aligned}
&\text{IN}[\Pi_1(\overline{\text{filter}_S(\text{label}_1(\text{SPLIT}(T_2))))] \\
(\text{Lemma 4.9}) &= \bigcup_{t \in \Pi_1(\overline{\text{filter}_S(\text{label}_1(\text{SPLIT}(T_2))))} \text{in}[t] \\
&= \bigcup_{t \in \{t_{21} \mid l\langle t_{21} \rangle t_{22} \in \overline{\text{SPLIT}(T_2)} \wedge x_2 \in \text{in}[t_{22}]\}} \text{in}[t]
\end{aligned}$$

there is a  $l\langle t_{21} \rangle t_{22} \in \overline{\text{SPLIT}(T_2)}$  such that  $x_1 \in \text{in}[t_{21}]$  and  $x_2 \in \text{in}[t_{22}]$ .

This means that  $\vdash x_1$  inhabits  $t_{21}$  and  $\vdash x_2$  inhabits  $t_{22}$ .

We can now make the following derivation

$$\text{in-seq} \frac{\text{in-tree} \frac{\vdash x_1 \text{ inhabits } t_{21}}{\vdash \langle l \rangle x_1 \langle /l \rangle \varepsilon \text{ inhabits } l\langle t_{21} \rangle} \quad \vdash x_2 \text{ inhabits } t_{22}}{\vdash \langle l \rangle x_1 \langle /l \rangle x_2 \text{ inhabits } l\langle t_{21} \rangle t_{22}}$$

and therefore  $\langle l \rangle x_1 \langle /l \rangle x_2 \in \text{in}[l\langle t_{21} \rangle t_{22}]$ .

Since  $\text{IN}[\text{SPLIT}(T_2)] = \bigcup_{l\langle t_{21} \rangle t_{22} \in \overline{\text{SPLIT}(T_2)}} \text{in}[t]$  we have that  $\langle l \rangle x_1 \langle /l \rangle x_2 \in \text{IN}[\text{SPLIT}(T_2)]$ ,

and since  $|\langle 1 \rangle x_1 \langle /1 \rangle x_2| \leq n$  we have that  $\langle 1 \rangle x_1 \langle /1 \rangle x_2 \in \text{IN}[\text{SPLIT}(T_2)]|_n$ .

We can therefore conclude that  $\text{IN}[\text{SPLIT}(T_1)]|_n \subseteq \text{IN}[\text{SPLIT}(T_2)]|_n$ .

Now we are ready to conclude that

$$\begin{aligned}
\text{IN}[T_1]|_n &= \{\varepsilon \mid \varepsilon \in \text{IN}[T_1]\}_n \cup \{\langle 1 \rangle x_1 \langle /1 \rangle x_2 \mid \langle 1 \rangle x_1 \langle /1 \rangle x_2 \in \text{IN}[T_1]\}_n \\
&= \{\varepsilon \mid \varepsilon \in \text{IN}[T_1]\} \cup \{\langle 1 \rangle x_1 \langle /1 \rangle x_2 \mid \langle 1 \rangle x_1 \langle /1 \rangle x_2 \in \text{IN}[\text{SPLIT}(T_1)]\}_n \\
&= \{\varepsilon \mid \text{ESP}(T_1) \neq \{\}\} \cup \{\langle 1 \rangle x_1 \langle /1 \rangle x_2 \mid \langle 1 \rangle x_1 \langle /1 \rangle x_2 \in \text{IN}[\text{SPLIT}(T_1)]|_n\} \\
&\subseteq \{\varepsilon \mid \text{ESP}(T_2) \neq \{\}\} \cup \{\langle 1 \rangle x_1 \langle /1 \rangle x_2 \mid \langle 1 \rangle x_1 \langle /1 \rangle x_2 \in \text{IN}[\text{SPLIT}(T_2)]|_n\} \\
&= \{\varepsilon \mid \varepsilon \in \text{IN}[T_2]\} \cup \{\langle 1 \rangle x_1 \langle /1 \rangle x_2 \mid \langle 1 \rangle x_1 \langle /1 \rangle x_2 \in \text{IN}[\text{SPLIT}(T_2)]\}_n \\
&= \{\varepsilon \mid \varepsilon \in \text{IN}[T_2]\} \cup \{\langle 1 \rangle x_1 \langle /1 \rangle x_2 \mid \langle 1 \rangle x_1 \langle /1 \rangle x_2 \in \text{IN}[T_2]\}_n \\
&= \text{IN}[T_2]|_n
\end{aligned}$$

This means that  $\models_n T_1 \prec T_2$ , and we can therefore conclude that  $\models_n \Gamma \Rightarrow \models_n T_1 \prec T_2$ .

We have now proved both the induction start and step, and we can therefore conclude that  $\forall n \in \mathbb{N}_0. \models_n \Gamma \Rightarrow \models_n T_1 \prec T_2$ , and therefore

$$\Gamma \models^s T_1 \prec T_2.$$

We have now proved all the cases in the rule induction, and we can therefore conclude that

$$\Gamma \vdash T_1 \prec T_2 \Rightarrow \Gamma \models^s T_1 \prec T_2.$$

■

### Proof 19.36 (Trans does not increase the data-size)

We will now prove Lemma 14.1 which states that

$$\forall (n, \langle d_1 \rangle d_2) \in \text{Trans}_\varphi(t, d). |d_1| < |d| \wedge |d_2| < |d|$$

by induction on the calltree of  $\text{Trans}_\varphi(t, d)$ .

*Induction Hypothesis:*

If  $\text{Trans}_\varphi(t', d')$  is called by  $\text{Trans}_\varphi(t, d)$  and  $(n', \langle d'_1 \rangle d'_2) \in \text{Trans}_\varphi(t', d')$  then we can assume that  $|d'_1| < |d'|$  and  $|d'_2| < |d'|$ .

We will consider each case in  $\text{Trans}_\varphi$  individually.

Case:  $\text{Trans}_\varphi(l(t_1), d) = \{0, d\bullet\}$ :

If  $\langle d_1 \rangle d_2 \in \text{Trans}_\varphi(t, d)$  then  $d = \langle d_1 \rangle$  and  $d_2 = \bullet$ , and therefore

$|d| = 1 + |d_1| > |d_1|$  and

$|d| = 1 + |d_1| > 1 = |e| = |d_2|$ .

Therefore the lemma is fulfilled in this case.

Case:  $\text{Trans}_\varphi(t_1 t_2, d_1 d_2)$ :

There are two sub-cases.

If  $\mathbf{Trans}_\varphi(t, d) = \{(|\mathbf{Split}_\varphi(t_1)| + n_2, d'_2) \mid (n_2, d'_2) \in \mathbf{Trans}(t_2, d_2)\}$

then we know that  $|d_1 d_2| = 1 + |d_1| + |d_2| > |d_2|$ .

If  $(n', \langle d'_1 \rangle d'_2) \in \mathbf{Trans}_\varphi(t_1 t_2, d_1 d_2)$  then  $(n' - |\mathbf{Split}_\varphi(t_1)|, \langle d'_1 \rangle d'_2) \in \mathbf{Trans}_\varphi(t_2, d_2)$ .

Now the induction hypothesis yields that  $|d_2| > |d'_1|$  and  $|d_2| > |d'_2|$ .

Therefore the lemma is fulfilled in this case.

If  $\mathbf{Trans}_\varphi(t, d) = \{(n, d'_{11} \langle d'_{12} d_2 \rangle) \mid (n, d'_{11} d'_{12}) \in \mathbf{Trans}_\varphi(t_1, d_1)\}$

and  $(n', \langle d'_1 \rangle d'_2) \in \mathbf{Trans}_\varphi(t, d)$  then  $d'_{11} = \langle d'_1 \rangle$  and  $d'_2 = d'_{12} d_2$ , where  $(n', d'_{11} d'_{12}) \in \mathbf{Trans}_\varphi(t_1, d_1)$ .

Now the induction hypothesis yields that  $|d_1| > |d'_{11}|$  and since

$|d'_{11}| = 1 + |d'_1|$  we have that

$|d_1 d_2| = 1 + |d_1| + |d_2| > |d_1| > |d'_1|$ .

The induction hypothesis also yields that  $|d_1| > |d'_{12}|$ .

Therefore  $|d_1 d_2| = 1 + |d_1| + |d_2| > 1 + |d'_{12}| + |d_2| = |d'_2|$ .

Therefore the lemma is fulfilled in this case.

Case:  $\mathbf{Trans}_\varphi(t_1 + t_2, d_1 \diamond) = \mathbf{Trans}_\varphi(t_1, d_1)$ :

Since  $|d_1 \diamond| = 1 + |d_1|$ , the induction hypothesis yields that the lemma is fulfilled in this case.

Case:  $\mathbf{Trans}_\varphi(t_1 + t_2, \diamond d_2) = \{(n + |\mathbf{Split}_\varphi(t_1)|, d') \mid (n, d') \in \mathbf{Trans}_\varphi(t_2, d_2)\}$ :

Since  $|\diamond d_2| = 1 + |d_2|$ , the induction hypothesis yields that the lemma is fulfilled in this case.

Case:  $\mathbf{Trans}_\varphi(\mu X.t_1, d) = \mathbf{Trans}_{\mu X.t_1/X::\varphi}(t_1, \mathbf{simplify}_{\mu X.t_1[\varphi]/X, d}(t_1[\mu X.t_1/X :: \varphi], d))$ :

In this case

$\mathbf{Trans}_\varphi(\mu X.t_1, d) = \mathbf{Trans}_{\mu X.t_1/X::\varphi}(t_1, \mathbf{simplify}_{\mu X.t_1[\varphi], d}(t_1[\mu X.t_1/X :: \varphi], d))$ .

since  $|\mathbf{simplify}_{\mu X.t_1[\varphi], d}(t_1[\mu X.t_1/X :: \varphi], d)| \leq |d|$ , the induction hypothesis yields that the lemma is fulfilled in this case.

Case:  $\mathbf{Trans}_\varphi(-, -) = \{\}$ :

In this case  $\mathbf{Trans}_\varphi(t, d) = \{\}$ , so the lemma is vacantly fulfilled in this case.

This concludes all the cases, and therefore we can conclude that

$$\forall (n, \langle d_1 \rangle d_2) \in \mathbf{Trans}_\varphi(t, d). |d_1| < |d| \wedge |d_2| < |d|$$

■

### Proof 19.37 (Soundness of $\mathbb{C}[\cdot]$ )

We will now prove Lemma 14.10 which states that

$$\begin{aligned} & (\forall (T'_1, T'_2) \in \Gamma. \forall D'_1. \mathbf{SIZE}(D'_1) \leq \mathbf{SIZE}(D_1) \Rightarrow \forall D'_2 \in \mathbf{c}_{T'_1, T'_2}(D'_1). \mathbf{TAG}(T'_2, D'_2) = \mathbf{TAG}(T'_1, D'_1)) \\ & \Rightarrow \forall D_2 \in \mathbb{C}[\Gamma \vdash T_1 <: T_2]. \mathbf{TAG}(T_2, D_2) = \mathbf{TAG}(T_1, D_1) \end{aligned}$$

by transfinite induction on  $\mathbf{SIZE}(D_1)$ .

We will handle each case in  $\mathbb{C}[\cdot]$  separately.

*Induction Hypothesis:*

If  $SIZE(D'_1) < SIZE(D_1)$  and  
 $\forall (T''_1, T''_2) \in \Gamma'. \forall D''_1. SIZE(D''_1) < SIZE(D'_1) \Rightarrow \forall D''_2 \in c_{T''_1, T''_2}(D''_1). TAG(T''_2, D''_2) = TAG(T''_1, D''_1)$   
then we can assume that  $\forall D'_2 \in C[\Gamma' \vdash T'_1 < : T'_2](D'_1). TAG(T'_2, D'_2) = TAG(T'_1, D'_1)$ .

Case: *sub-hyp*

Since  $nth_{n'_1}(T'_1) = nth_{n_1}(T_1)$  we have that  $TAG(T_1, (n_1, d_1)) = TAG(T'_1, (n'_1, d_1))$ .

The assumptions yields that  $(n'_2, d_2) \in c_{T'_1, T'_2}(n'_1, d_1) \Rightarrow TAG(T'_2, (n'_2, d_2)) = TAG(T'_1, (n'_1, d_1))$ .

Since  $nth_{n_2}(T_2) = nth_{n'_2}(T_2)$  we have that  $TAG(T_2, (n_2, d_2)) = TAG(T'_2, (n'_2, d_2))$ .

We therefore have that

$$\begin{aligned} & TAG(T_1, (n_1, d_1)) \\ &= TAG(T'_1, (n'_1, d_1)) \\ &= TAG(T'_2, (n'_2, d_2)) \\ &= TAG(T_2, (n_2, d_2)). \end{aligned}$$

Therefore the Lemma is fulfilled in this case.

Case: *sub-diff*

Since the result of  $C[\Gamma \vdash T_1 < : T_2]$  starts with *fix*  $c_{T_1, T_2}$  we know that any recursive call will have  $c_{T_1, T_2} = C[\Gamma \vdash T_1 < : T_2]$ .

If  $TAG(T_1, D_1) = \top$  then  $\{\}$  is returned and therefore the lemma is vacantly fulfilled.

If  $TAG(T_1, D_1) = \varepsilon$  then  $ESP(T_2)$  is returned and therefore Corollary 7.10 yields that  $\forall D_2 \in C[\Gamma \vdash T_1 < : T_2]. TAG(T_2, D_2) = \varepsilon = TAG(T_1, D_1)$ .

Therefore the lemma is fulfilled in this case.

We can therefore assume that  $TAG(T_1, D_1) \notin \{\top, \varepsilon\}$ .

The induction hypothesis yields that if  $SIZE(D'_1) < SIZE(D_1)$  then

$\forall D'_2 \in C[\Gamma \vdash T_1 < : T_2](D'_1). TAG(T_2, D'_2) = TAG(T_1, D'_1)$ .

Therefore the assumptions are fulfilled for  $\Gamma \cup \{(T_1, T_2)\}$  for all  $D'_1$  where  $SIZE(D'_1) < SIZE(D_1)$ .

Therefore the induction hypothesis yields that

$\forall (n'_2, d_{21}) \in C[\Gamma \vdash [t_{11}], \Pi_1(\text{label}_1(\text{SPLIT}(T_2)))](0, d_{11})$ .

$TAG(\Pi_1(\text{label}_1(\text{SPLIT}(T_2))), (n'_2, d_{21})) = TAG([t_{11}], (0, d_{11}))$ , and

$\forall (n'_2, d_{22}) \in C[\Gamma \vdash [t_{12}], \Pi_2(\text{label}_1(\text{SPLIT}(T_2)))](0, d_{12})$ .

$TAG(\Pi_2(\text{label}_1(\text{SPLIT}(T_2))), (n'_2, d_{22})) = TAG([t_{12}], (0, d_{12}))$ .

We therefore get that

$$\begin{aligned}
& \text{TAG}(T_1, D_1) \\
(\text{Lemma 10.18}) &= \text{TAG}(\text{SPLIT}(T_1), (n_1, \langle d_{11} \rangle d_{12})) \\
&= \text{tag}(l \langle t_{11} \rangle t_{12}, \langle d_{11} \rangle d_{12}) \\
&= \text{let } x_1 = \text{tag}(t_{11}, d_{11}), x_2 = \text{tag}(t_{12}, d_{12}) \text{ in } \langle 1 \rangle x_1 \langle /1 \rangle x_2 \text{ end} \\
&= \text{let } x_1 = \text{TAG}([t_{11}], (0, d_{11})), x_2 = \text{TAG}([t_{12}], (0, d_{12})) \text{ in } \langle 1 \rangle x_1 \langle /1 \rangle x_2 \text{ end} \\
(2 \times \text{IH}) &= \text{let } x_1 = \text{TAG}(\Pi_1(\text{label}_1(\text{SPLIT}(T_2))), (n'_2, d_{21})), \\
&\quad x_2 = \text{TAG}(\Pi_2(\text{label}_1(\text{SPLIT}(T_2))), (n'_2, d_{22})) \text{ in } \langle 1 \rangle x_1 \langle /1 \rangle x_2 \text{ end} \\
&= \text{let } x_1 = \text{TAG}(\Pi_1(\text{SPLIT}(T_2)), (\text{unlabel}(n'_2, l, \text{SPLIT}(T_2)), d_{21})), \\
&\quad x_2 = \text{TAG}(\Pi_2(\text{label}_1(\text{SPLIT}(T_2))), (\text{unlabel}(n'_2, l, \text{SPLIT}(T_2)), d_{22})) \\
&\quad \text{in } \langle 1 \rangle x_1 \langle /1 \rangle x_2 \text{ end} \\
&= \text{TAG}(\text{SPLIT}(T_2), (\text{unlabel}(n'_2, l, \text{SPLIT}(T_2)), \langle d_{21} \rangle d_{22})) \\
(\text{Lemma 10.21}) &= \text{TAG}(T_2, D_2)
\end{aligned}$$

Therefore the lemma is fulfilled in this case.

We have now proved the lemma in all the cases, and we can therefore conclude that

$$\begin{aligned}
& (\forall (T'_1, T'_2) \in \Gamma. \text{SIZE}(D'_1) \leq \text{SIZE}(D_1) \Rightarrow D'_2 \in \mathcal{C}_{T'_1, T'_2}(D'_1) \Rightarrow \text{TAG}(T'_2, D'_2) = \text{TAG}(T'_1, D'_1)) \\
& \Rightarrow D_2 \in \mathcal{C}[\Gamma \vdash T_1 \langle : T_2 \rangle] \Rightarrow \text{TAG}(T_2, D_2) = \text{TAG}(T_1, D_1).
\end{aligned}$$

■

### Proof 19.38 (Completeness of $\mathcal{C}[\cdot]$ )

We will now prove Lemma 14.12 which states that

$$\begin{aligned}
& \forall D_1. (\forall (T'_1, T'_2) \in \Gamma. \forall D'_1. \text{SIZE}(D'_1) \leq \text{SIZE}(D_1) \Rightarrow \forall j. \text{TAG}(T'_1, D'_1) \in \text{in}[\text{nth}_j(T'_2)]) \\
& \quad \Rightarrow \{(n'_2, d'_2) \in \mathcal{C}_{T'_1, T'_2}(D'_1) \mid n'_2 = j\} \neq \{\}) \\
& \Rightarrow \forall j. \text{TAG}(T_1, D_1) \in \text{IN}[\text{nth}_j(T_2)] \Rightarrow \{(n_2, d_2) \in \mathcal{C}[\Gamma \vdash T_1 \langle : T_2 \rangle](D_1) \mid n_2 = j\} \neq \{\}
\end{aligned}$$

by transfinite induction on  $\text{SIZE}(D_1)$ .

We will consider each case in  $\mathcal{C}[\Gamma \vdash T_1 \langle : T_2 \rangle]$  individually.

*Induction Hypothesis:*

If  $\text{SIZE}(D'_1) < \text{SIZE}(D_1)$  and

$$\forall (T''_1, T''_2) \in \Gamma. \forall D''_1. \text{SIZE}(D''_1) \leq \text{SIZE}(D'_1) \Rightarrow \forall j. \text{TAG}(T''_1, D''_1) \in \text{in}[\text{nth}_j(T''_2)] \Rightarrow \{(n''_2, d''_2) \in \mathcal{C}_{T''_1, T''_2}(D''_1) \mid n''_2 = j\} \neq \{\}$$

then we can assume that  $\forall j. \text{TAG}(T'_1, D'_1) \in \text{IN}[\text{nth}_j(T_2)] \Rightarrow \{(n'_2, d'_2) \in \mathcal{C}[\Gamma \vdash T'_1 \langle : T'_2 \rangle](D'_1) \mid n'_2 = j\} \neq \{\}$ .

*Case: sub-hyp*

Let  $j$  be chosen such that  $\text{TAG}(T_1, D_1) \in \text{in}[\text{nth}_j(T_2)]$ . Since  $\overline{T_1} = \overline{T'_1}$  there must be a  $n'_1$  such that  $\text{nth}_{n'_1}(T'_1) = \text{nth}_{n_1}(T_1)$ .

Since  $\mathbf{nth}_{n'_1}(T'_1) = \mathbf{nth}_{n_1}(T_1)$  we have that  $\mathbf{TAG}(T_1, (n_1, d_1)) = \mathbf{TAG}(T'_1, (n'_1, d_1))$ .

Since  $\overline{T_2} = \overline{T'_2}$  there must be a  $n'_2$  such that  $\mathbf{nth}_{n'_2}(T'_2) = \mathbf{nth}_j(T_2)$ .

This means that  $\mathbf{TAG}(T'_1, D'_1) = \mathbf{TAG}(T_1, D_1) \in \mathbf{in}[\mathbf{nth}_j(T_2)] = \mathbf{in}[\mathbf{nth}_{n'_2}(T'_2)]$ .

Therefore the assumptions yields that  $\{(n'_2, d_2) \in \mathbf{c}_{T'_1, T'_2}(n'_1, d_1) \mid n'_2 = n'_2 \neq \{\}\}$ .

Since  $\mathbf{nth}_j(T_2) = \mathbf{nth}_{n'_2}(T_2)$  we have that  $\{(n_2, d_2) \in \mathbf{C}[\Gamma \vdash T_1 <: T_2] \mid n_2 = j\} \neq \{\}$ .

Therefore the Lemma is fulfilled in this case.

Case: *sub-diff*

Since the result of  $\mathbf{C}[\Gamma \vdash T_1 <: T_2]$  starts with *fix*  $\mathbf{c}_{T_1, T_2}$  we know that any recursive call will have  $\mathbf{c}_{T_1, T_2} = \mathbf{C}[\Gamma \vdash T_1 <: T_2]$ .

If  $\mathbf{TAG}(T_1, D_1) = \top$  then the lemma is trivially fulfilled in this case.

If  $\mathbf{TAG}(T_1, D_1) = \varepsilon$  then Lemma 7.11 yields that

$\forall j. \varepsilon \in \mathbf{in}[\mathbf{nth}_j(T_2)] \Rightarrow \{(n_2, d_2) \in \mathbf{ESP}(T_2) \mid n_2 = j\} \neq \{\}$ .

Since  $\mathbf{ESP}(T_2)$  is returned, the lemma is fulfilled in this case.

We can therefore assume that  $\mathbf{TAG}(T_1, D_1) \notin \{\top, \varepsilon\}$ .

Therefore Lemma 10.19 yields that  $\mathbf{TRANS}(T_1, D_1) \neq \{\}$ .

Therefore there is a  $(n_1, \langle d_{11} \rangle d_{12}) \in \mathbf{TRANS}(T_1, D_2)$  and Lemma 10.18 yields that

$\mathbf{TAG}(\mathbf{SPLIT}(T_1), (n_1, \langle d_{11} \rangle d_{12})) = \mathbf{TAG}(T_1, D_1)$ .

If  $l \langle t_{11} \rangle t_{12} = \mathbf{nth}_{n_1}(\mathbf{SPLIT}(T_1))$  then

$$\begin{aligned} & \mathbf{TAG}(\mathbf{SPLIT}(T_1), (n_1, \langle d_{11} \rangle d_{12})) \\ &= \mathbf{let } x_1 = \mathbf{tag}(t_{11}, d_{11}), x_2 = \mathbf{tag}(t_2, d_{12}) \mathbf{ in } \langle l \rangle x_1 \langle /l \rangle x_2 \mathbf{ end} \\ &= \mathbf{let } x_1 = \mathbf{TAG}([t_{11}], (0, d_{11})), x_2 = \mathbf{TAG}([t_2], (0, d_{12})) \mathbf{ in } \langle l \rangle x_1 \langle /l \rangle x_2 \mathbf{ end} \end{aligned}$$

Let  $j$  be chosen such that  $\langle l \rangle x_1 \langle /l \rangle x_2 \in \mathbf{in}[\mathbf{nth}_j(T_2)]$ .

Corollary 10.23 yields that  $\mathbf{TAG}(T_1, D_1) \in \mathbf{IN}[\mathbf{Split}(\mathbf{nth}_j(T_2))]$ .

Given the form of  $\mathbf{SPLIT}(T_2)$  there must be a  $j'$  such that

$\langle l \rangle x_1 \langle /l \rangle x_2 \in \mathbf{IN}[\mathbf{nth}_{j'}(\mathbf{label}_1(\mathbf{Split}(\mathbf{nth}_j(T_2))))]$ .

We now select  $n_2 = j' + \sum_{i=0}^{j'-1} |\mathbf{label}_1(\mathbf{Split}(\mathbf{nth}_i(T_2)))|$ .

This means that  $\langle l \rangle x_1 \langle /l \rangle x_2 \in \mathbf{in}[\mathbf{nth}_{n_2}(\mathbf{SPLIT}(T_2))]$ , and

$\sum_{i=0}^{j'-1} |\mathbf{Split}(\mathbf{nth}_i(T_2))| \leq \mathbf{unlabel}(n_2, l, \mathbf{SPLIT}(T_2)) < \sum_{i=0}^j |\mathbf{Split}(\mathbf{nth}_i(T_2))|$ .

The form of  $\mathbf{label}_1(\mathbf{SPLIT}(T_2))$  yields that

$x_1 \in \mathbf{in}[\mathbf{nth}_{n_2}(\Pi_1(\mathbf{label}_1(\mathbf{SPLIT}(T_2))))]$  and

$x_2 \in \mathbf{in}[\mathbf{nth}_{n_2}(\Pi_2(\mathbf{label}_1(\mathbf{SPLIT}(T_2))))]$ .

Since  $\mathbf{SIZE}((0, d_{11})) < \mathbf{SIZE}(D_1)$  and  $\mathbf{SIZE}(0, d_{12}) < \mathbf{SIZE}(D_1)$  the induction hypothesis yields that

$\forall D'_1. \mathbf{SIZE}(D'_1) \leq \mathbf{SIZE}((0, d_{11})) \Rightarrow \forall j''. \mathbf{TAG}(T_1, D'_1) \in \mathbf{in}[\mathbf{nth}_{j''}(T_2)]. \{(n'_2, d'_2) \in \mathbf{C}[\Gamma \vdash T_1 <: T_2](D'_1) \mid n'_2 = j''\} \neq \{\}$ , and therefore the induction hypothesis yields that there is at least one

$(n_2, d_{21}) \in \mathbf{C}[\Gamma \cup \{(T_1, T_2)\} \vdash [t_{11}] <: \Pi_1(\mathbf{label}_1(\mathbf{SPLIT}(T_2)))](0, d_{11})$ , and one

$(n_2, d_{22}) \in \mathbf{C}[\Gamma \cup \{(T_1, T_2)\} \vdash [t_{12}] <: \Pi_2(\mathbf{label}_1(\mathbf{SPLIT}(T_2)))](0, d_{12})$ .

Lemma 14.10 yields that

$TAG(\Pi_1(\text{label}_1(\text{SPLIT}(T_2))), (n_2, d_{21})) = x_1$  and

$TAG(\Pi_2(\text{label}_1(\text{SPLIT}(T_2))), (n_2, d_{22})) = x_2$ .

Because of the form of  $\text{label}_1(\text{SPLIT}(T_2))$  we get that

$TAG(\text{label}_1(\text{SPLIT}(T_2)), (n_2, \langle d_{21} \rangle d_{22})) = \langle 1 \rangle x_1 \langle /1 \rangle x_2$ , and therefore

$TAG(\text{SPLIT}(T_2), (\text{unlabel}(n_2, l, \text{SPLIT}(T_2)), \langle d_{21} \rangle d_{22})) = \langle 1 \rangle x_1 \langle /1 \rangle x_2$ .

Now Lemma 10.22 yields that  $\{(n'_2, d_2) \in \text{TRANSINV}(T_2, (n_2, \langle d_{21} \rangle d_{22})) \mid n'_2 = j\} \neq \{\}$ .

Therefore the lemma is fulfilled in this case.

We have now proved all the cases, and we can therefore conclude that

$$\begin{aligned} \forall D_1 \quad & (\forall (T'_1, T'_2) \in \Gamma. \forall D'_1. \text{SIZE}(D'_1) \leq \text{SIZE}(D_1) \Rightarrow \forall j. \text{TAG}(T'_1, D'_1) \in \text{in}[\text{nth}_j(T'_2)]) \\ & \Rightarrow \{(n'_2, d'_2) \in \mathcal{C}_{T'_1, T'_2}(D'_1) \mid n'_2 = j\} \neq \{\}) \\ \Rightarrow \quad & \forall j. \text{TAG}(T_1, D_1) \in \text{IN}[\text{nth}_j(T_2)] \Rightarrow \{(n_2, d_2) \in \mathcal{C}[\Gamma \vdash T_1 \prec T_2](D_1) \mid n_2 = j\} \neq \{\}. \end{aligned}$$

■

## 19.2 Program Code

```
load "Int";
(* The signature of a set *)
signature SetSig =
sig
  type element
  type set
  val empty : set
  val member : element -> set -> bool
  val subset : set -> set -> bool
  val equal : set * set -> bool
  val insert : element * set -> set
  val singleton: element -> set
  val remove: element -> set -> set
  val cut : set -> set -> set
  val union : set -> set -> set
  val powerset : set -> set list
  val toList : set -> element list
  val fromList : element list -> set
  val toString : set -> string
end
(* A functor for setstructure generation *)
signature EQ =
sig
  type element
  val eq : element * element -> bool
  val elmToString : element -> string
end
functor SetFct(s : EQ) :> SetSig where type element = s.element =
struct
  type element = s.element
  type set = element list
  val empty = []
  fun member a = List.exists (fn y => s.eq(a,y))
  fun subset xs ys = List.all (fn x => member x ys) xs
  fun equal (s1,s2) = subset s1 s2 andalso subset s2 s1
  fun insert (a,s1) = if member a s1
                      then s1
                      else a::s1
end
```

```

fun singleton a = insert(a,empty)
fun remove a = List.filter (fn b => not (s.eq(a,b)))
val cut = foldl (fn (a,b) => remove a b)
val union = foldl insert
fun powerset [] = [[]]
  | powerset (x::xs) =
    let val yss = powerset xs
        in [x] :: yss @ (map (fn ys => insert (x, ys)) yss)
        end
val fromList = foldl insert empty
fun toList s1 = s1
fun toString xs = "{ "
  ^ (foldl (fn (x,b) => s.elmToString x ^ ", " ^ b) "" xs)
  ^ " }"
end
(* Defining sets of integers *)
structure EqInt : EQ =
struct
  type element = int
  val eq = op =
  val elmToString = Int.toString
end
structure SetInt = SetFct(EqInt)
(* Defining xml-types *)
datatype xmltype = XML0
  | XML1
  | XMLTree of string * xmltype
  | XMLSeq of xmltype * xmltype
  | XMLSum of xmltype * xmltype
  | XMLMu of int * xmltype
  | XMLVar of int

(* Pretty printing of xml-types *)
fun xmltypeToString XML0 = "0"
  | xmltypeToString XML1 = "1"
  | xmltypeToString (XMLTree (l,t1)) = l
  ^ "<"
  ^ (xmltypeToString t1)
  ^ ">"
  | xmltypeToString (XMLSeq (a,b)) = "("
  ^ (xmltypeToString a)
  ^ (xmltypeToString b)
  ^ ")"
  | xmltypeToString (XMLSum (a,b)) = "("
  ^ (xmltypeToString a)
  ^ " + "
  ^ (xmltypeToString b)
  ^ ")"
  | xmltypeToString (XMLMu (x,a)) = "("
  ^ "MU "
  ^ "x_{" ^ (Int.toString x) ^ "}"
  ^ ". "
  ^ (xmltypeToString a)
  ^ ")"
  | xmltypeToString (XMLVar x) = "x_{" ^ (Int.toString x) ^ "}"

fun newvar xs = (foldl Int.max 0 xs)+1
(* Define free variables *)
fun fv XML0 = SetInt.empty
  | fv XML1 = SetInt.empty
  | fv (XMLVar x) = SetInt.singleton x
  | fv (XMLTree (l,t1)) = fv(t1)
  | fv (XMLSeq (t1,t2)) = SetInt.union (fv t1) (fv t2)
  | fv (XMLSum (t1,t2)) = SetInt.union (fv t1) (fv t2)
  | fv (XMLMu (x,t1)) = SetInt.remove x (fv t1)
(* Define capture avoiding substitution for xml-types *)

```

```

fun subst XML0 _ = XML0
  | subst XML1 _ = XML1
  | subst (XMLVar y) (t',x) =
    if x = y
    then t'
    else XMLVar y
  | subst (XMLTree (l,t1)) S = XMLTree (l,subst t1 S)
  | subst (XMLSeq (t1,t2)) S = XMLSeq (subst t1 S,subst t2 S)
  | subst (XMLSum (t1,t2)) S = XMLSum (subst t1 S,subst t2 S)
  | subst (XMLMu (y,t1)) (t',x) =
    let val z = newvar(x::(SetInt.toList(SetInt.union (fv t') (fv(XMLMu (y,t1))))))
    in let val t1' = subst t1 (XMLVar z,y)
      in let val t1'' = subst t1' (t',x)
        in XMLMu (z,t1'') end
      end
    end
  end
end
(* Execute a sequence of substitutions *)
val SUBST = foldl (fn (S,t) => subst t S)

fun equiv (XML0, XML0) = true
  | equiv (XML1, XML1) = true
  | equiv ((XMLTree (l1,t1)), (XMLTree (l2,t2))) = l1 = l2 andalso (equiv (t1, t2))
  | equiv ((XMLSeq (t11,t12)), (XMLSeq (t21,t22))) =
    (equiv (t11, t21)) andalso (equiv (t12, t22))
  | equiv ((XMLSum (t11,t12)), (XMLSum (t21,t22))) =
    (equiv (t11, t21)) andalso (equiv (t12, t22))
  | equiv ((XMLMu (x,t1)), (XMLMu (y,t2))) =
    let val z = newvar (SetInt.toList(SetInt.union (fv t1) (fv t2)))
    in equiv (subst t1 (XMLVar z,x), subst t2 (XMLVar z,y))
    end
  | equiv (XMLVar x, XMLVar y) = x = y
  | equiv (_, _) = false

(* Defining sets of xml-types *)
structure EqXmltype : EQ =
struct
  type element = xmltype
  val eq = equiv
  val elmToString = xmltypeToString
end
structure SetXmltype = SetFct(EqXmltype)

(* Defining environments as a sets of pairs of liss of xml-types *)
structure EqEnv : EQ =
struct
  type element = xmltype list * xmltype list
  fun eq ((A,A'),(B,B')) = SetXmltype.equal (SetXmltype.fromList A ,SetXmltype.fromList B ) andalso
    SetXmltype.equal (SetXmltype.fromList A',SetXmltype.fromList B')
  fun elmToString (A,B) = "("
    ^ (SetXmltype.toString (SetXmltype.fromList A))
    ^ ", "
    ^ (SetXmltype.toString (SetXmltype.fromList B))
    ^ ")"
end
(* Set structure for environments, ie.
  set structure for pairs of lists of xml-types *)
structure SetEnv = SetFct(EqEnv)

(* Defining xml-data *)
datatype xmldata = DVoid
  | DTree of xmldata
  | DSeq of xmldata * xmldata
  | DSum1 of xmldata
  | DSum2 of xmldata

(* Pretty printing of xml-data *)

```

```

fun xmldataToString DVoid = "*"
  | xmldataToString (DTree d) = "<" ^ (xmldataToString d) ^ ">"
  | xmldataToString (DSeq (d1,d2)) = "(" ^ (xmldataToString d1) ^ ")" ^ (xmldataToString d2) ^ ")"
  | xmldataToString (DSum1 d1) = "(" ^ (xmldataToString d1) ^ "-)"
  | xmldataToString (DSum2 d2) = "-(" ^ (xmldataToString d2) ^ ")"

(* Defining sets of xml-data *)
structure EqData : EQ =
struct
  type element = xmldata
  val eq = op =
  val elmToString = xmldataToString
end
structure SetData = SetFct(EqData)

fun allpairs ds1 ds2 = foldl (fn (d1,ds) => (map (fn d2 => DSeq (d1,d2)) ds2) @ ds) [] ds1

(* The Empty Word Property *)
fun esp XML0 = SetData.empty
  | esp XML1 = SetData.singleton DVoid
  | esp (XMLTree (l,t1)) = SetData.empty
  | esp (XMLSeq (t1,t2)) =
  SetData.fromList (allpairs (SetData.toList (esp t1)) (SetData.toList (esp t2)))
  | esp (XMLSum (t1,t2)) =
  let val ds1 = esp t1
  in if SetData.subset ds1 SetData.empty
  then SetData.fromList (map (fn d => DSum2 d) (SetData.toList (esp t2)))
  else SetData.fromList (map (fn d => DSum1 d) (SetData.toList ds1))
  end
(* SetData.union (SetData.fromList (map (fn d => DSum1 d) (SetData.toList (esp t1))))
  (SetData.fromList (map (fn d => DSum2 d) (SetData.toList (esp t2)))) *)
  | esp (XMLMu (X,t1)) = esp t1
  | esp (XMLVar X) = SetData.empty

(* Defining sets of xml-Datas *)
structure EqDATA : EQ =
struct
  type element = int * xmldata
  val eq = op =
  val elmToString = (fn (n,d) => "(" ^ (Int.toString n) ^ "," ^ (xmldataToString d) ^ ")")
end
structure SetDATA = SetFct(EqDATA)

fun ESP [] = SetDATA.empty
  | ESP (t1::T1) = SetDATA.fromList ((map (fn d => (0,d)) (SetDATA.toList (esp t1)))
  @ (map (fn (n,d) => (n+1,d)) (SetDATA.toList (ESP T1))))

fun ziptype t2 [] = []
  | ziptype t2 ((XMLSeq (XMLTree (l,t11),t12))::T1) =
  (XMLSeq (XMLTree (l,t11), (XMLSeq (t12,t2)))) :: (ziptype t2 T1)
  | ziptype _ _ =
  raise Fail "ziptype: List of xml-types not wellformed"

(* Define type splitting *)
fun Split' phi XML0 = []
  | Split' phi XML1 = []
  | Split' phi (XMLVar x) = []
  | Split' phi (XMLTree (l,t1)) = [XMLSeq (XMLTree (l,SUBST t1 phi),XML1)]
  | Split' phi (XMLSeq (t1,t2)) =
  if not (SetData.subset (esp t1) SetData.empty)
  then (ziptype (SUBST t2 phi) (Split' phi t1)) @ (Split' phi t2)
  else (ziptype (SUBST t2 phi) (Split' phi t1))
  | Split' phi (XMLSum (t1,t2)) = (Split' phi t1) @ (Split' phi t2)
  | Split' phi (XMLMu (x,t1)) = Split' ((XMLMu (x,t1),x) :: phi) t1
val Split = Split' []
val SPLIT = fn T => List.concat (map Split T)

```

```

(* Define projections *)
fun PROJ0 [] = []
  | PROJ0 ((XMLSeq (XMLTree (l,t1),t2))::T1) = l::PROJ0 T1
  | PROJ0 _ = raise Fail "PROJ0: List is not wellformed"
fun PROJ1 [] = []
  | PROJ1 ((XMLSeq (XMLTree (l,t1),t2))::T1) = t1::PROJ1 T1
  | PROJ1 _ = raise Fail "PROJ1: List is not wellformed"
fun PROJ2 [] = []
  | PROJ2 ((XMLSeq (XMLTree (l,t1),t2))::T1) = t2::PROJ2 T1
  | PROJ2 _ = raise Fail "PROJ2: List is not wellformed"
(* Define filter functions *)
fun filter S = List.filter (fn t => SetXmltype.member t S)
fun label l [] = []
  | label l ((XMLSeq (XMLTree (l',t1),t2))::T1) =
    if l = l'
    then (XMLSeq (XMLTree (l',t1),t2))::(label l T1)
    else label l T1
  | label l _ = raise Fail "lfilter: List is not wellformed"
(* The algorithm *)
exception NotEq of string
datatype proof = SubHyp | SubDiff of (SetEnv.set * xmltype list * xmltype list * proof) list
fun XmlSub G T1 T2 =
  if SetEnv.member (T1,T2) G
  then SubHyp
  else if not (SetDATA.subset (ESP T1) SetDATA.empty) andalso
    (SetDATA.subset (ESP T2) SetDATA.empty)
    then raise NotEq "."
    else let val G' = SetEnv.insert ((T1,T2), G)
          val ST1 = SPLIT(T1)
          val ST2 = SPLIT(T2)
        in
          let val Ss = SetXmltype.powerset (SetXmltype.fromList ST2)
            in SubDiff (ProofListList G' ST1 ST2 Ss)
          end
        end
end
and ProofListList G [] ST2 Ss = []
  | ProofListList G (st1::ST1) ST2 Ss = (ProofList G st1 ST2 Ss) @ (ProofListList G ST1 ST2 Ss)
and ProofList G st1 ST2 [] = []
  | ProofList G st1 ST2 (S::Ss) = (Proof G st1 ST2 S) :: (ProofList G st1 ST2 Ss)
and Proof G (XMLSeq (XMLTree (l,t11),t12)) ST2 S =
  ((G,[t11],PROJ1(filter S (label l ST2))),
   XmlSub G [t11] (PROJ1 (filter S (label l ST2)))) handle NotEq x1 =>
  (G,[t12],PROJ2(filter (SetXmltype.union (SetXmltype.fromList ST2) S) (label l ST2)),
   XmlSub G [t12] (PROJ2 (filter (SetXmltype.cut (SetXmltype.fromList ST2) S) (label l ST2)))) handle NotEq x2 =>
  raise NotEq ("<^l^>" ^ x1 ^ "</" ^ l ^ ">" ^ x2)
  | Proof G st1 ST2 S = raise Fail "Proof: st1 is not wellformed"

datatype xmlsequence = XVoid
  | XTree of string * xmlsequence * xmlsequence

fun Xappend XVoid x2 = x2
  | Xappend (XTree (l,x11,x12)) x2 = XTree (l,x11,Xappend x12 x2)

(* Define Tag *)
fun Tag XML1 DVoid = XVoid
  | Tag (XMLTree (l,t1)) (DTree d1) = XTree (l,Tag t1 d1,XVoid)
  | Tag (XMLSeq (t1,t2)) (DSeq (d1,d2)) = Xappend (Tag t1 d1) (Tag t2 d2)
  | Tag (XMLSum (t1,t2)) (DSum1 d1) = Tag t1 d1
  | Tag (XMLSum (t1,t2)) (DSum2 d2) = Tag t2 d2
  | Tag (XMLMu (n,t1)) d = Tag (subst t1 (XMLMu (n,t1),n)) d
  | Tag _ _ = raise Fail "Tag: Data doesn't match type"

fun TAG [] D = raise Fail "TAG: Index out of bounds"
  | TAG (t1::T1) (0,d) = Tag t1 d
  | TAG (t1::T1) (n,d) = TAG T1 (n-1,d)

```

```

fun simplify t' d' (XMLSeq (t1,t2)) (DSeq (d1,d2)) =
  if Tag t1 d1 = XVoid
  then simplify t' d' t2 d2
  else d'
| simplify t' d' (XMLSum (t1,t2)) (DSum1 d1) = simplify t' d' t1 d1
| simplify t' d' (XMLSum (t1,t2)) (DSum2 d2) = simplify t' d' t2 d2
| simplify t' d' (XMLMu (n,t1)) d =
  if equiv ((XMLMu (n,t1)), t')
  then simplify (XMLMu (n,t1)) d (subst t1 (XMLMu (n,t1),n)) d
  else simplify t' d' (subst t1 (XMLMu (n,t1),n)) d
| simplify t' d' _ _ = d'

fun Trans phi (XMLTree (l,t)) d = SetDATA.singleton (0,DSeq (d,DVoid))
| Trans phi (XMLSeq (t1,t2)) (DSeq (d1,d2)) =
  if Tag t1 d1 = XVoid handle Fail s => false
  then SetDATA.fromList (map (fn (n,d) => (n + (length (Split' phi t1)),d)) (SetDATA.toList (Trans phi t2 d2)))
  else SetDATA.fromList (map (fn (n,(DSeq (d11,d12))) => (n,DSeq (d11,DSeq (d12,d2))))
    | _ => raise Fail "Trans: Result was not wellformed")
    (SetDATA.toList (Trans phi t1 d1)))
| Trans phi (XMLSum (t1,t2)) (DSum1 d1) = Trans phi t1 d1
| Trans phi (XMLSum (t1,t2)) (DSum2 d2) =
  SetDATA.fromList (map (fn (n,d) => (n + (length (Split' phi t1)),d)) (SetDATA.toList (Trans phi t2 d2)))
| Trans phi (XMLMu (n,t1)) d = Trans ((XMLMu (n,t1),n)::phi) t1
  (simplify (XMLMu (n,t1)) d (SUBST t1 ((XMLMu (n,t1),n)::phi)) d)
| Trans phi _ _ = SetDATA.empty

fun TRANS [] D = SetDATA.empty
| TRANS (t1::T1) (0,d) = Trans [] t1 d
| TRANS (t1::T1) (n,d) = SetDATA.fromList (map (fn (n',d') => (n' + (length (Split t1)),d'))
  (SetDATA.toList (TRANS T1 (n-1,d))))

fun TransInv phi (XMLTree (l,t)) (0,DSeq (d,DVoid)) = SetData.singleton d
| TransInv phi (XMLSeq (t1,t2)) (n,d) =
  if n < (length (Split' phi t1))
  then case d of (DSeq (d11,DSeq (d12,d2))) =>
    SetData.fromList (allpairs (SetData.toList (TransInv phi t1 (n,DSeq(d11,d12)))) [d2])
    | _ => SetData.empty
  else SetData.fromList (allpairs (SetData.toList (esp t1))
    (SetData.toList (TransInv phi t2 (n - (length (Split' phi t1)),d))))
| TransInv phi (XMLSum (t1,t2)) (n,d) =
  if n < (length (Split' phi t1))
  then SetData.fromList (map (fn d => DSum1 d)
    (SetData.toList (TransInv phi t1 (n,d))))
  else SetData.fromList (map (fn d => DSum2 d)
    (SetData.toList (TransInv phi t2 (n - (length (Split' phi t1)),d))))
| TransInv phi (XMLMu (n,t1)) D = TransInv ((XMLMu (n,t1),n)::phi) t1 D
| TransInv phi _ _ = SetData.empty

fun TRANSINV [] _ = SetDATA.empty
| TRANSINV (t1::T1) (n,d) =
  if n < length (Split t1)
  then SetDATA.fromList (map (fn d' => (0,d')) (SetData.toList (TransInv [] t1 (n,d))))
  else SetDATA.fromList (map (fn (n',d') => (n'+1,d'))
    (SetDATA.toList (TRANSINV T1 (n - (length (Split t1)),d))))

fun lookup t [] = raise Fail "lookup: Element not in list"
| lookup t (t1::T1) =
  if equiv (t,t1)
  then 0
  else 1 + (lookup t T1)

fun nth n [] = raise Fail "nth: index out of bounds"
| nth 0 (x::xs) = x
| nth n (x::xs) = nth (n-1) xs

```

```

fun unlabel n l [] = raise Fail "unlabel: Index has no match"
| unlabel n l ((XMLSeq (XMLTree (l',t1),t2)) :: T1) =
  if l=l' andalso n=0
  then 0
  else if l=l'
    then (unlabel (n-1) l T1) + 1
    else (unlabel n l T1) + 1
| unlabel n l (t1::T1) = (unlabel n l T1) + 1

fun fit _ [] = []
| fit (n1,d1) ((n2,d2)::Ds2) =
  if n1 = n2
  then (n1,DSeq (DTree d1,d2)) :: (fit (n1,d1) Ds2)
  else fit (n1,d1) Ds2

fun fitall [] _ = []
| fitall (D1::Ds1) Ds2 = (fit D1 Ds2) @ (fitall Ds1 Ds2)

(* The conversions *)
fun Cgen T1 T2 D1 = if TAG T1 D1 = XVoid
  then ESP T2
  else let val ST1 = SPLIT T1
          val ST2 = SPLIT T2
          val DS1 = SetDATA.toList (TRANS T1 D1)
        in
          case DS1
          of ((n1',DSeq (DTree d11,d12))::_) =>
            let val (XMLSeq (XMLTree (l,t11),t12)) = nth n1' ST1
              in
                let val D21 = Cgen [t11] (PROJ1 (label l ST2)) (0,d11)
                    val D22 = Cgen [t12] (PROJ2 (label l ST2)) (0,d12)
                  in
                    SetDATA.fromList
                      (List.concat
                        (map
                          (fn (n2',d2') => SetDATA.toList (TRANSINV T2 (unlabel n2' l ST2,d2'))
                            (fitall (SetDATA.toList D21) (SetDATA.toList D22))))
                      end
                    end
                  | _ => SetDATA.empty
                end
              handle Fail s => SetDATA.empty

(* Defining constructors and equality-operator *)
infix 6 ++
fun T1 ++ T2 = XMLSum (T1, T2)
infix 7 ^^
fun T1 ^^ T2 = XMLSeq (T1, T2)
infix 4 <:
fun t1 <: t2 = let
  val x = XmlSub SetEnv.empty [t1] [t2]
  in
    true
  end
  handle NotEq w => false

(* Defining the xml-types*)
val t_gender = XMLTree ("gender",XMLTree ("male",XML1) ++ XMLTree ("female",XML1))
val t_count = XMLMu (0,XML1 ++ XMLSeq (XMLTree ("s",XML1),XMLVar 0))
val t_age = XMLTree ("age",t_count)
val t_info = XMLSeq(t_gender,t_age)
val t1 = XMLMu (0,XML1 ++ XMLTree ("l",XMLVar 0) ^^ XMLVar 0)
val t2 = XMLMu (0,XML1 ++ XMLTree ("l",XML1) ^^ XMLVar 0)
val t3 = XMLMu (0,XML1 ++ XMLTree ("l",XMLVar 0))
val d2 = DSum2 (DSeq(DTree (DVoid), DSum2 (DSeq(DTree (DVoid), DSum1 DVoid))))
val d3 = DSum2 (DTree (DSum2 (DTree (DSum1 DVoid))))

```

```

(* Test execution *)
(* Proof search test *)
val test0 = (t1 <: t1) = true
val test1 = (t2 <: t2) = true
val test2 = (t3 <: t3) = true
val test3 = (t2 <: t1) = true
val test4 = (t3 <: t1) = true
val test5 = (t1 <: t2) = false (* error = "<1><1>.</1>.</1>." *)
val test6 = (t1 <: t3) = false (* error = "<1>.</1><1>.</1>." *)
(* TAG *)
val test7 = (TAG [t3] (0,d3)) = (XTree ("1",XTree ("1",XVoid,XVoid),XVoid))
val test8 = (TAG [t2] (0,d2)) = (XTree ("1",XVoid,XTree ("1",XVoid,XVoid)))
(* Conversion of xml-data *)
val test9 = length (SetDATA.toList (Cgen [t3] [t1] (0,d3))) = 1
val test10 = (TAG [t1] (nth 0 (SetDATA.toList (Cgen [t3] [t1] (0,d3)))))) = (XTree ("1", XTree ("1",XVoid,XVoid),XVoid))
val test11 = length (SetDATA.toList (Cgen [t2] [t1] (0,d2))) = 1
val test12 = (TAG [t1] (nth 0 (SetDATA.toList (Cgen [t2] [t1] (0,d2)))))) = (XTree ("1",XVoid,XTree ("1",XVoid,XVoid)))

;quit();

```

## 19.3 Output

This is the output from interpreting the code from Appendix 19.2 with [MOSML].

```

Moscow ML version 2.01 (January 2004)
Enter 'quit();' to quit.
[opening file "xmlsubtyping.sml"]
> val it = () : unit
File "xmlsubtyping.sml", line 397, characters 40-70:
!                               let val (XMLSeq (XMLTree (l,t11),t12)) = nth n1' ST1
!                               ~~~~~
! Warning: pattern matching is not exhaustive

> New type names: set, =xmltype, set/1, set/2, =xmldata, set/3, set/4, proof,
=xmlsequence
infix 6 ++
infix 7 ^^
infix 4 <:
structure EqInt :
{type element = int,
 val elmToString : int -> string,
 val eq : int * int -> bool}
structure SetInt :
{type element = int,
 type set = set,
 val cut : set -> set -> set,
 val empty : set,
 val equal : set * set -> bool,
 val fromList : int list -> set,
 val insert : int * set -> set,
 val member : int -> set -> bool,
 val powerset : set -> set list,
 val remove : int -> set -> set,
 val singleton : int -> set,
 val subset : set -> set -> bool,
 val toList : set -> int list,
 val toString : set -> string,
 val union : set -> set -> set}
structure EqXmltype :
{type element = xmltype,
 val elmToString : xmltype -> string,
 val eq : xmltype * xmltype -> bool}
structure SetXmltype :
{type element = xmltype,

```

```

type set = set/1,
val cut : set/1 -> set/1 -> set/1,
val empty : set/1,
val equal : set/1 * set/1 -> bool,
val fromList : xmltype list -> set/1,
val insert : xmltype * set/1 -> set/1,
val member : xmltype -> set/1 -> bool,
val powerset : set/1 -> set/1 list,
val remove : xmltype -> set/1 -> set/1,
val singleton : xmltype -> set/1,
val subset : set/1 -> set/1 -> bool,
val toList : set/1 -> xmltype list,
val toString : set/1 -> string,
val union : set/1 -> set/1 -> set/1}
structure EqEnv :
{type element = xmltype list * xmltype list,
 val elmToString : xmltype list * xmltype list -> string,
 val eq :
  (xmltype list * xmltype list) * (xmltype list * xmltype list) -> bool}
structure SetEnv :
{type element = xmltype list * xmltype list,
 type set = set/2,
 val cut : set/2 -> set/2 -> set/2,
 val empty : set/2,
 val equal : set/2 * set/2 -> bool,
 val fromList : (xmltype list * xmltype list) list -> set/2,
 val insert : (xmltype list * xmltype list) * set/2 -> set/2,
 val member : xmltype list * xmltype list -> set/2 -> bool,
 val powerset : set/2 -> set/2 list,
 val remove : xmltype list * xmltype list -> set/2 -> set/2,
 val singleton : xmltype list * xmltype list -> set/2,
 val subset : set/2 -> set/2 -> bool,
 val toList : set/2 -> (xmltype list * xmltype list) list,
 val toString : set/2 -> string,
 val union : set/2 -> set/2 -> set/2}
structure EqData :
{type element = xmldata,
 val elmToString : xmldata -> string,
 val eq : xmldata * xmldata -> bool}
structure SetData :
{type element = xmldata,
 type set = set/3,
 val cut : set/3 -> set/3 -> set/3,
 val empty : set/3,
 val equal : set/3 * set/3 -> bool,
 val fromList : xmldata list -> set/3,
 val insert : xmldata * set/3 -> set/3,
 val member : xmldata -> set/3 -> bool,
 val powerset : set/3 -> set/3 list,
 val remove : xmldata -> set/3 -> set/3,
 val singleton : xmldata -> set/3,
 val subset : set/3 -> set/3 -> bool,
 val toList : set/3 -> xmldata list,
 val toString : set/3 -> string,
 val union : set/3 -> set/3 -> set/3}
structure EqDATA :
{type element = int * xmldata,
 val elmToString : int * xmldata -> string,
 val eq : (int * xmldata) * (int * xmldata) -> bool}
structure SetDATA :
{type element = int * xmldata,
 type set = set/4,
 val cut : set/4 -> set/4 -> set/4,
 val empty : set/4,
 val equal : set/4 * set/4 -> bool,
 val fromList : (int * xmldata) list -> set/4,

```

```

val insert : (int * xmldata) * set/4 -> set/4,
val member : int * xmldata -> set/4 -> bool,
val powerset : set/4 -> set/4 list,
val remove : int * xmldata -> set/4 -> set/4,
val singleton : int * xmldata -> set/4,
val subset : set/4 -> set/4 -> bool,
val toList : set/4 -> (int * xmldata) list,
val toString : set/4 -> string,
val union : set/4 -> set/4 -> set/4}
functor SetFct :
!element.
{type element = element,
 val eq : element * element -> bool,
 val elmToString : element -> string}->
?set/5.
{type element = element,
 type set = set/5,
 val cut : set/5 -> set/5 -> set/5,
 val empty : set/5,
 val equal : set/5 * set/5 -> bool,
 val fromList : element list -> set/5,
 val insert : element * set/5 -> set/5,
 val member : element -> set/5 -> bool,
 val powerset : set/5 -> set/5 list,
 val remove : element -> set/5 -> set/5,
 val singleton : element -> set/5,
 val subset : set/5 -> set/5 -> bool,
 val toList : set/5 -> element list,
 val toString : set/5 -> string,
 val union : set/5 -> set/5 -> set/5}
signature SetSig =
/\element set/5.
{type element = element,
 type set = set/5,
 val empty : set/5,
 val member : element -> set/5 -> bool,
 val subset : set/5 -> set/5 -> bool,
 val equal : set/5 * set/5 -> bool,
 val insert : element * set/5 -> set/5,
 val singleton : element -> set/5,
 val remove : element -> set/5 -> set/5,
 val cut : set/5 -> set/5 -> set/5,
 val union : set/5 -> set/5 -> set/5,
 val powerset : set/5 -> set/5 list,
 val toList : set/5 -> element list,
 val fromList : element list -> set/5,
 val toString : set/5 -> string}
signature EQ =
/\element.
{type element = element,
 val eq : element * element -> bool,
 val elmToString : element -> string}
datatype xmltype =
(xmltype,
{con XML0 : xmltype,
 con XML1 : xmltype,
 con XMLMu : int * xmltype -> xmltype,
 con XMLSeq : xmltype * xmltype -> xmltype,
 con XMLSum : xmltype * xmltype -> xmltype,
 con XMLTree : string * xmltype -> xmltype,
 con XMLVar : int -> xmltype})}
datatype xmldata =
(xmldata,
{con DSeq : xmldata * xmldata -> xmldata,
 con DSum1 : xmldata -> xmldata,
 con DSum2 : xmldata -> xmldata,

```

```

    con DTree : xmldata -> xmldata,
    con DVoid : xmldata})
datatype proof =
(proof,
 {con SubDiff : (set/2 * xmlype list * xmlype list * proof) list -> proof,
  con SubHyp : proof})
datatype xmlsequence =
(xmlsequence,
 {con XTree : string * xmlsequence * xmlsequence -> xmlsequence,
  con XVoid : xmlsequence})
con XML0 = XML0 : xmlype
con XML1 = XML1 : xmlype
con XMLMu = fn : int * xmlype -> xmlype
con XMLSeq = fn : xmlype * xmlype -> xmlype
con XMLSum = fn : xmlype * xmlype -> xmlype
con XMLTree = fn : string * xmlype -> xmlype
con XMLVar = fn : int -> xmlype
val xmlypeToString = fn : xmlype -> string
val newvar = fn : int list -> int
val fv = fn : xmlype -> set
val subst = fn : xmlype -> xmlype * int -> xmlype
val SUBST = fn : xmlype -> (xmlype * int) list -> xmlype
val equiv = fn : xmlype * xmlype -> bool
con DSeq = fn : xmldata * xmldata -> xmldata
con DSum1 = fn : xmldata -> xmldata
con DSum2 = fn : xmldata -> xmldata
con DTree = fn : xmldata -> xmldata
con DVoid = DVoid : xmldata
val xmldataToString = fn : xmldata -> string
val allpairs = fn : xmldata list -> xmldata list -> xmldata list
val esp = fn : xmlype -> set/3
val ESP = fn : xmlype list -> set/4
val ziptype = fn : xmlype -> xmlype list -> xmlype list
val Split' = fn : (xmlype * int) list -> xmlype -> xmlype list
val Split = fn : xmlype -> xmlype list
val SPLIT = fn : xmlype list -> xmlype list
val PROJ0 = fn : xmlype list -> string list
val PROJ1 = fn : xmlype list -> xmlype list
val PROJ2 = fn : xmlype list -> xmlype list
val filter = fn : set/1 -> xmlype list -> xmlype list
val label = fn : string -> xmlype list -> xmlype list
exn NotEq = fn : string -> exn
con SubDiff = fn :
(set/2 * xmlype list * xmlype list * proof) list -> proof
con SubHyp = SubHyp : proof
val XmlSub = fn : set/2 -> xmlype list -> xmlype list -> proof
val ProofListList = fn :
set/2 -> xmlype list -> xmlype list -> set/1 list ->
(set/2 * xmlype list * xmlype list * proof) list
val ProofList = fn :
set/2 -> xmlype -> xmlype list -> set/1 list ->
(set/2 * xmlype list * xmlype list * proof) list
val Proof = fn :
set/2 -> xmlype -> xmlype list -> set/1 ->
set/2 * xmlype list * xmlype list * proof
con XTree = fn : string * xmlsequence * xmlsequence -> xmlsequence
con XVoid = XVoid : xmlsequence
val Xappend = fn : xmlsequence -> xmlsequence -> xmlsequence
val Tag = fn : xmlype -> xmldata -> xmlsequence
val TAG = fn : xmlype list -> int * xmldata -> xmlsequence
val simplify = fn : xmlype -> xmldata -> xmlype -> xmldata -> xmldata
val Trans = fn : (xmlype * int) list -> xmlype -> xmldata -> set/4
val TRANS = fn : xmlype list -> int * xmldata -> set/4
val TransInv = fn : (xmlype * int) list -> xmlype -> int * xmldata -> set/3
val TRANSINV = fn : xmlype list -> int * xmldata -> set/4
val lookup = fn : xmlype -> xmlype list -> int

```

```

val 'a nth = fn : int -> 'a list -> 'a
val unlabel = fn : int -> string -> xmltype list -> int
val ''a fit = fn :
''a * xmldata -> (''a * xmldata) list -> (''a * xmldata) list
val ''a fitall = fn :
(''a * xmldata) list -> (''a * xmldata) list -> (''a * xmldata) list
val Cgen = fn : xmltype list -> xmltype list -> int * xmldata -> set/4
val ++ = fn : xmltype * xmltype -> xmltype
val ^^ = fn : xmltype * xmltype -> xmltype
val <: = fn : xmltype * xmltype -> bool
val t_gender =
  XMLTree("gender", XMLSum(XMLTree("male", XML1), XMLTree("female", XML1))) :
xmltype
val t_count = XMLMu(0, XMLSum(XML1, XMLSeq(XMLTree("s", XML1), XMLVar 0))) :
xmltype
val t_age =
  XMLTree("age",
    XMLMu(0, XMLSum(XML1, XMLSeq(XMLTree("s", XML1), XMLVar 0)))) :
xmltype
val t_info =
  XMLSeq(XMLTree("gender",
    XMLSum(XMLTree("male", XML1), XMLTree("female", XML1))),
    XMLTree("age",
      XMLMu(0,
        XMLSum(XML1, XMLSeq(XMLTree("s", XML1), XMLVar 0)))))
  : xmltype
val t1 = XMLMu(0, XMLSum(XML1, XMLSeq(XMLTree("1", XMLVar 0), XMLVar 0))) :
xmltype
val t2 = XMLMu(0, XMLSum(XML1, XMLSeq(XMLTree("1", XML1), XMLVar 0))) :
xmltype
val t3 = XMLMu(0, XMLSum(XML1, XMLTree("1", XMLVar 0))) : xmltype
val d2 = DSum2(DSeq(DTree DVoid, DSum2(DSeq(DTree DVoid, DSum1 DVoid)))) :
xmldata
val d3 = DSum2(DTree(DSum2(DTree(DSum1 DVoid)))) : xmldata
val test0 = true : bool
val test1 = true : bool
val test2 = true : bool
val test3 = true : bool
val test4 = true : bool
val test5 = true : bool
val test6 = true : bool
val test7 = true : bool
val test8 = true : bool
val test9 = true : bool
val test10 = true : bool
val test11 = true : bool
val test12 = true : bool

```

## 19.4 Xml Data

The original xml-sequence and the serialized xml-data for the xml-type

$$t = \text{catalog} \langle \mu X.1 + \text{cd} \langle \text{title} \langle \text{string} \rangle \text{artist} \langle \text{string} \rangle \text{country} \langle \text{string} \rangle \text{price} \langle \text{float} \rangle \text{year} \langle \text{int} \rangle \rangle X \rangle$$

are included below.

The xml-data uses + for • and - for ◊.

xml-sequence:	xml-data:
<CATALOG>	<
<CD>	<-
<TITLE>Empire Burlesque</TITLE>	<Empire Burlesque>
<ARTIST>Bob Dylan</ARTIST>	<Bob Dylan>
<COUNTRY>USA</COUNTRY>	<USA>

<COMPANY>Columbia</COMPANY>	<Columbia>
<PRICE>10.90</PRICE>	<10.90>
<YEAR>1985</YEAR>	<1985>
</CD>	>
<CD>	<-
<TITLE>Hide your heart</TITLE>	<Hide your heart>
<ARTIST>Bonnie Tyler</ARTIST>	<Bonnie Tyler>
<COUNTRY>UK</COUNTRY>	<UK>
<COMPANY>CBS Records</COMPANY>	<CBS Records>
<PRICE>9.90</PRICE>	<9.90>
<YEAR>1988</YEAR>	<1988>
</CD>	>
<CD>	<-
<TITLE>Greatest Hits</TITLE>	<Greatest Hits>
<ARTIST>Dolly Parton</ARTIST>	<Dolly Parton>
<COUNTRY>USA</COUNTRY>	<USA>
<COMPANY>RCA</COMPANY>	<RCA>
<PRICE>9.90</PRICE>	<9.90>
<YEAR>1982</YEAR>	<1982>
</CD>	>
<CD>	<-
<TITLE>Still got the blues</TITLE>	<Still got the blues>
<ARTIST>Gary Moore</ARTIST>	<Gary Moore>
<COUNTRY>UK</COUNTRY>	<UK>
<COMPANY>Virgin records</COMPANY>	<Virgin records>
<PRICE>10.20</PRICE>	<10.20>
<YEAR>1990</YEAR>	<1990>
</CD>	>
<CD>	<-
<TITLE>Eros</TITLE>	<Eros>
<ARTIST>Eros Ramazzotti</ARTIST>	<Eros Ramazzotti>
<COUNTRY>EU</COUNTRY>	<EU>
<COMPANY>BMG</COMPANY>	<BMG>
<PRICE>9.90</PRICE>	<9.90>
<YEAR>1997</YEAR>	<1997>
</CD>	>
<CD>	<-
<TITLE>One night only</TITLE>	<One night only>
<ARTIST>Bee Gees</ARTIST>	<Bee Gees>
<COUNTRY>UK</COUNTRY>	<UK>
<COMPANY>Polydor</COMPANY>	<Polydor>
<PRICE>10.90</PRICE>	<10.90>
<YEAR>1998</YEAR>	<1998>
</CD>	>
<CD>	<-
<TITLE>Sylvias Mother</TITLE>	<Sylvias Mother>
<ARTIST>Dr.Hook</ARTIST>	<Dr.Hook>
<COUNTRY>UK</COUNTRY>	<UK>
<COMPANY>CBS</COMPANY>	<CBS>
<PRICE>8.10</PRICE>	<8.10>
<YEAR>1973</YEAR>	<1973>
</CD>	>
<CD>	<-
<TITLE>Maggie May</TITLE>	<Maggie May>
<ARTIST>Rod Stewart</ARTIST>	<Rod Stewart>
<COUNTRY>UK</COUNTRY>	<UK>
<COMPANY>Pickwick</COMPANY>	<Pickwick>
<PRICE>8.50</PRICE>	<8.50>
<YEAR>1990</YEAR>	<1990>
</CD>	>
<CD>	<-
<TITLE>Romanza</TITLE>	<Romanza>
<ARTIST>Andrea Bocelli</ARTIST>	<Andrea Bocelli>
<COUNTRY>EU</COUNTRY>	<EU>
<COMPANY>Polydor</COMPANY>	<Polydor>
<PRICE>10.80</PRICE>	<10.80>

```

<YEAR>1996</YEAR>
</CD>
<CD>
<TITLE>When a man loves a woman</TITLE>
<ARTIST>Percy Sledge</ARTIST>
<COUNTRY>USA</COUNTRY>
<COMPANY>Atlantic</COMPANY>
<PRICE>8.70</PRICE>
<YEAR>1987</YEAR>
</CD>
<CD>
<TITLE>Black angel</TITLE>
<ARTIST>Savage Rose</ARTIST>
<COUNTRY>EU</COUNTRY>
<COMPANY>Mega</COMPANY>
<PRICE>10.90</PRICE>
<YEAR>1995</YEAR>
</CD>
<CD>
<TITLE>1999 Grammy Nominees</TITLE>
<ARTIST>Many</ARTIST>
<COUNTRY>USA</COUNTRY>
<COMPANY>Grammy</COMPANY>
<PRICE>10.20</PRICE>
<YEAR>1999</YEAR>
</CD>
<CD>
<TITLE>For the good times</TITLE>
<ARTIST>Kenny Rogers</ARTIST>
<COUNTRY>UK</COUNTRY>
<COMPANY>Mucik Master</COMPANY>
<PRICE>8.70</PRICE>
<YEAR>1995</YEAR>
</CD>
<CD>
<TITLE>Big Willie style</TITLE>
<ARTIST>Will Smith</ARTIST>
<COUNTRY>USA</COUNTRY>
<COMPANY>Columbia</COMPANY>
<PRICE>9.90</PRICE>
<YEAR>1997</YEAR>
</CD>
<CD>
<TITLE>Tupelo Honey</TITLE>
<ARTIST>Van Morrison</ARTIST>
<COUNTRY>UK</COUNTRY>
<COMPANY>Polydor</COMPANY>
<PRICE>8.20</PRICE>
<YEAR>1971</YEAR>
</CD>
<CD>
<TITLE>Soulsville</TITLE>
<ARTIST>Jorn Hoel</ARTIST>
<COUNTRY>Norway</COUNTRY>
<COMPANY>WEA</COMPANY>
<PRICE>7.90</PRICE>
<YEAR>1996</YEAR>
</CD>
<CD>
<TITLE>The very best of</TITLE>
<ARTIST>Cat Stevens</ARTIST>
<COUNTRY>UK</COUNTRY>
<COMPANY>Island</COMPANY>
<PRICE>8.90</PRICE>
<YEAR>1990</YEAR>
</CD>
<1996>
>
-<
<When a man loves a woman>
<Percy Sledge>
<USA>
<Atlantic>
<8.70>
<1987>
>
-<
<Black angel>
<Savage Rose>
<EU>
<Mega>
<10.90>
<1995>
>
-<
<1999 Grammy Nominees>
<Many>
<USA>
<Grammy>
<10.20>
<1999>
>
-<
<For the good times>
<Kenny Rogers>
<UK>
<Mucik Master>
<8.70>
<1995>
>
-<
<Big Willie style>
<Will Smith>
<USA>
<Columbia>
<9.90>
<1997>
>
-<
<Tupelo Honey>
<Van Morrison>
<UK>
<Polydor>
<8.20>
<1971>
>
-<
<Soulsville>
<Jorn Hoel>
<Norway>
<WEA>
<7.90>
<1996>
>
-<
<The very best of>
<Cat Stevens>
<UK>
<Island>
<8.90>
<1990>
>

```

```

<CD>
<TITLE>Stop</TITLE>
<ARTIST>Sam Brown</ARTIST>
<COUNTRY>UK</COUNTRY>
<COMPANY>A and M</COMPANY>
<PRICE>8.90</PRICE>
<YEAR>1988</YEAR>
</CD>
<CD>
<TITLE>Bridge of Spies</TITLE>
<ARTIST>T'Pau</ARTIST>
<COUNTRY>UK</COUNTRY>
<COMPANY>Siren</COMPANY>
<PRICE>7.90</PRICE>
<YEAR>1987</YEAR>
</CD>
<CD>
<TITLE>Private Dancer</TITLE>
<ARTIST>Tina Turner</ARTIST>
<COUNTRY>UK</COUNTRY>
<COMPANY>Capitol</COMPANY>
<PRICE>8.90</PRICE>
<YEAR>1983</YEAR>
</CD>
<CD>
<TITLE>Midt om natten</TITLE>
<ARTIST>Kim Larsen</ARTIST>
<COUNTRY>EU</COUNTRY>
<COMPANY>Medley</COMPANY>
<PRICE>7.80</PRICE>
<YEAR>1983</YEAR>
</CD>
<CD>
<TITLE>Pavarotti Gala Concert</TITLE>
<ARTIST>Luciano Pavarotti</ARTIST>
<COUNTRY>UK</COUNTRY>
<COMPANY>DECCA</COMPANY>
<PRICE>9.90</PRICE>
<YEAR>1991</YEAR>
</CD>
<CD>
<TITLE>The dock of the bay</TITLE>
<ARTIST>Otis Redding</ARTIST>
<COUNTRY>USA</COUNTRY>
<COMPANY>Atlantic</COMPANY>
<PRICE>7.90</PRICE>
<YEAR>1987</YEAR>
</CD>
<CD>
<TITLE>Picture book</TITLE>
<ARTIST>Simply Red</ARTIST>
<COUNTRY>EU</COUNTRY>
<COMPANY>Elektra</COMPANY>
<PRICE>7.20</PRICE>
<YEAR>1985</YEAR>
</CD>
<CD>
<TITLE>Red</TITLE>
<ARTIST>The Communards</ARTIST>
<COUNTRY>UK</COUNTRY>
<COMPANY>London</COMPANY>
<PRICE>7.80</PRICE>
<YEAR>1987</YEAR>
</CD>
<CD>
<TITLE>Unchain my heart</TITLE>

```

```

-<
<Stop>
<Sam Brown>
<UK>
<A and M>
<8.90>
<1988>
>
-<
<Bridge of Spies>
<T'Pau>
<UK>
<Siren>
<7.90>
<1987>
>
-<
<Private Dancer>
<Tina Turner>
<UK>
<Capitol>
<8.90>
<1983>
>
-<
<Midt om natten>
<Kim Larsen>
<EU>
<Medley>
<7.80>
<1983>
>
-<
<Pavarotti Gala Concert>
<Luciano Pavarotti>
<UK>
<DECCA>
<9.90>
<1991>
>
-<
<The dock of the bay>
<Otis Redding>
<USA>
<Atlantic>
<7.90>
<1987>
>
-<
<Picture book>
<Simply Red>
<EU>
<Elektra>
<7.20>
<1985>
>
-<
<Red>
<The Communards>
<UK>
<London>
<7.80>
<1987>
>
-<
<Unchain my heart>

```

```
<ARTIST>Joe Cocker</ARTIST>  
<COUNTRY>USA</COUNTRY>  
<COMPANY>EMI</COMPANY>  
<PRICE>8.20</PRICE>  
<YEAR>1987</YEAR>  
</CD>  
</CATALOG>
```

```
<Joe Cocker>  
<USA>  
<EMI>  
<8.20>  
<1987>  
>  
+- >
```