Counting Instances of Software Components

Marc Bezem and Hoang Truong

Department of Informatics, University of Bergen,
PB.7800, N-5020 Bergen, Norway
{bezem,hoang}@ii.uib.no

Abstract. Component software is software that has been assembled from various pieces of standardized, reusable computer programs, so-called components. Executing component software creates instances of these components. For several reasons, for example, limited resources and/or application requirements, it can be important to have control over the number of such instances. Clearly, design-time or compile-time control is to be preferred to run-time control in cases where this is possible. We give an abstract component language and a type system which ensures that the number of simultaneously active instances of any component never exceeds a (sharp) bound expressed in the type. The language features instantiation and reuse of components, as well as sequential composition, choice and scope. Alternatively one can view the objects in the language as processes where the atomic actions are interpreted as either creating new, or reusing old instances.

1 Introduction

Component software is built from various components, possibly developed by third-parties [8], [10]. These third-party components may in turn use other components. Upon execution instances of all these components are created. The process of creating an instance of a component c does not only mean the allocation of memory space for c's code and data structures, the creation of instances of c's subcomponents (and so on), but possibly also the allocation of hardware resources. In many cases, resources are limited and components are required to have only a certain number of simultaneously active instances. For example, a serial output device can usually stand only one instance of a driver-component, serialized ID generators should be unique [5], [4]. Most servers can have only a certain number of clients.

When building component software it can easily happen that, unforseen by the developer, different instances of the same component are created. Creating more active instances than allowed can lead to errors. There are several ways to meet this challenge, ranging from testing to dynamic instantiation schemes. Type systems have traditionally been used for compile-time error-checking, cf. [3]. In component software, typing has been studied in relation to integrating components such as type-safe composition [9] or type-safe evolution [7]. In this paper we explore the possibility of a type system which allows one to detect *statically*, at development/composition time, whether or not the number of simultaneously active instances of specific components exceeds the allowed number.

For this purpose we have designed a component language where we have abstracted away many aspects of components and have kept only those that are relevant to instantiation.[1] The main features we have retained are instantiation and reuse, sequential composition, choice and scope. Reusing a component means here to use an existing instance of the component if there is already one, and to create a new instance only if there exists none. Though abstract, the strength of the primitives for composition is considerable. Sequential composition is associative. Choice allows us to model both conditionals and non-determinism (due to, e.g., user input). Scope is a mechanism to deallocate instances but it can also be used to model method calls.

This paper extends [2] in three main ways. First, we generalized the single-instance property to counting instances of components. Second, we have an additional primitive for reusing instead of always creating a new instance of a component. Third, we added a choice primitive to the language, which brings the language closer to practice.

The paper is organized as follows. Section 2 introduces the component language with its operational semantics. In Section 3 we define types and the typing relation. Properties of the type system and the operational semantics are presented in Section 4. Last, we outline a polynomial time type inference algorithm in Section 5. Technical proofs of Section 4 are delegated to the appendix.

2 A Component Language

2.1 Terms

We have two primitives (`new` and `reu`) for creating and (if possible) reusing an instance of a component, and three primitives for composition (sequential composition denoted by juxtaposition, $+$ for choice and $\{\ldots\}$ for scope. Together with the empty expression ϵ these generate so-called *component expressions*. A *declaration* $c \prec Exp$ states how the component c depends on subcomponents as expressed in the component expression *Exp*. If c has no subcomponents then *Exp* is ϵ and we call c a *primitive component*. Upon instantiation or reuse of c the expression *Exp* is executed. A *component program* consist of declarations and ends with an expression which sparks off the execution, see Section 2.2.

In the formal definition below, we use extended Backus-Naur Form with the following meta-symbols: infix | for choice and underlining for Kleene closure (zero or more iterations).

Definition 1 (Syntax). *Component programs, declarations and expressions are defined by the following syntax:*

[1] This should not be misunderstood as that other aspects are deemed uninteresting!

$$
\begin{array}{lcll}
\mathit{Prog} & ::= & \mathit{Decl}; \mathit{Exp} & \text{(Program)} \\
\mathit{Decl} & ::= & \mathit{Var} \mathrel{-\!\!\prec} \mathit{Exp}\ \underline{,\ \mathit{Var} \mathrel{-\!\!\prec} \mathit{Exp}} & \text{(Declarations)} \\
\mathit{Exp} & ::= & \epsilon & \text{(Empty Expression)} \\
 & | & \mathtt{new}\, \mathit{Var} & \text{(New Instantiation)} \\
 & | & \mathtt{reu}\, \mathit{Var} & \text{(Reuse Instantiation)} \\
 & | & (\mathit{Exp} + \mathit{Exp}) & \text{(Choice)} \\
 & | & \{\mathit{Exp}\} & \text{(Scope)} \\
 & | & \mathit{Exp}\ \mathit{Exp} & \text{(Sequential Composition)}
\end{array}
$$

We use $a, b, \ldots, z$ for component names from a set *Var* and $A, \ldots, E$ for expressions *Exp*. The following example is a well-formed component program:

$$d \mathrel{-\!\!\prec} \epsilon,\ e \mathrel{-\!\!\prec} \epsilon,\ a \mathrel{-\!\!\prec} \mathtt{new}\, d,\ b \mathrel{-\!\!\prec} (\mathtt{reu}\, d\{\mathtt{new}\, a\} + \mathtt{new}\, e\, \mathtt{new}\, a)\, \mathtt{reu}\, d;\ \mathtt{new}\, b\ .$$

In this example, d and e are primitive components. Component a uses one instance of component d. Component b has a choice expression before reuse of an instance of d. The first expression of the choice expression is $\mathtt{reu}\, d\{\mathtt{new}\, a\}$. We can view $\{\mathtt{new}\, a\}$ in this expression as function call $f()$ (in traditional programming languages). Function f then has body $\mathtt{new}\, a$, which means $f()$ needs a new instance of a to do its job. We abstract from the details of this job, the only relevant aspect here is that it involves a new instance of a which will be deallocated upon exiting f.

2.2 Operational Semantics

The operational semantics is modelled as a *transition system* where a state is a pair $\Sigma \circ E$, with Σ a non-empty stack of multisets over *Var* and E a component expression. Elements of the stack are separated by : and the stack is separated from the expression by $\circ$. Stacks are pushed and popped at the right end.

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 $\cup$ is union of multisets: $(M \cup N)(x) = max(M(x), N(x))$ with $M(x) = 0$ if $x \notin M$. The operation $\uplus$ is additive union of multisets: $(M \uplus N)(x) = M(x) + N(x)$ and we write $M + x$ for $M \uplus [x]$. When $x \in M$ we write $M - x$ for $M - [x]$. The additive union of all the multisets in a stack Σ will also be denoted by Σ, that is, if $\Sigma = M_1 : \ldots : M_n$ then also $\Sigma = M_1 \uplus \ldots \uplus M_n$.

Definition 2 (Transition rules). *Σ can be empty in the rules below.*

$$\frac{x \mathrel{-\!\!\prec} E \in \mathit{Prog}}{\Sigma : M \circ \mathtt{new}\, x \to \Sigma : (M + x) \circ E}\ \text{OS-new}$$

$$\frac{x \mathrel{-\!\!\prec} E \in \mathit{Prog}, x \notin \Sigma \cup M}{\Sigma : M \circ \mathtt{reu}\, x \to \Sigma : (M + x) \circ E}\ \text{OS-reu1} \qquad \frac{x \mathrel{-\!\!\prec} E \in \mathit{Prog}, x \in \Sigma \cup M}{\Sigma : M \circ \mathtt{reu}\, x \to \Sigma : M \circ E}\ \text{OS-reu2}$$

$$\frac{}{\Sigma \circ (A + B) \to \Sigma \circ A}\ \text{OS-choice1} \qquad \frac{}{\Sigma \circ (A + B) \to \Sigma \circ B}\ \text{OS-choice2}$$

$$\frac{\Sigma : [\,] \circ E \to \Sigma : M \circ \epsilon}{\Sigma \circ \{E\} \to \Sigma \circ \epsilon}\text{OS-scope} \qquad \frac{\Sigma : M \circ E \to \Sigma : M' \circ \epsilon}{\Sigma : M \circ EA \to \Sigma : M' \circ A}\text{OS-seq}$$

$$\frac{\Sigma \circ E \to \Sigma' \circ E' \quad \Sigma' \circ E' \to \Sigma'' \circ E''}{\Sigma \circ E \to \Sigma'' \circ E''}\text{OS-trans} \ .$$

The meaning of these transition rules can be described as follows. Rule OS-new adds an instance of the component x declared by $x \prec E$ to the multiset on top of the stack and starts executing E. Rule OS-reu1 does the same as OS-new as the component to be reused doesn't occur in the stack. OS-reu2 applies if component x does occur already in the stack with the effect that E is executed without adding x to the top of the stack. For the (inductive) scope rule, an empty multiset is pushed on the stack after which the expression between the scope delimiters is executed. If this execution terminates successfully, the scoped expression has been executed without changing the original stack. The rules OS-choice1,2 and OS-seq and OS-trans are self-explaining.

The example at the end of Section 2.1 can be used to illustrate the operational semantics. A formal derivation tree could be built using the Definition 2 but here we just show the main transitions. First, when executing $\mathtt{new}\, b$, an instance of b is added to the empty multiset on top of the stack and the execution continues using the declaring expression for b:

$[\,] \circ \mathtt{new}\, b \to [b] \circ (\,\mathtt{reu}\, d\{\,\mathtt{new}\, a\} + \mathtt{new}\, e\, \mathtt{new}\, a)\, \mathtt{reu}\, d \qquad (*)$

Now there are two options. If we chose rule OS-choice1 in $(*)$ we get:

$[b] \circ \mathtt{reu}\, d\{\,\mathtt{new}\, a\}\, \mathtt{reu}\, d \to [b, d] \circ \{\,\mathtt{new}\, a\}\, \mathtt{reu}\, d$

Now we meet a scoped expression and have to execute the expression $\mathtt{new}\, a$ inside the scope with an new empty multiset pushed on top of the stack, as a subsidiary derivation:

$[b, d] : [] \circ \mathtt{new}\, a \to [b, d] : [a] \circ \mathtt{new}\, d \to [b, d] : [a, d] \circ \epsilon$

Note the two instances of d here. By rules OS-scope and OS-reu2 we have:

$[b, d] \circ \{\,\mathtt{new}\, a\}\, \mathtt{reu}\, d \to [b, d] \circ \mathtt{reu}\, d \to [b, d] \circ \epsilon$

If we chose rule OS-choice2 in $(*)$ we proceed as follows:

$[b] \circ \mathtt{new}\, e\, \mathtt{new}\, a\, \mathtt{reu}\, d \to [b, e] \circ \mathtt{new}\, a\, \mathtt{reu}\, d \to [b, e, a] \circ \mathtt{new}\, d\, \mathtt{reu}\, d \to$
$[b, e, a, d] \circ \mathtt{reu}\, d \to [b, e, a, d] \circ \epsilon.$

In this example there are two possible runs and the numbers of active instances of each component are not the same during and at the end of the two runs. There are two $\mathtt{reu}\, d$'s in the above execution and only the first one creates an instance of d. The maximum for d is 2, for the others 1.

3 Type System

3.1 Types

We partition the set of all components $\mathbf{C} = \mathit{Var}$ into classes $\mathbf{C}_0, \ldots, \mathbf{C}_n$ such that each component in $\mathbf{C}_0$ can have an arbitrary number of active instances and each component in $\mathbf{C}_i$ with $i = 1..n$ can have at most i instances at a time. So $\mathbf{C} = \mathbf{C}_0 \cup \cdots \cup \mathbf{C}_n$ and $\mathbf{C}_i \cap \mathbf{C}_j = \emptyset$ for $0 \leq i < j \leq n$. Note that $\mathbf{C}_i$ may be empty for some i.

Definition 3 (Types). *Types of component expressions are quadruples* $X = \langle X^i, X^o, X^j, X^p \rangle$ *where* X^i, X^o, X^j *and* X^p *are finite multisets over* $\mathbf{C}$. *We let* $U, V, \ldots, Z$ *range over types.*

Let us first explain informally why multisets, which multisets and why four. The aim is to have a sharp upper bound of the number of simultaneously active instances of any component during the execution of the expression (X^i). Multisets are the right datastructure to collect and count such instances. In addition we want compositionality of typing, that is, we want the types to be computable from types of subexpressions. Since subexpressions may be scoped, it is necessary to have an sharp upper bound of the number of instances that are still active *after* the execution of an expression (X^o). Pairs $\langle X^i, X^o \rangle$ sufficed for the purpose of the paper [2]. Here we consider also reusing instances of components and this depends on whether there is already such an instance or not. More concretely, in a sequential composition EE' the behaviour of `reu`'s in E' depends on the instances that are active *after* the execution of E, which would violate compositionality. In order to save compositionality, we have to add more two more multisets to the types, denoted by X^j, X^p. These express the same bounds as X^i, X^o, but with respect to executing the expression in a state where every component has already one active instance. Finally, we have to explain the informal phrase 'sharp upper bound'. Since we have choice, there can be different runs of the same expression, with different numbers of active instances. Now 'upper bound' means an upper bound with respect to all possible runs and 'sharp' means that the upper bound is attained in at least one such run.

Based on the above intuitions, the following typings are easy:
$\mathtt{new}\,d : \langle [d], [d], [d], [d] \rangle$, $\{\mathtt{new}\,d\} : \langle [d], [], [d], [] \rangle$, $\mathtt{reu}\,d : \langle [d], [d], [], [] \rangle$,
$\mathtt{reu}\,d\{\mathtt{new}\,d\} : \langle [d,d], [d], [d], [] \rangle$, $\mathtt{reu}\,d\{\mathtt{new}\,a\} : \langle [a,d,d], [d], [a,d], [] \rangle$,
where $d \prec \epsilon$ and $a \prec \mathtt{new}\,d$ like in the example program in Section 2.1.

The intuitions from the above paragraph will be indispensable for understanding the typing rules later in this section, in particular the sequencing rule, but we have to prepare with some preliminary definitions.

A *basis* or an *environment* is an list of declarations: $x_1 \prec A_1, \ldots, x_n \prec A_n$ with distinct variables $x_i \neq x_j$ for all $i \neq j$, as in [1]. Let $\Gamma, \Delta, \ldots$ range over bases. When $\Gamma = x_1 \prec A_1, \ldots, x_n \prec A_n$, the set of variables $x_1, \ldots, x_n$ declared in Γ is the domain of Γ and is denoted by $Dom(\Gamma)$. A typing judgment is a triple of the form

$$\Gamma \vdash A : X$$

and it asserts that expression A has type X in the environment Γ. We write $\vdash A : X$ if there exists a Γ such that $\Gamma \vdash A : X$.

Notation: for types X and Y, let $X \subseteq Y$, $X + Y$ and $X \cup Y$ denote multiset inclusion, additive union and usual union, respectively, all component-wise. For any expression E, let $Var(E)$ denote the set of variables occurring in E.

Formal Typing Rules. Having built up some intuition about types for component expressions in the previous section we can now give formal typing rules.

Definition 4 (Typing rules). *Typing judgments $\Gamma \vdash A : X$ are derived by the following typing rules:*

$$\mathsf{Axiom}\ \frac{}{\vdash \epsilon : \langle [\,],[\,],[\,],[\,]\rangle} \qquad \mathsf{Weaken}\ \frac{\Gamma \vdash A : X \quad \Gamma \vdash B : Y \quad x \notin Dom(\Gamma)}{\Gamma, x \prec B \vdash A : X}$$

$$\mathsf{New}\ \frac{\Gamma \vdash A : X \quad x \notin Dom(\Gamma)}{\Gamma, x \prec A \vdash \mathtt{new}\,x : \langle X^i + x, X^o + x, X^j + x, X^p + x\rangle}$$

$$\mathsf{Reu}\ \frac{\Gamma \vdash A : X \quad x \notin Dom(\Gamma)}{\Gamma, x \prec A \vdash \mathtt{reu}\,x : \langle X^i + x, X^o + x, X^j, X^p\rangle}$$

$$\mathsf{Seq}\ \frac{\Gamma \vdash A : X \quad \Gamma \vdash B : Y \quad \forall k = 1..n. \forall c \in \mathbf{C}_k.(X^o \uplus Y^j)(c) \le k \quad A, B \neq \epsilon}{\Gamma \vdash AB : \langle X^i \cup (X^o \uplus Y^j) \cup Y^i, (X^o \uplus Y^p) \cup Y^o, X^j \cup (X^p \uplus Y^j), X^p \uplus Y^p\rangle}$$

$$\mathsf{Choice}\ \frac{\Gamma \vdash A : X \quad \Gamma \vdash B : Y}{\Gamma \vdash (A + B) : X \cup Y} \qquad \mathsf{Scope}\ \frac{\Gamma \vdash A : X}{\Gamma \vdash \{A\} : \langle X^i, [\,], X^j, [\,]\rangle}\ .$$

Besides the intuition given in the beginning of this section, some further explanation of these typing rules is in order. Rule Axiom requires no premise and is used to take-off. Rules New and Reu allow us to type expressions $\mathtt{new}\,x$ and $\mathtt{reu}\,x$, respectively. Weakening is used to expand bases so that we can combine typings in other rules. The side condition $x \notin Dom(\Gamma)$ prevents ambiguity and circularity. Rules Choice and Scope are easy to understand recalling the corresponding rules OS-choice and OS-scope of the operational semantics.

The most critical rule is Seq because sequencing two expressions can lead to increase in instances of the composed expression. Let us start with the first and the third component of type expression for AB. After expression A is executed, there are at most $X^o(x)$ instances of component x. Executing B can create at most $Y^i(x)$ instances of x if x is not in system state which is X^o. Otherwise $Y^j(x)$ instances of x will be created, meaning that there are at most $((X^o \uplus Y^j) \cup Y^i)(x)$ instances of x after the execution of A and during the execution of B. So we require the side condition $X^o(x) + Y^j(x) \le k$ for each $x \in \mathbf{C}_k$. In addition, because during executing A there are at most $X^i(x)$ instances of x created, the first component of type of AB is the maximum of $X^i(x)$ and $((X^o \uplus Y^j) \cup Y^i)(x)$. After executing AB it is easy to see that the surviving instances are total of those from A and B if we start from state with no instance of any component.

By similar reasoning when we start with a stack containing at least one instance of every component we can calculate the second and the last components in the type expression for AB and the whole type expression of AB is $\langle X^i \cup (X^o \uplus Y^j) \cup Y^i, (X^o \uplus Y^p) \cup Y^o, X^j \cup (X^p \uplus Y^j), X^p \uplus Y^p\rangle$.

Using the example in Section 2.1 with assumption that $\mathbf{C}_0 = \{c, d\}$, $\mathbf{C}_2 = \{a, b\}$, $\mathbf{C}_3 = \{e\}$, we derive type for $\mathtt{new}\,b$. Note that we omitted some side conditions as they can be checked easily and we shortened the rule names. Rule Axiom is simplified. Also $\Gamma = d \prec \epsilon, a \prec \mathtt{new}\,d, e \prec \epsilon$ and $\Gamma' = \Gamma, b \prec (\mathtt{reu}\,d\{\mathtt{new}\,a\} + \mathtt{new}\,e\,\mathtt{new}\,a)\,\mathtt{reu}\,d$ in the following examples.

$$\mathsf{Wea}\ \frac{\mathsf{New}\ \dfrac{\vdash \epsilon : \langle [\,],[\,],[\,],[\,]\rangle}{d \prec \epsilon \vdash \mathtt{reu}\,d : \langle [d],[d],[\,],[\,]\rangle} \quad \mathsf{New}\ \dfrac{\vdash \epsilon : \langle [\,],[\,],[\,],[\,]\rangle}{d \prec \epsilon \vdash \mathtt{new}\,d : \langle [d],[d],[d],[d]\rangle}}{d \prec \epsilon, a \prec \mathtt{new}\,d \vdash \mathtt{reu}\,d : \langle [d],[d],[\,],[\,]\rangle}$$

$$\mathsf{Sco}\ \dfrac{\mathsf{New}\ \dfrac{\mathsf{New}\ \dfrac{\vdash \epsilon : \langle [\,], [\,], [\,], [\,] \rangle}{d \prec \epsilon \vdash \mathtt{new}\, d : \langle [d], [d], [d], [d] \rangle}}{d \prec \epsilon, a \prec \mathtt{new}\, d \vdash \mathtt{new}\, a : \langle [a,d], [a,d], [a,d], [a,d] \rangle}}{d \prec \epsilon, a \prec \mathtt{new}\, d \vdash \{\, \mathtt{new}\, a\} : \langle [a,d], [\,], [a,d], [\,] \rangle}$$

Sequencing the above two derivation we have:
$d \prec \epsilon, a \prec \mathtt{new}\, d \vdash \mathtt{reu}\, d\{\mathtt{new}\, a\} : \langle [a,d,d], [d], [a,d], [] \rangle$.
We can weaken the above derivation to get:
$\Gamma \vdash \mathtt{reu}\, d\{\mathtt{new}\, a\} : \langle [a,d,d], [d], [a,d], [] \rangle$ We can also derive:

$$\mathsf{Seq}\ \dfrac{\dfrac{\cdots}{\Gamma \vdash \mathtt{new}\, e : \langle [e], [e], [e], [e] \rangle} \quad \dfrac{\cdots}{\Gamma \vdash \mathtt{new}\, a : \langle [a,d], [a,d], [a,d], [a,d] \rangle}}{\Gamma \vdash \mathtt{new}\, e\, \mathtt{new}\, a : \langle [a,d,e], [a,d,e], [a,d,e], [a,d,e] \rangle}$$

and we have: $\Gamma' \vdash \mathtt{new}\, b : \langle [a,b,d,d,e], [a,b,d,e], [a,b,d,e], [a,b,d,e] \rangle$.

In this example expression $\mathtt{new}\, b$ is typable. If $d \in \mathbf{C}_1$, the expression would not be typable as the side condition when sequencing $\mathtt{reu}\, d$ and $\{\, \mathtt{new}\, a\}$ would not be satisfied. Also, note that the above type derivation is not the only one but, as we will see later, the type for any expression is unique.

4 Properties

We start by giving some definitions and then state some properties of our type system. After that we will state some important properties relating types to states in the operational semantics. Proofs are delegated to Appendix A to improve the readability of this section.

Following [1] we fix some terminology on bases or environments.

Definition 5 (Bases). *Let $\Gamma = x_1 \prec A_1, \ldots, x_n \prec A_n$ be a basis.*

- *Γ is called* legal *if $\Gamma \vdash A : X$ for some expression A and type X.*
- *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 $\Delta = x_{i_1} \prec A_{i_1}, \ldots, x_{i_k} \prec A_{i_k}$ with $1 \leq i_1 < \ldots < i_k \leq n$. Note that the order is preserved.*
- *Δ is an* initial segment *of Γ, if $\Delta = x_1 \prec A_1, \ldots, x_j \prec A_j$ for some $1 \leq j \leq n$.*

In the sequel we assume that we are working with a well-typed program *Prog* and the set $\mathbf{C}$ of all components of this program are partitioned into n classes $\mathbf{C}_0, \mathbf{C}_1, \ldots, \mathbf{C}_n$ such that each component in $\mathbf{C}_i$ can have at most i instances for all $1 \leq i \leq n$ and each components in $\mathbf{C}_0$ can have any number of active instances.

The following lemma collects a number of simple properties of a typing judgment. It states that if $\Gamma \vdash A : X$, then the elements of each multiset of X and variables of A is in domain of Γ. It also shows some relations among multisets of A and any legal basis always has distinct declarations.

Lemma 1 (Legal typing). *If $\Gamma \vdash A : X$, then*

1. *elements of $Var(A)$, X^i, X^o, X^j and X^p are in $Dom(\Gamma)$,*

2. $\Gamma \vdash \epsilon : \langle [\,], [\,], [\,], [\,] \rangle$,
3. *every variable in* $Dom(\Gamma)$ *is declared only once in* Γ,
4. $\forall k = 1..n. \forall c \in \mathbf{C}_k.\ X^o(c) \leq X^i(c) \leq k, X^p(c) \leq X^j(c) \leq k$,
5. $\forall k = 1..n. \forall c \in \mathbf{C}_k.\ 0 \leq X^i(c) - X^j(c), X^o(c) - X^p(c) \leq 1$.

The following lemma is important in that it 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* [6]. Note that in the third clause the sequential decomposition in A and B may not be unique.

Lemma 2 (Generation).

1. *If* $\Gamma \vdash \mathtt{new}\,x : X$, *then* $x \in X^p$ *and there exists bases* Δ, Δ' *and expression* A *such that* $\Gamma = \Delta, x \prec A, \Delta'$, *and* $\Delta \vdash A : \langle X^i - x, X^o - x, X^j - x, X^p - x \rangle$.
2. *If* $\Gamma \vdash \mathtt{reu}\,x : X$, *then* $x \in X^o$ *and there exists bases* Δ, Δ' *and expression* A *such that* $\Gamma = \Delta, x \prec A, \Delta'$, *and* $\Delta \vdash A : \langle X^i - x, X^o - x, X^j, X^p \rangle$.
3. *If* $\Gamma \vdash AB : Z$ *with* $A, B \neq \epsilon$, *then there exists* X, Y *such that* $\Gamma \vdash A : X$, $\Gamma \vdash B : Y$, $Z = \langle X^i \cup (X^o \uplus Y^j) \cup Y^i, (X^o \uplus Y^p) \cup Y^o, X^j \cup (X^p \uplus Y^j), X^p \uplus Y^p \rangle$.
4. *If* $\Gamma \vdash (A + B) : Z$, *then there exists* X, Y *such that* $\Gamma \vdash A : X$, $\Gamma \vdash B : Y$ *and* $Z = X \cup Y$.
5. *If* $\Gamma \vdash \{A\} : \langle X^i, [\,], X^j, [\,] \rangle$, *then there exists multisets* X^o *and* X^p *such that* $\Gamma \vdash A : \langle X^i, X^o, X^j, X^p \rangle$.

The next lemma stresses the significance of the order of declarations in a legal basis in our type system. 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.

Lemma 3 (Legal monotonicity).

1. *If* $\Gamma = \Delta, x \prec E, \Delta'$ *is legal, then* $\Delta \vdash E : X$ *for some* X.
2. *If* $\Gamma \vdash E : X$, $\Gamma \subseteq \Gamma'$ *and* Γ' *is legal, then* $\Gamma' \vdash E : X$.

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

Lemma 4 (Strengthening). *If* $\Gamma, x \prec A \vdash B : Y$ *and* $x \notin Var(B)$, *then* $\Gamma \vdash B : Y$ *and* $x \notin Y^i$.

In our type system, when an expression has a type this type is unique. This property is stated in the following proposition.

Proposition 1 (Uniqueness of types). *If* $\Gamma \vdash A : X$ *and* $\Gamma \vdash A : Y$, *then* $X^i = Y^i$, $X^o = Y^o$, $X^j = Y^j$ *and* $X^p = Y^p$.

Now we state an important invariant of our operational semantics. During transition the total of instances in the stack does not reduce and the type expression does not increase. Moreover, relations between types and stacks of transitions in derivations allow us to prove the safety property afterwards. Recall that the additive union of all the multisets in a stack Σ will also be denoted by Σ.

Theorem 1 (Invariant of operational semantics). *Let $\Gamma \vdash C : Z$. Then we have for any derivation Δ of a transition $\Theta \circ C \to \Theta' \circ C'$ and any transition $\Sigma \circ A \to \Sigma' \circ B$ occurring in Δ (including the last!) that $\Gamma \vdash A : X$ and $\Gamma \vdash B : Y$ for types X, Y such that:*

– *$Y \subseteq X$, $\Sigma \subseteq \Sigma'$ and*

$$\Theta \uplus Z^i \supseteq \Sigma \uplus X^j \tag{1}$$

$$\Theta \uplus Z^i \supseteq \Sigma' \uplus Y^j \tag{2}$$

– *for any $c \notin \Sigma$:*

$$X^i(c) \geq (\Sigma' \uplus Y^j)(c) \tag{3}$$

$$X^o(c) \geq (\Sigma' \uplus Y^p)(c) \tag{4}$$

– *for any $c \in \Sigma$:*

$$(\Sigma \uplus X^j)(c) \geq (\Sigma' \uplus Y^j)(c) \tag{5}$$

$$(\Sigma \uplus X^p)(c) \geq (\Sigma' \uplus Y^p)(c) \tag{6}$$

Note that in inequality (1) of Theorem 1 (and similarly in other inequalities (2)-(6)) we have X^j, not X^i, in the right hand side. This is because, considering $[]\circ\, \mathtt{new}\, d\, \mathtt{reu}\, d \to [d]\circ\, \mathtt{reu}\, d$ with $d \prec \epsilon$, if the right hand side is X^i then $\Sigma \uplus Z^i = Z^i = [d] \subset \Sigma' \uplus X^i = [d] \uplus [d]$.

As a special case of this theorem the following corollary allows us to safely execute well-typed component programs. That is, during the execution of the programs the number of active instances of any component never exceeds the allowed number.

Corollary 1 (Safety). *Let $\Gamma \vdash C : Z$. Then for every transition $[\,]\circ C \to \Theta' \circ C'$ we have for any state $\Sigma \circ A$ occurring in the derivation of this transition (including $\Theta' \circ C'$!) that $k \geq Z^i(c) \geq \Sigma(c)$, where k is such that $c \in \mathbf{C}_k$.*

Proof. By Lemma 1 we have $k \geq Z^i(c)$ for all k and $c \in \mathbf{C}_k$. From Inequalities (1) and (2) of Theorem 1 we have $Z^i(c) \geq \Sigma(c)$ for all c.

5 Type Inference

So far we know that a well-typed program is safe to execute. Now given a well-formed program, if we know the type of its starting expression, then we know whether the program is safe to execute. The problem of finding a type/derivation of an expression, given a set of declarations, is the *type inference problem* [3] or typability problem [1]. Solving this problem relieves programmers from giving the types explicitly and having them checked. Types inferred also give information about component programs such as memory, resources they may use and hence guide the design of the component system.

One may argue that we can test the safe instantiation of a component program by executing all possible runs under our operational semantics. However, this process could be exponential or even non-terminating (in the case of unforeseen circular dependencies.)

Now let us see a solution for our type inference problem. Let *Prog* be the component program and E be the expression we need to find the type of. 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:

$$\text{if } \Gamma = \Delta, x \prec A, \Delta' \text{ then } \mathit{Var}(A) \subseteq \mathit{Dom}(\Delta) \tag{7}$$

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 (7). The considerations below are independent of which particular ordering is used as long as it satisfies (7).

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 2 reversely.

We can break down the problem of finding type for E by finding types of $\mathtt{new}\,x$ and $\mathtt{reu}\,x$ for all $x \in \mathit{Var}(E)$. Why? First of all we can recursively break down expression E into $E_1, \ldots, E_p$ for some p such that E_i is one of the forms: $\mathtt{new}\,x$, $\mathtt{reu}\,x$, $(\mathit{Exp} + \mathit{Exp})$, $\{\mathit{Exp}\}$ and $E = E_1 \ldots E_p$. By Definition 4 we can easily calculate type of E if we know types of all E_i. Moreover, $\mathit{Var}(E_i) \subseteq \mathit{Var}(E)$ so if we know types of $\mathtt{new}\,x$ for all $x \in \mathit{Var}(E)$ we can calculate types of E_i by doing few multisets operations in Definition 4. The type inference problem for E now becomes type inference problems of $\mathtt{new}\,x$ and $\mathtt{reu}\,x$ for all $x \in \mathit{Var}(E)$.

To find the type of $\mathtt{new}\,x$ or $\mathtt{reu}\,x$, we can 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.

References

1. H. Barendregt. Lambda Calculi with Types. In: Abramsky, Gabbay, Maibaum (Eds.), *Handbook of Logic in Computer Science*, Vol. II. Oxford University Press. 1992.
2. M. Bezem and H. Truong. A Type System for the Safe Instantiation of Components, In *Proceedings of FOCLASA '03*, Electronic Notes in Theoretical Computer Science, September 2003.
3. L. Cardelli. Type systems. In A. B. Tucker, editor, *The Computer Science and Engineering Handbook*, chapter 103, pages 2208-2236. CRC Press, 1997.
4. E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns - Elements of Reusable ObjectOriented Software*, Addison-Wesley, Reading, Mass., ISBN 0201633612, 1994.
5. E. Meijer and C. Szyperski. Overcoming Independent Extensibility Challenges, *Communications of the ACM*, Vol. 45, No. 10, pp. 41–44, October 2002.
6. B. Pierce. *Types and Programming Languages.* MIT Press, ISBN 0262162091, February 2002.
7. J. C. Seco, Adding Type Safety to Component Programming, In *Proceedings of The PhD Student's Workshop* in FMOODS'02, University of Twente, the Netherlands, March 2002.
8. C. Szyperski. *Component Software: Beyond Object-Oriented Programming*, 2nd edition, Addison-Wesley, ISBN 0201745720, 2002.
9. M. Zenger, Type-Safe Prototype-Based Component Evolution, *Proceedings of the European Conference on Object-Oriented Programming*, Malaga, Spain, June 2002.
10. M. Zenger, Programming Language Abstractions for Extensible Software Components, PhD Thesis, No. 2930, EPFL, Switzerland, March 2004.

A Proofs

In the sequel we use X^* for any of X^i, X^o, X^j and X^p.

Lemma 1 (Legal typing). By simultaneous induction on derivation. Recall that Lemma 1 has 5 clauses.

- Base case Axiom $\vdash \epsilon : \langle [], [], [], [] \rangle$ is trivial as $Var(\epsilon), X^i, X^o, X^j, X^p, Dom()$ are empty.
- Case Weaken

$$\mathsf{Weaken}\ \frac{\Gamma' \vdash A : X \quad \Gamma' \vdash B : Y \quad x \notin Dom(\Gamma')}{\Gamma', x \prec B \vdash A : X}$$

  Clause 3 follows by the side condition. The remaining clauses follow by IH.
- Case New

$$\mathsf{New}\ \frac{\Gamma' \vdash B : Y \quad x \notin Dom(\Gamma')}{\Gamma', x \prec B \vdash \mathtt{new}\, x : \langle Y^i + x, Y^o + x, Y^j + x, Y^p + x \rangle}$$

  with $\Gamma = \Gamma', x \prec B$, $X^* = Y^* + x$. Assume the lemma is correct for the premise of this rule, so elements of $Var(B)$, Y^* are in $Dom(\Gamma')$. Clause 1 holds easily as the new element x in $Var(\texttt{new}\,x)$ and X^* is in $Dom(\Gamma) = Dom(\Gamma', x \prec B) = Dom(\Gamma') \cup x$. Clause 2 $\Gamma', x \prec B \vdash \epsilon : \langle [\,], [\,], [\,], [\,] \rangle$ follows by applying Weaken:

$$\mathsf{Weaken}\ \frac{\Gamma' \vdash \epsilon : \langle [\,], [\,], [\,], [\,] \rangle \quad \Gamma' \vdash B : Y \quad x \notin Dom(\Gamma')}{\Gamma', x \prec B \vdash \epsilon : \langle [\,], [\,], [\,], [\,] \rangle}$$

Clause 3 follows by the side condition $x \notin Dom(\Gamma')$. Clause 4 follows by IH. The last clause 5 holds since $x \notin Y^i$ and $x \notin Y^j$.

– Case Reu

$$\mathsf{Reu}\ \frac{\Gamma' \vdash B : Y \quad x \notin Dom(\Gamma')}{\Gamma, x \prec B \vdash \texttt{reu}\,x : \langle Y^i + x, Y^o + x, Y^j, Y^p \rangle}$$

with $\Gamma = \Gamma', x \prec B$, $X = \langle Y^i + x, Y^o + x, Y^j, Y^p \rangle$. The proof is analogous to case New.

– Case Seq

$$\mathsf{Seq}\ \frac{\Gamma \vdash B : Y \quad \Gamma \vdash C : Z \quad \forall k = 1..n. \forall c \in \mathbf{C}_k. (Y^o \uplus Z^j)(c) \leq k \quad B, C \neq \epsilon}{\Gamma \vdash BC : \langle Y^i \cup (Y^o \uplus Z^j) \cup Z^i, (Y^o \uplus Z^p) \cup Z^o, Y^j \cup (Y^p \uplus Z^j), Y^p \uplus Z^p \rangle}$$

Clauses 1, 2 and 3 hold by IH. For clause 4 it is to see that: $((Y^o \uplus Z^p) \cup Z^o)(c) \leq (Y^i \cup (Y^o \uplus Z^j) \cup Z^i)(c) \leq k$ holds since $Z^p \subseteq Z^j$, $Z^o \subseteq Z^i$ and $(Y^o \uplus Z^j)(c) \leq k$ from side condition, Y^i, $Z^i \leq k$ by IH. Similarly, $(Y^p \uplus Z^p)(c) \leq (Y^j \cup (Y^p \uplus Z^j))(c) \leq k$ holds since $Z^p \subseteq Z^j$, for all k and $c \in \mathbf{C}_k$, and $(Y^p \uplus Z^j)(c) \leq (Y^o \uplus Z^j)(c) \leq k$. For clause 5, as $Y^i(c) \geq Y^j(c)$ and $Z^o(c) \geq Z^p(c)$ for all c, we get $0 \leq X^i(c) - X^j(c)$ immediately. In addition,

$$X^i(c) - X^j(c) = max \left\{ \begin{array}{l} Y^i(c) - (Y^j \cup (Y^p \uplus Z^j))(c), \\ (Y^o \uplus Z^j)(c) - (Y^j \cup (Y^p \uplus Z^j))(c), \\ Z^i(c) - (Y^j \cup (Y^p \uplus Z^j))(c) \end{array} \right\}$$

each of the three cases is less then or equals 1 so $X^i(c) - X^j(c) \leq 1$. Similarly, it is easy to see that $0 \leq X^o(c) - X^p(c) = (Y^o \uplus Z^p) \cup Z^o)(c) - (Y^p \uplus Z^p)(c) \leq 1$.

– Case Choice

$$\mathsf{Choice}\ \frac{\Gamma \vdash C : Z \quad \Gamma \vdash B : Y}{\Gamma \vdash (C + B) : Z \cup Y}$$

Analogous to case Seq. First three clauses are easy. Clause 4 holds because $max(Z^o(c), Y^o(c)) \leq max(Z^i(c), Y^i(c)) \leq k$ by IH.

– Case Scope:

$$\mathsf{Scope}\ \frac{\Gamma \vdash B : Y}{\Gamma \vdash \{B\} : \langle Y^i, [\,], Y^j, [\,] \rangle}$$

All clauses hold by IH.

□

Lemma 2 (Generation). By induction on derivation. Recall that the Generation Lemma has 5 clauses.

1. $\Gamma \vdash \texttt{new}\,x : X$ can only be derived by rule New or Weaken. If it is derived by rule New, then there is only one possibility:

$$\mathsf{New}\ \frac{\Delta \vdash A : Y \quad x \notin Dom(\Delta)}{\Delta, x \prec A \vdash \texttt{new}\,x : X}$$

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

If $\Gamma \vdash \mathtt{new}\, x : X$ is derived by rule Weaken:

$$\mathsf{Weaken}\ \frac{\Gamma' \vdash \mathtt{new}\, x : X \quad \Gamma' \vdash B : Y \quad y \notin Dom(\Gamma')}{\Gamma', y \prec B \vdash \mathtt{new}\, x : X}$$

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

2. Case $\Gamma \vdash \mathtt{reu}\, x : X$: analogous to clause 1.
3. $\Gamma \vdash AB : Z$ with $A, B \neq \epsilon$ can only be derived by rule Seq or rule Weaken. If $\Gamma \vdash AB : Z$ is derived by rule Seq with two component expressions A and B in the premise of the typing rule:

$$\mathsf{Seq}\ \frac{\Gamma \vdash A : X \quad \Gamma \vdash B : Y \quad \forall k = 1..n. \forall c \in \mathbf{C}_k.(X^o \uplus Y^j)(c) \leq k \quad A, B \neq \epsilon}{\Gamma \vdash AB : \langle X^i \cup (X^o \uplus Y^j) \cup Y^i, (X^o \uplus Y^p) \cup Y^o, X^j \cup (X^p \uplus Y^j), X^p \uplus Y^p\rangle}$$

then the proof is immediate.
If $\Gamma \vdash AB : Z$ is derived by rule Seq with two component expressions $A_1 \neq A$ and $B_1 \neq B$ such that $A_1 B_1 = AB$:

$$\mathsf{Seq}\ \frac{\begin{array}{c}\Gamma \vdash A_1 : X_1 \quad \Gamma \vdash B_1 : Y_1 \quad A_1, B_1 \neq \epsilon \\ \forall k = 1..n. \forall c \in \mathbf{C}_k.(X_1^o \uplus Y_1^j)(c) \leq k\end{array}}{\Gamma \vdash A_1 B_1 : \langle X_1^i \cup (X_1^o \uplus Y_1^j) \cup Y_1^i, (X_1^o \uplus Y_1^p) \cup Y_1^o, X_1^j \cup (X_1^p \uplus Y_1^j), X_1^p \uplus Y_1^p\rangle}$$

then there are two possibilities:
– $A = A_1 A_2$: then $B_1 = A_2 B$ and we have $\Gamma \vdash A_2 B : Y_1$.
By the IH applied to $\Gamma \vdash A_2 B : Y_1$ we get $\Gamma \vdash A_2 : X_2$ and $\Gamma \vdash B : Y$ with $Y_1 = \langle X_2^i \cup (X_2^o \uplus Y^j) \cup Y^i, (X_2^o \uplus Y^p) \cup Y^o, X_2^j \cup (X_2^p \uplus Y^j), X_2^p \uplus Y^p\rangle$. As the side condition $\forall k = 1..n. \forall c \in \mathbf{C}_k.(X_1^o \uplus X_2^j)(c) \leq (X_1^o \uplus (X_2^j \cup (X_2^p \uplus Y^j)))(c) = (X_1^o \uplus Y_1^j)(c) \leq k$ holds, we can apply rule Seq to $\Gamma \vdash A_1 : X_1$ and $\Gamma \vdash A_2 : X_2$ and get $\Gamma \vdash A : X$ with $X = \langle X_1^i \cup (X_1^o \uplus X_2^j) \cup X_2^i, (X_1^o \uplus X_2^p) \cup X_2^o, X_1^j \cup (X_1^p \uplus X_2^j), X_1^p \uplus X_2^p\rangle$. We still need to show that $Z = \langle X^i \cup (X^o \uplus Y^j) \cup Y^i, (X^o \uplus Y^p) \cup Y^o, X^j \cup (X^p \uplus Y^j), X^p \uplus Y^p\rangle$, that is we need to prove four equations:

$$\begin{array}{l} X^i \cup (X^o \uplus Y^j) \cup Y^i = X_1^i \cup (X_1^o \uplus Y_1^j) \cup Y_1^i, \\ (X^o \uplus Y^p) \cup Y^o = (X_1^o \uplus Y_1^p) \cup Y_1^o, \\ X^j \cup (X^p \uplus Y^j) = X_1^j \cup (X_1^p \uplus Y_1^j), \\ X^p \uplus Y^p = X_1^p \uplus Y_1^p \end{array}$$

We have:

$$\begin{array}{l} X^i \cup (X^o \uplus Y^j) \cup Y^i \\ = (X_1^i \cup (X_1^o \uplus X_2^j) \cup X_2^i) \cup (((X_1^o \uplus X_2^p) \cup X_2^o) \uplus Y^j) \cup Y^i \\ = X_1^i \cup X_2^i \cup Y^i \cup (X_1^o \uplus X_2^j) \cup (((X_1^o \uplus X_2^p) \cup X_2^o) \uplus Y^j) \\ = X_1^i \cup X_2^i \cup Y^i \cup (X_1^o \uplus X_2^j) \cup (X_1^o \uplus X_2^p \uplus Y^j) \cup (X_2^o \uplus Y^j) \\ = X_1^i \cup X_2^i \cup Y^i \cup (X_1^o \uplus (X_2^j \cup (X_2^p \uplus Y^j))) \cup (X_2^o \uplus Y^j) \\ = X_1^i \cup (X_1^o \uplus (X_2^j \cup (X_2^p \uplus Y^j))) \cup (X_2^i \cup (X_2^o \uplus Y^j) \cup Y^i) \\ = X_1^i \cup (X_1^o \uplus Y_1^j) \cup Y_1^i \end{array}$$

so the first equation holds. Similarly,

$$\begin{array}{l} (X^o \uplus Y^p) \cup Y^o \\ = (((X_1^o \uplus X_2^p) \cup X_2^o) \uplus Y^p) \cup Y^o \\ = (X_1^o \uplus X_2^p \uplus Y^p) \cup (X_2^o \uplus Y^p) \cup Y^o \\ = (X_1^o \uplus (X_2^p \uplus Y^p)) \cup ((X_2^o \uplus Y^p) \cup Y^o) \\ = (X_1^o \uplus Y_1^p) \cup Y_1^o \end{array}$$

so the second equation holds.

$$\begin{aligned}
&X^j \cup (X^p \uplus Y^j) \\
&= (X_1^j \cup (X_1^p \uplus X_2^j)) \cup ((X_1^p \uplus X_2^p) \uplus Y^j) \\
&= X_1^j \cup (X_1^p \uplus X_2^j) \cup (X_1^p \uplus X_2^p \uplus Y^j) \\
&= X_1^j \cup (X_1^p \uplus (X_2^j \cup (X_2^p \uplus Y^j))) \\
&= X_1^j \cup (X_1^p \uplus Y_1^j)
\end{aligned}$$

so the third equation holds. The last equation follows easily:

$$X^p \uplus Y^p = (X_1^p \uplus X_2^p) \uplus Y^p = X_1^p \uplus (X_2^p \uplus Y^p) = X_1^p \uplus Y_1^p \ .$$

- $B = B_0 B_1$: then $A_1 = AB_0$. By analogous reasoning as in the previous case we get the conclusions.

If $\Gamma \vdash AB : Z$ is derived by rule Weaken:

$$\textsf{Weaken}\ \frac{\Gamma' \vdash AB : Z \quad \Gamma' \vdash C : V \quad y \notin Dom(\Gamma')}{\Gamma', y \prec C \vdash AB : Z}$$

with $\Gamma = \Gamma', y \prec C$ then by the IH applied to $\Gamma' \vdash AB : Z$ we have $\Gamma' \vdash A : X$, $\Gamma' \vdash B : Y$, $Z = \langle X^i \cup (X^o \uplus Y^j) \cup Y^i, (X^o \uplus Y^p) \cup Y^o, X^j \cup (X^p \uplus Y^j), X^p \uplus Y^p \rangle$. Now weakening $\Gamma' \vdash A : X$ and $\Gamma' \vdash B : Y$ to $\Gamma = \Gamma', y \prec C$ we have all the conclusions.

4. $\Gamma \vdash (A+B) : Z$ can only be derived by rule Choice or rule Weaken. If it is derived by rule Choice, then there is only one possibility:

$$\textsf{Choice}\ \frac{\Gamma \vdash A : X \quad \Gamma \vdash B : Y}{\Gamma \vdash (A+B) : X \cup Y}$$

with $Z = X \cup Y$. The conclusions follows immediately.
If $\Gamma \vdash (A+B) : Z$ is derived by rule Weaken:

$$\textsf{Weaken}\ \frac{\Gamma' \vdash (A+B) : Z \quad \Gamma' \vdash E : V \quad x \notin Dom(\Gamma')}{\Gamma', x \prec E \vdash (A+B) : Z}$$

then the proof is analogous to the proof of case Weaken in the previous clause.
5. $\Gamma \vdash \{A\} : \langle X^i, [\,], X^j, [\,] \rangle$ can only be derived by rule Scope or rule Weaken. The proof is analogous to the proof of the previous clause.

□

Lemma 3 (Legal monotonicity).

1. The only way to extend Δ to $\Delta, x \prec E$ in a derivation is by applying the rule New, Reu or Weaken.

$$\textsf{New}\ \frac{\Gamma \vdash E : X \quad x \notin Dom(\Gamma)}{\Gamma, x \prec E \vdash \ \texttt{new}\, x : \langle X^i + x, X^o + x, X^j + x, X^p + x \rangle}$$

$$\textsf{Reu}\ \frac{\Gamma \vdash E : X \quad x \notin Dom(\Gamma)}{\Gamma, x \prec E \vdash \ \texttt{reu}\, x : \langle X^i + x, X^o + x, X^j, X^p \rangle}$$

$$\textsf{Weaken}\ \frac{\Gamma \vdash E : X \quad \Gamma \vdash B : Y \quad x \notin Dom(\Gamma)}{\Gamma, x \prec E \vdash B : Y}$$

Each of the rules has $\Delta \vdash E : X$ as a premise.

2. By induction on derivation of $\Gamma \vdash E : X$. We prove that for all Γ' legal such that $\Gamma \subseteq \Gamma'$ we have $\Gamma' \vdash E : X$.
   - Base case Axiom, $E = \epsilon$, then $\Gamma' \vdash \epsilon : \langle [\,],[\,],[\,],[\,] \rangle$ since Γ' is legal.
   - Case New, $E = \mathtt{new}\, x$

$$\mathsf{New}\ \frac{\Delta \vdash B : Y \quad x \notin Dom(\Delta)}{\Delta, x \prec B \vdash \mathtt{new}\, x : X}$$

   with $X = \langle Y^i + x, Y^o + x, Y^j + x, Y^p + x \rangle$ and $\Gamma = \Delta, x \prec B$. Because $\Gamma \subseteq \Gamma'$ with Γ' legal there exists Δ_1, Δ_2 such that $\Delta \subseteq \Delta_1$ and $\Delta_1, x \prec B, \Delta_2 = \Gamma'$, with all initial segments of Γ' are legal. By clause 1 we have $\Delta_1 \vdash B : Y$. As x occurs only once in Γ' we have $x \notin Dom(\Delta_1)$ and we can apply rule New to get $\Delta_1, x \prec B \vdash \mathtt{new}\, x : X$. Since Γ' is legal we can iterate rule Weaken to get $\Gamma' \vdash \mathtt{new}\, x : X$.
   - Case Reu, $E = \mathtt{new}\, x$: analogous to case New.
   - Case Weaken

$$\mathsf{Weaken}\ \frac{\Delta \vdash E : X \quad \Delta \vdash B : Y \quad x \notin Dom(\Delta)}{\Delta, x \prec B \vdash E : X}$$

   with $\Gamma = \Delta, x \prec B$. Because $\Gamma \subseteq \Gamma'$ and Γ' legal, we have $\Delta \subseteq \Gamma'$. By the IH we get immediately $\Gamma' \vdash E : X$.
   - Case Seq, $E = BC$ with $B, C \neq \epsilon$: by Generation Lemma we have $\Gamma \vdash B : Y$ and $\Gamma \vdash C : Z$. By the IH we have $\Gamma' \vdash B : Y$ and $\Gamma' \vdash C : Z$. As the side condition for $\Gamma \vdash BC : \langle Y^i \cup (Y^o \uplus Z^j) \cup Z^i, (Y^o \uplus Z^p) \cup Z^o, Y^j \cup (Y^p \uplus Z^j), Y^p \uplus Z^p \rangle$ holds we can apply rule Seq we get the conclusion.
   - Case Choice, $E = (B + C)$: analogous to the case Seq.
   - Case Scope, $E = \{B\}$: analogous to the case Seq.

$\square$

Lemma 4 (Strengthening). By induction on derivation. Let $\Gamma' = \Gamma, x \prec A$

- Case Axiom, $B = \epsilon$: does not apply since the basis is not empty.
- Case New, $B = \mathtt{new}\, x$: does not apply since $Var(B) = Var(\mathtt{new}\, x) = \{x\}$.
- Case Reu, $B = \mathtt{reu}\, x$: does not apply since $Var(B) = Var(\mathtt{reu}\, x) = \{x\}$.
- Case Weaken,

$$\mathsf{Weaken}\ \frac{\Gamma \vdash A : X \quad \Gamma \vdash B : Y \quad x \notin Dom(\Gamma)}{\Gamma, x \prec A \vdash B : Y}$$

  We have $\Gamma \vdash B : Y$ in the premise and $x \notin Y^i$ by IH.
- Case Seq, $B = B_1 B_2$:

$$\mathsf{Seq}\ \frac{\Gamma' \vdash B_1 : Y_1 \quad \Gamma' \vdash B_2 : Y_2 \quad \forall k = 1..n. \forall c \in \mathbf{C}_k. (Y_1^o \uplus Y_2^j)(c) \leq k \quad B_1, B_2 \neq \epsilon}{\Gamma' \vdash B_1 B_2 : \langle Y_1^i \cup (Y_1^o \uplus Y_2^j) \cup Y_2^i, (Y_1^o \uplus Y_2^p) \cup Y_2^o, Y_1^j \cup (Y_1^p \uplus Y_2^j), Y_1^p \uplus Y_2^p \rangle}$$

  with $Y = \langle Y_1^i \cup (Y_1^o \uplus Y_2^j) \cup Y_2^i, (Y_1^o \uplus Y_2^p) \cup Y_2^o, Y_1^j \cup (Y_1^p \uplus Y_2^j), Y_1^p \uplus Y_2^p \rangle$. Since $x \notin Var(B_1 B_2) = Var(B_1) \cup Var(B_2)$ we have $x \neq Var(B_1)$ and $x \notin Var(B_2)$. By IH we get $\Gamma \vdash B_1 : Y_1$ and $x \notin Y_1^i$, $\Gamma \vdash B_2 : Y_2$ and $x \notin Y_2^i$. As the side condition does not change at all, we can apply rule Seq to get the conclusion: $\Gamma \vdash B_1 B_2 : Y$.
- Case Choice: $B = (B_1 + B_2)$: analogous to the case Seq.
- Case Scope: $B = \{C\}$: analogous to the case Seq.

$\square$

Proposition 1 (Uniqueness of types). By induction on the derivation of $\Gamma \vdash A : X$.

- Base case Axiom we have $A = \epsilon$ and Γ is empty, so that only Axiom is applicable. Hence, $X = Y = \langle[\,],[\,],[\,],[\,]\rangle$.
- Case New: Let $\Gamma = \Gamma', x \prec B$ such that:

$$\mathsf{New}\ \frac{\Gamma' \vdash B : U \quad x \notin Dom(\Gamma)}{\Gamma', x \prec B \vdash \texttt{new}\, x : X}$$

  with $X^* = U^* + x$ and $\Gamma = \Gamma', x \prec B$. Assume this Proposition 1 holds for the premise of this rule and let $\Gamma \vdash \texttt{new}\, x : Y$. By Generation Lemma 2, $x \in Y^*$, $\Gamma = \Delta_1, x \prec C, \Delta_2$ and $\Delta_1 \vdash C : \langle Y^i - x, Y^o - x, Y^j - x, Y^p - x\rangle$ for some Δ_1, Δ_2, C.
  By Lemma 1, there is only one declaration of x in Γ. This means $\Delta_1 = \Gamma'$, $C = B$ and Δ_2 is empty, so $\Gamma' \vdash B : \langle Y^i - x, Y^o - x, Y^j - x, Y^p - x\rangle$. By IH we have $X^* - x = Y^* - x$, i.e. $X = Y$.
- Case Reu: analogous to case New.
- Case Weaken: Let $\Gamma = \Gamma', x \prec B$ such that:

$$\mathsf{Weaken}\ \frac{\Gamma' \vdash A : X \quad \Gamma' \vdash B : Z \quad x \notin Dom(\Gamma')}{\Gamma', x \prec B \vdash A : X}$$

  Assume this Proposition 1 holds for the two premises and let $\Gamma \vdash A : Y$. Since $\Gamma' \vdash A : X$ and $x \notin Dom(\Gamma')$ we have $x \notin Var(A)$. By Lemma 4 applied to $\Gamma', x \prec B \vdash A : Y$ we get $\Gamma' \vdash A : Y$. By IH we have the conclusion $X = Y$.
- Case Seq: Let $\Gamma \vdash B_1 B_2 : X$ with $B_1, B_2 \neq \epsilon$ such that:

$$\mathsf{Seq}\ \frac{\begin{array}{c}\Gamma \vdash B_1 : Y_1 \quad \Gamma \vdash B_2 : Y_2 \quad B_1, B_2 \neq \epsilon \\ \forall k = 1..n. \forall c \in \mathbf{C}_k. (Y_1^o \uplus Y_2^j)(c) \leq k\end{array}}{\Gamma \vdash B_1 B_2 : \langle Y_1^i \cup (Y_1^o \uplus Y_2^j) \cup Y_2^i, (Y_1^o \uplus Y_2^p) \cup Y_2^o, Y_1^j \cup (Y_1^p \uplus Y_2^j), Y_1^p \uplus Y_2^p\rangle}$$

  By Generation Lemma 2 applied to $\Gamma \vdash B_1 B_2 : Y$ we have $\Gamma \vdash B_1 : V_1$, $\Gamma \vdash B_2 : V_2$, $Y = \langle V_1^i \cup (V_1^o \uplus V_2^j) \cup V_2^i, (V_1^o \uplus V_2^p) \cup V_2^o, V_1^j \cup (V_1^p \uplus V_2^j), V_1^p \uplus V_2^p\rangle$. By the IH, we have $Y_1 = V_1$ and $Y_2 = V_2$. Hence, $X = Y = \langle Y_1^i \cup (Y_1^o \uplus Y_2^j) \cup Y_2^i, (Y_1^o \uplus Y_2^p) \cup Y_2^o, Y_1^j \cup (Y_1^p \uplus Y_2^j), Y_1^p \uplus Y_2^p\rangle$.
- Case Choice: analogous to case Seq.
- Case Scope: analogous to case Seq.

$\square$

Theorem 1 (Invariant of operational semantics). By induction of the derivation of $\to$. Recall that we have three cases with two sub cases each.

- Base case OS-new:

$$\frac{x \prec B \ \in \ Prog}{\Omega : M \circ \texttt{new}\, x \to \Omega : (M + x) \circ B}\ \text{OS-new}$$

  with $\Sigma = \Omega : M$, $\Sigma' = \Omega : (M + x)$ and $\Sigma' = \Sigma + x \supset \Sigma$.
  From $\Gamma \vdash \texttt{new}\, x : X$ we get, by Generation Lemma 2, $\Gamma \vdash B : Y$ with $Y = \langle X^i - x, X^o - x, X^j - x, X^p - x\rangle \subseteq X$. Inequality (1) is trivial. For Inequality (2), we have: $\Sigma' \uplus Y^j = (\Sigma + x) \uplus (X^j - x) = \Sigma \uplus X^j \subseteq \Sigma \uplus X^i = \Theta \uplus X^i$.

For $c \notin \Sigma$: $(\Sigma' \uplus Y^j)(c) = ((\Sigma + x) \uplus (X^j - x))(c) = (\Sigma \uplus X^j)(c) = X^j(c) \subseteq X^i(c)$ and we have (3). Analogously, for $c \notin \Sigma$: $(\Sigma' \uplus Y^p)(c) = ((\Sigma + x) \uplus (X^p - x))(c) = (\Sigma \uplus X^p)(c) = X^p(c) \subseteq X^o(c)$ and we have (4).
For $c \in \Gamma$: $(\Sigma' \uplus Y^j)(c) = ((\Sigma + x) \uplus (X^j - x))(c) = (\Sigma \uplus X^j)(c)$ and we have (5). Similarly, we get (6).

– Base case OS-reu1:

$$\frac{x \prec B \in \mathit{Prog},\ x \notin \Omega \cup M}{\Omega : M \circ \mathtt{reu}\, x \to \Omega : (M + x) \circ B} \text{ OS-reu2}$$

with $\Sigma = \Omega : M$, $\Sigma' = \Omega : (M + x)$ and $\Sigma' = \Sigma + x \supset \Sigma$.
From $\Gamma \vdash \mathtt{reu}\, x : X$ we get, by Generation Lemma 2, $\Gamma \vdash B : Y$ with $Y = \langle X^i - x, X^o - x, X^j, X^p \rangle \subseteq X$ and we have $X^j \subseteq (X^i - x)$ and $X^p \subseteq (X^o - x)$ by Lemma 1. Inequality (1) is trivial. For Inequality (2), as B well-typed we have $X^i - x \supseteq X^j$ by Lemma 1 and then $\Sigma' \uplus Y^j = (\Sigma + x) \uplus X^j \subseteq (\Sigma + x) \uplus (X^i - x) = \Sigma \uplus X^i = \Theta \uplus X^i$.
For $c \notin \Sigma$:$(\Sigma' \uplus Y^j)(c) = ((\Sigma + x) \uplus X^j)(c) \subseteq ((\Sigma + x) \uplus (X^i - x))(c) = X^i(c)$ and we have (3). Analogously, for $c \notin \Sigma$:$(\Sigma' \uplus Y^p)(c) = ((\Sigma + x) \uplus X^p)(c) \subseteq ((\Sigma + x) \uplus (X^o - x))(c) = X^o(c)$ and we have (4).
Inequalities (5) and (6) are hold trivially as $c \in \Sigma$ implies $c \neq x$.

– Base case OS-reu2:

$$\frac{x \prec B \in \mathit{Prog},\ x \in \Sigma}{\Sigma \circ \mathtt{reu}\, x \to \Sigma \circ B} \text{ OS-reu1}$$

with $\Sigma' = \Sigma$.
From $\Gamma \vdash \mathtt{reu}\, x : X$ we get, by Generation Lemma 2, $\Gamma \vdash B : Y$ with $Y = \langle X^i - x, X^o - x, X^j, X^p \rangle \subseteq X$. Inequality (1) is trivial. For Inequality (2), as B well-typed we have $X^i - x \supseteq X^j$ by Lemma 1 and then $\Sigma' \uplus Y^j = \Sigma \uplus X^j \subseteq \Sigma \uplus (X^i - x) \subseteq \Sigma \uplus X^i = \Theta \uplus X^i$.
For $c \notin \Sigma$: $(\Sigma' \uplus Y^j)(c) = (\Sigma \uplus X^j)(c) = X^j(c) \subseteq X^i(c)$ and we have (3). Analogously, for $c \notin \Sigma$:$(\Sigma' \uplus Y^p)(c) = (\Sigma \uplus X^p)(c) = X^p(c) \subseteq X^o(c)$ and we have (4).
For $c \in \Gamma$: $(\Sigma' \uplus Y^j)(c) = (\Sigma \uplus X^j)(c) = (\Sigma \uplus X^j)(c)$ and we have (5). Similarly, we get (6).

– Base case OS-choice1:

$$\frac{}{\Sigma \circ (B + C) \to \Sigma \circ B} \text{ OS-choice1}$$

We have $A = (B + C)$ and $\Sigma' = \Sigma$. By Generation Lemma 2 applied to $\Gamma \vdash (B + C) : X$ we have $\Gamma \vdash B : Y$, $\Gamma \vdash C : Z$ with $Y \subseteq Y \cup Z = X$. Hence all inequalities (1-6) hold immediately.

– Base case OS-choice2: symmetric to case OS-choice1.
– Induction case OS-scope:

$$\frac{\Theta : [\,] \circ E \to \Theta : M \circ \epsilon}{\Theta \circ \{E\} \to \Theta \circ \epsilon} \text{ OS-scope}$$

with $C = \{E\}, \Theta = \Theta', \Sigma = \Theta : [\,], \Sigma' = \Theta : M$.
For Inequalities (1) and (2) we only need to prove three inequalities: $\Theta \uplus Z^i \supseteq (\Theta \uplus [\,]) \uplus Z^j$, $\Theta \uplus Z^i \supseteq \Theta' \uplus [\,]$ and $\Theta \uplus Z^i \supseteq \Theta' \uplus [\,]$. The first one holds by Lemma 1. The second one holds from IH. The third one is trivial.
Item 2 and 3 are trivial.

– Induction case OS-seq:

$$\frac{\Omega : M \circ E \to \Omega : M' \circ \epsilon}{\Omega : M \circ EC' \to \Omega : M' \circ C'} \text{ OS-seq}$$

with $\Theta = \Omega : M$, $\Theta' = \Omega : M'$ and $C = EC'$. Assume the theorem is correct for the premise of this rule we have $\Gamma \vdash E : U$, $\Theta \subseteq \Theta'$, for $x \notin \Theta$: $U^i \supseteq \Theta'$ and $U^o \supseteq \Theta'$, and for $x \in \Theta$: $\Theta \uplus U^j \supseteq \Theta'$ and $\Theta \uplus U^p \supseteq \Theta'$. In addition $\Gamma \vdash EC' : Z$ well-typed so we assume $\Gamma \vdash C' : Z'$.
For item 1, the first two conclusions hold trivially. Suppose $\Sigma \circ A \to \Sigma' \circ B$ in Δ excluding the last transition, we have, by IH, $\Theta \uplus U^i \supseteq \Sigma \uplus X^j$. Hence $\Theta \uplus Z^i = \Theta \uplus (U^i \cup (U^o \uplus Z'^j) \cup Z'^i) \supseteq \Theta \uplus U^i \supseteq \Sigma \uplus X^j$. We still need to prove the case $\Sigma' = \Theta'$ and $B = C'$ i.e. $\Theta \uplus Z^i \supseteq \Theta' \uplus Z'^j$. This inequality holds easily when $c \notin \Theta$, by IH, we have $U^o(c) \geq \Theta'(c)$. If $c \in \Theta$, we have $(\Theta \uplus U^p)(c) \geq \Theta'(c)$ by IH, so $(\Theta \uplus Z^i)(c) \geq (\Theta \uplus Z^j)(c) \geq (\Theta \uplus (U^p \uplus Z'^j))(c) \geq (\Theta' \uplus Z'^j)(c)$.
For item 2, we have to prove that for $c \notin \Theta$: $(U^i \cup (U^o \uplus Z'^j) \cup Z'^i) \supseteq \Theta' \uplus Z'^j$ and $(U^o \uplus Z'^p) \cup Z'^o \supseteq \Theta' \uplus Z'^p$. Both inequalities follow from IH: $U^o \supseteq \Theta'$.
For item 3, we have to prove that for $c \in \Theta$: $\Theta \uplus (U^j \cup (U^p \uplus Z'^j)) \supseteq \Theta' \uplus Z'^j$ and $\Theta \uplus (U^p \uplus Z'^p) \supseteq \Theta' \uplus Z'^p$. Both inequalities follow from IH: for $x \in \Theta$: $\Theta \uplus U^p \supseteq \Theta'$.

– Induction case OS-trans:

$$\frac{\Theta \circ C \to \Omega \circ E \quad \Omega \circ E \to \Theta' \circ C'}{\Theta \circ C \to \Theta' \circ C'} \text{ OS-trans}$$

Assume the theorem is correct for the premise of this rule we have $\Gamma \vdash C : Z$, $\Gamma \vdash C' : Z'$, $\Gamma \vdash E : U$, $Z \supseteq U \supseteq Z'$, $\Theta \subseteq \Omega \subseteq \Theta'$, and for any $\Sigma \circ A$ such that $\Gamma \vdash A : X$, in derivation Δ of $\Theta \circ C \to \Omega \circ E$:

$$\Theta \uplus Z^i \supseteq \Sigma \uplus X^j$$

for any $\Sigma' \circ A'$ such that $\Gamma \vdash A' : X'$, in derivation Δ' of $\Omega \circ E \to \Theta' \circ C'$:

$$\Omega \uplus U^i \supseteq \Sigma' \uplus X'^j$$

for $c \notin \Theta$:

$$Z^i \supseteq \Omega \uplus U^j \text{ and } Z^o \supseteq \Omega \uplus U^p$$

for $c \in \Theta$:

$$\Theta \uplus Z^j \supseteq \Omega \uplus U^j \text{ and } \Theta \uplus Z^p \supseteq \Omega \uplus U^p$$

for $c \notin \Omega$:

$$U^i \supseteq \Theta' \uplus Z'^j \text{ and } U^o \supseteq \Theta' \uplus Z'^p$$

for $c \in \Omega$:

$$\Omega \uplus U^j \supseteq \Theta' \uplus Z'^j \text{ and } \Omega \uplus U^p \supseteq \Theta' \uplus Z'^p$$

For item 1, the first two conclusions by transitivity. The inequalities (1) and (2) hold easily for any $\Sigma \circ A$ in Δ. We still need to prove the two inequalities for any $\Sigma' \circ A'$ in Δ'. If $c \notin \Omega$, then $c \notin \Theta$ and $X^i(c) \geq U^i(c) \geq (\Sigma' \uplus X'^j)(c)$. If $c \in \Omega$, then $(\Theta \uplus Z^i)(c) \geq (\Omega \uplus U^j)(c) \geq (\Sigma' \uplus X'^j)(c)$.
For item 2, we only have to prove that for $c \notin \Theta$: $Z^i \supseteq \Theta' \uplus Z'^j$ and $Z^o \supseteq \Theta' \uplus Z'^p$. If $c \in \Omega$ the both inequalities follow by transitivity and IH. If $c \notin \Omega$ then because $Z \supseteq U$ we can apply transitivity to get both inequalities.
For item 3, we only have to prove that for $c \in \Theta$: $\Theta \uplus Z^j \supseteq \Theta' \uplus Z'^j$ and $\Theta \uplus Z^p \supseteq \Theta' \uplus Z'^p$. As $\Theta \subseteq \Omega$ we have $c \in \Omega$ and both inequalities follow by transitivity.

□