A Type System for the Safe Instantiation of Components [1]

Marc Bezem [2] & Hoang Truong [3]

Institute for Informatics
University of Bergen
PB.7800, 5020 Bergen, Norway

Abstract

Component composition can lead to multiple instances of the same component. Some components can have only one instance loaded at a time, for example, when a unique external resource is used. We give an abstract component language and a type system ensuring the safe instantiation of components. Language features are instantiation, composition and a simple scope mechanism for discharging instances.

1 Introduction

Imagine a computer program composed from several components. These components, possibly purchased from different vendors, may use other components, which on their turn use other components, and so on. In order to analyze this process, the phrase 'to use a component' is somewhat too loose and we prefer to speak of 'to create an instance of a component', usually denoted by the primitive `new` c, where c denotes the component in question. The semantics of `new` c is, roughly, the allocation of the resources to run an instance of c. This does not only mean allocating memory space for c's data structures and the like, but this also means creating instances of the components used by c. (The exact moment on which instances of subcomponents are created depends on the binding regime, see [15].)

In the situation sketched above it can easily happen that, unforseen by the composer, different instances of the same component are created. For many components this is no problem at all. However, there exist components which do not allow multiple instances running side-by-side [4,10], for example, in case the component uses a unique external resource like a printer or serial

[1] This research is supported by the Research Council of Norway (NFR).

[2] Email: `bezem@ii.uib.no`

[3] Email: `hoang@ii.uib.no`

output device or in case the component is a centralized function such as issuing serialized ID numbers [7]. In [10], Meijer points out that the only option for all but the first request to multiple versions of a component that cannot exist side-by-side, is to fail. The single instance property of component can be controlled by the component implementation itself using the singleton pattern [7]. However, in component software, the context in which a component will be used should not be anticipated completely, therefore there are cases we need to control the number of instances of a particular component on a higher level[5]. The aim of this paper is to develop a type system which allows one to detect *statically*, at development/composition time, whether or not multiple instances of certain components are running side-by-side.

For this purpose we have designed a rudimentary component language where we have abstracted away many aspects of components. The main features we have retained are instantiation, (sequential) composition and scope. On this abstraction level there is little difference between components and classes on one hand, and instances and objects on the other. Please keep in mind that we report on ongoing research here: more sophisticated language features will be included in the near future.

The simple binding mechanism for components used here bears similarity to `let` binding in functional languages such as ML [11], and hence to lambda abstraction and application. However, the types used here are completely different. To some extent it turned out to be possible to develop our type theory along the lines of so-called Pure Type Systems (PTSs), see [3]. This increased our confidence in the abstractions chosen and can be viewed as a tribute to the generality of PTSs.

There are several works on type systems for components, see e.g. [14,16]. However, they do not address the issue of single instantiation of components.

The intuition behind our types bears some similarity to so-called linear types [2,6]. Linear types usually express that a value will be used *exactly* once within its scope, as opposed to *at most* once in our case. This difference is reflected in our Weakening and Start rules (see below). Nevertheless, the possible connection with linear types must still be explored.

This paper is organized as follows. In the next section we develop the component language with its terms and operational semantics. We define types and typing relation in Section 3. Then we prove some properties of the type system in Section 4. We outline a polynomial time type inference algorithm in Section 5 before ending with some conclusions and indications of future research. Technical proofs are delegated to the appendix.

2 A component language

2.1 Terms

Since we are mainly interested in instantiation, composition and scope, we have abstracted in our component language from all other aspects of components. At this level of abstraction we do not detail how components are connected as in many architecture description languages like Wright [1], Darwin [9], SADL [12], and ACME [8].

A component may or may not allow multiple instances running side-by-side. We distinguish between these two kinds of components by using two disjoint sets **S** and **E** for such 'side-by-side' and 'exclusive' components, respectively. So, a component in **S** can have arbitrarily many instances, while a component in **E** can have at most one instance at a time.

We use extended Backus-Naur Form with the following meta-symbols: ϵ for the empty expression, infix $|$ for choice, postfix * for Kleene closure (zero or more iterations) and round brackets for grouping. The curly brackets { and } are part of the component language and are used as scope-delimiters.

Definition 2.1 [Component programs] The component programs are given by the following abstract syntax, with x ranging over $\mathbf{S} \cup \mathbf{E}$.

$$\begin{aligned} \mathit{Prog} &::= \mathit{Decl};\ \mathit{Exp} \\ \mathit{Decl} &::= (x \prec \mathit{Exp})^* \\ \mathit{Exp} &::= \epsilon \mid \mathtt{new}\, x\, \mathit{Exp} \mid \{\mathit{Exp}\}\mathit{Exp} \end{aligned}$$

The above grammar for expressions Exp generates the same expressions as $\mathit{Exp} ::= \epsilon \mid \mathtt{new}\, x \mid \mathit{Exp}\ \mathit{Exp} \mid \{\mathit{Exp}\}$ but has the technical advantage of non-ambiguity.

Thus a program consists of a list of so-called component declarations followed by an expression Exp which sparks off the execution. For a precise definition of the operational semantics the reader is referred to the next section. In particular the example there will help to elucidate the typing rules that are to follow.

A component declaration $x \prec \mathit{Exp}$ states that the component x is composed from the component expression Exp. In particular, $x \prec \epsilon$ states that x is a primitive component. A list of declarations with distinct variables is called a *basis*, as in PTSs [3]. We use Γ, Δ,... to range over bases.

According to the above syntax, component expressions can have several forms. The simplest, the empty expression ϵ is used for declaring primitive components. Otherwise, expressions can be a sequence of $\mathtt{new}\, x$'s with or without matching, possibly nested, scope-delimiters. The purpose of putting components in a scope is that, during their lifetime, these components need to collaborate with each other and can be deallocated afterwards. We use

$A, \ldots, E, Exp$ to range over expressions.

A program is *deterministic* if there is at most one declaration of every component. In this paper we only work with deterministic programs.

As an example, let $\mathbf{S} = \{c, d\}$ and $\mathbf{E} = \{a, b\}$. A well-formed program $Prog$ in our syntax could be as follows:

$$\begin{aligned}
&d \multimap \epsilon\\
&a \multimap \mathtt{new}\, d\\
&b \multimap \mathtt{new}\, a\\
&c \multimap \mathtt{new}\, d\, \{\mathtt{new}\, b\, \mathtt{new}\, d\, \}\mathtt{new}\, a\, ;\\
&\mathtt{new}\, c
\end{aligned}$$

We will return to this example in the following sections.

2.2 Operational semantics

In this section we give the operational semantics for our language. We use so-called *post-expressions*, which are obtained by deleting zero or more consecutive opening {'s in the prefix of an expression.

Definition 2.2 [Post-expressions] The post-expressions are given by the abstract syntax $P ::= (Exp\})^* Exp$, with Exp as in Definition 2.1:

We also give a formal definition of concatenation of two expressions.

Definition 2.3 [Expression concatenation] The concatenated expression of two expressions A and B, written $A@B$, is defined recursively as follows:

$$\begin{aligned}
\epsilon @ B &= B\\
(\mathtt{new}\, x\, A_1) @ B &= \mathtt{new}\, x\, (A_1 @ B)\\
(\{A_1\} A_2) @ B &= \{A_1\}(A_2 @ B)
\end{aligned}$$

Concatenation of an expression with a post-expression is defined in the same way as concatenation of two expressions, and results in again a post-expression.

The operational semantics is modelled as a *transition system* where a state is a stack $\mathbb{S}$ of multisets followed by a post-expression P. Stacks and post-expressions are separated by ' $\propto$ '. Elements of the stack are separated by ':'. Stacks are pushed and popped at the right end. The empty stack is denoted by $\emptyset$. When the depth of stack is at least one, we use the multiset M to denote the top of the stack and $\mathbb{S}$ to denote the rest of the stack. When the depth of a stack is one, we write just the multiset M, as in the second transition rule below. Multisets are denoted by $[\ldots]$, where sets are denoted, as usual, by

$\{\ldots\}$. $M(x)$ is the multiplicity of element x in multiset M. The operation $\uplus$ is additive union of multisets.

Definition 2.4 [Transition rules] The transition rules are given as follows:

$$
\begin{array}{lcl}
\emptyset \propto P & \xrightarrow{terminate} & failure \\
M \propto \epsilon & \xrightarrow{terminate} & success \\
\mathbb{S} : M : M' \propto \epsilon & \xrightarrow{terminate} & failure \\
\mathbb{S} : M \propto \mathtt{new}\, x\, P & \xrightarrow{x \prec E \in Prg} & \mathbb{S} : (M \uplus [x]) \propto E@P \\
\mathbb{S} : M \propto \{P & \xrightarrow{push} & \mathbb{S} : M : [\,] \propto P \\
\mathbb{S} : M \propto \}P & \xrightarrow{pop} & \mathbb{S} \propto P
\end{array}
$$

The transition rules can be explained as follows. When the stack is empty the transition process terminates to failure. When the input is ϵ, the transition process also terminates. In this case, if the depth of the stack is one, the program succeeds, else it fails. When the input is $\mathtt{new}\, x$, x is added to the multiset at the top of the stack and $\mathtt{new}\, x$ is replaced by the declaration of x. The last two rules are for scope. When entering a new scope, that is, when the first element of the input is '{', we push a new empty multiset $[\,]$ to the stack. When leaving a scope, that is, when the first element of the input is '}', the multiset at the top of the stack is popped. This means that all instances created in this scope have been discarded.

The example in Section 2.1 is again used to illustrate our operational semantics. The transition steps are showed as follows:

$$
\begin{array}{l}
[\,] \propto \mathtt{new}\, c \xrightarrow{c \prec \mathtt{new}\, d\, \{\mathtt{new}\, b\, \mathtt{new}\, d\, \}\mathtt{new}\, a} \\
[c] \propto \mathtt{new}\, d\, \{\mathtt{new}\, b\, \mathtt{new}\, d\, \}\mathtt{new}\, a \xrightarrow{d \prec \epsilon} \\
[c, d] \propto \{\mathtt{new}\, b\, \mathtt{new}\, d\, \}\mathtt{new}\, a \xrightarrow{push} \\
[c, d] : [\,] \propto \mathtt{new}\, b\, \mathtt{new}\, d\, \}\mathtt{new}\, a \xrightarrow{b \prec \mathtt{new}\, a} \\
[c, d] : [b] \propto \mathtt{new}\, a\, \mathtt{new}\, d\, \}\mathtt{new}\, a \xrightarrow{a \prec \mathtt{new}\, d} \\
[c, d] : [a, b] \propto \mathtt{new}\, d\, \mathtt{new}\, d\, \}\mathtt{new}\, a \xrightarrow{d \prec \epsilon} \\
[c, d] : [a, b, d] \propto \mathtt{new}\, d\, \}\mathtt{new}\, a \xrightarrow{d \prec \epsilon} \\
[c, d] : [a, b, d, d] \propto \}\mathtt{new}\, a \xrightarrow{pop} \\
[c, d] \propto \mathtt{new}\, a \xrightarrow{a \prec \mathtt{new}\, d} \\
[a, c, d] \propto \mathtt{new}\, d \xrightarrow{d \prec \epsilon} \\
[a, c, d, d] \propto \epsilon \xrightarrow{terminate} success
\end{array}
$$

In this example, exclusive component a is instantiated two times. However, for reason of scope, there is at most one instance of a at each moment.

If d had been exclusive, the execution of program would fail at $[c,d]$: $[a,b]$, $\texttt{new}\, d\, \texttt{new}\, d\, \}\texttt{new}\, a$. Continuing loading would duplicate exclusive component d.

3 Type system

3.1 Types

A component expression E may use several components. Among the latter there are instances that exist for the whole lifetime of E, whereas other instances live only for a while and are then discharged. Therefore we use two sets to represent the type of a component expression. The first set X^i collects all components instantiated during the lifetime of the expression and the second set X^o consists of those components that have instances surviving the execution of the expression.

Definition 3.1 [Types] The set of *types* for component expressions consists of pairs of sets

$$X^i|X^o$$

where $X^o, X^i \subseteq \mathbf{S} \cup \mathbf{E}$. Types are denoted by super- and subscripted capitals $U, \ldots, Z$.

3.2 Typing relation

Before defining the typing relation we give a formal definition of *domain*, the set of variables in a basis.

Definition 3.2 [Basis variables] The set of declared variables in basis Γ, written $Dom(\Gamma)$, is defined inductively as follows:

$$\begin{aligned} Dom() &= \emptyset \\ Dom(x \prec A, \Gamma) &= \{x\} \cup Dom(\Gamma) \end{aligned}$$

For example suppose

$$\Gamma = d \prec \epsilon, a \prec \texttt{new}\, d\,, b \prec \texttt{new}\, a\,, c \prec \texttt{new}\, d\, \{\texttt{new}\, b\, \texttt{new}\, d\, \}\texttt{new}\, a$$

then we have

$$Dom(\Gamma) = \{a, b, c, d\}$$

A *typing triple* $\Gamma \vdash A : X^i|X^o$, also called *typing* for short, expresses that, given basis Γ, the component expression A has type $X^i|X^o$. The *typing relation* is an inductively defined set of typing triples. It is defined in the usual way by giving *typing rules* to construct *derivation trees* for valid typings.

Definition 3.3 [Typing rules]

$$Axiom\ \frac{}{\vdash \epsilon : \emptyset | \emptyset}$$

$$Start\ \frac{\Gamma \vdash A : X^i | X^o}{\Gamma, x \prec A \vdash \mathtt{new}\, x\ : X^i \cup \{x\} | X^o \cup \{x\}}\ x \notin Dom(\Gamma)$$

$$Weakening\ \frac{\Gamma \vdash A : X^i | X^o \quad \Gamma \vdash A_1 : Y^i | Y^o}{\Gamma, x \prec A_1 \vdash A : X^i | X^o}\ x \notin Dom(\Gamma)$$

$$Sequencing\ \frac{\Gamma \vdash \mathtt{new}\, x\ : X^i | X^o \quad \Gamma \vdash A : Y^i | Y^o}{\Gamma \vdash \mathtt{new}\, x\, A : X^i \cup Y^i | X^o \cup Y^o}\ X^o \cap \mathbf{E} \cap Y^i = \emptyset, A \neq \epsilon$$

$$Scope\ \frac{\Gamma \vdash A_1 : X^i | X^o \quad \Gamma \vdash A_2 : Y^i | Y^o}{\Gamma \vdash \{A_1\} A_2 : X^i \cup Y^i | Y^o}$$

Let us briefly explain the above five typing rules. Rule Axiom requires no premise and is used to take-off. Rule Start allows us to type a new instance of a component. The combination of Axiom and Start allows us to type instances of primitive components $x \prec \epsilon$. Weakening is used to expand bases so that we can combine typings in rule Sequencing and Scope, which allow us to type component compositions with a prefix $\mathtt{new}\, x$ and a scoped expression, respectively. The side condition $x \notin Dom(\Gamma)$ prevents ambiguity and circularity. The side condition $X^o \cap \mathbf{E} \cap Y^i = \emptyset$ prevents exclusive components from being instantiated more than once in the same scope.

Continuing the example in Section 2.1, we show the *type derivation tree* for $\{\mathtt{new}\, b\, \mathtt{new}\, d\,\}\mathtt{new}\, a$. First, a typing for $\mathtt{new}\, b$ can be derived as follows: (The names of the typing rules are shortened to their first three letters.)

$$Sta\ \cfrac{Sta\ \cfrac{Sta\ \cfrac{Axi\ \cfrac{}{\vdash \epsilon : \emptyset|\emptyset}}{d \prec \epsilon \vdash \mathtt{new}\, d\ : \{d\}|\{d\}}}{d \prec \epsilon, a \prec \mathtt{new}\, d\ \vdash \mathtt{new}\, a\ : \{a, d\}|\{a, d\}}}{d \prec \epsilon, a \prec \mathtt{new}\, d\,, b \prec \mathtt{new}\, a\ \vdash \mathtt{new}\, b\ : \{a, b, d\}|\{a, b, d\}}$$

The typing for $\mathtt{new}\, d$ can be weakened as follows:

$$Wea\ \cfrac{Sta\ \cfrac{Axi\ \cfrac{}{\vdash \epsilon : \emptyset|\emptyset}}{d \prec \epsilon \vdash \mathtt{new}\, d\ : \{d\}|\{d\}} \quad Sta\ \cfrac{Axi\ \cfrac{}{\vdash \epsilon : \emptyset|\emptyset}}{d \prec \epsilon \vdash \mathtt{new}\, d\ : \{d\}|\{d\}}}{d \prec \epsilon, a \prec \mathtt{new}\, d\ \vdash \mathtt{new}\, d\ : \{d\}|\{d\}}$$

Using this result we can derive yet another typing for $\mathtt{new}\, d$ by weakening:

$$Wea\ \cfrac{\cfrac{\dots}{d \prec \epsilon, a \prec \mathtt{new}\, d \vdash \mathtt{new}\, d\ : \{d\}|\{d\}} \quad \cfrac{\dots}{d \prec \epsilon, a \prec \mathtt{new}\, d \vdash \mathtt{new}\, a\ : \{a, d\}|\{a, d\}}}{d \prec \epsilon, a \prec \mathtt{new}\, d\,, b \prec \mathtt{new}\, a \vdash \mathtt{new}\, d\ : \{d\}|\{d\}}$$

Similarly, we can weaken the typing for $\mathtt{new}\, a$:

$$d \prec \epsilon, a \prec \mathtt{new}\, d\,, b \prec \mathtt{new}\, a\ \vdash \mathtt{new}\, a\ : \{a, d\}|\{a, d\}$$

Now, let $\Gamma = d \prec \epsilon, a \prec \texttt{new}\, d\,, b \prec \texttt{new}\, a\,.$ Based on the previous typings we can derive a type for $\{\texttt{new}\, b\, \texttt{new}\, d\,\}\texttt{new}\, a$ as follows:

$$
Sco\ \dfrac{Seq\ \dfrac{\dfrac{\dots}{\Gamma \vdash \texttt{new}\, b\ :\{a,b,d\}|\{a,b,d\}} \quad \dfrac{\dots}{\Gamma \vdash \texttt{new}\, d\ :\{d\}|\{d\}}}{\Gamma \vdash \texttt{new}\, b\, \texttt{new}\, d\ :\{a,b,d\}|\{a,b,d\}} \qquad \dfrac{\dots}{\Gamma \vdash \texttt{new}\, a\ :\{a,d\}|\{a,d\}}}{\Gamma \vdash \{\texttt{new}\, b\, \texttt{new}\, d\,\}\texttt{new}\, a\ :\{a,b,d\}|\{a,d\}}
$$

Note that the side condition $X^o \cap \mathbf{E} \cap Y^i$ of rule Sequencing is satisfied since $d \notin \mathbf{E}$.

Having defined types, terms and typing rules, we can now define the notion of a well-typed program.

Definition 3.4 [Well-typed program] A well-formed program $P = Decl;\ Exp$ is well-typed if Exp can be typed in a basis built from $Decl$. Here it is understood that the declarations may have to be reordered to form a legal basis.

4 Properties of the type system

In this section we will state some properties of our type system. The invariant theorem and its correctness corollary at the end of the section relate the type system to the operational semantics. In order to prove the invariant property some definitions and lemmas are needed. Some technical proofs of lemmas are delegated to Appendix A, to improve the readability of this section.

Definition 4.1 [Bases] Let $\Gamma = x_1 \prec A_1, \dots, x_n \prec A_n$ be a basis and let A be an expression.

- Γ is called *legal* if $\Gamma \vdash A : X^i|X^o$ for some A, X^i, and X^o.
- A declaration $x \prec A$ *is in* Γ, notation $x \prec A \in \Gamma$, if $x \equiv x_i$ and $A \equiv A_i$ for some i.
- Δ is *part* of Γ, notation $\Delta \subseteq \Gamma$, if $\Delta = x_{i_1} \prec A_{i_1}, \dots, x_{i_k} \prec A_{i_k}$ with $1 \le i_1 < \dots < i_k \le n$. Note that the order is preserved.
- Δ is an *initial segment* of Γ, notation $\Delta \sqsubseteq \Gamma$, if $\Delta = x_1 \prec A_1, \dots, x_j \prec A_j$ for some $1 \le j \le n$.

Definition 4.2 [Expression variables] The set of variables occurring in an expression A, written $Var(A)$, is defined inductively as follows:

$$
\begin{aligned}
Var(\epsilon) &= \emptyset \\
Var(\texttt{new}\, x\ A) &= \{x\} \cup Var(A) \\
Var(\{A_1\}A_2) &= Var(A_1) \cup Var(A_2)
\end{aligned}
$$

For example, the set of expression variables of $\texttt{new}\, d\,\{\texttt{new}\, b\, \texttt{new}\, d\,\}\texttt{new}\, a$ is:

$$Var(\texttt{new}\, d\,\{\texttt{new}\, b\, \texttt{new}\, d\,\}\texttt{new}\, a\,) = \{a,b,d\}$$

For convenience we shall abbreviate from now on $X \cup \{x\}$ by $X{+}x$ and $X \setminus \{x\}$ by $X{-}x$. The latter abbreviation will only be used in cases where actually $x \in X$.

The following lemma collects a number of easy properties of a typing. These will be frequently be used in the sequel. It states that if an expression A has type $X^i|X^o$ in a basis Γ then the set of the variables of expression A is subset of the domain of Γ. Moreover, any legal basis always has its declarations distinct.

Lemma 4.3 (Legal basis properties) *If $\Gamma \vdash A : X^i|X^o$, then $Var(A) \cup X^o \subseteq X^i \subseteq Dom(\Gamma)$, $\Gamma \vdash \epsilon : \emptyset|\emptyset$, and every variable in $Dom(\Gamma)$ is declared only once in Γ.*

Proof (Sketch) By induction on derivation (full proof in Appendix A). □

The next lemma allows us to find the last typing rule applied to derive the type of an expression and hence it allows us to recursively calculate the types of well-typed expressions. We will return to this issue in Section 5, Type Inference. This lemma is sometimes called the *inversion lemma of the typing relation* [13].

Lemma 4.4 (Generation)

(i) *If $\Gamma \vdash \texttt{new}\,x\ : X^i|X^o$, then $x \in X^o$ and there exists Δ, Δ', A such that $\Gamma = \Delta, x{\prec} A, \Delta'$, and $\Delta \vdash A : X^i{-}x|X^o{-}x$.*

(ii) *If $\Gamma \vdash \texttt{new}\,x\,A : Z^i|Z^o$ with $A \neq \epsilon$, then there exists X^i, X^o, Y^i, Y^o such that $\Gamma \vdash \texttt{new}\,x\ : X^i|X^o$, $\Gamma \vdash A : Y^i|Y^o$, $Z^i = X^i \cup Y^i$, $Z^o = X^o \cup Y^o$, and $X^o \cap \mathbf{E} \cap Y^i = \emptyset$.*

(iii) *If $\Gamma \vdash \{B\}C : Z^i|Z^o$, then there exists X^i, X^o, Y^i, Y^o such that $\Gamma \vdash B : X^i|X^o$, $\Gamma \vdash C : Y^i|Y^o$, $Z^i = X^i \cup Y^i$, $Z^o = Y^o$.*

Proof (Sketch) All three items are proved by induction on derivation (full proof in Appendix A). □

In our type system the order of declarations in a legal basis is significant. The initial segment Δ of a legal basis Γ is a legal basis for the expression of the consecutive declaration after Δ. Besides, because of the weakening rule, there can be many legal bases under which a well-typed expression can be derived. These properties are stated in the following lemma.

Lemma 4.5 (Legal monotonicity)

(i) *If $\Gamma = \Delta, x{\prec} A, \Delta'$ is legal, then $\Delta \vdash A : X^i|X^o$ for some X^i and X^o.*

(ii) *If $\Gamma \vdash A : X^i|X^o$, $\Gamma \subseteq \Gamma'$ and Γ' is legal, then $\Gamma' \vdash A : X^i|X^o$.*

Proof (Sketch)

(i) The only way to extend Δ to $\Delta, x{\prec}\ A$ in a derivation is by applying the rule Start or Weakening. Each of the rules has $\Delta \vdash A : X^i|X^o$ as a premise.

(ii) By induction on the derivation of $\Gamma \vdash A : X^i|X^o$ (see Appendix A).

□

The following lemma can be viewed as the inverse of the previous legal monotonicity lemma. Under certain conditions we can contract the legal basis so that the expression is still well-typed in the new basis.

Lemma 4.6 (Strengthening) *If* $\Gamma, x{\prec} A \vdash B : Y^i|Y^o$ *and* $x \notin Var(B)$, *then* $\Gamma \vdash B : Y^i|Y^o$ *and* $x \notin Y^i$.

Proof (Sketch) By induction on derivation (full proof in Appendix A). □

In our simple type system, every term has a unique type if it has a type at all. This property is stated and proved in the following proposition.

Proposition 4.7 (Uniqueness of types) *If* $\Gamma \vdash A : X^i|X^o$ *and* $\Gamma \vdash A : Y^i|Y^o$, *then* $X^i = Y^i$ *and* $X^o = Y^o$.

Proof. By induction on the derivation of $\Gamma \vdash A : X^i|X^o$.

Base step: In the case of Axiom we have $A = \epsilon$ and Γ is empty, so that only Axiom is applicable. Hence, $X^i = Y^i = \emptyset$ and $X^o = Y^o = \emptyset$.

Induction step:

- Case Start: Let $\Gamma = \Gamma', x{\prec} B$ such that:

$$Start\ \frac{\Gamma' \vdash B : X^i{-}x|X^o{-}x}{\Gamma', x{\prec} B \vdash \mathtt{new}\, x\ : X^i|X^o}\ x \notin Dom(\Gamma')$$

with $x \in X^i$ and $x \in X^o$. Assume Proposition 4.7 holds for the premise and let $\Gamma \vdash \mathtt{new}\, x\ : Y^i|Y^o$. By Generation Lemma we have $x \in Y^o$ and $\Gamma = \Delta_1, x{\prec} C, \Delta_2$ and $\Delta_1 \vdash C : Y^i{-}x|Y^o{-}x$ for some Δ_1, Δ_2, C.

By Lemma 4.3, there is only one declaration of x in Γ. This means $\Delta_1 = \Gamma'$, $C = B$ and Δ_2 is empty, so $\Gamma' \vdash B : Y^i{-}x|Y^o{-}x$. By IH we have $X^i{-}x = Y^i{-}x$, $X^o{-}x = Y^o{-}x$. So $X^i = Y^i$, $X^o = Y^o$ as by Lemma 4.3 also $x \in Y^o$.

- Case Weakening: Let $\Gamma = \Gamma', x{\prec} B$ such that:

$$Weakening\ \frac{\Gamma' \vdash A : X^i|X^o \quad \Gamma' \vdash B : Z^i|Z^o}{\Gamma', x{\prec} B \vdash A : X^i|X^o}\ x \notin Dom(\Gamma'), B \neq \epsilon$$

Assume Proposition 4.7 holds for the two premises and let $\Gamma = \Gamma', x{\prec} B \vdash A : Y^i|Y^o$. Since $\Gamma' \vdash A : X^i|X^o$ we have $x \notin Var(A)$. By Lemma 4.6 applied to $\Gamma', x{\prec} B \vdash A : Y^i|Y^o$ we get $\Gamma' \vdash A : Y^i|Y^o$. By IH we have the conclusion $X^i = Y^i$ and $X^o = Y^o$.

- Case Sequencing: Let $\Gamma \vdash \mathtt{new}\, x\, B : X^i|X^o$ with $B \neq \epsilon$ be inferred by:

$$Sequencing\ \frac{\Gamma \vdash \mathtt{new}\, x\ : V^i|V^o \quad \Gamma \vdash B : W^i|W^o}{\Gamma \vdash \mathtt{new}\, x\, B : V^i \cup W^i|V^o \cup W^o}\ V^o \cap \mathbf{E} \cap W^i = \emptyset, B \neq \epsilon$$

By Generation Lemma 4.4 applied to $\Gamma \vdash \mathtt{new}\, x\, B : Y^i|Y^o$ we have $\Gamma \vdash \mathtt{new}\, x : V_1^i|V_1^o$, $\Gamma \vdash B : W_1^i|W_1^o$, $Y^i = V_1^i \cup W_1^i$, and $Y^o = V_1^o \cup W_1^o$ for some V_1^i, V_1^o, W_1^i, W_1^o. By the IH, we have $V^i = V_1^i$, $V^o = V_1^o$, $W^i = W_1^i$, and $W^o = W_1^o$. Hence, $X^i = Y^i = V^i \cup W^i$ and $X^o = Y^o = V^o \cup W^o$ follow.

- Case Scope: analogous to case Sequencing.

□

The following lemma plays a role in the invariant of the operational semantics. If we replace a component $\mathtt{new}\, x$ in a well-typed expression by the declaration of x in the typing basis, then the new expression is again well-typed in the same basis.

Lemma 4.8 (Substitution) *Suppose* $\Gamma \vdash \mathtt{new}\, x\, B : Z^i|Z^o$*, then*

(i) $x \prec A \in \Gamma$ *and* $\Gamma \vdash A : X^i|X^o$ *for some* A*,* X^i*, and* X^o*.*

(ii) $\Gamma \vdash B : Y^i|Y^o$ *for some* Y^i *and* Y^o*.*

(iii) $\Gamma \vdash A@B : X^i \cup Y^i|X^o \cup Y^o$*.*

Proof. In Appendix A. □

Now, we give some definitions before stating the invariant theorem and correctness corollary for our type system. In the rest of this section, we assume that we are working with some well-typed program *Prog* and two disjoint sets $\mathbf{S}$ and $\mathbf{E}$ of side-by-side and exclusive components, respectively.

Definition 4.9 [Single multiset, projection]

- A multiset M is *single* if the multiplicity of every element of M is 1. Thus, a single multiset is a set and set operations apply.
- The projection of a multiset M by a set $\mathbf{E}$, notation $M|_{\mathbf{E}}$, is the multiset obtained by removing from M all elements that are not in $\mathbf{E}$:

$$(M|_{\mathbf{E}})(x) = \begin{cases} M(x) & \text{if } x \in \mathbf{E} \\ 0 & \text{otherwise} \end{cases}$$

Definition 4.10 [Stack union projection] Suppose we have a stack of multisets $\mathbb{S} = S_1 : \ldots : S_n$. The multiset of exclusive elements in stack $\mathbb{S}$, written $\mathbb{S}|_{\mathbf{E}}$, is defined as follows:

$$\mathbb{S}|_{\mathbf{E}} = (S_1 \uplus \cdots \uplus S_n)|_{\mathbf{E}}$$

Theorem 4.11 (Invariant) *Let* Γ *be a basis. Assume stack* $\mathbb{S} = S_1 : \ldots : S_n$ *with* $\mathbb{S}|_{\mathbf{E}}$ *single, and expressions* A_j *with* $\Gamma \vdash A_j : X_j^i|X_j^o$ *for all* $1 \leq j \leq n$*, such that*

$$(S_1 : .. : S_k)|_{\mathbf{E}} \cap X_k^i = \emptyset \quad \textit{for all} \quad 1 \leq k \leq n \tag{1}$$

Then, either we have termination or there exists a unique stack $\mathbb{S}' = S_1' : \ldots : S_m'$ *with* $\mathbb{S}'|_{\mathbf{E}}$ *single, and unique expressions* B_j *with* $\Gamma \vdash B_j : Y_j^i|Y_j^o$ *for all*

$1 \leq j \leq m$, *such that*

$$\mathbb{S} \ \propto \ A_n\}A_{n-1}\ldots\}A_1 \rightarrow \mathbb{S}' \ \propto \ B_m\}B_{m-1}\ldots\}B_1$$

and

$$(S'_1 : \ldots : S'_k)|_{\mathbf{E}} \cap Y^i_k = \emptyset \quad \textit{for all} \quad 1 \leq k \leq m \qquad (2)$$

Proof. By examining all possible transitions.

- If $A_n = \epsilon$ and $n = 1$, then the transition terminates.
- If $A_n = \epsilon$ and $n > 1$, then the transition step is:

$$S_1 : \ldots : S_n \ \propto \}A_{n-1}\ldots\}A_1 \xrightarrow{pop} S_1 : \ldots : S_{n-1} \ \propto \ A_{n-1}\}\ldots\}A_1$$

  All the conclusions follow immediately since $m = n - 1$.
- If $A_n = \mathtt{new}\, x$, by Generation Lemma, we have $x \prec A' \in \Gamma$. Then, the transition step is:

$$S_1 : \ldots : S_n \ \propto \ \mathtt{new}\, x\,\}A_{n-1}\ldots\}A_1 \xrightarrow{x \prec A' \in \Gamma}$$
$$S_1 : \ldots : (S_n \uplus [x]) \ \propto \ A'\}A_{n-1}\ldots\}A_1$$

  We have $m = n$, $B_m = A'$, and $B_j = A_j$ for $1 \leq j \leq n - 1$. Thus, $\Gamma \vdash B_j : X^i_j|X^o_j$ follows immediately for $1 \leq j \leq m - 1$. $\Gamma \vdash B_m : Y^i_m|Y^o_m$ also follows from Lemma 4.4 applied to $\Gamma \vdash \mathtt{new}\, x \ : X^i_m|X^o_m$ by taking $Y^i_m = X^i_m - x$ and $Y^o_m = X^o_m - x$.

  Equation (2) holds for $k \leq m-1$ by assumption and we only have to prove Equation (2) for $k = m$. If $x \notin \mathbf{E}$, the proof is trivial since $(S'_1 : \ldots : S'_n)|_{\mathbf{E}} = (S'_1 : \ldots : S'_m)|_{\mathbf{E}}$ and $Y^i_m \subseteq X^i_m$. Otherwise, since $Y^i_m = X^i_m - x$, so $x \notin Y^i_m$. By Equation (1) with $k = n$ and $x \in X^i_n$ we have $x \notin (S_1 : \ldots : S_n)$. Hence, $(S'_1 : \ldots : S'_m)|_{\mathbf{E}} \cap Y^i_m = \emptyset$ holds as the new exclusive variable x only occurs in the left of intersection operation. The conclusion $\mathbb{S}'|_{\mathbf{E}}$ single also follows as there is only one new $x \in S'_m$ and $x \notin (S_1 : .. : S_{n-1})|_{\mathbf{E}} = (S'_1 : .. : S'_{m-1})|_{\mathbf{E}}$.
- If $A_n = \mathtt{new}\, x\, C$ with $C \neq \epsilon$, then the transition step is:

$$S_1 : \ldots : S_n \ \propto \ \mathtt{new}\, x\, C\}A_{n-1}\ldots\}A_1 \xrightarrow{x \prec A' \in \Gamma}$$
$$S_1 : \ldots : (S_n \uplus [x]) \ \propto \ A'@C\}A_{n-1}\ldots\}A_1$$

  We have $m = n$, $B_n = A'@C$, and $B_j = A_j$ for $1 \leq j \leq n - 1$. Thus, $\Gamma \vdash B_j : Y^i_j|Y^o_j$ follows immediately for $1 \leq j \leq n - 1$. $\Gamma \vdash A'@C : Y^i_n|Y^o_n$ also follows from Lemma 4.8 applied to $\Gamma \vdash \mathtt{new}\, x\, C : X^i_m|X^o_m$. The remaining proof is analogous to the previous case.

- If $A_n = \{C\}D$, then the transition step is:

$$\begin{aligned} S_1 : \ldots : S_n \ &\propto \ \{C\}D\}A_{n-1}\ldots\}A_1 \xrightarrow{push} \\ S_1 : \ldots : S_n : [\,] \ &\propto \ C\}D\}A_{n-1}\}\ldots\}A_1 \end{aligned}$$

We have $m = n+1$, $B_m = C$, $B_n = D$ and $B_j = A_j$ with $1 \le j \le n-1$. Thus, $\Gamma \vdash B_j : Y^i_j|Y^o_j$ follows immediately for $1 \le j \le n-1$. $\Gamma \vdash B_m : Y^i_m|Y^o_m$ and $\Gamma \vdash B_n : Y^i_n|Y^o_n$ also follow from Lemma 4.4 applied to $\Gamma \vdash \{C\}D : X^i_m|X^o_m$. The remaining conclusions follow trivially.

□

Corollary 4.12 (Correctness) *Starting with the stack* $[\,]$, *containing only the empty multiset, and a well-typed expression,* $\mathbb{S}|_{\mathbf{E}}$ *is single in every consecutive state, that is, of every exclusive component there is at most one instance at a time.*

Proof. Follows from iterating the previous theorem starting with $n = 1$. □

5 Type inference

In this section we sketch a polynomial time type inference algorithm. The type inference problem, or more precisely, an instance of this problem, is to determine, given basis Γ and expression A, a type $X^i|X^o$ such that $\Gamma \vdash A : X^i|X^o$. By Proposition 4.7 we know that such a type is unique if it exists. Inferring such types automatically relieves the programmer from the task to give the types explicitly and have them checked. The types inferred can be expected to guide the design of the component program. Moreover, by the correctness result Corollary 4.12, a well-typed expression can be safely executed. The latter could also be tested by running the operational semantics according to the rules in Definition 2.4. However, running these rules could be exponential (by iterated duplication, for example), so that a polynomial time type inference algorithm is to be preferred.

Let *Prog* be a component program. A necessary (but not sufficient) condition for type inference is that the declarations in *Prog* can be reordered into a basis Γ such that, for any declaration $x \prec A$ in Γ, the variables occurring in A are already declared previously in Γ. In other words:

$$\text{if } \Gamma = \Delta, x \prec A, \Delta' \text{ then } Var(A) \subseteq Dom(\Delta) \tag{3}$$

The existence of such a reordering can be detected in polynomial time by an analysis of the dependency graph associated with the declarations in *Prog*. From now on we assume that Γ is a basis consisting of all declarations in *Prog* and satisfying (3). The considerations below are independent of which particular ordering is used as long as it satisfies (3).

The basic idea behind the type inference algorithm is to exploit the fact that the typing rules are syntax-directed, or, in other words, to use the Generation Lemma 4.4 reversely. By applying clause 2 and 3 of this lemma to expression of the forms $\texttt{new}\,x\,A$ with $A \neq \epsilon$ and $\{B\}C$, respectively, we can break down any instance of the type inference problem to instances where the expression is simply of the form $\texttt{new}\,x$. We can then look up the declaration of x in the basis Γ. If no declaration of x can be found then no type can be inferred. Otherwise $\Gamma = \Delta, x \!\prec A, \Delta'$ for some Δ, Δ' and A and clause 1 of the Generation Lemma allows us to reduce the problem to inferring the type of A in Δ, together with the additional task of checking if Δ' legally extends $\Delta, x \!\prec A$. Here some care has to be taken in order to stay polynomial. A naive recursive algorithm could behave exponentially by generating recursively duplicate instances of the same type inference problem. Duplication can, however, be avoided by storing solved instances. Observe that all instances are of the form: infer the type of A in Δ, where Δ is an initial segment of the basis of the original type inference problem and A is a sub-expression of one of its constituents. There are polynomially many of such instances and hence type inference can be done in polynomial time. This finishes the sketch. We hope finally to be able to infer types in cubic or even quadratic time.

6 Conclusions and future research

We have designed a component language and a type system which allows one to detect statically whether or not multiple instances of certain components are running side-by-side. The language features instantiation, (sequential) composition and scope. For the future we plan to include more sophisticated language features such as explicit dispose operators, connectivity, concurrency features and versioning.

References

[1] R. Allen and G. Garlan. Formalizing Architectural Connection. In *Proceedings of the Sixteenth International Conference on Software Engineering*, Sorrento, Italy, May 1994.

[2] H. Baker. 'Use-Once' Variables and Linear Objects – Storage Management, Reflection and Multi-Threading. *ACM SIGPLAN Notices 30*, January 1995.

[3] H. Barendregt. Lambda Calculi with Types. In: Abramsky, Gabbay, Maibaum (Eds.), *Handbook of Logic in Computer Science*, Vol. II. Oxford University Press. 1992.

[4] D. Box and C. Sells. *Essential .NET, Volume I: The Common Language Runtime*, Addison-Wesley, ISBN 0201734117, November 2002.

[5] J. Cheesman and J. Daniels, *UML Components: A Simple Process for Specifying Component-Based Software*, Addison-Wesley, ISBN 0201708515, 2000.

[6] M. Fähndrich and R. DeLine. Adoption and Focus: Practical Linear Types for Imperative Programming, *Proceedings of the SIGPLAN'02 Conference on Programming Language Design and Implementation*, Jun 2002.

[7] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns - Elements of Reusable ObjectOriented Software*, Addison-Wesley, Reading, Mass., ISBN 0201633612, 1994.

[8] D. Garlan, R. Monroe, and D. Wile. ACME: An Architecture Description Interchange Language. In *Proceedings of CASCON'97*, November 1997.

[9] J. Magee, N. Dulay, S. Eisenbach, and J. Kramer. Specifying Distributed Software Architectures. In *Proceedings of the Fifth European Software Engineering Conference (ESEC95)*, Barcelona, September 1995.

[10] E. Meijer and C. Szyperski. Overcoming Independent Extensibility Challenges, *Communications of the ACM*, Vol. 45, No. 10, pp. 41–44, October 2002.

[11] R. Milner et alii. *The Definition of Standard ML (Revised)*, MIT Press, ISBN 0262631814, 1997.

[12] M. Moriconi, X. Qian, and R. A. Riemenschneider. Correct Architecture Refinement. *IEEE Transactions on Software Engineering*, April 1995.

[13] B. Pierce. *Types and Programming Languages.* MIT Press, ISBN 0262162091, February 2002.

[14] J. C. Seco and L. Caires, A Basic Model of Typed Components, *Lecture Notes in Computer Science*, Vol. 1850, 2000.

[15] C. Szyperski. *Component Software: Beyond Object-Oriented Programming*, 2nd edition, Addison-Wesley, ISBN 0201745720, 2002.

[16] M. Zenger, Type-Safe Prototype-Based Component Evolution, *Proceedings of the European Conference on Object-Oriented Programming*, Malaga, Spain, June 2002.

A Appendix

Lemma 4.3 (Legal basis properties)

If $\Gamma \vdash A : X^i|X^o$, then $Var(A) \cup X^o \subseteq X^i \subseteq Dom(\Gamma)$, $\Gamma \vdash \epsilon : \emptyset|\emptyset$, and every variable in $Dom(\Gamma)$ is declared only once in Γ.

Proof. By induction on derivation.

Base case Axiom: $\vdash \epsilon : \emptyset|\emptyset$ is trivial as $Var(\epsilon) = X^o = X^i = Dom() = \emptyset$.

Induction step: We have to consider four cases corresponding to the four typing rules.

- Case Start:

$$\mathit{Start}\ \frac{\Gamma \vdash A : X^i|X^o}{\Gamma, x{\prec} A \vdash \mathtt{new}\, x\ : X^i{+}x|X^o{+}x}\ x \notin \mathit{Dom}(\Gamma)$$

  Assume the lemma is correct for the premise of this rule, so $Var(A) \cup X^o \subseteq X^i \subseteq \mathit{Dom}(\Gamma)$, $\Gamma \vdash \epsilon : \emptyset|\emptyset$ and every variable is declared at most once in Γ. Then, $Var(\mathtt{new}\, x\ ) = \{x\} \subseteq X^o{+}x \subseteq X^i{+}x \subseteq \mathit{Dom}(\Gamma){+}x = \mathit{Dom}(\Gamma, x{\prec} A)$. Moreover, $\Gamma, x{\prec} A \vdash \epsilon : \emptyset|\emptyset$ follows by applying Weakening:

$$\mathit{Weakening}\ \frac{\Gamma \vdash \epsilon : \emptyset|\emptyset \quad \Gamma \vdash A : X^i|X^o}{\Gamma, x{\prec} A \vdash \epsilon : \emptyset|\emptyset}\ x \notin \mathit{Dom}(\Gamma)$$

  The last conclusion: every variable in $\Gamma, x{\prec} A$ is declared at most once, follows by the side condition $x \notin \mathit{Dom}(\Gamma)$.

- Case Weakening:

$$\mathit{Weakening}\ \frac{\Gamma \vdash A : X^i|X^o \quad \Gamma \vdash B : Y^i|Y^o}{\Gamma, x{\prec} B \vdash A : X^i|X^o}\ x \notin \mathit{Dom}(\Gamma)$$

  Assume the lemma is correct for the two premises of this rule, so $Var(A) \cup X^o \subseteq X^i \subseteq \mathit{Dom}(\Gamma)$, $Var(B) \cup Y^o \subseteq Y^i \subseteq \mathit{Dom}(\Gamma)$, $\Gamma \vdash \epsilon : \emptyset|\emptyset$ and every variable is declared at most once in Γ. We have $Var(A) \cup X^o \subseteq X^i \subseteq \mathit{Dom}(\Gamma) \subseteq \mathit{Dom}(\Gamma, x{\prec} B)$. The last two conclusions are proved in the same way as in the case Start.

- Case Sequencing:

$$\mathit{Sequencing}\ \frac{\Gamma \vdash \mathtt{new}\, x\ : X^i|X^o \quad \Gamma \vdash A : Y^i|Y^o}{\Gamma \vdash \mathtt{new}\, x\, A : X^i \cup Y^i|X^o \cup Y^o}\ X^o \cap \mathbf{E} \cap Y^i = \emptyset, A \neq \epsilon$$

  Assume the lemma holds for the two premises of this rule, so $Var(\mathtt{new}\, x\ ) \cup X^o \subseteq X^i \subseteq \mathit{Dom}(\Gamma)$, $Var(A) \cup Y^o \subseteq Y^i \subseteq \mathit{Dom}(\Gamma)$, $\Gamma \vdash \epsilon : \emptyset|\emptyset$ and every variable is declared at most once in Γ. Note that $Var(\mathtt{new}\, x\, A) = \{x\} \cup Var(A)$. We have $Var(\mathtt{new}\, x\, A) \cup X^o \cup Y^o \subseteq X^i \cup Y^i \subseteq \mathit{Dom}(\Gamma)$. The remaining conclusions are the IHs themselves.

- Case Scope:

$$\mathit{Scope}\ \frac{\Gamma \vdash A_1 : X^i|X^o \quad \Gamma \vdash A_2 : Y^i|Y^o}{\Gamma \vdash \{A_1\}A_2 : X^i \cup Y^i|Y^o}$$

  Assume the lemma holds for the two premises of this rule, so $Var(A_1) \cup X^o \subseteq X^i \subseteq \mathit{Dom}(\Gamma)$, $Var(A_2) \cup Y^o \subseteq Y^i \subseteq \mathit{Dom}(\Gamma)$, $\Gamma \vdash \epsilon : \emptyset|\emptyset$ and every variable is declared at most once in Γ. Note that $Var(\{A_1\}A_2) = Var(A_1) \cup Var(A_2)$. We have $Var(\{A_1\}A_2) \cup Y^o \subseteq X^i \cup Y^i \subseteq \mathit{Dom}(\Gamma)$. The remaining conclusions are the IHs themselves.

□

Lemma 4.4 (Generation)

(i) If $\Gamma \vdash \mathtt{new}\, x\ : X^i|X^o$, then $x \in X^o$ and there exists Δ, Δ', A such that $\Gamma = \Delta, x \prec A, \Delta'$, and $\Delta \vdash A : X^i{-}x|X^o{-}x$.

(ii) If $\Gamma \vdash \mathtt{new}\, x\, A : Z^i|Z^o$ with $A \neq \epsilon$, then there exists X^i, X^o, Y^i, Y^o such that $\Gamma \vdash \mathtt{new}\, x\ : X^i|X^o$, $\Gamma \vdash A : Y^i|Y^o$, $Z^i = X^i \cup Y^i$, $Z^o = X^o \cup Y^o$, and $X^o \cap \mathbf{E} \cap Y^i = \emptyset$.

(iii) If $\Gamma \vdash \{B\}C : Z^i|Z^o$, then there exists X^i, X^o, Y^i, Y^o such that $\Gamma \vdash B : X^i|X^o$, $\Gamma \vdash C : Y^i|Y^o$, $Z^i = X^i \cup Y^i$, $Z^o = Y^o$.

Proof. All three items are proved by induction on derivation.

(i) $\Gamma \vdash \mathtt{new}\, x\ : X^i|X^o$ can only be derived by rule Start or rule Weakening. If it is derived by rule Start, then there is only one possibility:

$$\mathit{Start}\ \frac{\Delta \vdash A : X^i{-}x|X^o{-}x}{\Gamma \vdash \mathtt{new}\, x\ : X^i|X^o}\ x \notin \mathit{Dom}(\Delta)$$

with $x \in X^o$ and $\Gamma = \Delta, x \prec A$, so that Δ' is empty.

If $\Gamma \vdash \mathtt{new}\, x\ : X^i|X^o$ is derived by rule Weakening:

$$\mathit{Weakening}\ \frac{\Gamma' \vdash \mathtt{new}\, x\ : X^i|X^o \quad \Gamma' \vdash B : Y^i|Y^o}{\Gamma', y \prec B \vdash \mathtt{new}\, x\ : X^i|X^o}\ y \notin \mathit{Dom}(\Gamma')$$

then $\Gamma' \vdash \mathtt{new}\, x\ : X^i|X^o$ and by the IH applied to $\Gamma' \vdash \mathtt{new}\, x\ : X^i|X^o$ we have $x \in X^o$, $\Gamma' = \Delta_1, x \prec A', \Delta_2$ and $\Delta_1 \vdash \mathtt{new}\, x\ : X^i{-}x|X^o{-}x$ for some Δ_1, Δ_2, and A'. Now take $\Delta = \Delta_1$, $\Delta' = \Delta_2, y \prec B$, $A = A'$ and we have all the conclusions.

(ii) Similarly, $\Gamma \vdash \mathtt{new}\, x\, A : Z^i|Z^o$ with $A \neq \epsilon$ can only be derived by rule Sequencing or rule Weakening. The proof for case Sequencing is immediate. If $\Gamma \vdash \mathtt{new}\, x\, A : Z^i|Z^o$ is derived by rule Weakening:

$$\mathit{Weakening}\ \frac{\Gamma' \vdash \mathtt{new}\, x\, A : Z^i|Z^o \quad \Gamma' \vdash B : V^i|V^o}{\Gamma', y \prec B \vdash \mathtt{new}\, x\, A : Z^i|Z^o}\ y \notin \mathit{Dom}(\Gamma')$$

then $\Gamma = \Gamma', y \prec B$ and by the IH applied to $\Gamma' \vdash \mathtt{new}\, x\, A : Z^i|Z^o$ we have $\Gamma' \vdash \mathtt{new}\, x\ : X^i|X^o$, $\Gamma' \vdash A : Y^i|Y^o$, $Z^i = X^i \cup Y^i$, $Z^o = X^o \cup Y^o$, and $X^o \cap \mathbf{E} \cap Y^i = \emptyset$. Now weakening $\Gamma' \vdash \mathtt{new}\, x\ : X^i|X^o$ and $\Gamma' \vdash A : Y^i|Y^o$ to $\Gamma = \Gamma', y \prec B$ we have all the conclusions.

(iii) Similarly, $\Gamma \vdash \{B\}C : Z^i|Z^o$ can only be derived by rule Scope or rule Weakening. The proof is analogous to that of the previous case.

□

Lemma 4.5 (Legal monotonicity)

(i) If $\Gamma = \Delta, x \prec A, \Delta'$ is legal, then $\Delta \vdash A : X^i|X^o$ for some X^i and X^o.

(ii) If $\Gamma \vdash A : X^i|X^o$, $\Gamma \subseteq \Gamma'$ and Γ' is legal, then $\Gamma' \vdash A : X^i|X^o$.

Proof.

(i) The only way to extend Δ to $\Delta, x \prec A$ in a derivation is by applying the rule Start or Weakening.

$$\mathit{Weakening}\ \frac{\Delta \vdash \epsilon : \emptyset|\emptyset \quad \Delta \vdash A : X^i|X^o}{\Delta, x \prec A \vdash \epsilon : \emptyset|\emptyset}\ x \notin \mathit{Dom}(\Delta)$$

$$\mathit{Start}\ \frac{\Delta \vdash A : X^i|X^o}{\Delta, x \prec A \vdash \mathtt{new}\, x\ : X^i{+}x|X^o{+}x}\ x \notin \mathit{Dom}(\Delta)$$

Each of the rules has $\Delta \vdash A : X^i|X^o$ as a premise.

(ii) By induction on derivation of $\Gamma \vdash A : X^i|X^o$. We prove that for all Γ' legal such that $\Gamma \subseteq \Gamma'$ we have $\Gamma' \vdash A : X^i|X^o$.

Base case Axiom $A = \epsilon$: then $\Gamma' \vdash \epsilon : \emptyset|\emptyset$ since Γ' is legal.

Case Start $A = \mathtt{new}\, x$:

$$\mathit{Start}\ \frac{\Delta \vdash B : Y^i|Y^o}{\Gamma = \Delta, x \prec B \vdash \mathtt{new}\, x\ : Y^i{+}x|Y^o{+}x}\ x \notin \mathit{Dom}(\Delta)$$

Let $\Gamma \subseteq \Gamma'$ with Γ' legal. Then there exists Δ_1, Δ_2, Δ_3 such that $\Delta_1, \Delta, \Delta_2, x \prec B, \Delta_3 = \Gamma'$, with all initial segments of Γ' are legal. By IH we have $\Delta_1, \Delta, \Delta_2 \vdash B : Y^i|Y^o$. As x occurs only once in Γ' we have $x \notin \mathit{Dom}(\Delta_1, \Delta, \Delta_2)$ and we can apply rule Start to get $\Delta_1, \Delta, \Delta_2, x \prec B \vdash \mathtt{new}\, x\ : Y^i{+}x|Y^o{+}x$. Since Γ' is legal we can iterate weakening to get $\Gamma' \vdash \mathtt{new}\, x\ : Y^i{+}x|Y^o{+}x$.

Case Weakening:

$$\mathit{Weakening}\ \frac{\Delta \vdash A : X^i|X^o \quad \Delta \vdash B : Y^i|Y^o}{\Gamma = \Delta, y \prec B \vdash A : X^i|X^o}\ x \notin \mathit{Dom}(\Delta)$$

Let $\Gamma \subseteq \Gamma'$ with Γ' legal, so also $\Delta \subseteq \Gamma'$. By IH we get immediately $\Gamma' \vdash A : X^i|X^o$.

Case Sequencing $A = \mathtt{new}\, x\, B$ with $B \neq \epsilon$: by Generation Lemma we have $\Gamma \vdash \mathtt{new}\, x\ : V^i|V^o$ and $\Gamma \vdash B : Y^i|Y^o$ with $X^i = V^i \cup Y^i$, $X^o = V^o \cup Y^o$, and $V^o \cap \mathbf{E} \cap Y^i = \emptyset$. By IHs we have $\Gamma' \vdash \mathtt{new}\, x\ : V^i|V^o$ and $\Gamma' \vdash B : Y^i|Y^o$. Apply rule Sequencing and we get the conclusion.

Case $A = \{A_1\}A_2$: analogous to the case Sequencing.

□

Lemma 4.6 (Strengthening)

If $\Gamma, x \prec A \vdash B : Y^i|Y^o$ and $x \notin \mathit{Var}(B)$, then $\Gamma \vdash B : Y^i|Y^o$ and $x \notin Y^i$.

Proof. By induction on derivation.

- Case Axiom: does not apply since the basis is not empty.
- Case Start: does not apply since $\mathit{Var}(B) = \mathit{Var}(\mathtt{new}\, x\,) = \{x\}$.

- Case Weakening:

$$Weakening\ \frac{\Gamma \vdash B : Y^i|Y^o \quad \Gamma \vdash A : X^i|X^o}{\Gamma, x \prec A \vdash B : Y^i|Y^o}\ x \notin Dom(\Gamma)$$

  Then we get $\Gamma \vdash B : Y^i|Y^o$ in the premise. Moreover, $x \notin Y^i$ since $Y^i \subseteq Dom(\Gamma)$ and x is declared only once in $\Gamma, x \prec A$, both by Lemma 4.3.

- Case Sequencing:

$$Sequencing\ \frac{\Gamma, x \prec A \vdash \mathtt{new}\, y : V^i|V^o \quad \Gamma, x \prec A \vdash C : Z^i|Z^o}{\Gamma, x \prec A \vdash \mathtt{new}\, y\, C : Y^i|Y^o}\ V^o \cap \mathbf{E} \cap Z^i = \emptyset, C \neq \epsilon$$

  for V^i, V^o, Z^i, Z^o such that $Y^i = V^i \cup Z^i$ and $Y^o = V^o \cup Z^o$. Since $x \notin Var(\mathtt{new}\, y\, C) = \{y\} \cup Var(C)$ we have $x \neq y$ and $x \notin Var(C)$. By IHs we get $\Gamma \vdash \mathtt{new}\, y : V^i|V^o$ and $x \notin V^i$, $\Gamma \vdash C : Z^i|Z^o$ and $x \notin Z^i$. So by applying rule Sequencing we get the conclusion: $\Gamma \vdash \mathtt{new}\, y\, C : Y^i|Y^o$.

- Case $A = \{A_1\}A_2$: analogous to the case Sequencing.

□

Lemma 4.8 (Substitution)

Suppose $\Gamma \vdash \mathtt{new}\, x\, B : Z^i|Z^o$, then

(i) $x \prec A \in \Gamma$ and $\Gamma \vdash A : X^i|X^o$ for some A, X^i, and X^o.

(ii) $\Gamma \vdash B : Y^i|Y^o$ for some Y^i and Y^o.

(iii) $\Gamma \vdash A@B : X^i \cup Y^i|X^o \cup Y^o$.

Proof. If $B = \epsilon$ then the lemma trivially holds. So we assume $B \neq \epsilon$.

(i) If $\Gamma \vdash \mathtt{new}\, x\, B : Z^i|Z^o$ with $B \neq \epsilon$, then by Generation Lemma 4.4, there exists V^i, V^o, Y^i, Y^o such that $\Gamma \vdash \mathtt{new}\, x : V^i|V^o$, $\Gamma \vdash B : Y^i|Y^o$, $Z^i = V^i \cup Y^i$, $Z^o = V^o \cup Y^o$, and $V^o \cap \mathbf{E} \cap Y^i = \emptyset$. Apply the same lemma to $\Gamma \vdash \mathtt{new}\, x : V^i|V^o$ and we have $x \in V^o$, $\Gamma = \Delta, x \prec A, \Delta'$ and $\Delta \vdash A : V^i - x|V^o - x$. So $x \prec A \in \Gamma$ is proved.

    Applying Lemma 4.5 to $\Delta \subseteq \Gamma$ and $\Delta \vdash A : V^i - x|V^o - x$, we have $\Gamma \vdash A : X^i|X^o$ with $X^i = V^i - x$ and $X^o = V^o - x$.

(ii) Immediate from Generation Lemma.

(iii) By induction on structure of A.

- Base step: Case $A = \epsilon$: Then $X^i = X^o = \emptyset$ by rule Axiom $\vdash \epsilon : \emptyset|\emptyset$, Lemma 4.5, and Proposition 4.7.
- Case $A = \mathtt{new}\, y$: Since $\mathtt{new}\, y\, @B = \mathtt{new}\, y\, B$, applying rule Sequencing to $\Gamma \vdash \mathtt{new}\, y : X^i|X^o$ and $\Gamma \vdash B : Y^i|Y^o$ received in the proofs of the previous clauses we have:

$$Sequencing\ \frac{\Gamma \vdash \mathtt{new}\, y : X^i|X^o \quad \Gamma \vdash B : Y^i|Y^o}{\Gamma \vdash \mathtt{new}\, y\, B : X^i \cup Y^i|X^o \cup Y^o}\ X^o \cap \mathbf{E} \cap Y^i = \emptyset, B \neq \epsilon$$

The side condition holds since $X^o = V^o - x$ and $V^o \cap \mathbf{E} \cap Y^i = \emptyset$ also in the proof of the first clause.

- Case $A = \texttt{new}\, y\, C$ with $C \neq \epsilon$: By clause 1 we have $\Gamma \vdash \texttt{new}\, y\, C : X^i | X^o$. By Generation Lemma, we have $\Gamma \vdash \texttt{new}\, y\ : X_1^i | X_1^o$, $\Gamma \vdash C : X_2^i | X_2^o$ for some X_1^i, X_1^o, X_2^i, X_2^o such that $X^i = X_1^i \cup X_2^i$, $X^o = X_1^o \cup X_2^o$ and $X_1^o \cap \mathbf{E} \cap X_2^i = \emptyset$. By IH for clause 3 we have $\Gamma \vdash C@B : X_2^i \cup Y^i | X_2^o \cup Y^o$. By rule Sequencing we infer $\Gamma \vdash \texttt{new}\, y\, (C@B) : X_1^i \cup X_2^i \cup Y^i | X_1^o \cup X_2^o \cup Y^o$. The side condition for this rule $X_1^o \cap \mathbf{E} \cap (X_2^i \cup Y^i) = \emptyset$ holds since $X_1^o \cap \mathbf{E} \cap X_2^i = \emptyset$ holds above and $X_1^o \cap \mathbf{E} \cap Y^i = \emptyset$ holds from $X^o \cap \mathbf{E} \cap Y^i = \emptyset$ and $X_1^o \subseteq X^i$. Finally, observe that $X_1^i \cup X_2^i \cup Y^i = X^i \cup Y^i$, $X_1^o \cup X_2^o \cup Y^o = X^o \cup Y^o$, and $A@B = \texttt{new}\, y\, (C@B)$.
- Case $A = \{C\}D$: By clause 1 we have $\Gamma \vdash \{C\}D : X^i | X^o$. By Generation Lemma, we have $\Gamma \vdash C : X_1^i | X_1^o$, $\Gamma \vdash D : X_2^i | X_2^o$ for some X_1^i, X_1^o, X_2^i, X_2^o such that $X^i = X_1^i \cup X_2^i$ and $X^o = X_2^o$. By IH for clause 3 we have $\Gamma \vdash D@B : X_2^i \cup Y^i | X_2^o \cup Y^o$. By rule Scope we infer $\Gamma \vdash \{C\}(D@B) : X_1^i \cup X_2^i \cup Y^i | X_2^o \cup Y^o$. Finally, observe that $X_1^i \cup X_2^i \cup Y^i = X^i \cup Y^i$, $X_2^o \cup Y^o = X^o \cup Y^o$, and $A@B = (\{C\}D)@B = \{C\}(D@B)$.

$\square$