Alma Mater Studiorum · Università di Bologna

SCUOLA DI SCIENZE
Corso di Laurea Magistrale in Informatica

Deadlock Analysis of Asynchronous Sequential Processes

Relatore:
Chiar.mo Prof.
Cosimo Laneve

Correlatore:
Elena Giachino

Presentata da:
Vincenzo Mastandrea

Sessione II
Anno Accademico 2013/2014

Abstract

In this thesis we present ad study an object-oriented language, characterized by two different types of objects, passive and active objects, of which we define the operational syntax and semantics. For this language we also define the type system, that will be used for the type checking and for the extraction of behavioral types, which are an abstract description of the behavior of the methods, used in deadlock analysis. Programs can manifest deadlock due to the errors of the programmer. To statically identify possible unintended behaviors we studied and implemented a technique for the analysis of deadlock based on behavioral types.

Contents

List of Figures

Introduzione

I sistemi moderni sono progettati per supportare un elevato grado di parallelismo permettendo alle varie componenti del sistema di operare contemporaneamente. Quando questi sistemi mostrano anche un elevato numero di risorse e una mole cospiqua di dati condivisi, i deadlock rappresentano una minaccia insidiosa e ricorrente. Un classico esempio si ha quando due processi che detengono in modo esclusivo una risorsa richiedono l'accesso alla risorsa detenuta dall'altro processo, in questo caso si ha che la corretta terminazione di ciascuna delle due attivitá dipende dalla terminazione dell'altra; la presenza di una dipendenza circolare rende quidni impossibile la terminazione. I deadlock possono essere particolarmente difficili da rilevare nei sistemi con ricorsione e con creazione dinamica delle risorse. Un caso particolare risulta essere un sistema adattivo che crea un numero illimitato di processi, come ad esempio possono essere le applicazioni server. In questi sistemi, i protocolli di interazione sono estremamente complessi e le soluzioni allo stato dell'arte danno risposte imprecise o non scalabili.

Perseguendo l'obiettivo di definire un linguaggio "leggero", sufficientemente semplice per facilitare la dimostrazione della proprietá di base, é stato progettato e studiato un linguaggio orientato agli oggetti, con oggetti attivi e passivi, e successivamente é stata sviluppata una tecnica per l'analisi di deadlock. Sistemi distribuiti ad oggetti sono generalmente basati sull'invocazione di metodi remoti che fanno affidamento sul concetto di thread, che risulta essere indipendente dalla struttura degli oggetti. Questo lavoro punta ad unificare il concetto di thread e quello di oggetto, infatti, come si vedrá di seguito piú nel dettaglio, ogni oggetto appartiene ad una sola attivitá (ci si riferirá con il termine attivitá invece che processo per esprimere l'unitá di elaborazione) ognuna associata ad un singolo thread.

Le attivitá comunicano mediante invocazioni di metodo asincrone, permettendo al chiamante e al chiamato di eseguire altre operazioni nel lasso di tempo che intercorre fra l'invio della richiesta e la sua elaborazione, aumentando la concorrenza. Nel lavoro proposto verrá presentato anche il concetto di variabili con tipo "futuro", necessario per rappresentare i risultati attesi da invocazioni asincrone.

Come accennato precedentemente, il fine principale di questo lavoro di tesi é progettare e studiare un linguaggio orientato agli oggetti chiamato classAsp, dove ASP é l'acronimo di Asynchronous Sequential Processes.

Si é partiti da un classico linguaggio orientato agli oggetti, puramente sequenziale, e lo si é esteso introducendo i concetti di oggetto attivo o passivo.

Rispetto un comune oggetto (oggetto passivo), al momento della creazione di un oggetto attivo viene avviato un nuovo thread che si occuperá dell'esecuzione

di tutti i metodi di classe invocati su tale oggetto, che risulteranno pertanto essere asincroni rispetto al thread principale. In questo caso il risultato del metodo invocato sará rappresentato da un tipo futuro finché il valore corrispondente non verrá calcolato.

A differenza di altri lavori simili a questo, la sincronizzazione risulta essere implicita: non vi sono, infatti, comandi espliciti per forzare le operazioni di sincronizzazione, che vengono rimandate il piú possibile. In tale linguaggio, ogni sincronizzazione viene eseguita nell'esatto momento in cui il valore restituito da un metodo asincrono viene utilizzato all'interno di un'operazione.

Gli aspetti innovativi di classAsp pertanto includono una formalizzazione delle seguenti caratteristiche in un contesto orientato agli oggetti: futuri e invocazioni di metodo asincrone, sincronizzazioni guidate dal flusso di dati e l'unificazione del concetto di oggetto e attivitá.

In questo lavoro verrá presentata la sintassi e la semantica operazionale proposte per tale linguaggio e il sistema di tipi che verrá utilizzato per il controllo sui tipi e per l'estrazione dei *behavioral type*. I *behavioral type* rappresentano una descrizione astratta del comportamento dei metodi e verranno usati nell'analisi di deadlock. Come sappiamo i programmi possono manifestare deadlock a causa di errori del programmatore. Per identificare staticamente possibili comportamenti non desiderati abbiamo studiato e implementato una tecnica per l'analisi di deadlock basata su tali *behavioral type*.

L'idea alla base dell'analisi di deadlock proposta consiste nell'estrarre dai *behavioral type*, individuati mediante il sistema di tipi, coppie di dipendenze fra attivitá. Tali coppie verranno analizzate con l'intento di individuare dipendenze circolari al loro interno. La presenza o l'assenza di tali dipendenze circlari indicherá la possibilitá di verificarsi o meno di deadlock durante l'esecuzione del programma.

La tesi é organizzata come segue. Nel Capitolo 2 verrá definita la sintassi e la semantica operazionale del linguaggio classAspe verranno mostrati alcuni esempi di esecuzione. Il Capitolo 3 mostrerá il sistema di tipi e verranno mostrati alcuni esempi di type checking. Il Capitolo 4 definisce il concetto di "lam" e i principi base dell'analisi di deadlock. In Appendice A verranno riportate le dimostrazioni formali per la correttezza di quanto detto nel Capitolo 3

Chapter 1

Intruduction

1.1 Introduction

Modern systems are designed to support a high degree of parallelism by letting as many system components as possible operate concurrently. When such systems also exhibit a high degree of resource and data sharing then deadlocks represent an insidious and recurring threat. In particular, deadlocks arise as a consequence of exclusive resource access and circular wait for accessing resources. A standard example is when two processes are exclusively holding a different resource and are requesting access to the resource held by the other. That is, the correct termination of each of the two process activities depends on the termination of the other. The presence of a circular dependency makes termination impossible. Deadlocks may be particularly hard to detect in systems with unbounded (mutual) recursion and dynamic resource creation. A paradigm case is an adaptive system that creates an unbounded number of processes such as server applications. In these systems, the interaction protocols are extremely complex and stateof-the-art solutions either give imprecise answers or do not scale. Pursuing the aim of defining a language "light", sufficiently small to facilitate the demonstration of basic properties, an object-oriented language has been achieved with active and passive objects. Was also developed a technique for the analysis of deadlocks. Distributed object systems are generally based on remote method calls between objects and rely on a concept of threads rather orthogonal to the object structure. This work formalizes a way of unifying the notion of threads and objects: each object belongs to a single activity (we use activity rather than process for expressing the unit of distribution) and each activity is associated a single thread. Then, activities communicate by asynchronous method calls allowing both sender and callee to perform operations between the request sending and its treatment, thus increasing concurrency. Futures are introduced to represent awaited results of such asynchronous calls. We design an object-oriented language named ASP: Asynchronous Sequential Processes. We start from a purely sequential and classical object-oriented language and extend it with the concept of active and passive object. Method calls on active objects are asynchronous: their results are represented by futures until the corresponding response is returned. Automatic synchronization of activities comes from wait-by-necessity: a wait automatically occurs upon a strict operation on a future. Innovative aspects of ASP include a formalization of the following features in an object oriented context: futures together with asynchronous method calls, data-driven synchronization, and unification of the notions of processes and objects.

1.2 State of the art

Analisi statica di deadlock. The analysis of deaclock, based on the types, has been heavily studied. Some proposals involve calculations of processes [1–4], but some contributions also concern the deadlock in object-oriented programs. [5–7].

Kobayashi, on works abount pi-calculus [1, 8, 9], defines dependencies between channels using "capability" and "constrain" of type and verifies the absence of circularity.

In any case, his technique is different from the technique that will be presented and a complete comparison would require the application of our technique to the pi-calculus, which has a different sync pattern than the one that will be described later.

The work of Suenaga [2, 3] apply the technique of Kobayashi to concurrent languages with resources. It is easy to extract the LAM model from a program in a language with interrupts [2] and, therefore, it is equally easy to verify the presence of deadlocks. In any case, a precise comparison with [2] has not been done. The language of [3] using mutable references, which are a well-known aspect which, together with the concurrency, leads to a non-deterministic behavior. In other contributions, a type system computes a partial order of locks in a program and a theorem shows that the task follow that order A difference with the work mentioned above is the use of behavioral types. The behavioral types are terms in an algebraic process [10]. The use of a simple algebraic process (that is finite state) to describe protocols (for communication or synchronization) is not new. This is the case of exchange solutions in SSDL [11], which are based on CSP [12] and the pi-calculus [13], or based on behavioral types on [14] and in [15], using CCS [16]. Other static approaches, which are not based on types, are [17, 18] where circular dependencies within the processes are recognized as erroneous configurations, but the dynamic creation of names is not treated.

Languages and Behavioral types. Terms similar to LAM were studied in [19] for a language of the family Creol [20] with the aim of controlling the presence of deadlock. [In particular, the language of [19] is `Javalike` with future data types and the operation `get` described in [21].] The technique of derivation of LAM from real programs described in [22] has already been prototyped in [23]. The inference system, when applied to real programs requires some additional annotations, in order to overcome the difficulties related to the use of structured data types and iterations.

Chapter 2

The language Class-based ASP

2.1 Syntax

Figure 2.1 displays classAsp syntax, where an overlined element corresponds to any finite sequence of such element. The elements of the sequence are separated by commas, except for $\overline{C}$, which has no separator. For example $\overline{T}$ means a (possibly empty) sequence $T_1, \cdots, T_n$. When we write $\overline{T\ x\ ;}$ we mean a sequence $T_1\ x_1\ ;\ \cdots\ ;\ T_n\ x_n\ ;$ when the sequence is not empty; we mean the empty sequence otherwise.

P	::=	$\overline{I}\ \overline{C}\ \{\overline{T\ x\ ;}\ s\}$	program
T	::=	`D` | `I`	type
I	::=	`interface I` $\{\overline{S\ ;}\}$	interface
S	::=	T `m`$(\overline{T\ x})$	method signature
C	::=	`class C`$(\overline{T\ x})$ [`implements` $\overline{\texttt{I}}$] $\{\overline{M}\}$	class
M	::=	$S\{\overline{T\ x\ ;}\ s\}$	method definition
s	::=	`skip` | $x = z$ `;` s | `if` e $\{s\}$ `else` $\{s\}$ `;` s	statement
	|	`return` e	
z	::=	e | e.`m`$(\overline{e})$ | `new C`$(\overline{e})$	expression with side effects
	|	`newActive C`$(\overline{e})$ | *arithmetic-bool-exp*	
e	::=	v | x | `this`	expression
v	::=	`null` | *primitive-val*	value

Figure 2.1: Class-based classAsp

The syntax of Class Based classAsp is based on syntax of `ABS`. Below program components are described step by step

$$P \ ::= \ \overline{I}\ \overline{C}\ \{\overline{T\ x\ ;}\ s\}$$

The program P is composed of three blocks, the first two blocks are lists of interface and class declarations (resp. I and C) while the third block represents the *main function*, which is composed of a series of variable declarations followed by one or more statements.

$$T \ ::= \ \texttt{D} \mid \texttt{I}$$

T stands for type, which can be primitive `D` such as Int, Bool, String, or an interface.

$$I \ ::= \ \texttt{interface I}\ \{\overline{S\ ;}\}$$

An interface is composed of a header and a body, the header consists of the keyword *interface* and the name of the interface, the body consists of a set of method signature.

$$S \;::=\; T\ \mathtt{m}(\overline{T\ x})$$

A method signature consists of a type, which will be the type of data returned, the method name and the declaration of the parameters of the method, which comprise a list of pairs variable type - name.

$$C \;::=\; \mathtt{class}\ \mathtt{C}(\overline{T\ x})\ [\mathtt{implements}\ \overline{\mathtt{I}}]\ \{\,\overline{M}\,\}$$

A class be composed of a header and a body. The header comprise the keyword *class*, the class name and the declaration of the parameters, which are represented by a list of pairs type - variable name. The parameters of a class correspond to the fields of that class. The header of the class has an optional component used in case it implements an interface, this component consists of the keyword *interface*,which is followed by a list of interfaces names that this class implements. The body of the class includes a list of method definitions.

$$M \;::=\; S\{\,\overline{T\ x\ \mathtt{;}}\ s\,\}$$

The class method is composed of the method signature, and a body. The body is made up of two blocks, the first block consists of the declarations of local variables,which are expressed by type-value pairs, and the second block is formed by one or more statements.

$$\begin{array}{lcl} s & ::= & \mathtt{skip} \mid x = z\ \mathtt{;}\ s \mid \mathtt{if}\ e\ \{\,s\,\}\ \mathtt{else}\ \{\,s\,\}\ \mathtt{;}\ s \mid \mathtt{return}\ e \\ z & ::= & e \mid e.\mathtt{m}(\overline{e}) \mid \mathtt{new}\ \mathtt{C}(\overline{e}) \mid \mathtt{newActive}\ \mathtt{C}(\overline{e}) \mid \textit{arithmetic-bool-exp} \\ e & ::= & v \mid x \mid \mathtt{this} \\ v & ::= & \mathtt{null} \mid \textit{primitive-val} \end{array}$$

This is the classic definition of the statements for a programming language.

$$z \;::=\; e \mid e.\mathtt{m}(\overline{e}) \mid \mathtt{new}\ \mathtt{C}(\overline{e}) \mid \mathtt{newActive}\ \mathtt{C}(\overline{e}) \mid \textit{arithmetic-bool-exp}$$

The right side of an assignment can be made, as often happens in a programming language, by primitive values, arithmetic and boolean expressions, method invocations and class constructors. As we see from the syntax definition, to create an object of a given class can be done in two ways. The first way is to use the primitive $\mathtt{new}\ \mathtt{C}(\overline{e})$, that creates a passive object, which is an object that whenever one of its methods is invoked the execution of the process that called it is blocked and waits for result. The second way is to use the primitive $\mathtt{newActive}\ \mathtt{C}(\overline{e})$ which creates an active object, whenever a method is called on that object it will be executed concurrently and asynchronously with the process that invoked it, which will continue its execution.

2.2 Semantics

$$
\begin{array}{rcl}
cn & ::= & \epsilon \mid \mathit{fut}(f, \mathit{val}) \mid \mathit{ob}(\alpha, o, \sigma, p, q) \mid cn\ cn \\
p & ::= & \{\ell \mid s\} \mid \texttt{idle} \mid \texttt{error} \\
q & ::= & \epsilon \mid \texttt{error} :: q \mid \{\ell \mid s\} :: q \\
z & ::= & \bullet \mid \dots
\end{array}
\qquad
\begin{array}{rcl}
\sigma & ::= & [\cdots, o \mapsto [\overline{x : v}], \cdots] \\
\ell & ::= & [\cdots, x \mapsto v, \cdots]
\end{array}
\qquad
\begin{array}{rcl}
\mathit{val} & ::= & (v, \sigma) \mid \bot \\
v & ::= & o \mid f \mid \alpha \mid \dots
\end{array}
$$

Figure 2.2: Runtime syntax of classAsp.

The classAsp semantics is defined as a transition relation between *configurations*, noted cn and defined in Figure 2.2. Configurations are sets of elements – therefore we assume associativity and commutativity – and are denoted by the juxtaposition of the elements $cn\ cn$; the empty configuration is denoted by ε. The transition relation uses three infinite sets of names: *object names*, ranged over by o, o', $\cdots$, *activity names*, ranged over by α, β, $\cdots$, and *future names*, ranged over by f, f', $\cdots$. Object names are partitioned according to the class and the active object they belong to. We assume there are infinitely many object names per class and the function, fresh(C) returns a new object name of class C. Given an object name o, the function class(o) returns its class. The function fresh( ) returns either a fresh active object name or a fresh future name; the context will disambiguate between the two cases.

Runtime values also contain object and future names or undefined values, which are denoted by $\bot$. With an abuse of notation, we range over runtime values with v, v', $\cdots$. We finally use σ and ℓ, possibly indexed, to range over maps from fields to runtime values and local variables to runtime values, respectively. The map ℓ also binds the special name destiny to a future value.

The elements of configurations are:

- *objects* $\mathit{ob}(\alpha, o, \sigma, p, q)$ where α is an activity name; o is the name of the active object in the activity α; σ returns the values of object's fields, p is either idle, representing inactivity, or is the *active process* $\{\ell \mid s\}$, where ℓ returns the values of local identifiers and s is the body of the method; q is a stack of processes to evaluate.

- *future binders* $\mathit{fut}(f, v)$ where v, called *the reply value* may be also $\bot$ meaning that the value has still not being computed.

The most relevant transition relation rules are collected in Figure 2.3. They define transitions of objects $\mathit{ob}(\alpha, o, \sigma, p, q)$ according to the shape of the statement in p.

The following auxiliary functions and assumptions are used in the semantic rules (we assume a fixed classAsp program):

- $\mathrm{dom}(\ell)$ and $\mathrm{dom}(\sigma)$ return the domain of ℓ and σ, respectively.
- $\mathit{fields}(\mathtt{C}) = \overline{T\ x}$ returns the field of class $\mathtt{C}$.
- $\ell[x \mapsto v]$ is the function such that $(\ell[x \mapsto v])(x) = v$ and $(\ell[x \mapsto v])(y) = \ell(y)$, when $y \neq x$. Similarly for $\sigma[x \mapsto v]$.
- $[\![e]\!]_{(\sigma+\ell)}$ returns the value of e by computing the arithmetic and boolean expressions and retrieving the value of the identifiers that is stored either in σ or in ℓ. Since σ and ℓ are assumed to have disjoint domains, we denote the union map with $\sigma+\ell$. $[\![\overline{e}]\!]_{(\sigma+\ell)}$ returns the tuple of values of $\overline{e}$. If one of the terms of an expression is a future, the expression can not be computed, the computation of the expression is postponed until the future will not be solved.
- $\mathtt{C.m}$ returns the term $(\overline{T\ x})\{\overline{T'\ z}; s\}$ that contains the arguments and the body of the method $\mathtt{m}$ in the class $\mathtt{C}$.
- $\mathrm{bind}(o, \mathtt{m}, \overline{v}, \mathtt{C}) = \{[\overline{x} \mapsto \overline{v}, \overline{z} \mapsto \bot] \mid s[^{o}/_{\mathtt{this}}]\}$, where $\mathtt{C.m} = (\overline{T\ x})\{\overline{T'\ z}; s\}$. Binding may not succeed, in this case we get the `error` process.
- serialize defined as follows:

  $\mathrm{serialize}(o, \sigma) = o \mapsto \sigma(o) \cup \mathrm{serialize}(\sigma(o), \sigma)$
  $\mathrm{serialize}([\overline{x \mapsto v}], \sigma) = \bigcup_{v' \in \overline{v}} \mathrm{serialize}(v', \sigma)$
  $\mathrm{serialize}(f, \sigma) = \mathrm{serialize}(\alpha, \sigma) = \mathrm{serialize}(\mathtt{null}, \sigma) = \mathrm{serialize}(\mathit{primitive\text{-}val}, \sigma) = \varnothing$

- we will assume the equivalence: skip ; s ≡ s.

(Activate)

$$\frac{}{ob(\alpha, o, \sigma, \texttt{idle}, \{\ell \mid s\} :: q) \rightarrow ob(\alpha, o, \sigma, \{\ell \mid s\}, q)}$$

(Skip)

$$\frac{}{ob(\alpha, o, \sigma, \{\ell \mid skip\}, q) \rightarrow ob(\alpha, o, \sigma, idle, q)}$$

(Assign-Local)

$$\frac{x \in \mathrm{dom}(\ell) \quad v = [\![e]\!]_{(\sigma+\ell)}}{ob(\alpha, o, \sigma, \{\ell \mid x = e \ ; \ s\}, q) \rightarrow ob(\alpha, o, \sigma, \{\ell[x \mapsto v] \mid s\}, q)}$$

(Assign-Field)

$$\frac{x \in \mathrm{dom}(\sigma) \quad v = [\![e]\!]_{(\sigma+\ell)} \quad \sigma' = \sigma[o \mapsto (\sigma(o)[x \mapsto v])] \quad l(this) = o}{ob(\alpha, o, \sigma, \{\ell \mid x = e \ ; \ s\}, q) \rightarrow ob(\alpha, o, \sigma', \{\ell \mid s\}, q)}$$

(If-Then)

$$\frac{0 \neq [\![e]\!]_{(\sigma+\ell)}}{ob(\alpha, o, \sigma, \{\ell \mid \texttt{if } e \ \{ s \} \ \texttt{else} \ \{ s' \} \ ; \ s''\}, q) \rightarrow ob(\alpha, o, \sigma, \{\ell \mid s \ ; \ s''\}, q)}$$

(If-Else)

$$\frac{0 = [\![e]\!]_{(\sigma+\ell)}}{ob(\alpha, o, \sigma, \{\ell \mid \texttt{if } e \ \{ s \} \ \texttt{else} \ \{ s' \} \ ; \ s''\}, q) \rightarrow ob(\alpha, o, \sigma, \{\ell \mid s' \ ; \ s''\}, q)}$$

(New-Object)

$$\frac{fields(\texttt{C}) = \overline{T \ x} \quad o' = \mathrm{fresh}(\texttt{C}) \quad \sigma' = \sigma \cup \{o' \mapsto [\overline{x \mapsto v}]\} \quad [\![\overline{e}]\!]_{(\sigma+\ell)} \overline{v}}{ob(\alpha, o, \sigma, \{\ell \mid x = \texttt{new C}(\overline{e}) \ ; \ s \ \}, q) \rightarrow ob(\alpha, o, \sigma', \{\ell \mid x = o' \ ; \ s \ \}, q)}$$

(New-Active)

$$\frac{fields(\texttt{C}) = \overline{T \ x} \quad o' = \mathrm{fresh}(\texttt{C}) \quad \gamma = \mathrm{fresh}(\ ) \quad \sigma' = \{o' \mapsto [\overline{x \mapsto v}], \mathrm{serialise}(\overline{v}, \sigma)\} \quad [\![\overline{e}]\!]_{(\sigma+\ell)} = \overline{v}}{ob(\alpha, o, \sigma, \{\ell \mid x = \texttt{newActive C}(\overline{e}) \ ; \ s \ \}, q) \rightarrow ob(\alpha, o, \sigma, \{\ell \mid x = o' \ ; \ s \ \}, q) \quad ob(\gamma, o', \sigma', \texttt{idle}, \epsilon)}$$

(Invk-Active)

$$\frac{\begin{array}{c}[\![e]\!]_{(\sigma+\ell)} = o' \quad [\![\overline{e}]\!]_{(\sigma+\ell)} = \overline{v} \quad f = \mathrm{fresh}(\ ) \\ \mathrm{dom}(\sigma) \cap \mathrm{dom}(\sigma') = \varnothing \\ \mathrm{bind}(o', m, \overline{v}, \mathrm{class}(o')) = \{\ell'' \mid s'\} \\ \sigma'' = \sigma' \cup \mathrm{serialize}(\overline{v}, \sigma) \quad \ell' = [\texttt{destiny} \mapsto f, this \mapsto o'] \cup \ell''\end{array}}{\begin{array}{c}ob(\alpha, o, \sigma, \{\ell \mid x = e.\texttt{m}(\overline{e}) \ ; \ s \ \}, q) \ \ ob(\beta, o', \sigma', p', q') \\ \rightarrow ob(\alpha, o, \sigma, \{\ell \mid x = f \ ; \ s \ \}, q) \ \ ob(\beta, o', \sigma'', p', q' :: \{\ell' \mid s'\}) \ \ fut(f, \bot)\end{array}}$$

(Invk-Passive)

$$\frac{\begin{array}{c}o \in \mathrm{dom}(\sigma) \quad [\![e]\!]_{(\sigma+\ell)} = o' \\ [\![\overline{e}]\!]_{(\sigma+\ell)} = \overline{v} \quad f = \mathrm{fresh}(\ ) \\ \mathrm{bind}(o', m, \overline{v}, \mathrm{class}(o')) = \{\ell'' \mid s'\} \\ \ell' = [\texttt{destiny} \mapsto f, this \mapsto o'] \cup \ell''\end{array}}{\begin{array}{c}ob(\alpha, o, \sigma, \{\ell \mid x = e.\texttt{m}(\overline{e}) \ ; \ s \ \}, q) \\ \rightarrow ob(\alpha, o, \sigma, \{\ell' \mid s'\}, \{\ell \mid x = f \ ; \ s \ \} :: q)\end{array}}$$

(ReturnLocal)

$$\frac{v = [\![e]\!]_{(\sigma+\ell)} \quad f = \ell(\texttt{destiny})}{ob(\alpha, o, \sigma, \{\ell \mid \texttt{return } e\}, \{\ell' \mid x = f \ ; \ s \ \} :: q) \rightarrow ob(\alpha, o, \sigma, \{\ell' \mid x = v \ ; \ s \ \}, q)}$$

(Return)

$$\frac{v = [\![e]\!]_{(\sigma+\ell)} \quad f = \ell(\texttt{destiny})}{ob(\alpha, o, \sigma, \{\ell \mid \texttt{return } e\}, q) \ \ fut(f, \bot) \rightarrow ob(\alpha, o, \sigma, \texttt{idle}, q) \ \ fut(f, (v, \mathrm{serialize}(v, \sigma)))}$$

(GetLocal)

$$\frac{\ell(x) = f \quad \ell' = \ell[x \mapsto v] \quad \sigma'' = \sigma \cup \sigma' \quad \mathrm{dom}(\sigma) \cap \mathrm{dom}(\sigma') = \varnothing}{ob(\alpha, o, \sigma, \{\ell \mid s\}, q) \ \ fut(f, (v, \sigma')) \rightarrow ob(\alpha, o, \sigma'', \{\ell' \mid s\}, q) \ \ fut(f, (v, \sigma'))}$$

(Get)

$$\frac{\sigma(o) = [\overline{x \mapsto v}, x \mapsto f] \quad \sigma'' = \sigma[o \mapsto (\sigma(o)[x \mapsto v])] \cup \sigma' \quad \mathrm{dom}(\sigma) \cap \mathrm{dom}(\sigma') = \varnothing}{ob(\alpha, o, \sigma, \{\ell \mid s\}, q) \ \ fut(f, (v, \sigma')) \rightarrow ob(\alpha, o, \sigma'', \{\ell \mid s\}, q) \ \ fut(f, (v, \sigma'))}$$

Figure 2.3: Semantics of classAsp.

2.3 Rules

We try to describe each rule.

(Activate)

$$\frac{}{ob(\alpha, o, \sigma, \texttt{idle}, \{\ell \mid s\} :: q) \rightarrow ob(\alpha, o, \sigma, \{\ell \mid s\}, q)}$$

The rule Activate, given a configuration consisting of an inactive object ($p = \texttt{idle}$), takes the next item from the stack of processes to evaluate ($\{\ell \mid s\} :: q$) and puts it as an active process ($p = \{\ell \mid s\}$)

(Assign-Local)

$$\frac{x \in \mathrm{dom}(\ell) \quad v = [\![e]\!]_{(\sigma+\ell)}}{ob(\alpha, o, \sigma, \{\ell \mid x = e \ ; \ s\}, q) \rightarrow ob(\alpha, o, \sigma, \{\ell[x \mapsto v] \mid s\}, q)}$$

If x is a local variable of the method execution ($x \in \mathrm{dom}(\ell)$), v is the result of evaluating an expression ($v = [\![e]\!]_{(\sigma+\ell)}$), and the active process is $\{\ell \mid x = e \ ; \ s\}$, we have that the rule Assign-Local modifies the function ℓ such that x is mapped to v ($\ell[x \mapsto v]$), and we go to the evaluation of the next statement; the active process becomes $\{\ell[x \mapsto v] \mid s\}$.

(Assign-Field)

$$\frac{\begin{array}{l} x \in \mathrm{dom}(\sigma) \quad v = [\![e]\!]_{(\sigma+\ell)} \\ \sigma' = \sigma[o \mapsto (\sigma(o)[x \mapsto v])] \\ l(this) = o \end{array}}{ob(\alpha, o, \sigma, \{\ell \mid x = e \ ; \ s\}, q) \rightarrow ob(\alpha, o, \sigma', \{\ell \mid s\}, q)}$$

If x is the field of an object o ($x \in \mathrm{dom}(\sigma)$), which was created during the execution of the α activity, `this` corresponds to the object o ($l(this) = o$), v is the result of evaluating an expression ($v = [\![e]\!]_{(\sigma+\ell)}$), and the active process is $\{\ell \mid x = e \ ; \ s\}$, it follows that the rule Assign-Field modifies the function σ in order to have that the field x is mapped to v and twe go to the evaluation of the next statement; he active process becomes $\{\ell \mid s\}$.

(New-Object)

$$\frac{\mathit{fields}(\mathtt{C}) = \overline{T\ x} \quad o' = \text{fresh}(\mathtt{C}) \quad \sigma' = \sigma \cup \{o' \mapsto [\overline{x \mapsto v}]\} \quad [\![\overline{e}]\!]_{(\sigma+\ell)} = \overline{v}}{ob(\alpha, o, \sigma, \{\ell \mid x = \mathtt{new}\ \mathtt{C}(\overline{e})\ ;\ s\ \}, q) \rightarrow ob(\alpha, o, \sigma', \{\ell \mid x = o'\ ;\ s\ \}, q)}$$

If o' is a new name for an object of `class C` , not yet used, obtained from the function fresh(C) ($o' = \text{fresh}(\mathtt{C})$), $\overline{T\ x}$ is the set of fields of the `class C` , given from the function $\mathit{fields}(\mathtt{C})$, $\overline{v}$ is the set of values obtained from the evaluation of expressions, and the evaluation context is $x = \mathtt{new}\ \mathtt{C}(\overline{e})$, it follows that the rule New-Object changes the function σ to map the fields of the object o that match the parameters passed to the constructor, to the values v. The evaluation context becomes $x = o'$.

(New-Active)

$$\frac{\mathit{fields}(\mathtt{C}) = \overline{T\ x} \quad o' = \text{fresh}(\mathtt{C}) \quad \gamma = \text{fresh}(\ ) \quad \sigma' = \{o' \mapsto [\overline{x \mapsto v}], \text{serialise}(\overline{v}, \sigma)\} \quad [\![\overline{e}]\!]_{(\sigma+\ell)} = \overline{v}}{\begin{array}{c} ob(\alpha, o, \sigma, \{\ell \mid x = \mathtt{newActive}\ \mathtt{C}(\overline{e})\ ;\ s\ \}, q) \\ \rightarrow ob(\alpha, o, \sigma, \{\ell \mid x = o'\ ;\ s\ \}, q) \quad ob(\gamma, o', \sigma', \mathtt{idle}, \epsilon) \end{array}}$$

We are almost in the same situation of the rule New-Object but we have γ which is a fresh activity name and the evaluation context is $x = \mathtt{newActive}\ \mathtt{C}(\overline{e})$. The rule New-Acrive adds a new element to the configuration, that is a new *object* ($ob(\gamma, o', \sigma', \mathtt{idle}, \epsilon)$), where: the activity name is γ; the active object is o'; σ' is a new function that returns the values of objects fields; the status of process is `idle` and the stack of process to evaluate is empty.
To understand more clearly the semantics of this rule is appropriate to describe in more detail the function σ'.
We have that $\sigma' = \{o' \mapsto [\overline{x = v}], \text{serialise}(\overline{v}, \sigma)\}$ has a rule that maps the fields of object o', that match the parameters passed to the constructor, to the values v. The function serialise$(\overline{v}, \sigma)$ add a new rule to function σ', like the rule of o', for each field of o' that is an object and for the fields of the fields of o' that are an object and so on recursively.

(Invk-Active)

$$\frac{\begin{array}{c}[\![e]\!]_{(\sigma+\ell)} = o' \quad [\![\overline{e}]\!]_{(\sigma+\ell)} = \overline{v} \quad f = \text{fresh}(\ ) \quad \text{dom}(\sigma) \cap \text{dom}(\sigma') = \varnothing \\ \text{bind}(o', m, \overline{v}, \text{class}(o')) = \{\ell'' \mid s'\} \\ \sigma'' = \sigma' \cup \text{serialize}(\overline{v}, \sigma) \quad \ell' = [\texttt{destiny} \mapsto f, this \mapsto o'] \cup \ell''\end{array}}{\begin{array}{c}ob(\alpha, o, \sigma, \{\ell \mid x = e.\texttt{m}(\overline{e}) \ ; \ s\ \}, q)\ ob(\beta, o', \sigma', p', q') \\ \rightarrow ob(\alpha, o, \sigma, \{\ell \mid x = f \ ; \ s\ \}, q)\ ob(\beta, o', \sigma'', p', q' :: \{\ell' \mid s'\})\ fut(f, \bot)\end{array}}$$

If the configuration has two *objects* $ob(\alpha, o, \sigma, \{\ell \mid x = e.\texttt{m}(\overline{e}); s\}, q)\ \ ob(\beta, o', \sigma', p', q')$, where: the domain of σ is different from that of σ' ($\text{dom}(\sigma) \cap \text{dom}(\sigma') = \varnothing$); $\overline{v}$ is the set of values passed as parameters of the method $\texttt{m}(\overline{e})$; f is a new name for a future, not yet used, obtained from the function fresh( ) ($f = \text{fresh}(\ )$) and the method $\texttt{m}(\overline{e})$ is invoked on an active object ($[\![e]\!]_{(\sigma+\ell)} = o'$) it follows that the rule Invk-Active add a new element to the configuration, that is a *future binders*, and modifies the two objects in the way that will be described below. In the first object changes only the evaluation context ($x = e.\texttt{m}(\overline{e})$), which becomes $x = f$ while the second object undergoes two major changes, as we can see both the function σ' and the stack of processes to be evaluated are changed. The function σ' was updated by adding a new rule $o \mapsto [\overline{x \mapsto v}]$ for each object passed as a parameter to the method `C.m`() and for each field of this objects that is an object too, recursively. The rule Invk-Active adds to the stack of processes to be evaluated a new element $\{\ell' \mid s'\}$ where ℓ' is a function in which all method's parameters are mapped to the values ($[\![\overline{e}]\!]_{(\sigma+\ell)} = \overline{v}$) obtained from the evaluation of expressions passed in the invocation of the method `C.m`(), all local variables of this method are mapped to $\bot$, the special variable `destiny` is mapped to f and *this* becomes the active object in the activity β (o').

(Invk-Passive)

$$\frac{\begin{array}{c}o \in \text{dom}(\sigma) \quad [\![e]\!]_{(\sigma+\ell)} = o' \quad [\![\overline{e}]\!]_{(\sigma+\ell)} = \overline{v} \quad f = \text{fresh}(\ ) \\ \text{bind}(o', m, \overline{v}, \text{class}(o')) = \{\ell'' \mid s'\} \quad \ell' = [\texttt{destiny} \mapsto f, this \mapsto o'] \cup \ell''\end{array}}{ob(\alpha, o, \sigma, \{\ell \mid x = e.\texttt{m}(\overline{e}) \ ; \ s\ \}, q) \rightarrow ob(\alpha, o, \sigma, \{\ell' \mid s'\}, \{\ell \mid x = f \ ; \ s\ \} :: q)}$$

The rule Invk-Passive adds the value of p to the stack of processes to be evaluated, replacing the evaluation context $x = e.\texttt{m}(\overline{e})$ with $x = f$. This rule also changes the active process which becomes $\{\ell' \mid s'\}$ where ℓ' is a function in which all method's parameters are mapped to the values ($[\![\overline{e}]\!]_{(\sigma+\ell)} = \overline{v}$) obtained from the evaluation of expressions passed in the invocation of the method `C.m`(), all local variables of this method are mapped to $\bot$, and *this* becomes the passive object ($[\![e]\!]_{(\sigma+\ell)} = o'$) on which the method `C.m`() was invoked.

$$\frac{v = [\![e]\!]_{(\sigma+\ell)} \quad f = \ell(\texttt{destiny})}{ob(\alpha, o, \sigma, \{\ell \mid \texttt{return}\ e\}, \{\ell' \mid x = f\ ;\ s\ \} :: q) \rightarrow ob(\alpha, o, \sigma, \{\ell' \mid x = v\ ;\ s\ \}, q)}\ (\textsc{ReturnLocal})$$

$$\frac{v = [\![e]\!]_{(\sigma+\ell)} \quad f = \ell(\texttt{destiny})}{ob(\alpha, o, \sigma, \{\ell \mid \texttt{return}\ e\}, q)\ \mathit{fut}(f, \bot) \rightarrow ob(\alpha, o, \sigma, \texttt{idle}, q)\ \mathit{fut}(f, (v, \text{serialize}(v, \sigma)))}\ (\textsc{Return})$$

If the evaluation context is `return` e there are two possibilities, if in configuration doesn't exist the element $\mathit{fut}(f, \bot)$ the rule Return-Local is applied and this means that we are executing a method on a passive object, otherwise it is necessary to apply the rule Return that means that we are executing a method on an active object. In the first case, the evaluation context that was previously placed in the stack of processes to be evaluated, using the rule Activate-Passive, becomes the active process. In the second case the application of rule Return makes inactive the α activity and in the future binding replaces $\bot$ with the result of function serialize(v, σ).

$$\frac{\ell(x) = f \quad \ell' = \ell[x \mapsto v] \quad \sigma'' = \sigma \cup \sigma' \quad \text{dom}(\sigma) \cap \text{dom}(\sigma') = \varnothing}{ob(\alpha, o, \sigma, \{\ell \mid s\}, q)\ \mathit{fut}(f, (v, \sigma')) \rightarrow ob(\alpha, o, \sigma'', \{\ell' \mid s\}, q)\ \mathit{fut}(f, (v, \sigma'))}\ (\textsc{GetLocal})$$

$$\frac{\sigma(o) = [\overline{x \mapsto v}, x \mapsto f] \quad \sigma'' = \sigma[o \mapsto (\sigma(o)[x \mapsto v])] \cup \sigma' \quad \text{dom}(\sigma) \cap \text{dom}(\sigma') = \varnothing}{ob(\alpha, o, \sigma, \{\ell \mid s\}, q)\ \mathit{fut}(f, (v, \sigma')) \rightarrow ob(\alpha, o, \sigma'', \{\ell' \mid s\}, q)\ \mathit{fut}(f, (v, \sigma'))}\ (\textsc{Get})$$

The rule GetLocal and Get are used when a synchronization can be performed. In the two rules we can see that in the configuration we need to have a future binders in which the reply value is not $\bot$, but it's a pair of elements (v, σ'), where v rappresents the value return from a method, and σ' is a function that maps object to object field. The difference between the rules the rule GetLocal and Get is that in the first rule the reply value must be associated to a local variable, instead in the second rule it's must be associated to a object's field.

2.4 Samples of concurrent programs

In this section, we will present some example of code with the aim of better understanding the operational semantics described in the previous sections. Are presented cases of proper execution and cases of deadlock, in which will be clearly visible the impossibility of applying any of the rules presented

Example 1

```
1   class Z(int z){ int getZ(){ return z;} }
2   class O(int x, A y){
3       int getX(){ return x;}
4       A getY(){ return y;}
5     }
6
7   class R() {
8     Z foo(O u){
9       Z h;
10      A c;
11      c = u.getY();
12      h = c.bar( );
13      return h;
14    }
15  }
16
17  class A() {
18    Z bar() {
19      Z g;
20      g = new Z(2);
21      return g;
22    }
23    A m(A target) {
24      O og; R r; R a; Z b; A d;
25      og = new O(4,target);
26      a = newActive R( );
27      b = a.foo(og);
28      d = b.getY();
29      return d;
30    }
31  }
32
33  //MAIN
34  {
35      A a; A b;
36      a = newActive A();
37      b = a.m(a);
38  }
```

$ob(\alpha, o, \epsilon, \texttt{idle}, \{a \mapsto \bot, b \mapsto \bot this \mapsto o \mid a = \texttt{newActive A() ; } s\})$

$\rightarrow$ (Activate)
$ob(\alpha, o, \epsilon, \{a \mapsto \bot, b \mapsto \bot, this \mapsto o \mid a = \texttt{newActive A() ; } s\}, \epsilon)$

$\rightarrow^2$ (New-Active) (Assign-Local)
$ob(\alpha, o, \epsilon, \{a \mapsto o_1, b \mapsto \bot, this \mapsto o \mid b = a.\texttt{m}(a)\}, \epsilon)$
$ob(\gamma, o_1, \epsilon, \texttt{idle}, \epsilon)$

$\rightarrow^3$ (Invk-Active) (Activate) (Assign-Local)
$ob(\alpha, o, \epsilon, \{a \mapsto o_1, b \mapsto f \mid \texttt{skip}\}, \epsilon)$
$ob(\gamma, o_1, \epsilon, \{destiny \mapsto f, this \mapsto o_1, target \mapsto o_1, og \mapsto \bot, a \mapsto \bot, b \mapsto \bot, d \mapsto \bot \mid og = \texttt{new O}(4, target) \texttt{ ; } s_1'\}, \epsilon)$
$fut(f, \bot)$

$\rightarrow^2$ (New Object) (Assign Local)
$ob(\alpha, o, \epsilon, \{a \mapsto o_1, b \mapsto f \mid \texttt{skip}\}, \epsilon)$
$ob(\gamma, o_1, \{o_2 \mapsto [x = 4, y = o_1]\}, \{destiny \mapsto f, this \mapsto o_1, target \mapsto o_1, og \mapsto o_2, a \mapsto \bot, b \mapsto \bot, d \mapsto \bot \mid a = \texttt{newActive R() ; } s_2'\}, \epsilon)$
$fut(f, \bot)$

$\rightarrow^2$ (New Active) (Assign Local)
$ob(\alpha, o, \epsilon, \{a \mapsto o_1, b \mapsto f \mid \texttt{skip}\}, \epsilon)$
$ob(\gamma, o_1, \{o_2 \mapsto [x = 4, y = o_1]\}, \{destiny \mapsto f, this \mapsto o_1, target \mapsto o_1, og \mapsto o_2, a \mapsto o_3, b \mapsto \bot, d \mapsto \bot \mid b = a.foo(og) \texttt{ ; } s_3'\}, \epsilon)$
$fut(f, \bot)$
$ob(\beta, o_3, \epsilon, \texttt{idle}, \epsilon)$

$\rightarrow^2$ (Invk-Active) (Activate)
$ob(\alpha, o, \epsilon, \{a \mapsto o_1, b \mapsto f \mid \texttt{skip}\}, \epsilon)$
$ob(\gamma, o_1, \{o_2 \mapsto [x = 4, y = o_1]\}, \{destiny \mapsto f, this \mapsto o_1, target \mapsto o_1, og \mapsto o_2, a \mapsto o_3, b \mapsto \bot, d \mapsto \bot \mid b = f_1 \texttt{ ; } s_3'\}, \epsilon)$
$fut(f, \bot)$
$ob(\beta, o_3, \{o_2 \mapsto [x = 4, y = o_1]\}, \{destiny \mapsto f_1, this \mapsto o_2, h \mapsto \bot, c \mapsto \bot, u \mapsto o_2 \mid c = u.getY() \texttt{ ; } s_1''\}, \epsilon)$
$fut(f_1, \bot)$

$\rightarrow$ (Assign-Local)
$ob(\alpha, o, \epsilon, \{a \mapsto o_1, b \mapsto f \mid \texttt{skip}\}, \epsilon)$
$ob(\gamma, o_1, \{o_2 \mapsto [x = 4, y = o_1]\}, \{destiny \mapsto f, this \mapsto o_1, target \mapsto o_1, og \mapsto o_2, a \mapsto o_3, b \mapsto f_1, d \mapsto \bot \mid d = b.y \texttt{ ; } s_4'\}, \epsilon)$
$fut(f, \bot)$
$ob(\beta, o_3, \{o_2 \mapsto [x = 4, y = o_1]\}, \{destiny \mapsto f_1, this \mapsto o_2, h \mapsto \bot, c \mapsto \bot, u \mapsto o_2 \mid c = u.y \texttt{ ; } s_1''\}, \epsilon)$
$fut(f_1, \bot)$

$\rightarrow$ (Invk-Passive)

$ob(\alpha, o, \epsilon, \{a \mapsto o_1, b \mapsto f \mid \texttt{skip}\}, \epsilon)$
$ob(\gamma, o_1, \{o_2 \mapsto [x = 4, y = o_1]\}, \{destiny \mapsto f, this \mapsto o_1, target \mapsto o_1, og \mapsto o_2, a \mapsto o_3, b \mapsto \bot, d \mapsto \bot \mid b = f_1 \ ; \ s'_3\}, \epsilon)$
$fut(f, \bot)$
$ob(\beta, o_3, \{o_2 \mapsto [x = 4, y = o_1]\}, this \mapsto o_2 \mid \texttt{return}\ y, \{destiny \mapsto f_1, this \mapsto o_2, h \mapsto \bot, c \mapsto \bot, u \mapsto o_2 \mid c = \bullet \ ; \ s''_1\})$
$fut(f_1, \bot)$

$\rightarrow$ (Return-Local)

$ob(\alpha, o, \epsilon, \{a \mapsto o_1, b \mapsto f \mid \texttt{skip}\}, \epsilon)$
$ob(\gamma, o_1, \{o_2 \mapsto [x = 4, y = o_1]\}, \{destiny \mapsto f, this \mapsto o_1, target \mapsto o_1, og \mapsto o_2, a \mapsto o_3, b \mapsto \bot, d \mapsto \bot \mid b = f_1 \ ; \ s'_3\}, \epsilon)$
$fut(f, \bot)$
$ob(\beta, o_3, \{o_2 \mapsto [x = 4, y = o_1]\}, \{destiny \mapsto f_1, this \mapsto o_2, h \mapsto \bot, c \mapsto \bot, u \mapsto o_2 \mid c = o_1 \ ; \ s''_1\}, \epsilon)$
$fut(f_1, \bot)$

$\rightarrow$ (Assign-Local)
$ob(\alpha, o, \epsilon, \{a \mapsto o_1, b \mapsto f \mid \texttt{skip}\}, \epsilon)$
$ob(\gamma, o_1, \{o_2 \mapsto [x = 4, y = o_1]\}, \{destiny \mapsto f, this \mapsto o_1, target \mapsto o_1, og \mapsto o_2, a \mapsto o_3, b \mapsto f_1, d \mapsto \bot \mid d = b.\texttt{g}etY() \ ; \ s'_4\}, \epsilon)$
$fut(f, \bot)$
$ob(\beta, o_3, \{o_2 \mapsto [x = 4, y = o_1]\}, \{destiny \mapsto f_1, this \mapsto o_2, h \mapsto \bot, c \mapsto o_1, u \mapsto o_2 \mid h = c.\texttt{b}ar() \ ; \ s''_2\}, \epsilon)$
$fut(f_1, \bot)$

$\rightarrow$ (Invk-Active)
$ob(\alpha, o, \epsilon, \{a \mapsto o_1, b \mapsto f \mid \texttt{skip}\}, \epsilon)$
$ob(\gamma, o_1, \{o_2 \mapsto [x = 4, y = o_1]\}, \{destiny \mapsto f, this \mapsto o_1, target \mapsto o_1, og \mapsto o_2, a \mapsto o_3, b \mapsto f_1, d \mapsto \bot \mid d = b.\texttt{g}etY() \ ; \ s'_4\}, \{g \mapsto \bot, destiny \mapsto f_2, this \mapsto o_1 \mid g = \texttt{new Z}(2) \ ; \ s'''_1\})$
$fut(f, \bot)$
$ob(\beta, o_3, \{o_2 \mapsto [x = 4, y = o_1]\}, \{destiny \mapsto f_1, this \mapsto o_2, h \mapsto \bot, c \mapsto o_1, u \mapsto o_2 \mid h = f_2 \ ; \ s''_2\}, \epsilon)$
$fut(f_1, \bot)$
$fut(f_2, \bot)$

No Rule to Apply!

Example 2

```
1   class Z(int z){
2       int getZ(){ return z;}
3   }
4
5   class O(int x, A y){
6
7       int getX(){ return x;}
8
9       A getY(){ return y;}
10  }
11
12  class R() {
13    Z foo(O u){
14      Z h;
15      A c;
16      c = u.getY();
17      h = c.bar( );
18      return h;
19    }
20  }
21
22  class A() {
23
24    Z bar() {
25      Z g;
26      g = new Z(2);
27      return g;
28    }
29
30    A m(A target) {
31      O og; R r; R a; Z b; A d;
32      og = new O(4,target);
33      a = newActive R( );
34      b = a.foo(og);
35      d = b.getY();
36      return d;
37    }
38  }
39
40  //MAIN
41  {
42      A a; A b; A c;
43      a = newActive A();
44      c = new A();
45      b = a.m(c);
46  }
```

$ob(\alpha, o, \epsilon, \{a \mapsto \bot, b \mapsto \bot, c \mapsto \bot \mid \text{a = new Active A(); } s_1\}, \epsilon)$

$\rightarrow^2$ (New Active) (Assign Local)
$ob(\alpha, o, \epsilon, \{a \mapsto o_1, b \mapsto \bot, c \mapsto \bot \mid \text{c = new A(); } s_2\}, \epsilon)$
$ob(\gamma, o_1, \epsilon, \texttt{idle}, \epsilon)$

$\rightarrow^2$ (New Object) (Assign Local)
$ob(\alpha, o, \epsilon, \{a \mapsto o_1, b \mapsto \bot, c \mapsto o_4 \mid \text{b = a.m(c); } s_3\}, \epsilon)$
$ob(\gamma, o_1, \epsilon, \texttt{idle}, \epsilon)$

$\rightarrow^3$ (Invk-Active) (Activate) (Assign-Local)
$ob(\alpha, o, \epsilon, \{a \mapsto o_1, b \mapsto f, c \mapsto o_4 \mid \text{skip;}\}, \epsilon)$
$ob(\gamma, o_1, \epsilon, \{destiny \mapsto f, target \mapsto o_4, og, a, b, d \mapsto \bot \mid \text{og = new O(4, target); } s_1'\}, \epsilon)$
$fut(f, \bot)$

$\rightarrow^2$ (New Object) (Assign Local)
$ob(\alpha, o, \epsilon, \{a \mapsto o_1, b \mapsto f, c \mapsto o_4 \mid \text{skip;}\}, \epsilon)$
$ob(\gamma, o_1, \{o_2 \mapsto [x = 4, y = o_4]\}, \{destiny \mapsto f, target \mapsto o_4, og \mapsto o_2, a, b, d \mapsto \bot \mid$
$\text{a = new Active R(); } s_2'\}, \epsilon)$
$fut(f, \bot)$

$\rightarrow^2$ (New Active) (Assign Local)
$ob(\alpha, o, \epsilon, \{a \mapsto o_1, b \mapsto f, c \mapsto o_4 \mid \text{skip;}\}, \epsilon)$
$ob(\gamma, o_1, \{o_2 \mapsto [x = 4, y = o_4]\}, \{destiny \mapsto f, target \mapsto o_4, og \mapsto o_2, a \mapsto o_3, b, d \mapsto \bot \mid$
$\text{b = a.foo(og); } s_3'\}, \epsilon)$
$fut(f, \bot)$
$ob(\beta, o_3, \epsilon, \texttt{idle}, \epsilon)$

$\rightarrow^3$ (Invk-Active) (Activate) (Assign-Local)
$ob(\alpha, o, \epsilon, \{a \mapsto o_1, b \mapsto f, c \mapsto o_4 \mid \text{skip;}\}, \epsilon)$
$ob(\gamma, o_1, \{o_2 \mapsto [x = 4, y = o_4]\}, \{destiny \mapsto f, target \mapsto o_4, og \mapsto o_2, a \mapsto o_3, b \mapsto f_1, d \mapsto$
$\bot \mid \text{d = b.y; } s_4'\}, \epsilon)$
$fut(f, \bot)$
$ob(\beta, o_3, \{o_2 \mapsto [x = 4, y = o_4]\}, \{destiny \mapsto f_1, h, c \mapsto \bot, u \mapsto o_2 \mid \text{c = u.y; } s_1''\}, \epsilon)$
$fut(f_1, \bot)$

$\rightarrow$ (Assign Local)
$ob(\alpha, o, \epsilon, \{a \mapsto o_1, b \mapsto f, c \mapsto o_4 \mid \text{skip;}\}, \epsilon)$
$ob(\gamma, o_1, \{o_2 \mapsto [x = 4, y = o_4]\}, \{destiny \mapsto f, target \mapsto o_4, og \mapsto o_2, a \mapsto o_3, b \mapsto f_1, d \mapsto$
$\bot \mid \text{d = b.y; } s_4'\}, \epsilon)$
$fut(f, \bot)$
$ob(\beta, o_3, \{o_2 \mapsto [x = 4, y = o_4]\}, \{destiny \mapsto f_1, h \mapsto \bot, c \mapsto o_4, u \mapsto o_2 \mid \text{h = c.bar(); } s_2''\}, \epsilon)$
$fut(f_1, \bot)$

$\rightarrow$ (Invk-Passive)
$ob(\alpha, o, \epsilon, \{a \mapsto o_1, b \mapsto f, c \mapsto o_4 \mid \text{skip;}\}, \epsilon)$
$ob(\gamma, o_1, \{o_2 \mapsto [x = 4, y = o_4]\}, \{destiny \mapsto f, target \mapsto o_4, og \mapsto o_2, a \mapsto o_3, b \mapsto f_1, d \mapsto$
$\bot \mid \text{d = b.y; } s_4'\}, \epsilon)$
$fut(f, \bot)$

$ob(\beta, o_3, \{o_2 \mapsto [x = 4, y = o_4]\}, \{this \mapsto o_4, g \mapsto \bot \mid \text{g = new Z(2); } s'''_1\}, \{destiny \mapsto f_1, h \mapsto \bot, c \mapsto o_4, u \mapsto o_2 \mid \text{h} = \bullet \text{ ; } s''_2\})$
$fut(f_1, \bot)$

$\to^2$ (New Object)
$ob(\alpha, o, \epsilon, \{a \mapsto o_1, b \mapsto f, c \mapsto o_4 \mid \text{skip;}\}, \epsilon)$
$ob(\gamma, o_1, \{o_2 \mapsto [x = 4, y = o_4]\}, \{destiny \mapsto f, target \mapsto o_4, og \mapsto o_2, a \mapsto o_3, b \mapsto f_1, d \mapsto \bot \mid \text{d = b.y; } s'_4\}, \epsilon)$
$fut(f, \bot)$
$ob(\beta, o_3, \{o_2 \mapsto [x = 4, y = o_4], o_5 \mapsto [z = 2]\}, \{this \mapsto o_4, g \mapsto o_5 \mid \text{return g;}\}, \{destiny \mapsto f_1, h \mapsto \bot, c \mapsto o_4, u \mapsto o_2 \mid \text{h} = \bullet \text{ ; } s''_2\})$
$fut(f_1, \bot)$

$\to$ (Return Local)
$ob(\alpha, o, \epsilon, \{a \mapsto o_1, b \mapsto f, c \mapsto o_4 \mid \text{skip;}\}, \epsilon)$
$ob(\gamma, o_1, \{o_2 \mapsto [x = 4, y = o_4]\}, \{destiny \mapsto f, target \mapsto o_4, og \mapsto o_2, a \mapsto o_3, b \mapsto f_1, d \mapsto \bot \mid \text{d = b.y; } s'_4\}, \epsilon)$
$fut(f, \bot)$
$ob(\beta, o_3, \{o_2 \mapsto [x = 4, y = o_4], o_5 \mapsto [z = 2]\}, \{destiny \mapsto f_1, h \mapsto \bot, c \mapsto o_4, u \mapsto o_2 \mid \text{h} = o_5 \text{ ; } s''_2\}, \epsilon)$
$fut(f_1, \bot)$

$\to$ (Assign Local)
$ob(\alpha, o, \epsilon, \{a \mapsto o_1, b \mapsto f, c \mapsto o_4 \mid \text{skip;}\}, \epsilon)$
$ob(\gamma, o_1, \{o_2 \mapsto [x = 4, y = o_4]\}, \{destiny \mapsto f, target \mapsto o_4, og \mapsto o_2, a \mapsto o_3, b \mapsto f_1, d \mapsto \bot \mid \text{d = b.y; } s'_4\}, \epsilon)$
$fut(f, \bot)$
$ob(\beta, o_3, \{o_2 \mapsto [x = 4, y = o_4], o_5 \mapsto [z = 2]\}, \{destiny \mapsto f_1, h \mapsto o_5, c \mapsto o_4, u \mapsto o_2 \mid \text{return u;}\}, \epsilon)$
$fut(f_1, \bot)$

$\to$ (Return)
$ob(\alpha, o, \epsilon, \{a \mapsto o_1, b \mapsto f, c \mapsto o_4 \mid \text{skip;}\}, \epsilon)$
$ob(\gamma, o_1, \{o_2 \mapsto [x = 4, y = o_4]\}, \{destiny \mapsto f, target \mapsto o_4, og \mapsto o_2, a \mapsto o_3, b \mapsto f_1, d \mapsto \bot \mid \text{d = b.y; } s'_4\}, \epsilon)$
$fut(f, \bot)$
$ob(\beta, o_3, \{o_2 \mapsto [x = 4, y = o_4], o_5 \mapsto [z = 2]\}, \texttt{idle}, \epsilon)$
$fut(f_1, (o_2, \{o_2 \mapsto [x = 4, y = o_4]\}))$

$\to$ (Get-Local)
$ob(\alpha, o, \epsilon, \{a \mapsto o_1, b \mapsto f, c \mapsto o_4 \mid \text{skip;}\}, \epsilon)$
$ob(\gamma, o_1, \{o_2 \mapsto [x = 4, y = o_4]\}, \{destiny \mapsto f, target \mapsto o_4, og \mapsto o_2, a \mapsto o_3, b \mapsto o_2, d \mapsto \bot \mid \text{d = b.y; } s'_4\}, \epsilon)$
$fut(f, \bot)$
$ob(\beta, o_3, \{o_2 \mapsto [x = 4, y = o_4], o_5 \mapsto [z = 2]\}, \texttt{idle}, \epsilon)$
$fut(f_1, (o_2, \{o_2 \mapsto [x = 4, y = o_4]\}))$

$\to$ (Assign-Local)
$ob(\alpha, o, \epsilon, \{a \mapsto o_1, b \mapsto f, c \mapsto o_4 \mid \text{skip;}\}, \epsilon)$

$ob(\gamma, o_1, \{o_2 \mapsto [x = 4, y = o_4]\}, \{destiny \mapsto f, target \mapsto o_4, og \mapsto o_2, a \mapsto o_3, b \mapsto o_2, d \mapsto o_4 \mid \text{return d;}\}, \epsilon)$
$fut(f, \bot)$
$ob(\beta, o_3, \{o_2 \mapsto [x = 4, y = o_4], o_5 \mapsto [z = 2]\}, \texttt{idle}, \epsilon)$
$fut(f_1, (o_2, \{o_2 \mapsto [x = 4, y = o_4]\}))$

$\rightarrow$ (Return)
$ob(\alpha, o, \epsilon, \{a \mapsto o_1, b \mapsto f, c \mapsto o_4 \mid \text{skip;}\}, \epsilon)$
$ob(\gamma, o_1, \{o_2 \mapsto [x = 4, y = o_4]\}, \texttt{idle}, \epsilon)$
$fut(f, (o_2, \{o_2 \mapsto [x = 4, y = o_4]\}))$
$ob(\beta, o_3, \{o_2 \mapsto [x = 4, y = o_4], o_5 \mapsto [z = 2]\}, \texttt{idle}, \epsilon)$
$fut(f_1, (o_2, \{o_2 \mapsto [x = 4, y = o_4]\}))$

$\rightarrow$ (Get-Local)
$ob(\alpha, o, \epsilon, \{a \mapsto o_1, b \mapsto o_4, c \mapsto o_4 \mid \text{skip;}\}, \epsilon)$
$ob(\gamma, o_1, \{o_2 \mapsto [x = 4, y = o_4]\}, \texttt{idle}, \epsilon)$
$fut(f, (o_2, \{o_2 \mapsto [x = 4, y = o_4]\}))$
$ob(\beta, o_3, \{o_2 \mapsto [x = 4, y = o_4], o_5 \mapsto [z = 2]\}, \texttt{idle}, \epsilon)$
$fut(f_1, (o_2, \{o_2 \mapsto [x = 4, y = o_4]\}))$

Example 3

```
1
2       class A {
3         int ac; int ah;
4
5         int metA1(int ad){
6           int al;
7           int ae;
8           ae = ab + aa;
9           ah = ogB.metB1(this);
10          al = ad + ae;
11          ac = 2 + ah;
12          return ac + al;
13        }
14
15        int getAh(){ return ah; }
16
17        int getAc(){ return ac; }
18      }
19
20      class B {
21        int bb; int ba;
22
23        int metB1(A ogA){
24          ba = ogA.getAh();
25          bb = 3 + ba;
26          return bb;
27        }
28      }
29
30      //MAIN
31      {
32          A z; B w;
33          int x;
34          int y
35          w = newActive B();
36          z = newActive A(3,2,w);
37          y = z.metA1(5);
38          x = z.getAc();
39          return x;
40      }
```

$ob(\alpha, o, \epsilon, \texttt{idle}, \{z \mapsto \bot, w \mapsto \bot, x \mapsto \bot, y \mapsto \bot, this \mapsto o \mid w = \texttt{newActive B() ; } s\})$

$\rightarrow$ (ACTIVATE)
$ob(\alpha, o, \epsilon, \{z \mapsto \bot, w \mapsto \bot, x \mapsto \bot, y \mapsto \bot, this \mapsto o \mid w = \texttt{newActive B() ; } s\}, \epsilon)$

$\rightarrow^2$ (New-Active) (Assign-Local)

$ob(\alpha, o, \epsilon, \{z \mapsto \bot, w \mapsto o_1, x \mapsto \bot, y \mapsto \bot, this \mapsto o \mid z = \texttt{newActive A}(3, 2, w)\ ;\ s_1\}, \epsilon)$
$ob(\beta, o_1, \{o_1 \mapsto [bb \mapsto \bot, ba \mapsto \bot]\}, \texttt{idle}, \epsilon)$

$\rightarrow^2$ (New-Active) (Assign-Local)

$ob(\alpha, o, \epsilon, \{z \mapsto o_2, w \mapsto o_1, x \mapsto \bot, y \mapsto \bot, this \mapsto o \mid y = z.\mathtt{m}etA1(5)\ ;\ s_2\}, \epsilon)$
$ob(\beta, o_1, \{o_1 \mapsto [bb \mapsto \bot, ba \mapsto \bot]\}, \texttt{idle}, \epsilon)$
$ob(\gamma, o_2, \{o_2 \mapsto [ac \mapsto \bot, ah \mapsto \bot, aa \mapsto 3, ab \mapsto 2, ogB \mapsto o_1]\}, \texttt{idle}, \epsilon)$

$\rightarrow^3$ (Invk-Active) (Activate) (Assign-Local)

$ob(\alpha, o, \epsilon, \{z \mapsto o_2, w \mapsto o_1, x \mapsto \bot, y \mapsto f, this \mapsto o \mid x = z.\mathtt{g}etAc()\ ;\ s_3\}, \epsilon)$
$ob(\beta, o_1, \{o_1 \mapsto [bb \mapsto \bot, ba \mapsto \bot]\}, \texttt{idle}, \epsilon)$
$ob(\gamma, o_2, \{o_2 \mapsto [ac \mapsto \bot, ah \mapsto \bot, aa \mapsto 3, ab \mapsto 2, ogB \mapsto o_1], o_1 \mapsto [bb \mapsto \bot, ba \mapsto \bot]\}, \{destiny \mapsto f, this \mapsto o_2, al \mapsto \bot, ae \mapsto \bot, ad \mapsto 5 \mid ae = ab + aa\ ;\ s'\}, \epsilon)$
$fut(f, \bot)$

$\rightarrow^2$ (Invk-Active) (Assign-Local)

$ob(\alpha, o, \epsilon, \{z \mapsto o_2, w \mapsto o_1, x \mapsto f_1, y \mapsto f, this \mapsto o \mid \texttt{return}\ x\}, \epsilon)$
$ob(\beta, o_1, \{o_1 \mapsto [bb \mapsto \bot, ba \mapsto \bot]\}, \texttt{idle}, \epsilon)$
$ob(\gamma, o_2, \{o_2 \mapsto [ac \mapsto \bot, ah \mapsto \bot, aa \mapsto 3, ab \mapsto 2, ogB \mapsto o_1], o_1 \mapsto [bb \mapsto \bot, ba \mapsto \bot]\}, \{destiny \mapsto f, this \mapsto o_2, al \mapsto \bot, ae \mapsto \bot, ad \mapsto 5 \mid ae = ab + aa\ ;\ s'\}, \{destiny \mapsto f_1, this \mapsto o_2 \mid \texttt{return}\ ac\})$
$fut(f, \bot)$
$fut(f_1, \bot)$

$\rightarrow$ (Assign-Local)

$ob(\alpha, o, \epsilon, \{z \mapsto o_2, w \mapsto o_1, x \mapsto f_1, y \mapsto f, this \mapsto o \mid \texttt{return}\ x\}, \epsilon)$
$ob(\beta, o_1, \{o_1 \mapsto [bb \mapsto \bot, ba \mapsto \bot]\}, \texttt{idle}, \epsilon)$
$ob(\gamma, o_2, \{o_2 \mapsto [ac \mapsto \bot, ah \mapsto \bot, aa \mapsto 3, ab \mapsto 2, ogB \mapsto o_1], o_1 \mapsto [bb \mapsto \bot, ba \mapsto \bot]\}, \{destiny \mapsto f, this \mapsto o_2, al \mapsto \bot, ae \mapsto 5, ad \mapsto 5 \mid ah = ogB.\mathtt{m}etB1(this)\ ;\ s'_1\}, \{destiny \mapsto f_1, this \mapsto o_2 \mid \texttt{return}\ ac\})$
$fut(f, \bot)$
$fut(f_1, \bot)$

$\rightarrow^3$ (Invk-Active) (Activate) (Assign-Field)

$ob(\alpha, o, \epsilon, \{z \mapsto o_2, w \mapsto o_1, x \mapsto f_1, y \mapsto f, this \mapsto o \mid \texttt{return}\ x\}, \epsilon)$
$ob(\beta, o_1, \{o_1 \mapsto [bb \mapsto \bot, ba \mapsto \bot], o_2 \mapsto [ac \mapsto \bot, ah \mapsto f_2, aa \mapsto 3, ab \mapsto 2, ogB \mapsto o_1]\}, \{destiny \mapsto f_2, this \mapsto o_2, ogA \mapsto o_2 \mid ba = ogA.\mathtt{g}etAh()\ ;\ s''\}, \epsilon)$
$ob(\gamma, o_2, \{o_2 \mapsto [ac \mapsto \bot, ah \mapsto f_2, aa \mapsto 3, ab \mapsto 2, ogB \mapsto o_1], o_1 \mapsto [bb \mapsto \bot, ba \mapsto \bot]\}, \{destiny \mapsto f, this \mapsto o_2, al \mapsto \bot, ae \mapsto 5, ad \mapsto 5 \mid al = ad + ae\ ;\ s'_2\}, \{destiny \mapsto f_1, this \mapsto o_2 \mid \texttt{return}\ ac\})$
$fut(f, \bot)$

$fut(f_1, \bot)$
$fut(f_2, \bot)$

$\rightarrow$ (Assign-Local)

$ob(\alpha, o, \epsilon, \{z \mapsto o_2, w \mapsto o_1, x \mapsto f_1, y \mapsto f, this \mapsto o \mid \texttt{return}\ x\}, \epsilon)$
$ob(\beta, o_1, \{o_1 \mapsto [bb \mapsto \bot, ba \mapsto \bot], o_2 \mapsto [ac \mapsto \bot, ah \mapsto f_2, aa \mapsto 3, ab \mapsto 2, ogB \mapsto o_1]\}, \{destiny \mapsto f_2, this \mapsto o_2, ogA \mapsto o_2 \mid ba = ogA.\texttt{g}etAh()\ ;\ s''\}, \epsilon)$
$ob(\gamma, o_2, \{o_2 \mapsto [ac \mapsto \bot, ah \mapsto f_2, aa \mapsto 3, ab \mapsto 2, ogB \mapsto o_1], o_1 \mapsto [bb \mapsto \bot, ba \mapsto \bot]\}, \{destiny \mapsto f, this \mapsto o_2, al \mapsto 10, ae \mapsto 5, ad \mapsto 5 \mid ac = 2 + ah\ ;\ s'_3\}, \{destiny \mapsto f_1, this \mapsto o_2 \mid \texttt{return}\ ac\})$
$fut(f, \bot)$
$fut(f_1, \bot)$
$fut(f_2, \bot)$

$\rightarrow^2$ (Invk-Active) (Assign-Field)

$ob(\alpha, o, \epsilon, \{z \mapsto o_2, w \mapsto o_1, x \mapsto f_1, y \mapsto f, this \mapsto o \mid \texttt{return}\ x\}, \epsilon)$
$ob(\beta, o_1, \{o_1 \mapsto [bb \mapsto \bot, ba \mapsto f_3], o_2 \mapsto [ac \mapsto \bot, ah \mapsto f_2, aa \mapsto 3, ab \mapsto 2, ogB \mapsto o_1]\}, \{destiny \mapsto f_2, this \mapsto o_2, ogA \mapsto o_2 \mid bb = 3 + ab\ ;\ s''_1\}, \epsilon)$
$ob(\gamma, o_2, \{o_2 \mapsto [ac \mapsto \bot, ah \mapsto f_2, aa \mapsto 3, ab \mapsto 2, ogB \mapsto o_1], o_1 \mapsto [bb \mapsto \bot, ba \mapsto f_3]\}, \{destiny \mapsto f, this \mapsto o_2, al \mapsto 10, ae \mapsto 5, ad \mapsto 5 \mid ac = 2 + ah\ ;\ s'_3\}, \{destiny \mapsto f_3, this \mapsto o_2 \mid \texttt{return}\ ah\} :: \{destiny \mapsto f_1, this \mapsto o_2 \mid \texttt{return}\ ac\})$
$fut(f, \bot)$
$fut(f_1, \bot)$
$fut(f_2, \bot)$
$fut(f_3, \bot)$

Example 4

```
1
2       class A(B ogB, int aa){
3         int ac;
4         int ag;
5
6         int metA1(int ad){
7           int ae;
8           ac = ad + aa;
9           ae = agB.metB1(ac,this);
10          ag = ac *3;
11          ac = ae + 2;
12          return ae;
13        }
14      }
15
16      class B(int ba) {
17
18        int metB1(int bb, A ogA){
19          int bc;
20          int bd;
21          bc = ba + bb;
22          bd = odA.metA1(bc);
23          bc = bc + 3;
24          return bc;
25        }
26      }
27
28      //MAIN
29      {
30          A a;
31          B b;
32          int z;
33          b = new B(2);
34          a = new A(b,3);
35          z = b.metB1(1,a);
36      }
```

$ob(\alpha, o, \epsilon, \texttt{idle}, \{a \mapsto \bot, b \mapsto \bot, z \mapsto \bot, this \mapsto o \mid b = \texttt{new B(2)} \ ; \ s_1\})$

$\rightarrow$ (Activate)

$ob(\alpha, o, \epsilon, \{a \mapsto \bot, b \mapsto \bot, z \mapsto \bot, this \mapsto o \mid b = \texttt{new B(2)} \ ; \ s_2\}, \epsilon)$

$\rightarrow^2$ (New-Object) (Assign-Local)

$ob(\alpha, o, \{o_1 \mapsto [ba \mapsto 2]\}, \{a \mapsto \bot, b \mapsto o_1, z \mapsto \bot, this \mapsto o \mid a = newA(b,3)\ ;\ s_2\}, \epsilon)$

$\to^2$ (New-Object) (Assign-Local)

$ob(\alpha, o, \{o_1 \mapsto [ba \mapsto 2], o_2 \mapsto [ogB \mapsto o_1, aa \mapsto 3, ac \mapsto \bot, ag \mapsto \bot]\}, \{a \mapsto o_2, b \mapsto o_1, z \mapsto \bot, this \mapsto o \mid z = b.\mathtt{m}etB1(1,a)\}, \epsilon)$

$\to$ (Invk-Passive)

$ob(\alpha, o, \{o_1 \mapsto [ba \mapsto 2], o_2 \mapsto [ogB \mapsto o_1, aa \mapsto 3, ac \mapsto \bot, ag \mapsto \bot]\}, \{bb \mapsto 1, ogA \mapsto o_2, bc \mapsto \bot, bd \mapsto \bot, this \mapsto o_1 \mid bc = ba + bb\ ;\ s_1'\}, \{a \mapsto o_2, b \mapsto o_1, z \mapsto \bot, this \mapsto o \mid z = \bullet\})$

$\to$ (Assign-Local)

$ob(\alpha, o, \{o_1 \mapsto [ba \mapsto 2], o_2 \mapsto [ogB \mapsto o_1, aa \mapsto 3, ac \mapsto \bot, ag \mapsto \bot]\}, \{bb \mapsto 1, ogA \mapsto o_2, bc \mapsto 3, bd \mapsto \bot, this \mapsto o_1 \mid bd = ogA.\mathtt{m}etA1(bc)\ ;\ s_2'\}, \{a \mapsto o_2, b \mapsto o_1, z \mapsto \bot, this \mapsto o \mid z = \bullet\})$

$\to$ (Invk-Passive)

$ob(\alpha, o, \{o_1 \mapsto [ba \mapsto 2], o_2 \mapsto [ogB \mapsto o_1, aa \mapsto 3, ac \mapsto \bot, ag \mapsto \bot]\}, \{ad \mapsto 3, ae \mapsto \bot, this \mapsto o_2 \mid ac = ad + aa\ ;\ s_1''\}, \{bb \mapsto 1, ogA \mapsto o_2, bc \mapsto 3, bd \mapsto \bot, this \mapsto o_1 \mid bd = \bullet\ ;\ s_2'\} :: \{a \mapsto o_2, b \mapsto o_1, z \mapsto \bot, this \mapsto o \mid z = \bullet\})$

$\to$ (Assign-Field)

$ob(\alpha, o, \{o_1 \mapsto [ba \mapsto 2], o_2 \mapsto [ogB \mapsto o_1, aa \mapsto 3, ac \mapsto 6, ag \mapsto \bot]\}, \{ad \mapsto 3, ae \mapsto \bot, this \mapsto o_2 \mid ae = ogB.\mathtt{m}etB1(ac, this)\ ;\ s_2''\}, \{bb \mapsto 1, ogA \mapsto o_2, bc \mapsto 3, bd \mapsto \bot, this \mapsto o_1 \mid bd = \bullet\ ;\ s_2'\} :: \{a \mapsto o_2, b \mapsto o_1, z \mapsto \bot, this \mapsto o \mid z = \bullet\})$

$\to$ (Invk-Passive)

$ob(\alpha, o, \{o_1 \mapsto [ba \mapsto 2], o_2 \mapsto [ogB \mapsto o_1, aa \mapsto 3, ac \mapsto 6, ag \mapsto \bot]\}, \{bb \mapsto 6, ogA \mapsto o_2, bc \mapsto \bot, bd \mapsto \bot, this \mapsto o_1 \mid bc = ba + bb\ ;\ s_1'''\}, \{ad \mapsto 3, ae \mapsto \bot, this \mapsto o_2 \mid ae = \bullet\ ;\ s_2''\} :: \{bb \mapsto 1, ogA \mapsto o_2, bc \mapsto 3, bd \mapsto \bot, this \mapsto o_1 \mid bd = \bullet\ ;\ s_2'\} :: \{a \mapsto o_2, b \mapsto o_1, z \mapsto \bot, this \mapsto o \mid z = \bullet\})$

$\to$ (Assign-Local)

$ob(\alpha, o, \{o_1 \mapsto [ba \mapsto 2], o_2 \mapsto [ogB \mapsto o_1, aa \mapsto 3, ac \mapsto 6, ag \mapsto \bot]\}, \{bb \mapsto 6, ogA \mapsto o_2, bc \mapsto 8, bd \mapsto \bot, this \mapsto o_1 \mid bd = ogA.\mathtt{m}etA1(bc)\ ;\ s_2'''\}, \{ad \mapsto 3, ae \mapsto \bot, this \mapsto o_2 \mid ae = \bullet\ ;\ s_2''\} :: \{bb \mapsto 1, ogA \mapsto o_2, bc \mapsto 3, bd \mapsto \bot, this \mapsto o_1 \mid bd = \bullet\ ;\ s_2'\} :: \{a \mapsto o_2, b \mapsto o_1, z \mapsto \bot, this \mapsto o \mid z = \bullet\})$

$\to$ (Invk-Passive)
ecc...

Example 5

```
1
2       class A(B ogB, int aa){
3         int ac;
4         int ag;
5
6         int metA1(int ad){
7           int ae;
8           ac = ad + aa;
9           ae = agB.metB1(ac,this);
10          ag = ac *3;
11          ac = ae + 2;
12          return ae;
13        }
14      }
15
16      class B(int ba) {
17
18        int metB1(int bb, A ogA){
19          int bc;
20          int bd;
21          bc = ba + bb;
22          bd = odA.metA1(bc);
23          bc = bc + 3;
24          return bc;
25        }
26      }
27
28      //MAIN
29      {
30          A a;
31          B b;
32          int z;
33          b = newActiveB(2);
34          a = new A(b,3);
35          z = b.metB1(1,a);
36      }
```

$ob(\alpha, o, \epsilon, \texttt{idle}, \{a \mapsto \bot, b \mapsto \bot, z \mapsto \bot, this \mapsto o \mid b = \texttt{newActive B}(2)\ ;\ s_1\})$

$\rightarrow$ (Activate)

$ob(\alpha, o, \epsilon, \{a \mapsto \bot, b \mapsto \bot, z \mapsto \bot, this \mapsto o \mid b = \texttt{new B}(2)\ ;\ s_1\}, \epsilon)$

$\rightarrow^2$ (New-Active) (Assign-Local)

$ob(\alpha, o, \epsilon, \{a \mapsto \bot, b \mapsto o_1, z \mapsto \bot, this \mapsto o \mid a = \texttt{new A}(b, 3)\ ;\ s_2\}, \epsilon)$
$ob(\beta, o_1, \{o_1 \mapsto [ba \mapsto 2]\}, \texttt{idle}, \epsilon)$

$\rightarrow^2$ (New-Object) (Assign-Local)

$ob(\alpha, o, \{o_2 \mapsto [ogB \mapsto o_1, aa \mapsto 3, ac \mapsto \bot, ag \mapsto \bot]\}, \{a \mapsto o_2, b \mapsto o_1, z \mapsto \bot, this \mapsto o \mid z = b.\mathtt{m}etB1(1, a)\}, \epsilon)$
$ob(\beta, o_1, \{o_1 \mapsto [ba \mapsto 2]\}, \texttt{idle}, \epsilon)$

$\rightarrow^2$ (Invk-Active) (Assign-Local)

$ob(\alpha, o, \{o_2 \mapsto [ogB \mapsto o_1, aa \mapsto 3, ac \mapsto \bot, ag \mapsto \bot]\}, \{a \mapsto o_2, b \mapsto o_1, z \mapsto f, this \mapsto o \mid \texttt{skip}\}, \epsilon)$
$ob(\beta, o_1, \{o_1 \mapsto [ba \mapsto 2], o_2 \mapsto [ogB \mapsto o_1, aa \mapsto 3, ac \mapsto \bot, ag \mapsto \bot]\}, \texttt{idle}, \{bb \mapsto 1, ogA \mapsto o_2, bc \mapsto \bot, bd \mapsto \bot, this \mapsto o_1, destiny \mapsto f \mid bc = ba + bb\ ;\ s_1'\})$
$fut(f, \bot)$

$\rightarrow$ (Activate)
$ob(\alpha, o, \{o_2 \mapsto [ogB \mapsto o_1, aa \mapsto 3, ac \mapsto \bot, ag \mapsto \bot]\}, \{a \mapsto o_2, b \mapsto o_1, z \mapsto f, this \mapsto$
$, \mid \texttt{skip}\}, \epsilon)$
$ob(\beta, o_1, \{o_1 \mapsto [ba \mapsto 2], o_2 \mapsto [ogB \mapsto o_1, aa \mapsto 3, ac \mapsto \bot, ag \mapsto \bot]\}, \{bb \mapsto 1, ogA \mapsto o_2, bc \mapsto \bot, bd \mapsto \bot, this \mapsto o_1, destiny \mapsto f \mid bc = ba + bb\ ;\ s_1'\}, \epsilon)$
$fut(f, \bot)$

$\rightarrow$ (Assign-Local)

$ob(\alpha, o, \{o_2 \mapsto [ogB \mapsto o_1, aa \mapsto 3, ac \mapsto \bot, ag \mapsto \bot]\}, \{a \mapsto o_2, b \mapsto o_1, z \mapsto f, this \mapsto o \mid \texttt{skip}\}, \epsilon)$
$ob(\beta, o_1, \{o_1 \mapsto [ba \mapsto 2], o_2 \mapsto [ogB \mapsto o_1, aa \mapsto 3, ac \mapsto \bot, ag \mapsto \bot]\}, \{bb \mapsto 1, ogA \mapsto o_2, bc \mapsto 3, bd \mapsto \bot, this \mapsto o_1, destiny \mapsto f \mid bd = odA.\mathtt{m}etA1(bc)\ ;\ s_2'\}, \epsilon)$
$fut(f, \bot)$

$\rightarrow$ (Invk-Passive)

$ob(\alpha, o, \{o_2 \mapsto [ogB \mapsto o_1, aa \mapsto 3, ac \mapsto \bot, ag \mapsto \bot]\}, \{a \mapsto o_2, b \mapsto o_1, z \mapsto f, this \mapsto o \mid \texttt{skip}\}, \epsilon)$
$ob(\beta, o_1, \{o_1 \mapsto [ba \mapsto 2], o_2 \mapsto [ogB \mapsto o_1, aa \mapsto 3, ac \mapsto \bot, ag \mapsto \bot]\}, \{ad \mapsto 3, ae \mapsto \bot, this \mapsto o_2 \mid ac = ad + aa\ ;\ s_1''\}, \{bb \mapsto 1, ogA \mapsto o_2, bc \mapsto 3, bd \mapsto \bot, this \mapsto o_1, destiny \mapsto f \mid bd = \bullet\ ;\ s_2'\})$
$fut(f, \bot)$

$\rightarrow$ (Assign-Field)

$ob(\alpha, o, \{o_2 \mapsto [ogB \mapsto o_1, aa \mapsto 3, ac \mapsto \bot, ag \mapsto \bot]\}, \{a \mapsto o_2, b \mapsto o_1, z \mapsto f, this \mapsto o \mid \texttt{skip}\}, \epsilon)$
$ob(\beta, o_1, \{o_1 \mapsto [ba \mapsto 2], o_2 \mapsto [ogB \mapsto o_1, aa \mapsto 3, ac \mapsto 6, ag \mapsto \bot]\}, \{ad \mapsto 3, ae \mapsto \bot, this \mapsto o_2 \mid ae = ogB.\mathtt{m}etB1(ac, this)\ ;\ s_2''\}, \{bb \mapsto 1, ogA \mapsto o_2, bc \mapsto 3, bd \mapsto \bot, this \mapsto o_1, destiny \mapsto f \mid bd = \bullet\ ;\ s_2'\})$

$fut(f, \bot)$

$\rightarrow$ (INVK-PASSIVE) Note: Calling a method of an active object that needs Invk-Passive rule

$ob(\alpha, o, \{o_2 \mapsto [ogB \mapsto o_1, aa \mapsto 3, ac \mapsto \bot, ag \mapsto \bot]\}, \{a \mapsto o_2, b \mapsto o_1, z \mapsto f, this \mapsto o \mid \texttt{skip}\}, \epsilon)$
$ob(\beta, o_1, \{o_1 \mapsto [ba \mapsto 2], o_2 \mapsto [ogB \mapsto o_1, aa \mapsto 3, ac \mapsto 6, ag \mapsto \bot]\}, \{bb \mapsto 6, ogA \mapsto o_2, bc \mapsto \bot, bd \mapsto \bot, this \mapsto o_1 \mid bc = ba + bb \ ; \ s_1'''\}, \{ad \mapsto 3, ae \mapsto \bot, this \mapsto o_2 \mid ae = \bullet \ ; \ s_2''\} :: \{bb \mapsto 1, ogA \mapsto o_2, bc \mapsto 3, bd \mapsto \bot, this \mapsto o_1, destiny \mapsto f \mid bd = \bullet \ ; \ s_2'\})$
$fut(f, \bot)$

ecc...

Chapter 3

Type Checking

3.1 Type Rule

$\mathbb{r}$	::=	$\_ \mid X \mid [act{:}\alpha, \overline{x}{:}\overline{\mathbb{r}}] \mid \alpha \rightsquigarrow \mathbb{r}$	future record
$\mathbb{b}$	::=	$\mathtt{0} \mid (\alpha, \alpha') \mid \mathtt{C.m}\ (\mathbb{r}, \overline{\mathbb{s}}) \to \mathbb{r}' \mid [\alpha = \alpha']\mathbb{b} \mid [\alpha \neq \alpha']\mathbb{b}$	contract
	$\mid$	$\mathbb{b} \fatsemi \mathbb{b} \mid \mathbb{b} \& \mathbb{b}$	
$\mathbb{x}$	::=	$\mathbb{r} \mid f$	extended future record
$\mathbb{z}$	::=	$(\mathbb{r}, \mathbb{b}) \mid (\mathbb{r}, \mathtt{0})^{\checkmark}$	future reference values

Figure 3.1: Syntax of future records and contracts.

The dedlock detection framework presented in this work relies on abstract description, called behavioral types that are estracted from programs by an inference system. The syntax of these descriptions, which is defined in Figure 3.1, use *record names X, Y, Z* $\cdots$. Future records $\mathbb{r}$, which encode the values of expressions in behavioral types, may be one of the following:

- a dummy value $\_$ that models primitive types;
- a record name X that represents a place-holder for a value and can be instantiated by substitutions;
- $[act{:}\alpha, \overline{x}{:}\overline{\mathbb{r}}]$ that defines an object with its activity name α and the values for fields and parameters of the object;
- and $\alpha \rightsquigarrow \mathbb{r}$, which specifies that accessing $\mathbb{r}$ requires control of the activity α (and that the control is to be relesed once the method has been evaluated). The future record $\alpha \rightsquigarrow \mathbb{r}$ is associated with method invocatio: α is the activity of the object on which the method is invoked. The name α in $[act{:}\alpha, \overline{x}{:}\overline{\mathbb{r}}]$ and $\alpha \rightsquigarrow \mathbb{r}$ will be called *root* of the future record.

Behavioral types $\mathbb{b}$ collect the method invocations and the activity dependencies inside statements. The Behavioral types can be one of the following possibilities:

- $\mathtt{0}$ that rapresents the empty behavior;
- (α, α') represents the dependency introduced by a synchronization operation;

– $\mathtt{C.m}\ (\mathbb{r}, \overline{\mathbb{s}}) \to \mathbb{r}'$ specifies that a method $\mathtt{m}$ of class $\mathtt{C}$ is going to be invoked on an object $\mathbb{r}$, with argument $\overline{\mathbb{r}}$, and an object $\mathbb{r}'$ will be returned;

– $[\alpha = \alpha']\mathbb{b}$ amd $[\alpha \neq \alpha']\mathbb{b}$ indicate that the behavioral type $\mathbb{b}$ dipends on a constraint upon the activity of the caller and the callee. The behavioral type $\mathbb{b}$ is valid if and only if the activity α, that is the activity of the coller is respectively the same or different from α', that is the activity of the callee.

The composite behaviors $\mathbb{b} \fatsemi \mathbb{b}'$ defines the abstract behavior of sequential compositions, while $\mathbb{b} \& \mathbb{b}'$ defines the parallelism among behaviors.

We need to define two additional syntactic categories: $\mathbb{x}$ of future record values and $\mathbb{z}$ of typing values. The former one extends future records with *future references f*, which are used to carry out the *alias analysis*. In particular, every local variable of methods and every object field and parameters of future type is associated to a future reference. Assignments between these terms, such as x = y, amounts to copying future references instead of the corresponding values (x and y become aliases). The category $\mathbb{z}$ collects the typing values of future references, which are either $(\mathbb{r}, \mathbb{b})$ or $(\mathbb{r}, \mathtt{0})^{\checkmark}$.

The abstract behavior of methods is defined by *method behavioral type* $([act{:}\alpha, \overline{x{:}\mathbb{r}}], \overline{\mathbb{s}})\{\mathbb{b}\}\mathbb{r}'$ where $[act{:}\alpha, \overline{x{:}\mathbb{r}}]$ is future record of the receiver of the method in which α and $\mathbb{r}$ are respectively its activity and the future record of its field, $\mathbb{s}$ are the future records of the arguments, $\mathbb{b}$ is the abstract behavior of the body, and $\mathbb{r}'$ is the future record of the returned object.

The following auxiliary operators are used:

– *fields*($\mathtt{C}$) and *param*($\mathtt{C}$) return the sequence of fields and their types of a class $\mathtt{C}$.

– *types*(e) returns the type of an expression e, which is either an interface (when e is an object) or a data type;

– *class*(I) returns the unique class implementing I; and

– *mname*($\overline{M}$) returns the sequence of method names in the sequence $\overline{M}$ of method declarations.

– $\Gamma[x \mapsto v]$ is the function such that $(\Gamma[x \mapsto v])(x) = v$ and $(\Gamma[x \mapsto v])(y) = \Gamma(y)$, when $y \neq x$.

– $unsync(\Gamma) \stackrel{def}{=} \mathbb{b}_1 \& \cdots \& \mathbb{b}_n$ where $\{\mathbb{b}_1, \cdots, \mathbb{b}_n\} = \{\mathbb{b}' \mid \text{there are } f, \mathbb{r} : \Gamma(f) = (\mathbb{r}, \mathbb{b}')\}$

The judgments of the inference algorithm have a typing context Γ mapping variables to extended future records, future names to future name values and methods to their signatures. They have the following form:

- $\Gamma \vdash e : \mathbb{x}$ for pure expressions e and $\Gamma \vdash f : \mathbb{z}$ for future names f, where $\mathbb{x}$ and $\mathbb{z}$ are their inferred values.
- $\Gamma \vdash z : \mathbb{r}, \mathbb{b} \rhd \Gamma'$ for expressions with side effects z, where $\mathbb{x}$ are their inferred values, e, $\mathbb{b}$ is the behavioral type for z created by the inference rules, and Γ' is the environment Γ *with updates* of variables and future names. We use the same judgment for pure expressions; in this case $\mathbb{b} = \mathtt{0}$, and $\Gamma' = \Gamma$.
- for statements s: $\Gamma \vdash s : \mathbb{b} \rhd \Gamma'$ where $\mathbb{b}$ is the behavioral type for z created by the inference rules, and Γ' is the environment obtained after the execution of the statement. The environment may change because of variable updates.

We discuss below in more detail the proposed typing rules:

Expressions and addresses

$$\text{(T-Var)}\ \frac{\Gamma(x) = \mathbb{x}}{\Gamma \vdash x : \mathbb{x}} \qquad \text{(T-Fut)}\ \frac{\Gamma(f) = \mathbb{z}}{\Gamma \vdash f : \mathbb{z}} \qquad \text{(T-Field)}\ \frac{x \notin \mathrm{dom}(\Gamma) \qquad \Gamma(\mathtt{this}.x) = \mathbb{r}}{\Gamma \vdash x : \mathbb{r}}$$

$$\text{(T-Val)}\ \frac{e \quad \textit{primitive value}}{\Gamma \vdash e : \_} \qquad \text{(T-Pure)}\ \frac{\Gamma \vdash e : \mathbb{r}}{\Gamma \vdash e : \mathbb{r}, \mathtt{0} \rhd \Gamma}$$

Figure 3.2: Typing rules for expressions and addresses.

The typing rules for expressions and addresses are reported in Figure 3.2. They are straightforward, therefore we skip their discussion, except for the rule (T-Pure) lifts the judgment of a pure expression to a judgment similar to those for expressions with side-effects. This expedient allows us to simplify rules for statements.

(T-Value) not only performs the dereference of variables and return the future record stored in the future name of the variable, but also represents the synchronization point. In fact, as previously said, in the proposed language, the synchronizations are implicit and are postponed

Expressions with side effects

(T-Value)
$$\frac{\begin{array}{c}\Gamma \vdash x : f \\ \Gamma \vdash f : (\alpha' \rightsquigarrow \mathtt{r}, \mathbb{b}) \\ \Gamma \vdash \mathtt{this} : [act{:}\alpha, \overline{X{:}\mathbb{s}}] \\ \Gamma' = \Gamma[f \mapsto (\alpha' \rightsquigarrow \mathtt{r}, \mathtt{0})^{\checkmark}]\end{array}}{\Gamma \vdash x : \mathtt{r}, [\alpha \neq \alpha']\big(\mathbb{b}\&(\alpha, \alpha')\big) \rhd \Gamma'}$$

(T-Value-Tick)
$$\frac{\Gamma \vdash x : f \quad \Gamma \vdash f : (\alpha' \rightsquigarrow \mathtt{r}, \mathtt{0})^{\checkmark}}{\Gamma \vdash x : \mathtt{r}, \mathtt{0} \rhd \Gamma}$$

(T-Exp)
$$\frac{\Gamma \vdash e : \mathtt{r}, \mathbb{b} \rhd \Gamma' \quad \Gamma' \vdash e' : \mathtt{r}, \mathbb{b}' \rhd \Gamma''}{\Gamma \vdash e \oplus e' : \mathtt{r}, \mathbb{b}\&\mathbb{b}' \rhd \Gamma''}$$

(T-NewActive)
$$\frac{\mathit{fields}(\mathtt{C}) = \overline{T\ x} \qquad \alpha \text{ fresh} \quad \big(\Gamma_i \vdash e_i : \mathtt{r}_i, \mathbb{b}_i \rhd \Gamma_{i+1}\big)^{i=1\ldots n}}{\Gamma_1 \vdash \mathtt{newActive}\ \ \mathtt{C}(e_1 \ldots, e_n) : [act{:}\alpha, x_1{:}\mathtt{r}_1, \ldots, x_n{:}\mathtt{r}_n], \mathbb{b}_1\&\ldots\&\mathbb{b}_n \rhd \Gamma_{n+1}}$$

(T-New)
$$\frac{\mathit{fields}(\mathtt{C}) = \overline{T\ x} \quad \Gamma_1 \vdash \mathtt{this} : [act{:}\alpha, \overline{z{:}\mathbb{s}}] \quad \big(\Gamma_i \vdash e_i : \mathtt{r}_i, \mathbb{b}_i \rhd \Gamma_{i+1}\big)^{i=1\ldots n}}{\Gamma_1 \vdash \mathtt{new}\ \mathtt{C}(e_1 \ldots, e_n) : [act{:}\alpha, x_1{:}\mathtt{r}_1, \ldots, x_n{:}\mathtt{r}_n], \mathbb{b}_1\&\ldots\&\mathbb{b}_n \rhd \Gamma_{n+1}}$$

(T-Invk)
$$\frac{\begin{array}{c}\Gamma_0 \vdash \mathtt{this} : [act{:}\alpha', \overline{z{:}\mathbb{s}}] \\ \Gamma_0 \vdash e : [act{:}\alpha'', \overline{x{:}\mathtt{r}}], \mathbb{b}_0 \rhd \Gamma_1 \qquad \Gamma_0 \vdash \mathtt{C.m} : ([act{:}\alpha'', \overline{x{:}\mathtt{r}}], \overline{\mathbb{s}}) \to \mathtt{r}' \\ \big(\Gamma_i \vdash e_i : \mathtt{r}_i, \mathbb{b}_i \rhd \Gamma_{i+1}\big)^{i=1\ldots n} \qquad \mathit{class}(\mathit{types}(e)) = \mathtt{C} \qquad \mathit{fields}(\mathtt{C}) = \overline{T\ x} \qquad f \text{ fresh} \\ \Gamma' = \Gamma_{n+1}\Big[f \mapsto \Big(\alpha'' \rightsquigarrow \mathtt{r}',\ [\alpha \neq \alpha']\big(\mathtt{C.m}\ ([act{:}\alpha'', \overline{x{:}\mathtt{r}}], \overline{\mathbb{s}}) \to \mathtt{r}'\big)\Big)\Big]\end{array}}{\Gamma_0 \vdash e.\mathtt{m}(e_1 \ldots, e_n) : f, \mathbb{b}_0\&\ldots\&\mathbb{b}_n \fatsemi [\alpha' = \alpha'']\Big(\big(\mathtt{C.m}\ ([act{:}\alpha'', \overline{x{:}\mathtt{r}}], \overline{\mathbb{s}}) \to \mathtt{r}'\big)\&\mathit{unsync}\big(\Gamma_{n+1}\big)\Big) \rhd \Gamma'}$$

Figure 3.3: Typing rules for expressions with side effects

until that are not strictly necessary. The synchronization returns the behavioral type $[\alpha \neq \alpha'](\mathbb{b}\&(\alpha, \alpha'))$, where $\mathbb{b}$ is stored in the future name of x, (α, α') represents a dependency between the activity of the object executing the expression and the root of the expression and the constraint $[\alpha \neq \alpha']$ specifies that the behavior type is valid only if the method that return the required value is an asynchronous method. The behavior type $\mathbb{b}$ may have two shapes: either (i) $\mathbb{b} = \mathtt{C.m}\ (\mathtt{r}, \overline{\mathbb{s}}) \to \mathtt{r}'$ or (ii) $\mathbb{b} = \mathtt{0}$. We also observe that the rule updates the environment by check-marking the value of the future name of x and by replacing the contract with $\mathtt{0}$ (because the synchronisation has been already performed). This allows subsequent operations on the same future name not to modify the contract – see (T-Value-Tick).

(T-Exp) derives behavioral type for an arithmetic or boolean expression. This behavioral type consist of the behavioral types of each sub-expression related by the operator $\&$.

(T-NewActive) returns a record with a new activity name. In this rule the behavior type $\mathbb{b}_1 \& \ldots \& \mathbb{b}_n + 1$ and the environment Γ_{n+1} derived from synchronization of each parameters.

(T-New) in this rule in contrast with (T-NewActive), the activity of the returned record is the same of the object executing the expression.

(T-Invk) derives behavior type for a method invocations. In this rule the behavior type is composed of $\mathbb{b}_0 \& \ldots \mathbb{b}_n$ which represents the behavior type obtained by the synchronization of the *this* and all parameters of the method, in sequence with the behavioral type of the method invocation $[\alpha' = \alpha''](\texttt{C.m}\ ([act{:}\alpha'', \overline{x{:}\mathbb{r}}], \overline{\mathbb{s}}) \to \mathbb{r}')\&unsync(\Gamma_{n+1}))$. The subterm $[\alpha' = \alpha'']$ is a constraint which tells us that the behavioral type is valid only if the activity of the coller is the same of the callee. The subterm $unsync(\Gamma)$ lets us collect all the contracts in Γ that are stored in future names that are not check-marked.

Statements

(T-Field-Record)
$$\frac{x \notin \mathrm{dom}(\Gamma) \qquad \Gamma(\texttt{this}.x) = \mathbb{r} \qquad \Gamma \vdash z : \mathbb{r}}{\Gamma \vdash x = z : \mathbb{b} \ \rhd\ \Gamma'}$$

(T-Var-Record)
$$\frac{\Gamma \vdash z : \mathbb{r}}{\Gamma \vdash x = z : \texttt{0} \ \rhd\ \Gamma[x \mapsto \mathbb{r}]}$$

(T-Var-Future)
$$\frac{\Gamma \vdash z : f}{\Gamma \vdash x = z : \texttt{0} \ \rhd\ \Gamma[x \mapsto f]}$$

(T-Var-Expression)
$$\frac{\Gamma \vdash e : \mathbb{x}, \mathbb{b} \ \rhd\ \Gamma' \qquad e \text{ is not a variable}}{\Gamma \vdash x = e : \mathbb{b} \ \rhd\ \Gamma'[x \mapsto \mathbb{x}]}$$

(T-Seq)
$$\frac{\Gamma \vdash s_1 : \mathbb{b}_1 \ \rhd\ \Gamma_1 \qquad \Gamma_1 \vdash s_2 : \mathbb{b}_2 \ \rhd\ \Gamma_2}{\Gamma \vdash s_1; s_2 : \mathbb{b}_1 \fatsemi \mathbb{b}_2 \ \rhd\ \Gamma_2}$$

(T-Return)
$$\frac{\Gamma \vdash e : \mathbb{r} \qquad \Gamma(\texttt{destiny}) = \mathbb{r}}{\Gamma \vdash \texttt{return}\ e : \texttt{0} \ \rhd\ \Gamma}$$

(T-If)
$$\frac{\begin{array}{c}\Gamma \vdash e : \texttt{Bool} \qquad \Gamma \vdash s_1 : \mathbb{b}_1 \ \rhd\ \Gamma_1 \qquad \Gamma \vdash s_2 : \mathbb{b}_2 \ \rhd\ \Gamma_2 \\ (\bigwedge_{x \in \mathrm{dom}(\Gamma)} \Gamma_1(x) = \Gamma_2(x)) \wedge (\bigwedge_{x \in \texttt{Fut}(\Gamma)} \Gamma_1(\Gamma_1(x)) = \Gamma_2(\Gamma_2(x))) \\ \Gamma' = \Gamma_1 + \Gamma_2|_{\{f \mid f \notin \Gamma_2(\texttt{Fut}(\Gamma))\}}\end{array}}{\Gamma \vdash \texttt{if}\ e\ \{\, s_1 \,\}\ \texttt{else}\ \{\, s_2 \,\} : \mathbb{b}_1 \& \mathbb{b}_2 \ \rhd\ \Gamma'}$$

Figure 3.4: Typing rules for statements

The inference rules for statements are collected in Figure 3.4. The first four rules define the inference of contracts for assignment. There are two types of assignments: those updating fields and parameters of the *this* object and the other ones. For every type, we need to address the cases of updates

with values (rules (T-Field-Record) and (T-Var-Record)), with expressions (with side effects) (rule (T-Var-Expression)), or future names (rule (T-Var-Future)). Rules for fields and parameters updates enforce that their future records are unchanging.

(T-Var-Future) define the management of aliases: future variables are always updated with future names and never with future names values.

(T-Seq) defines the sequential composition of contracts.

(T-Return) constrains the record of destiny, which is an identifier introduced by (T-Method) , shown in Figure 3.5, for storing the return record.

Method,class and program

(T-Method)

$$\frac{\mathit{fields}(\mathtt{C}) = \overline{T_f\ x} \qquad \Gamma(\mathtt{C.m}) = ([\mathit{act}{:}\alpha', \overline{x{:}\mathbb{r}}], \overline{\mathbb{s}})\{\mathbb{b}\&\mathit{unsync}(\Gamma')\}\mathbb{r}' \qquad \Gamma + \mathtt{this} : [\mathit{act}{:}\alpha',\ \overline{x{:}\mathbb{r}}] + \overline{y : \mathbb{s}} + \mathtt{destiny} : \mathbb{r}' \vdash s : \mathbb{b} \rhd \Gamma'}{\mathtt{C}, \Gamma \vdash \mathtt{T\ m}\ (\overline{T\ y})\{\overline{T_l\ w};\ s\}\ :\ ([\mathit{act}{:}\alpha', \overline{x{:}\mathbb{r}}], \overline{\mathbb{s}})\{\mathbb{b}\&\mathit{unsync}(\Gamma')\}\mathbb{r}'}$$

(T-Class)

$$\frac{\mathtt{C}, \Gamma \vdash \overline{M} : \overline{\mathbb{B}}}{\Gamma \vdash \mathtt{class\ C}(\overline{T\ x})\ \{\overline{M}\}\ :\ \overline{\mathtt{C}.\mathit{mname}(M) \mapsto \mathbb{B}}}$$

(T-Program)

$$\frac{\Gamma \vdash \overline{C} : \overline{S} \qquad \Gamma + \mathtt{this} : [\mathit{act}{:}\ \mathit{start}] \vdash s : \mathbb{b} \rhd \Gamma'}{\Gamma \vdash \overline{C}\ \{\overline{T\ x}\ \mathtt{;}\ s\} : \mathbb{b}\&\mathit{unsync}(\Gamma')}$$

Figure 3.5: Typing rules for method,class and program

The rules for method and class declarations are defined in Figure 3.5.

(T-Method) derives the behavioral type of $\mathtt{T\ m}\ (\overline{T\ y})\{\overline{T_l\ w};\ s\}$ by typing s in an environment extended with `this`, `destiny` (that will be set by `return` statements, see (T-Return)). As discussed above, the abstract behavior of the method body is a behavioral type, which is $\mathbb{b}\&\mathit{unsync}(\Gamma')$ for (T-Method). This term $\mathit{unsync}(\Gamma')$ collects all the contracts in Γ' *that are stored in future names that are not checkmarked*. In fact, these contracts correspond to asynchronous invocations without any synchronization in the body. These invocations will be evaluated *after* the termination of the body – they are the *unsynchronised behavioral type*.

(T-Class) yields an *abstract class table* that associates a behavioral type of method with every method name. It is this abstract class table that is used by our analyzers.

(T-Program) derives the contract of a classAsp program by typing the main function in the same way as it was a body of a method.

The behavioral type class tables of the classes in a program derived by the rule (T-Class), will be noted bct. We will address the behavioral type of `m` of class `C` by bct(`C.m`).

3.2 Examples

3.2.1 Example 1

```
1   class A() {
2
3     int metA1(B b) {
4       int aa;
5       int ab;
6       aa = b.metB1();
7       ab = aa + 1;
8       return ab;
9     }
10
11    int get(){ return 2;}
12  }
13
14  class B(A a) {
15
16    int metB1() {
17      int z;
18      int g;
19      z = a.get();
20      g = z + 1;
21      return g;
22    }
23  }
24
25  //MAIN
26  {
27    A a;
28    B b;
29    int x;
30    a = newActive A();
31    b = newActive B(a);
32    x = a.metA1(b);
33  }
```

MAIN

$$\text{(T-Program)}\ \frac{\text{(T-Seq)}\ \dfrac{\Delta \qquad \Delta' \qquad \Delta''}{\Gamma + this : [act : start] \vdash \text{a = newActive A(); b = newActive B(a); x = a.metA1(b)} : \mathbb{b}_1 \fatsemi \mathbb{b}_2 \fatsemi \mathbb{b}_3 \ \triangleright\ \Gamma'}}{\Gamma + this : [act : start] \vdash \text{a = newActive A(); b = newActive B(a); x = a.metA1(b)} : \mathbb{b} \ \triangleright\ \Gamma'}}{\Gamma \vdash \overline{C}\ \{\overline{T\ x}\ ;\ s\} : \overline{S},\ \mathbb{b} \& unsync(\Gamma')}$$

Δ

$$\text{(T-Var-Record)}\ \frac{\text{(T-NewActive)}\ \dfrac{\alpha_1\ fresh}{\Gamma + this : [act : start] \vdash \text{newActive A()} : [act : \alpha_1], \mathtt{0} \ \triangleright\ \Gamma_1}}{\Gamma + this : [act : start] \vdash \text{a = newActive A()} : \mathbb{b}_1 \ \triangleright\ \Gamma_1[\text{a} \mapsto [act : \alpha_1]]}$$

Δ'

$$\text{(T-Var-Record)}\ \frac{\text{(T-NewActive)}\ \dfrac{\Gamma_1 \vdash a : [act : \alpha_1] \qquad \alpha_2\ fresh}{\Gamma_1 \vdash \text{newActive B(a)} : \big[act : \alpha_2, a : [act : \alpha_1]\big], \mathtt{0} \ \triangleright\ \Gamma_2}}{\Gamma_1 \vdash \text{b = newActive B(a)} : \mathbb{b}_2 \ \triangleright\ \Gamma_2\Big[\text{b} \mapsto \big[act : \alpha_2, a : [act : \alpha_1]\big]\Big]}$$

Δ''

$$\Gamma_2 \vdash a : [act : \alpha_1] \quad \Gamma_2 \vdash this : [act : start] \quad \Gamma_2 \vdash b : \big[act : \alpha_2, a : [act : \alpha_1]\big] \quad f \ fresh$$
$$\Gamma_2 \vdash \texttt{A.metA1:}\ \big([act{:}\alpha_1], b : \big[act : \alpha_2, a : [act : \alpha_1]\big]\big) \to \_$$
$$\Gamma' = \Gamma_2 + \Big\{ f \mapsto \Big(\alpha_1 \rightsquigarrow \_\ ,\ \texttt{A.metA1}\big([act{:}\alpha_1]\ ,\ b : \big[act : \alpha_2, a : [act : \alpha_1]\big]\big) \to \_\Big) \Big\}$$

(T-Invk)
$$\Gamma_2 \vdash \text{a.metA1(b)} : f,\ [start = \alpha_1]\Big(\Big(\texttt{A.metA1}\big([act{:}\alpha_1], b : \big[act : \alpha_2, a : [act : \alpha_1]\big]\big) \to \_\Big) \& unsync(\Gamma_2)\Big) \ \triangleright\ \Gamma'$$

(T-Var-Future)
$$\frac{\Gamma_2 \vdash \text{a.metA1(b)} : f, \mathbb{b}_3 \ \triangleright\ \Gamma'}{\Gamma_2 \vdash\ \text{x= a.metA1(b)} : \mathbb{b}_3 \ \triangleright\ \Gamma'[x \mapsto f]}$$

where:

$$\Gamma_1 = \Gamma + \Big\{ a \mapsto [act : \alpha_1] \Big\}$$

$$\Gamma_2 = \Gamma_1 + \Big\{ b \mapsto \big[act : \alpha_2, a : [act : \alpha_1]\big] \Big\}$$

$$\Gamma' = \Gamma_2 + \Big\{ x \mapsto f\ ,\ f \mapsto \Big(\alpha_1 \rightsquigarrow \_\ ,\ \texttt{A.metA1}\big([act{:}\alpha_1]\ ,\ b : \big[act : \alpha_2, a : [act : \alpha_1]\big]\big) \to \_\Big) \Big\}$$

$$\mathbb{b} = \texttt{0}\ \texttt{;}\ \texttt{0}\ \texttt{;}\ [start = \alpha_1]\Big(\Big(\texttt{A.metA1}\big([act{:}\alpha_1], b : \big[act : \alpha_2, a : [act : \alpha_1]\big]\big) \to \_\Big) \& unsync(\Gamma_2)\Big) \iff$$

$$\iff\ [start = \alpha_1]\Big(\Big(\texttt{A.metA1}\big([act{:}\alpha_1], b : \big[act : \alpha_2, a : [act : \alpha_1]\big]\big) \to \_\Big))$$

The Behavioural Type for the mail is:

$$\mathbb{b}\ \&\ unsync(\Gamma') \iff \mathbb{b}\ \&\ \Big(\texttt{A.metA1}\big([act{:}\alpha_1]\ ,\ b : \big[act : \alpha_2, a : [act : \alpha_1]\big]\big) \to \_\Big)$$

A.metA1(B b)

$$\dfrac{\dfrac{\dfrac{\Gamma_2 \vdash ab : \_ \quad \Gamma_2(\texttt{destiny}) = \_}{\Gamma_2 \vdash \text{return ab} : \texttt{0} \rhd \Gamma'}\ \text{(T-Seq)}}{\Gamma_2 \vdash \text{return ab} : \mathbb{b}_3 \rhd \Gamma'}}{\Gamma + \texttt{this} : [act{:}\alpha_1] + b : \big[act : \alpha_2, a : [act : \alpha_3]\big] + \texttt{destiny} : \_ \vdash \text{aa} = \text{b.metB1(); ab} = \text{aa} + 1\text{; return ab} : \mathbb{b}_1 \fatsemi \mathbb{b}_2 \fatsemi \mathbb{b}_3 \rhd \Gamma'}$$

$$\dfrac{\Gamma + \texttt{this} : [act{:}\alpha_1] + b : \big[act : \alpha_2, a : [act : \alpha_3]\big] + \texttt{destiny} : \_ \vdash \text{aa} = \text{b.metB1(); ab} = \text{aa} + 1\text{; return ab} : \mathbb{b} \rhd \Gamma'}{\text{A,}\Gamma \vdash \text{int metA1(B b) \{int aa; int bb; s;\}} : \big([act : \alpha_1], \big[act : \alpha_2, a : [act : \alpha_3]\big]\big)\{\mathbb{b} \& unsync(\Gamma')\}\_}\ \text{(T-Method)}$$

where:

s = { aa = b.metB1(); ab = aa + 1; return ab; }

Δ

$$\dfrac{\dfrac{\Gamma \vdash b : \big[act : \alpha_2, a : [act : \alpha_3]\big] \quad \Gamma \vdash this : [act : \alpha_1] \quad f\ fresh \quad \Gamma \vdash \texttt{B}.\text{metB1: } \big(\big[act : \alpha_2, a : [act : \alpha_3]\big]\big) \to \_ \quad \Gamma_1 = \Gamma + \Big\{f \mapsto \big(\alpha_2 \rightsquigarrow \_\ ,\ \texttt{B}.\text{metB1}\big(\big[act : \alpha_2, a : [act : \alpha_3]\big]\big) \to \_\big)\Big\}}{\Gamma + \texttt{this} : [act{:}\alpha_1] + b : \big[act : \alpha_2, a : [act : \alpha_3]\big] + \texttt{destiny} : \_ \vdash \text{b.metB1()} : f,\ [\alpha_1 = \alpha_2]\ \big(\texttt{B}.\text{metB1}\big(\big[act : \alpha_2, a : [act : \alpha_3]\big]\big) \to \_\big) \rhd \Gamma_1}\ \text{(T-Invk)}}{\Gamma + \texttt{this} : [act{:}\alpha_1] + b : \big[act : \alpha_2, a : [act : \alpha_3]\big] + \texttt{destiny} : \_ \vdash \text{b.metB1()} : f, \mathbb{b}_1 \rhd \Gamma_1}\ \text{(T-Var-Future)}$$

$$\Gamma + \texttt{this} : [act{:}\alpha_1] + b : \big[act : \alpha_2, a : [act : \alpha_3]\big] + \texttt{destiny} : \_ \vdash \text{aa} = \text{b.metB1()} : \mathbb{b}_1 \rhd \Gamma_1[\text{aa} \mapsto f]$$

Δ'

$$\dfrac{\dfrac{\Gamma_1 \vdash f : \Big(\alpha_2 \rightsquigarrow \_\ ,\ \texttt{B.metB1}\big(\big[act : \alpha_2, a : [act : \alpha_3]\big]\big) \to \_\Big) \quad \Gamma_1 \vdash this : [act : \alpha_1] \quad \Gamma_1 \vdash \text{aa} : f \quad \Gamma_2 = \Gamma_1\Big[f \mapsto (\alpha_2 \rightsquigarrow \_\ ,\ \texttt{0})^{\checkmark}\Big]}{\Gamma_1 \vdash \text{aa} : \_,\ [\alpha_1 \neq \alpha_2]\ \Big(\Big(\texttt{B.metB1}\big(\big[act : \alpha_2, a : [act : \alpha_3]\big]\big) \to \_\Big)\&(\alpha_1, \alpha_2)\Big)\ \rhd\ \Gamma_2}\ \text{(T-Value)} \quad \dfrac{\dfrac{\Gamma_2 \vdash 1 : \_}{\Gamma_2 \vdash 1 : \_,\ \texttt{0}\ \rhd\ \Gamma_2}\ \text{(T-Pure)}}{\Gamma_2 \vdash 1 : \mathbb{r},\ \mathbb{b}''_2\ \rhd\ \Gamma_2}}{\dfrac{\Gamma_1 \vdash \text{aa} : \_,\ \mathbb{b}'_2\ \rhd\ \Gamma_2 \qquad \Gamma_2 \vdash 1 : \mathbb{r},\ \mathbb{b}''_2\ \rhd\ \Gamma_2}{\Gamma_1 \vdash \text{aa} + 1 : \_,\ \mathbb{b}'_2\ \&\ \mathbb{b}''_2\ \rhd\ \Gamma_2}\ \text{(T-Exp)}}\ \text{(T-Var-Record)}$$
$$\Gamma_1 \vdash \text{ab} = \text{aa} + 1 : \mathbb{b}_2\ \rhd\ \Gamma_2[\ \text{ab} \mapsto \_]$$

where:

$$\Gamma_1 = \Gamma + \Big\{\ \text{aa} \mapsto f\ ,\ f \mapsto \Big(\alpha_2 \rightsquigarrow \_\ ,\ \texttt{B.metB1}\big(\big[act : \alpha_2, a : [act : \alpha_3]\big]\big) \to \_\Big)\Big\}$$

$$\Gamma_2 = \Gamma_1\big[f \mapsto (\alpha_2 \rightsquigarrow \_\ , \texttt{0})^{\checkmark}\big] + \{ab \mapsto \_\}$$

$$\Gamma_2 = \Gamma'$$

$$\mathbb{b} = \ \mathbb{b}' \ \texttt{;}\ \mathbb{b}'' \ \texttt{;}\ \texttt{0}$$

$$\mathbb{b}' = [\alpha_1 = \alpha_2]\ \Big(\texttt{B.metB1}\big(\big[act : \alpha_2, a : [act : \alpha_3]\big]\big) \to \_\Big)$$

$$\mathbb{b}'' = [\alpha_1 \neq \alpha_2]\ \Big(\Big(\texttt{B.metB1}\big(\big[act : \alpha_2, a : [act : \alpha_3]\big]\big) \to \_\Big)\&(\alpha_1, \alpha_2)\Big)$$

A.get()

$$\dfrac{\dfrac{\Gamma_2 \vdash 2 : \_ \quad \Gamma(\texttt{destiny}) = \_}{\Gamma + \texttt{this} : [act : \alpha_1] + \texttt{destiny} : \_ \vdash \text{return g} : \texttt{0}\ \rhd\ \Gamma}}{\Gamma + \texttt{this} : [act : \alpha_1] + \texttt{destiny} : \_ \vdash \text{return g} : \mathbb{b}\ \rhd\ \Gamma}$$
$$\text{A,}\Gamma \vdash \text{int get() \{return 2\}}\ :\ ([act : \alpha_1])\{\mathbb{b}\&\mathit{unsync}(\Gamma)\}\_$$

B.metB1()

$$\dfrac{\dfrac{\dfrac{\Gamma_2 \vdash \text{g} : \_ \quad \Gamma(\texttt{destiny}) = \_}{\Gamma_2 \vdash \text{return g} : \texttt{0} \rhd \Gamma'}\ \text{(T-Seq)}}{\Gamma_2 \vdash \text{return g} : \mathbb{b}_3 \rhd \Gamma'} \qquad \Delta \qquad \Delta'}{\Gamma + \texttt{this} : \big[act : \alpha_2, a : [act : \alpha_3]\big] + \texttt{destiny} : \_ \vdash \text{z} = \text{a.get(); g} = \text{z} + 1\text{; return g} : \mathbb{b}_1 \fatsemi \mathbb{b}_2 \fatsemi \mathbb{b}_3 \rhd \Gamma'}$$

$$\dfrac{\Gamma + \texttt{this} : \big[act : \alpha_2, a : [act : \alpha_1]\big] + \texttt{destiny} : \_ \vdash \text{z} = \text{a.get(); g} = \text{z} + 1\text{; return g} : \mathbb{b} \rhd \Gamma'}{\text{B,}\Gamma \vdash \text{int metB1() \{int z; int g; z = a.get(); g = z + 1; return g\}} : \big([act : \alpha_2, a : [act : \alpha_1]]\big)\{\mathbb{b} \& \mathit{unsync}(\Gamma)\}\_}\ \text{(T-Method)}$$

where:

$\text{s} = \{\ \text{z} = \text{a.get(); g} = \text{z} + 1\text{; return g;}\ \}$

Δ

$$\dfrac{\dfrac{\begin{array}{c}\Gamma \vdash a : [act : \alpha_1] \quad \Gamma \vdash \texttt{A}.\text{get:}\,\big([act : \alpha_2]\big) \to \_ \\ \Gamma \vdash this : \big[act : \alpha_2, a : [act : \alpha_1]\big] \quad f\ fresh \\ \Gamma_1 = \Gamma + \Big\{ f \mapsto \big(\alpha_1 \rightsquigarrow \_\ ,\ \texttt{A}.\text{get}\big([act : \alpha_1]\big) \to \_\big)\Big\}\end{array}}{\begin{array}{c}\Gamma + \texttt{this} : \big[act : \alpha_2, a : [act : \alpha_1]\big] + \texttt{destiny} : \_ \vdash \\ \text{a.get()} : f,\ \ [\alpha_2 = \alpha_1]\ \big(\texttt{A}.\text{get}\big([act : \alpha_1]\big) \to \_\big) \rhd \Gamma_1\end{array}}\ \text{(T-Invk)}}{\Gamma + \texttt{this} : \big[act : \alpha_2, a : [act : \alpha_1]\big] + \texttt{destiny} : \_ \vdash \text{a.get()} : f, \mathbb{b}_1 \rhd \Gamma_1}\ \text{(T-Var-Future)}$$

$$\Gamma + \texttt{this} : \big[act : \alpha_2, a : [act : \alpha_1]\big] + \texttt{destiny} : \_ \vdash \text{z} = \text{a.get()} : \mathbb{b}_1 \rhd \Gamma_1[\text{z} \mapsto f]$$

Δ'

$$\dfrac{\dfrac{\dfrac{\Gamma_1 \vdash f : \Big(\alpha_1 \rightsquigarrow \_\ ,\ \mathtt{A}.\mathrm{get}\big([act : \alpha_1]\big) \to \_\Big)}{\Gamma_1 \vdash this : \big[act : \alpha_2, a : [act : \alpha_1]\big] \quad \Gamma_1 \vdash \mathrm{z} : f \qquad \Gamma_2 = \Gamma_1\Big[f \mapsto (\alpha_1 \rightsquigarrow \_\ ,\ \mathtt{0})^{\checkmark}\Big]}\ \text{(T-Value)} \qquad \dfrac{\Gamma_2 \vdash 1 : \_}{\Gamma_2 \vdash 1 : \_,\ \mathtt{0} \rhd \Gamma_2}\ \text{(T-Pure)}}{\Gamma_1 \vdash \mathrm{z} : \_,\ [\alpha_2 \neq \alpha_1]\ \Big(\big(\mathtt{A}.\mathrm{get}\big([act : \alpha_1]\big) \to \_\big)\&(\alpha_2, \alpha_1)\Big) \rhd \Gamma_2} \qquad \dfrac{}{\Gamma_2 \vdash 1 : \mathbb{r},\ \mathbb{b}''_2 \rhd \Gamma_2}}{\Gamma_1 \vdash \mathrm{z} : \_,\ \mathbb{b}'_2 \rhd \Gamma_2}\ \text{(T-Exp)} \qquad \dfrac{}{\Gamma_1 \vdash \mathrm{z} + 1 : \_,\ \mathbb{b}'_2 \,\&\, \mathbb{b}''_2 \rhd \Gamma_2}$$

$$\text{(T-Var-Record)}\ \dfrac{\Gamma_1 \vdash \mathrm{z} + 1 : \_,\ \mathbb{b}'_2 \,\&\, \mathbb{b}''_2 \rhd \Gamma_2}{\Gamma_1 \vdash \mathrm{g} = \mathrm{z} + 1 : \mathbb{b}_2 \rhd \Gamma_2[\ \mathrm{g} \mapsto \_]}$$

where:

$$\Gamma_1 = \Gamma + \Big\{\ \mathrm{z} \mapsto f\ ,\ f \mapsto \Big(\alpha_1 \rightsquigarrow \_\ ,\ \mathtt{A}.\mathrm{get}\big([act : \alpha_3]\big) \to \_\Big)\Big\}$$

$$\Gamma_2 = \Gamma_1\big[f \mapsto (\alpha_1 \rightsquigarrow \_\ , \mathtt{0})^{\checkmark}\big] + \{\ \mathrm{g} \mapsto \_\}$$

$$\Gamma_2 = \Gamma'$$

$$\mathbb{b} = \ [\alpha_2 = \alpha_1]\ \Big(\mathtt{A}.\mathrm{get}\big([act : \alpha_1]\big) \to \_\Big)\ \mathbin{⨟}\ [\alpha_2 \neq \alpha_1]\ \Big(\big(\mathtt{A}.\mathrm{get}\big([act : \alpha_1]\big) \to \_\big)\&(\alpha_2, \alpha_1)\Big)\ \mathbin{⨟}\ \mathtt{0}$$

3.2.2 Example 2

```
1   class A() {
2
3     int metA1(B b) {
4       int aa;
5       int ab;
6       aa = b.metB1();
7       ab = aa + 1;
8       return ab;
9     }
10
11    int get(){ return 2;}
12  }
13
14  class B(A a) {
15
16    int metB1() {
17      int z;
18      int g;
19      z = a.get();
20      g = z + 1;
21      return g;
22    }
23  }
24
25  //MAIN
26  {
27    A a;
28    A am;
29    B b;
30    int x;
31    a = newActive A();
32    am = newActive A();
33    b = newActive B(am);
34    x = a.metA1(b);
35  }
```

MAIN

$$\textsc{(T-Program)}\ \frac{\textsc{(T-Seq)}\ \dfrac{\Delta \qquad \Delta' \qquad \Delta'' \qquad \Delta'''}{\Gamma + this : [act : start] \vdash \mathrm{s} : \mathbb{b}_1 \fatsemi \mathbb{b}_2 \fatsemi \mathbb{b}_3 \fatsemi \mathbb{b}_4 \ \rhd\ \Gamma'}}{\Gamma + this : [act : start] \vdash \mathrm{s} : \mathbb{b} \ \rhd\ \Gamma'}$$

where: $s = \{\ \mathrm{a} = \text{newActive A(); am} = \text{newActive A(); b} = \text{newActive B(am); x} = \text{a.metA1(b)}\ \}$

Δ

$$\textsc{(T-Var-Record)}\ \frac{\textsc{(T-NewActive)}\ \dfrac{\alpha_1\ fresh}{\Gamma + this : [act : start] \vdash \text{newActive A()} : [act : \alpha_1], \mathtt{0} \ \rhd\ \Gamma_1}}{\Gamma + this : [act : start] \vdash \mathrm{a} = \text{newActive A()} : \mathbb{b}_1 \ \rhd\ \Gamma_1[\mathrm{a} \mapsto [act : \alpha_1]]}$$

Δ'

$$\textsc{(T-Var-Record)}\ \frac{\textsc{(T-NewActive)}\ \dfrac{\alpha_2\ fresh}{\Gamma_1 \vdash \text{newActive A()} : [act : \alpha_2], \mathtt{0} \ \rhd\ \Gamma_2}}{\Gamma_1 \vdash \mathrm{am} = \text{newActive A()} : \mathbb{b}_2 \ \rhd\ \Gamma_2[\mathrm{am} \mapsto [act : \alpha_2]]}$$

Δ''

$$\textsc{(T-Var-Record)}\ \frac{\textsc{(T-NewActive)}\ \dfrac{\Gamma_2 \vdash am : [act : \alpha_2] \qquad \alpha_3\ fresh}{\Gamma_2 \vdash \text{newActive B(am)} : \big[act : \alpha_3, am : [act : \alpha_2]\big], \mathtt{0} \ \rhd\ \Gamma_3}}{\Gamma_2 \vdash \mathrm{b} = \text{newActive B(am)} : \mathbb{b}_3 \ \rhd\ \Gamma_3\Big[\mathrm{b} \mapsto \big[act : \alpha_3, am : [act : \alpha_2]\big]\Big]}$$

Δ'''

$$\Gamma_3 \vdash a : [act : \alpha_1] \quad \Gamma_3 \vdash this : [act : start] \quad \Gamma_3 \vdash b : \big[act : \alpha_3, am : [act : \alpha_2]\big] \quad f\ fresh$$
$$\Gamma_3 \vdash \texttt{A}.\text{metA1: } \big([act{:}\alpha_1], b : \big[act : \alpha_3, am : [act : \alpha_2]\big]\big) \to \_$$
$$\Gamma' = \Gamma_3 + \Big\{ f \mapsto \Big(\alpha_1 \rightsquigarrow \_\ ,\ \texttt{A}.\text{metA1}\big([act{:}\alpha_1]\ ,\ b : \big[act : \alpha_3, am : [act : \alpha_2]\big]\big) \to \_\Big) \Big\}$$

(T-Invk)
$$\Gamma_3 \vdash \text{a.metA1(b)} : f,\ [start = \alpha_1]\Big(\Big(\texttt{A}.\text{metA1}\big([act{:}\alpha_1], b : \big[act : \alpha_3, am : [act : \alpha_2]\big]\big) \to \_\Big)\&unsync(\Gamma_3)\Big)\ \triangleright\ \Gamma'$$

(T-Var-Future)
$$\Gamma_3 \vdash \text{a.metA1(b)} : f, \mathbb{b}_4\ \triangleright\ \Gamma'$$
$$\Gamma_3 \vdash\ \text{x} = \text{a.metA1(b)} : \mathbb{b}_4\ \triangleright\ \Gamma'[x \mapsto f]$$

where:

$$\Gamma_1 = \Gamma + \Big\{ a \mapsto [act : \alpha_1] \Big\}$$

$$\Gamma_2 = \Gamma_1 + \Big\{ am \mapsto [act : \alpha_2] \Big\}$$

$$\Gamma_3 = \Gamma_2 + \Big\{ b \mapsto \big[act : \alpha_3, am : [act : \alpha_2]\big] \Big\}$$

$$\Gamma' = \Gamma_3 + \Big\{ x \mapsto f\ ,\ f \mapsto \Big(\alpha_1 \rightsquigarrow \_\ ,\ \texttt{A}.\text{metA1}\big(\alpha, [act{:}\alpha_1]\ ,\ b : \big[act : \alpha_3, am : [act : \alpha_2]\big]\big) \to \_\Big) \Big\}$$

$$\mathbb{b} = \texttt{0} \fatsemi \texttt{0} \fatsemi \texttt{0} \fatsemi [start = \alpha_1]\Big(\Big(\texttt{A}.\text{metA1}\big([act{:}\alpha_1], b : \big[act : \alpha_3, am : [act : \alpha_2]\big]\big) \to \_\Big)\&unsync(\Gamma_3)\Big)$$

A.metA1(B b)

$$\dfrac{\dfrac{\dfrac{\Gamma_2 \vdash ab : \_ \quad \Gamma_2(\texttt{destiny}) = \_}{\Gamma_2 \vdash \text{return ab} : \texttt{0} \rhd \Gamma'}\ \text{(T-Seq)}}{\Gamma_2 \vdash \text{return ab} : \mathbb{b}_3 \rhd \Gamma'}}{\dfrac{\dfrac{\Gamma + \texttt{this} : [act{:}\alpha_1] + b : \big[act : \alpha_2, a : [act : \alpha_3]\big] + \texttt{destiny} : \_ \vdash \text{aa} = \text{b.metB1(); ab} = \text{aa} + 1\text{; return ab} : \mathbb{b}_1 \fatsemi \mathbb{b}_2 \fatsemi \mathbb{b}_3 \rhd \Gamma'}{\Gamma + \texttt{this} : [act{:}\alpha_1] + b : \big[act : \alpha_2, a : [act : \alpha_3]\big] + \texttt{destiny} : \_ \vdash \text{aa} = \text{b.metB1(); ab} = \text{aa} + 1\text{; return ab} : \mathbb{b} \rhd \Gamma'}\ \text{(T-Method)}}{\text{A,}\Gamma \vdash \text{int metA1(B b) \{int aa; int bb; s;\}} : \big([act : \alpha_1], \big[act : \alpha_2, a : [act : \alpha_3]\big]\big)\big\{\mathbb{b}\&unsync(\Gamma)\big\}\_}$$

where:

s = { aa = b.metB1(); ab = aa + 1; return ab; }

Δ

$$\dfrac{\dfrac{\Gamma \vdash b : \big[act : \alpha_2, a : [act : \alpha_3]\big] \quad \Gamma \vdash this : [act : \alpha_1] \quad f\ fresh \quad \Gamma \vdash \text{B.metB1: } \big(\big[act : \alpha_2, a : [act : \alpha_3]\big]\big) \to \_ \quad \Gamma_1 = \Gamma + \Big\{f \mapsto \big(\alpha_2 \rightsquigarrow \_\ ,\ \text{B.metB1}\big(\big[act : \alpha_2, a : [act : \alpha_3]\big]\big) \to \_\big)\Big\}}{\Gamma + \texttt{this} : [act{:}\alpha_1] + b : \big[act : \alpha_2, a : [act : \alpha_3]\big] + \texttt{destiny} : \_ \vdash \text{b.metB1()} : f,\ [\alpha_1 = \alpha_2]\ \Big(\text{B.metB1}\big(\big[act : \alpha_2, a : [act : \alpha_3]\big]\big) \to \_\Big)\ \rhd \Gamma_1}\ \text{(T-Invk)}}{\Gamma + \texttt{this} : [act{:}\alpha_1] + b : \big[act : \alpha_2, a : [act : \alpha_3]\big] + \texttt{destiny} : \_ \vdash \text{b.metB1()} : f, \mathbb{b}_1 \rhd \Gamma_1}\ \text{(T-Var-Future)}$$

$$\Gamma + \texttt{this} : [act{:}\alpha_1] + b : \big[act : \alpha_2, a : [act : \alpha_3]\big] + \texttt{destiny} : \_ \vdash \text{aa} = \text{b.metB1()} : \mathbb{b}_1 \rhd \Gamma_1[\text{aa} \mapsto f]$$

Δ'

$$\dfrac{\dfrac{\Gamma_1 \vdash f : \Big(\alpha_2 \rightsquigarrow \_\ ,\ \texttt{B.metB1}\big(\big[act : \alpha_2, a : [act : \alpha_3]\big]\big) \to \_\Big) \quad \Gamma_1 \vdash this : [act : \alpha_1] \quad \Gamma_1 \vdash \text{aa} : f \quad \Gamma_2 = \Gamma_1\Big[f \mapsto (\alpha_2 \rightsquigarrow \_\ ,\ \texttt{0})^{\checkmark}\Big]}{\Gamma_1 \vdash \text{aa} : \_,\ [\alpha_1 \neq \alpha_2]\ \Big(\Big(\texttt{B.metB1}\big(\big[act : \alpha_2, a : [act : \alpha_3]\big]\big) \to \_\Big)\&(\alpha_1, \alpha_2)\Big)\ \rhd\ \Gamma_2}\ \text{(T-Value)} \quad \dfrac{\Gamma_2 \vdash 1 : \_}{\Gamma_2 \vdash 1 : \_,\ \texttt{0}\ \rhd\ \Gamma_2}\ \text{(T-Pure)}}{\Gamma_1 \vdash \text{aa} : \_,\ \mathbb{b}_2'\ \rhd\ \Gamma_2} \quad \dfrac{}{\Gamma_2 \vdash 1 : \mathbb{r},\ \mathbb{b}_2''\ \rhd\ \Gamma_2}$$

$$\text{(T-Exp)}\ \dfrac{\Gamma_1 \vdash \text{aa} : \_,\ \mathbb{b}_2'\ \rhd\ \Gamma_2 \qquad \Gamma_2 \vdash 1 : \mathbb{r},\ \mathbb{b}_2''\ \rhd\ \Gamma_2}{\Gamma_1 \vdash \text{aa} + 1 : \_,\ \mathbb{b}_2' \,\&\, \mathbb{b}_2''\ \rhd\ \Gamma_2}$$

$$\text{(T-Var-Record)}\ \dfrac{\Gamma_1 \vdash \text{aa} + 1 : \_,\ \mathbb{b}_2' \,\&\, \mathbb{b}_2''\ \rhd\ \Gamma_2}{\Gamma_1 \vdash \text{ab} = \text{aa} + 1 : \mathbb{b}_2\ \rhd\ \Gamma_2[\ \text{ab} \mapsto \_]}$$

where:

$$\Gamma_1 = \Gamma + \Big\{\ \text{aa} \mapsto f\ ,\ f \mapsto \Big(\alpha_2 \rightsquigarrow \_\ ,\ \texttt{B.metB1}\big(\big[act : \alpha_2, a : [act : \alpha_3]\big]\big) \to \_\Big)\Big\}$$

$$\Gamma_2 = \Gamma_1\big[f \mapsto (\alpha_2 \rightsquigarrow \_\ , \texttt{0})^{\checkmark}\big] + \{ab \mapsto \_\}$$

$$\Gamma_2 = \Gamma'$$

$$\mathbb{b} = \ \mathbb{b}' \ \texttt{;}\ \mathbb{b}'' \ \texttt{;}\ \texttt{0}$$

$$\mathbb{b}' = [\alpha_1 = \alpha_2]\ \Big(\texttt{B.metB1}\big(\big[act : \alpha_2, a : [act : \alpha_3]\big]\big) \to \_\Big)$$

$$\mathbb{b}'' = [\alpha_1 \neq \alpha_2]\ \Big(\Big(\texttt{B.metB1}\big(\big[act : \alpha_2, a : [act : \alpha_3]\big]\big) \to \_\Big)\&(\alpha_1, \alpha_2)\Big)$$

A.get()

$$\dfrac{\dfrac{\Gamma_2 \vdash 2 : \_ \quad \Gamma(\texttt{destiny}) = \_}{\Gamma + \texttt{this} : [act : \alpha_1] + \texttt{destiny} : \_ \vdash \text{return g} : \texttt{0}\ \rhd\ \Gamma}}{\Gamma + \texttt{this} : [act : \alpha_1] + \texttt{destiny} : \_ \vdash \text{return g} : \mathbb{b}\ \rhd\ \Gamma}$$

$$\text{A,}\Gamma \vdash \text{int get() \{return 2\}}\ :\ ([act : \alpha_1])\{\mathbb{b}\&\mathit{unsync}(\Gamma)\}\_$$

B.metB1()

$$\dfrac{\Delta \qquad \Delta' \qquad \dfrac{\dfrac{\Gamma_2 \vdash \mathrm{g} : \_ \quad \Gamma(\texttt{destiny}) = \_}{\Gamma_2 \vdash \text{return g} : \mathtt{0} \ \triangleright\ \Gamma'}\ (\text{T-Seq})}{\Gamma_2 \vdash \text{return g} : \mathbb{b}_3 \ \triangleright\ \Gamma'}}{\dfrac{\Gamma + \texttt{this} : \big[act : \alpha_2, a : [act : \alpha_3]\big] + \texttt{destiny} : \_ \vdash \mathrm{z} = \text{a.get(); g} = \mathrm{z} + 1\text{; return g} : \mathbb{b}_1 \fatsemi \mathbb{b}_2 \fatsemi \mathbb{b}_3 \ \triangleright\ \Gamma'}{\dfrac{\Gamma + \texttt{this} : \big[act : \alpha_2, a : [act : \alpha_1]\big] + \texttt{destiny} : \_ \vdash \mathrm{z} = \text{a.get(); g} = \mathrm{z} + 1\text{; return g} : \mathbb{b} \ \triangleright\ \Gamma'}{\mathrm{B}, \Gamma \vdash \text{int metB1() \{int z; int g; z = a.get(); g = z + 1; return g\}} \ :\ \big(\big[act : \alpha_2, a : [act : \alpha_1]\big]\big)\big\{\mathbb{b} \& unsync(\Gamma)\big\}\_}\ (\text{T-Method})}}$$

where:

$\quad$ s = { z = a.get(); g = z + 1; return g; }

Δ

$$\dfrac{\dfrac{\begin{array}{c}\Gamma \vdash a : [act : \alpha_1] \quad \Gamma \vdash \mathtt{A}\text{.get: } \big([act : \alpha_2]\big) \to \_ \\ \Gamma \vdash this : \big[act : \alpha_2, a : [act : \alpha_1]\big] \quad f\ fresh \\ \Gamma_1 = \Gamma + \Big\{ f \mapsto \Big(\alpha_1 \rightsquigarrow \_\ ,\ \mathtt{A}\text{.get}\big([act : \alpha_1]\big) \to \_ \Big) \Big\}\end{array}}{\begin{array}{c}\Gamma + \texttt{this} : \big[act : \alpha_2, a : [act : \alpha_1]\big] + \texttt{destiny} : \_ \vdash \\ \text{a.get()} :\ f,\ \ [\alpha_2 = \alpha_1]\ \Big(\mathtt{A}\text{.get}\big([act : \alpha_1]\big) \to \_\Big)\ \triangleright \Gamma_1\end{array}}\ (\text{T-Invk})}{\dfrac{\Gamma + \texttt{this} : \big[act : \alpha_2, a : [act : \alpha_1]\big] + \texttt{destiny} : \_ \vdash \text{a.get()} : f, \mathbb{b}_1 \ \triangleright\ \Gamma_1}{\Gamma + \texttt{this} : \big[act : \alpha_2, a : [act : \alpha_1]\big] + \texttt{destiny} : \_ \vdash \mathrm{z} = \text{a.get()} : \mathbb{b}_1 \ \triangleright\ \Gamma_1[\mathrm{z} \mapsto f]}\ (\text{T-Var-Future})}$$

Δ'

$$\dfrac{\dfrac{\dfrac{\Gamma_1 \vdash f : \Big(\alpha_1 \rightsquigarrow \_\ ,\ \mathtt{A.get}\big([act : \alpha_1]\big) \to \_\Big) \qquad \Gamma_1 \vdash this : \big[act : \alpha_2, a : [act : \alpha_1]\big] \quad \Gamma_1 \vdash \mathrm{z} : f \quad \Gamma_2 = \Gamma_1\big[f \mapsto (\alpha_1 \rightsquigarrow \_\ ,\ 0)^{\checkmark}\big]}{\Gamma_1 \vdash \mathrm{z} : \_,\ [\alpha_2 \neq \alpha_1]\ \Big(\big(\mathtt{A.get}\big([act : \alpha_1]\big) \to \_\big)\&(\alpha_2, \alpha_1)\Big)\ \rhd\ \Gamma_2}\ \text{(T-Value)}}{\Gamma_1 \vdash \mathrm{z} : \_,\ \mathbb{b}'_2\ \rhd\ \Gamma_2}\ \text{(T-Exp)} \qquad \dfrac{\dfrac{\Gamma_2 \vdash 1 : \_}{\Gamma_2 \vdash 1 : \_,\ 0\ \rhd\ \Gamma_2}\ \text{(T-Pure)}}{\Gamma_2 \vdash 1 : \mathbb{r},\ \mathbb{b}''_2\ \rhd\ \Gamma_2}}{\Gamma_1 \vdash \mathrm{aa} + 1 : \_,\ \mathbb{b}'_2\ \&\ \mathbb{b}''_2\ \rhd\ \Gamma_2}\ \text{(T-Var-Record)}$$

$$\Gamma_1 \vdash \mathrm{g} = \mathrm{z} + 1 : \mathbb{b}_2\ \rhd\ \Gamma_2[\ \mathrm{ab} \mapsto \_]$$

where:

$$\Gamma_1 = \Gamma + \Big\{\ \mathrm{z} \mapsto f\ ,\ f \mapsto \Big(\alpha_1 \rightsquigarrow \_\ ,\ \mathtt{A.get}\big([act : \alpha_3]\big) \to \_\Big)\Big\}$$

$$\Gamma_2 = \Gamma_1\big[f \mapsto (\alpha_1 \rightsquigarrow \_\ , 0)^{\checkmark}\big] + \{\ \mathrm{g} \mapsto \_\}$$

$$\Gamma_2 = \Gamma'$$

$$\mathbb{b} = \ [\alpha_2 = \alpha_1]\ \Big(\mathtt{A.get}\big([act : \alpha_1]\big) \to \_\Big)\ \mathbin{⨟}\ [\alpha_2 \neq \alpha_1]\ \Big(\big(\mathtt{A.get}\big([act : \alpha_1]\big) \to \_\big)\&(\alpha_2, \alpha_1)\Big)\ \mathbin{⨟}\ 0$$

3.2.3 Example 3

```
1   class A(){
2     int metA1() {
3       B b;
4       C c;
5       int x;
6       b = newActive B();
7       c = newActive C(b);
8       x = b.metB1(c);
9       return x;
10    }
11  }
12
13  class B() {
14    int metB1(C c) {
15      int ba;
16      int bb;
17      ba = c.metC1();
18      bb = ba + 1;
19      return bb;
20    }
21
22    int get(){ return 2;}
23  }
24
25  class C(B b) {
26    int metC1() {
27      int z;
28      int g;
29      z = b.get();
30      g = z + 1;
31      return g;
32    }
33  }
34
35  //MAIN
36  {
37    A a;
38    B ogB;
39    int k;
40    int y;
41    int m;
42    a = newActive A();
43    k = a.metA1();
44    ogB = newActive B();
45    y = ogB.get();
46    m = k + 1 * y;
47  }
```

MAIN

$$\text{(T-Seq)}\ \frac{\Delta \qquad \Delta_1 \qquad \Delta_2 \qquad \Delta_3 \qquad \Delta_4}{\Gamma + this : [act : start] \vdash \text{s} : \mathbb{b}_1 \fatsemi \mathbb{b}_2 \fatsemi \mathbb{b}_3 \fatsemi \mathbb{b}_4 \fatsemi \mathbb{b}_5 \ \triangleright\ \Gamma'}$$

$$\text{(T-Program)}\ \frac{\Gamma + this : [act : start] \vdash \text{s} : \mathbb{b} \ \triangleright\ \Gamma'}{\Gamma + this : [act : start] \vdash \text{s} : \mathbb{b} \ \triangleright\ \Gamma'}$$

where:
s = { a = newActive A(); k = a.metA1(); ogB = newActive B(); y = ogB.get(); m = k + 1 * y; }

Δ

$$\text{(T-NewActive)}\ \frac{\alpha_1\ fresh}{\Gamma + this : [act : start] \vdash \text{newActive A}() : [act : \alpha_1], \mathtt{0} \ \triangleright\ \Gamma_1}$$

$$\text{(T-Var-Record)}\ \frac{\Gamma + this : [act : start] \vdash \text{newActive A}() : [act : \alpha_1], \mathtt{0} \ \triangleright\ \Gamma_1}{\Gamma + this : [act : start] \vdash \text{a} = \text{newActive A}() : \mathbb{b}_1 \ \triangleright\ \Gamma_1[\text{a} \mapsto [act : \alpha_1]]}$$

Δ_1

$$\text{(T-Invk)}\ \frac{\begin{array}{c}\Gamma_1 \vdash a : [act : \alpha_1] \quad \Gamma_1 \vdash this : [act : start] \quad f\ fresh \\ \Gamma_1 \vdash \mathtt{A}.\text{metA1: } \big([act{:}\alpha_1]\big) \to \_ \\ \Gamma_2 = \Gamma_1 + \Big\{ f \mapsto \Big(\alpha_1 \rightsquigarrow \_\ ,\ \mathtt{A}.\text{metA1}\big([act{:}\alpha_1]\big) \to \_ \Big) \Big\}\end{array}}{\Gamma_1 \vdash \text{a.metA1}() : f,\ [start = \alpha_1]\Big(\Big(\mathtt{A}.\text{metA1}\big([act{:}\alpha_1]\big) \to \_ \Big) \& unsync(\Gamma_1) \Big) \ \triangleright\ \Gamma_2}$$

$$\text{(T-Var-Future)}\ \frac{\Gamma_1 \vdash \text{a.metA1}() : f, \mathbb{b}_2 \ \triangleright\ \Gamma_2}{\Gamma_1 \vdash \ \text{k= a.metA1}() : \mathbb{b}_2 \ \triangleright\ \Gamma_2[k \mapsto f]}$$

Δ_2

$$\text{(T-NewActive)}\ \frac{\alpha_2\ fresh}{\Gamma_2 \vdash \text{newActive B}() : [act : \alpha_2], \mathtt{0} \ \triangleright\ \Gamma_3}$$

$$\text{(T-Var-Record)}\ \frac{\Gamma_2 \vdash \text{newActive B}() : [act : \alpha_2], \mathtt{0} \ \triangleright\ \Gamma_3}{\Gamma_2 \vdash \text{ogB} = \text{newActive B}() : \mathbb{b}_3 \ \triangleright\ \Gamma_3\Big[\text{ogB} \mapsto [act : \alpha_2]\Big]}$$

Δ_3

$$\dfrac{\dfrac{\Gamma_3 \vdash ogB : [act : \alpha_2] \quad \Gamma_3 \vdash this : [act : start] \quad f\ fresh \quad \Gamma_3 \vdash \texttt{B.get:}\ \big([act{:}\alpha_2]\big) \to \_ \quad \Gamma_4 = \Gamma_3 + \Big\{ f_1 \mapsto \big(\alpha_2 \rightsquigarrow \_\ ,\ \texttt{B.get}\big([act{:}\alpha_2]\big) \to \_\big)\Big\}}{\Gamma_3 \vdash \text{ogB.get}() : f_1,\ [start = \alpha_2]\Big(\big(\texttt{B.get}\big([act{:}\alpha_2]\big) \to \_\big) \& unsync(\Gamma_3)\Big)\ \rhd\ \Gamma_4}\ \text{(T-Invk)}}{\Gamma_3 \vdash \text{ogB.get}() : f_1, \mathbb{b}_4\ \rhd\ \Gamma_4}$$

$$\dfrac{\Gamma_3 \vdash \text{ogB.get}() : f_1, \mathbb{b}_4\ \rhd\ \Gamma_4}{\Gamma_3 \vdash\ \text{y} = \text{ogB.get}() : \mathbb{b}_4\ \rhd\ \Gamma_4[y \mapsto f_1]}\ \text{(T-Var-Future)}$$

Δ_4

$$\dfrac{\dfrac{\dfrac{\dfrac{\Gamma_5 \vdash 1 : \_}{\Gamma_5 \vdash 1 : \_,\ \texttt{0}\ \rhd\ \Gamma_5}\ \text{(T-Pure)}}{\Gamma_5 \vdash 1 : \mathbb{r},\ \mathbb{b}_5''\ \rhd\ \Gamma_5}}{\Gamma_4 \vdash \text{k} + 1\ \text{*}\ \text{y} : \_,\ \mathbb{b}_5' \ \&\ \mathbb{b}_5'' \ \&\ \mathbb{b}_5'''\ \rhd\ \Gamma'}\ \text{(T-Var-Record)}}{\Gamma_4 \vdash \text{m} = \text{k} + 1\ \text{*}\ \text{y} : \mathbb{b}_5\ \rhd\ \Gamma'[\ \text{m} \mapsto \_]}$$

Σ (T-Exp) Σ'

Σ

$$\dfrac{\dfrac{\Gamma_4 \vdash f : \big(\alpha_1 \rightsquigarrow \_\ ,\ \texttt{A.metA1}\big([act : \alpha_1]\big) \to \_\big) \quad \Gamma_4 \vdash this : [act : start] \quad \Gamma_4 \vdash \text{k} : f \quad \Gamma_5 = \Gamma_4\Big[f \mapsto (\alpha_1 \rightsquigarrow \_\ ,\ \texttt{0})^{\checkmark}\Big]}{\Gamma_4 \vdash \text{k} : \_,\ [start \neq \alpha_1]\ \Big(\big(\texttt{A.metA1}\big([act : \alpha_1]\big) \to \_\big) \& (start, \alpha_1)\Big)\ \rhd\ \Gamma_5}\ \text{(T-Value)}}{\Gamma_4 \vdash \text{k} : \_,\ \mathbb{b}_5'\ \rhd\ \Gamma_5}$$

Σ'

$$\frac{\begin{array}{c}\Gamma_5 \vdash f_1 : \Big(\alpha_2 \rightsquigarrow \_\ ,\ \mathtt{B}.\mathrm{get}\big([act : \alpha_2]\big) \to \_\Big) \\ \Gamma_5 \vdash this : [act : start] \quad \Gamma_5 \vdash \mathrm{y} : f_1 \qquad \Gamma' = \Gamma_5\Big[f_1 \mapsto (\alpha_2 \rightsquigarrow \_\ ,\ \mathtt{0})^{\checkmark}\Big]\end{array}}{\Gamma_5 \vdash \mathrm{y} : \_,\ [start \neq \alpha_2]\ \Big(\Big(\mathtt{B}.\mathrm{get}\big([act : \alpha_2]\big) \to \_\Big)\&(start, \alpha_2)\Big)\ \rhd\ \Gamma'}\ \text{(T-Value)}$$

$$\frac{}{\Gamma_5 \vdash \mathrm{y} : \_,\ \mathbb{b}_5'''\ \rhd\ \Gamma'}$$

where:

$$\Gamma_1 = \Gamma + \Big\{\mathrm{a} \mapsto [act : \alpha_1]\Big\}$$

$$\Gamma_2 = \Gamma_1 + \Big\{k \mapsto f\ , f \mapsto \Big(\alpha_1 \rightsquigarrow \_\ ,\ \mathtt{A}.\mathrm{metA1}\big([act{:}\alpha_1]\big) \to \_\Big)\Big\}$$

$$\Gamma_3 = \Gamma_2 + \Big\{\mathrm{ogB} \mapsto [act : \alpha_2]\Big\}$$

$$\Gamma_4 = \Gamma_3 + \Big\{y \mapsto f_1\ , f_1 \mapsto \Big(\alpha_2 \rightsquigarrow \_\ ,\ \mathtt{B}.\mathrm{get}\big([act{:}\alpha_2]\big) \to \_\Big)\Big\}$$

$$\Gamma_5 = \Gamma_4\Big[f \mapsto (\alpha_1 \rightsquigarrow \_\ ,\ \mathtt{0})^{\checkmark}\Big]$$

$$\Gamma' = \Gamma_5\Big[f_1 \mapsto (\alpha_2 \rightsquigarrow \_\ ,\ \mathtt{0})^{\checkmark}\Big]$$

$$\mathbb{b} = \mathtt{0}\ \fatsemi\ \mathbb{b}_2\ \fatsemi\ \mathtt{0}\ \fatsemi\ \mathbb{b}_4\ \fatsemi\ (\mathbb{b}_5'\ \&\ \mathtt{0}\ \&\ \mathbb{b}_5''')$$

$$\mathbb{b}_2 = [start = \alpha_1]\Big(\Big(\mathtt{A}.\mathrm{metA1}\big([act{:}\alpha_1]\big) \to \_\Big)\&unsync(\Gamma_1)\Big)$$

$$\mathbb{b}_4 = [start = \alpha_2]\Big(\Big(\mathtt{B}.\mathrm{get}\big([act{:}\alpha_2]\big) \to \_\Big)\&unsync(\Gamma_3)\Big)$$

$$\mathbb{b}_5' = [start \neq \alpha_1]\ \Big(\Big(\mathtt{A}.\mathrm{metA1}\big([act : \alpha_1]\big) \to \_\Big)\&(start, \alpha_1)\Big)$$
$$\mathbb{b}_5' = [start \neq \alpha_2]\ \Big(\Big(\mathtt{B}.\mathrm{get}\big([act : \alpha_2]\big) \to \_\Big)\&(start, \alpha_2)\Big)$$

A.metA1()

$$\text{(T-Method)}\ \frac{\text{(T-Seq)}\ \dfrac{\Delta \qquad \Delta' \qquad \Delta'' \qquad \Delta'''}{\Gamma + this : [act : \alpha_1] + \texttt{destiny} : \_ \vdash \text{s} : \mathbb{b}_1 \fatsemi \mathbb{b}_2 \fatsemi \mathbb{b}_3 \fatsemi \mathbb{b}_4 \ \rhd\ \Gamma'}}{\Gamma + this : [act : \alpha_1] + \texttt{destiny} : \_ \vdash \text{s} : \mathbb{b} \ \rhd\ \Gamma'}}{\text{A,}\Gamma \vdash \text{int metA1() } \{\text{B b; C c; int x; s;}\}\big([act : \alpha_1]\big)\big\{\mathbb{b} \& unsync(\Gamma)\big\}\_}$$

where: s = { b = newActive B(); c = newActive C(b); x = b.metB1(c); return x; }

Δ

$$\text{(T-Var-Record)}\ \frac{\text{(T-NewActive)}\ \dfrac{\alpha_2\ fresh}{\Gamma + this : [act : \alpha_1] + \texttt{destiny} : \_ \vdash \text{newActive B()} : [act : \alpha_2], \texttt{0} \ \rhd\ \Gamma_1}}{\Gamma + this : [act : \alpha_1] + \texttt{destiny} : \_ \vdash \text{b} = \text{newActive B()} : \mathbb{b}_1 \ \rhd\ \Gamma_1[\text{b} \mapsto [act : \alpha_2]]}$$

Δ'

$$\text{(T-Var-Record)}\ \frac{\text{(T-NewActive)}\ \dfrac{\Gamma_1 \vdash b : [act : \alpha_2] \qquad \alpha_3\ fresh}{\Gamma_1 \vdash \text{newActive C(b)} : \big[act : \alpha_3, b : [act : \alpha_2]\big], \texttt{0} \ \rhd\ \Gamma_2}}{\Gamma_1 \vdash \text{c} = \text{newActive C(b)} : \mathbb{b}_2 \ \rhd\ \Gamma_2\Big[\text{c} \mapsto \big[act : \alpha_3, b : [act : \alpha_2]\big]\Big]}$$

Δ''

$$\text{(T-Invk)}\ \frac{\begin{array}{c}\Gamma_2 \vdash b : [act : \alpha_2] \quad \Gamma_2 \vdash this : [act : \alpha_1] \quad \Gamma_2 \vdash c : \big[act : \alpha_3, b : [act : \alpha_2]\big] \quad f\ fresh \\ \Gamma_2 \vdash \texttt{B}.\text{metB1: } \big([act{:}\alpha_2], c : \big[act : \alpha_3, b : [act : \alpha_2]\big]\big) \to \_ \\ \Gamma_3 = \Gamma_2 + \Big\{f \mapsto \Big(\alpha_2 \rightsquigarrow \_\ ,\ \texttt{B}.\text{metB1}\big([act{:}\alpha_2]\ ,\ c : \big[act : \alpha_3, b : [act : \alpha_2]\big]\big) \to \_\Big)\Big\}\end{array}}{\Gamma_2 \vdash \text{b.metB1(c)} : f,\ [\alpha_1 = \alpha_2]\Big(\Big(\texttt{B}.\text{metB1}\big([act{:}\alpha_2], c : \big[act : \alpha_3, b : [act : \alpha_2]\big]\big) \to \_\Big) \& unsync(\Gamma_2)\Big) \ \rhd\ \Gamma_3}$$

$$\text{(T-Var-Future)}\ \frac{\Gamma_2 \vdash \text{b.metB1(c)} : f, \mathbb{b}_3 \ \rhd\ \Gamma_3}{\Gamma_2 \vdash\ \text{x= b.metB1(c)} : \mathbb{b}_3 \ \rhd\ \Gamma_3[x \mapsto f]}$$

Δ'''

$$\dfrac{\dfrac{\Gamma_3 \vdash x : \_ \quad \Gamma_3(\texttt{destiny}) = \_}{\Gamma_3 \vdash \text{return x} : \texttt{0} \ \triangleright \ \Gamma'}}{\Gamma_3 \vdash \text{return x} : \mathbb{b}_4 \ \triangleright \ \Gamma'}$$

where:

$$\Gamma_1 = \Gamma + \Big\{\text{b} \mapsto [act : \alpha_2]\Big\}$$

$$\Gamma_2 = \Gamma_1 + \Big\{\text{c} \mapsto \big[act : \alpha_3, b : [act : \alpha_2]\big]\Big\}$$

$$\Gamma_3 = \Gamma_2 + \Big\{x \mapsto f \ , \ f \mapsto \Big(\alpha_2 \rightsquigarrow \_ \ , \ \texttt{B}.\text{metB1}\big([act{:}\alpha_2] \ , \ c : \big[act : \alpha_3, b : [act : \alpha_2]\big]\big) \to \_\Big)\Big\}$$

$$\mathbb{b} = \texttt{0} \ \texttt{;} \ \texttt{0} \ \texttt{;} \ [\alpha_1 = \alpha_2]\Big(\Big(\texttt{B}.\text{metB1}\big([act{:}\alpha_2], c : \big[act : \alpha_3, b : [act : \alpha_2]\big]\big) \to \_\Big)\&unsync(\Gamma_2)\Big) \ \texttt{;} \ \texttt{0}$$

B.metB1(C c)

$$\dfrac{\dfrac{\dfrac{\Gamma_2 \vdash ab : \_ \quad \Gamma_2(\texttt{destiny}) = \_}{\Gamma_2 \vdash \text{return bb} : \texttt{0} \rhd \Gamma'}\ \text{(T-Seq)}}{\Gamma_2 \vdash \text{return bb} : \mathbb{b}_3 \rhd \Gamma'}}{\dfrac{\dfrac{\Gamma + \texttt{this} : [act{:}\alpha_1] + c : \big[act : \alpha_2, a : [act : \alpha_3]\big] + \texttt{destiny} : \_ \vdash \text{ba} = \text{c.metC1(); bb} = \text{ba} + 1\text{; return bb} : \mathbb{b}_1 \fatsemi \mathbb{b}_2 \fatsemi \mathbb{b}_3 \rhd \Gamma'}{\Gamma + \texttt{this} : [act{:}\alpha_1] + c : \big[act : \alpha_2, b : [act : \alpha_3]\big] + \texttt{destiny} : \_ \vdash \text{ba} = \text{c.metC1(); bb} = \text{ba} + 1\text{; return bb} : \mathbb{b} \rhd \Gamma'}\ \text{(T-Method)}}{\text{A,}\Gamma \vdash \text{int metB1(C c) \{int ba; int bb; s;\}} : \big([act : \alpha_1], \big[act : \alpha_2, b : [act : \alpha_3]\big]\big)\big\{\mathbb{b} \& unsync(\Gamma)\big\}\_}$$

where:

$\text{s} = \{\ \text{ba} = \text{c.metC1(); bb} = \text{ba} + 1\text{; return bb;}\ \}$

Δ

$$\dfrac{\dfrac{\Gamma \vdash c : \big[act : \alpha_2, b : [act : \alpha_3]\big] \quad \Gamma \vdash this : [act : \alpha_1] \quad f\ fresh \quad \Gamma \vdash \text{B.metB1:}\ \big(\big[act : \alpha_2, a : [act : \alpha_3]\big]\big) \to \_ \quad \Gamma_1 = \Gamma + \Big\{f \mapsto \Big(\alpha_2 \rightsquigarrow \_\ ,\ \text{C.metC1}\big(\big[act : \alpha_2, b : [act : \alpha_3]\big]\big) \to \_\Big)\Big\}}{\Gamma + \texttt{this} : [act{:}\alpha_1] + c : \big[act : \alpha_2, b : [act : \alpha_3]\big] + \texttt{destiny} : \_ \vdash \text{b.metB1()} : f,\ [\alpha_1 = \alpha_2]\ \Big(\text{C.metC1}\big(\big[act : \alpha_2, b : [act : \alpha_3]\big]\big) \to \_\Big)\ \rhd \Gamma_1}\ \text{(T-Invk)}}{\Gamma + \texttt{this} : [act{:}\alpha_1] + c : \big[act : \alpha_2, b : [act : \alpha_3]\big] + \texttt{destiny} : \_ \vdash \text{c.metC1()} : f, \mathbb{b}_1 \rhd \Gamma_1}\ \text{(T-Var-Future)}$$

$$\Gamma + \texttt{this} : [act{:}\alpha_1] + c : \big[act : \alpha_2, b : [act : \alpha_3]\big] + \texttt{destiny} : \_ \vdash \text{ba} = \text{c.metC1()} : \mathbb{b}_1 \rhd \Gamma_1[\text{ba} \mapsto f]$$

Δ'

$$\dfrac{\dfrac{\Gamma_1 \vdash f : \Big(\alpha_2 \rightsquigarrow \_\ ,\ \texttt{C.metC1}\big([act : \alpha_2, b : [act : \alpha_3]]\big) \to \_\Big) \quad \Gamma_1 \vdash this : [act : \alpha_1] \quad \Gamma_1 \vdash \texttt{bb} : f \quad \Gamma_2 = \Gamma_1\Big[f \mapsto (\alpha_2 \rightsquigarrow \_\ ,\ \texttt{0})^{\checkmark}\Big]}{\Gamma_1 \vdash \texttt{ba} : \_,\ [\alpha_1 \neq \alpha_2]\ \Big(\big(\texttt{C.metC1}\big([act : \alpha_2, b : [act : \alpha_3]]\big) \to \_\big)\&(\alpha_1, \alpha_2)\Big)\ \rhd\ \Gamma_2}\ \text{(T-Value)} \quad \dfrac{\dfrac{\Gamma_2 \vdash 1 : \_}{\Gamma_2 \vdash 1 : \_,\ \texttt{0}\ \rhd\ \Gamma_2}\ \text{(T-Pure)}}{\Gamma_2 \vdash 1 : \mathbb{r},\ \mathbb{b}''_2\ \rhd\ \Gamma_2}}{\Gamma_1 \vdash \texttt{ba} : \_,\ \mathbb{b}'_2\ \rhd\ \Gamma_2}\ \text{(T-Exp)}$$

$$\dfrac{\Gamma_1 \vdash \texttt{ba} + 1 : \_,\ \mathbb{b}'_2\ \&\ \mathbb{b}''_2\ \rhd\ \Gamma_2}{\Gamma_1 \vdash \texttt{bb} = \texttt{ba} + 1 : \mathbb{b}_2\ \rhd\ \Gamma_2[\ \texttt{bb} \mapsto \_]}\ \text{(T-Var-Record)}$$

where:

$$\Gamma_1 = \Gamma + \Big\{\ \texttt{ba} \mapsto f\ ,\ f \mapsto \Big(\alpha_2 \rightsquigarrow \_\ ,\ \texttt{C.metC1}\big([act : \alpha_2, b : [act : \alpha_3]]\big) \to \_\Big)\Big\}$$

$$\Gamma_2 = \Gamma_1\big[f \mapsto (\alpha_2 \rightsquigarrow \_\ , \texttt{0})^{\checkmark}\big] + \{bb \mapsto \_\}$$

$$\Gamma_2 = \Gamma'$$

$$\mathbb{b} = \ \mathbb{b}'\ \fatsemi\ \mathbb{b}''\ \fatsemi\ \texttt{0}$$

$$\mathbb{b}' = [\alpha_1 = \alpha_2]\ \Big(\texttt{C.metC1}\big([act : \alpha_2, b : [act : \alpha_3]]\big) \to \_\Big)$$

$$\mathbb{b}'' = [\alpha_1 \neq \alpha_2]\ \Big(\big(\texttt{C.metC1}\big([act : \alpha_2, b : [act : \alpha_3]]\big) \to \_\big)\&(\alpha_1, \alpha_2)\Big)$$

B.get()

$$\dfrac{\dfrac{\dfrac{\Gamma_2 \vdash\ 2 : \_ \quad \Gamma(\texttt{destiny}) = \_}{\Gamma + \texttt{this} : [act : \alpha_1] + \texttt{destiny} : \_ \vdash \text{return } 2 : \texttt{0}\ \rhd\ \Gamma}}{\Gamma + \texttt{this} : [act : \alpha_1] + \texttt{destiny} : \_ \vdash \text{return } 2 : \mathbb{b}\ \rhd\ \Gamma}}{\text{A,}\Gamma \vdash \text{int get() } \{\text{return } 2\}\ :\ ([act : \alpha_1])\{\mathbb{b}\&unsync(\Gamma)\}\_}$$

C.metC1()

$$\dfrac{\dfrac{\dfrac{\Gamma_2 \vdash \text{g} : \_ \quad \Gamma(\texttt{destiny}) = \_}{\Gamma_2 \vdash \text{return g} : \texttt{0} \rhd \Gamma'}\ \text{(T-Seq)}}{\Gamma_2 \vdash \text{return g} : \mathbb{b}_3 \rhd \Gamma'} \qquad \Delta \qquad \Delta'}{\Gamma + \texttt{this} : \big[act : \alpha_2, b : [act : \alpha_3]\big] + \texttt{destiny} : \_ \vdash \text{z} = \text{a.get(); g} = \text{z} + 1\text{; return g} : \mathbb{b}_1 \fatsemi \mathbb{b}_2 \fatsemi \mathbb{b}_3 \rhd \Gamma'}$$

$$\dfrac{\Gamma + \texttt{this} : \big[act : \alpha_2, b : [act : \alpha_1]\big] + \texttt{destiny} : \_ \vdash \text{z} = \text{a.get(); g} = \text{z} + 1\text{; return g} : \mathbb{b} \rhd \Gamma'}{\text{B,}\Gamma \vdash \text{int metC1() \{int z; int g; z = b.get(); g = z + 1; return g\}} : \big(\big[act : \alpha_2, b : [act : \alpha_1]\big]\big)\big\{\mathbb{b} \& \mathit{unsync}(\Gamma)\big\}\_}\ \text{(T-Method)}$$

where:

$\text{s} = \{\ \text{z} = \text{b.get(); g} = \text{z} + 1\text{; return g;}\ \}$

Δ

$$\dfrac{\begin{array}{c}\Gamma \vdash b : [act : \alpha_1] \quad \Gamma \vdash \texttt{B}.\text{get:}\ \big([act : \alpha_2]\big) \to \_ \\ \Gamma \vdash this : \big[act : \alpha_2, b : [act : \alpha_1]\big] \quad f\ fresh \\ \Gamma_1 = \Gamma + \Big\{ f \mapsto \Big(\alpha_1 \rightsquigarrow \_\ ,\ \texttt{B}.\text{get}\big([act : \alpha_1]\big) \to \_\Big)\Big\}\end{array}}{\begin{array}{c}\Gamma + \texttt{this} : \big[act : \alpha_2, b : [act : \alpha_1]\big] + \texttt{destiny} : \_ \vdash \\ \text{b.get()} : f,\ \ [\alpha_2 = \alpha_1]\ \Big(\texttt{B}.\text{get}\big([act : \alpha_1]\big) \to \_\Big) \rhd \Gamma_1\end{array}}\ \text{(T-Invk)}$$

$$\dfrac{\Gamma + \texttt{this} : \big[act : \alpha_2, b : [act : \alpha_1]\big] + \texttt{destiny} : \_ \vdash \text{b.get()} : f, \mathbb{b}_1 \rhd \Gamma_1}{\Gamma + \texttt{this} : \big[act : \alpha_2, b : [act : \alpha_1]\big] + \texttt{destiny} : \_ \vdash \text{z} = \text{b.get()} : \mathbb{b}_1 \rhd \Gamma_1[\text{z} \mapsto f]}\ \text{(T-Var-Future)}$$

Δ'

$$\dfrac{\dfrac{\dfrac{\Gamma_1 \vdash f : \Big(\alpha_1 \rightsquigarrow \_\ ,\ \mathtt{B.get}\big([act : \alpha_1]\big) \to \_\Big)}{\Gamma_1 \vdash this : \big[act : \alpha_2, b : [act : \alpha_1]\big] \quad \Gamma_1 \vdash \mathrm{z} : f \qquad \Gamma_2 = \Gamma_1\Big[f \mapsto (\alpha_1 \rightsquigarrow \_\ ,\ \mathtt{0})^{\checkmark}\Big]}\ \text{(T-Value)} \qquad \dfrac{\Gamma_2 \vdash 1 : \_}{\Gamma_2 \vdash 1 : \_,\ \mathtt{0} \rhd \Gamma_2}\ \text{(T-Pure)}}{\Gamma_1 \vdash \mathrm{z} : \_,\ [\alpha_2 \neq \alpha_1]\ \Big(\big(\mathtt{B.get}\big([act : \alpha_1]\big) \to \_\big)\&(\alpha_2, \alpha_1)\Big) \rhd \Gamma_2} \qquad \dfrac{}{\Gamma_2 \vdash 1 : \mathbb{r},\ \mathbb{b}_2'' \rhd \Gamma_2}}{\Gamma_1 \vdash \mathrm{z} : \_,\ \mathbb{b}_2' \rhd \Gamma_2}\ \text{(T-Exp)} \qquad \dfrac{}{\Gamma_2 \vdash 1 : \mathbb{r},\ \mathbb{b}_2'' \rhd \Gamma_2}$$

$$\dfrac{\Gamma_1 \vdash \mathrm{z} + 1 : \_,\ \mathbb{b}_2' \ \&\ \mathbb{b}_2'' \rhd \Gamma_2}{\Gamma_1 \vdash \mathrm{g} = \mathrm{z} + 1 : \mathbb{b}_2 \rhd \Gamma_2[\ \mathrm{g} \mapsto \_]}\ \text{(T-Var-Record)}$$

where:

$$\Gamma_1 = \Gamma + \Big\{\ \mathrm{z} \mapsto f\ ,\ f \mapsto \Big(\alpha_1 \rightsquigarrow \_\ ,\ \mathtt{B.get}\big([act : \alpha_3]\big) \to \_\Big)\Big\}$$

$$\Gamma_2 = \Gamma_1\big[f \mapsto (\alpha_1 \rightsquigarrow \_\ , \mathtt{0})^{\checkmark}\big] + \{\ \mathrm{g} \mapsto \_\}$$

$$\Gamma_2 = \Gamma'$$

$$\mathbb{b} = \ [\alpha_2 = \alpha_1]\ \Big(\mathtt{B.get}\big([act : \alpha_1]\big) \to \_\Big)\ \mathbin{;}\ [\alpha_2 \neq \alpha_1]\ \Big(\big(\mathtt{B.get}\big([act : \alpha_1]\big) \to \_\big)\&(\alpha_2, \alpha_1)\Big)\ \mathbin{;}\ \mathtt{0}$$

Chapter 4

The language of lams

4.1 Introduction

Deadlock-freedom of concurrent programs has been largly investigated in the litterature [bibliografia]. The proposed algorithms automatically detect deadlocks by building graphs of dependencies (a, b) between resources, meaning that the release of a resource referenced by a depends on the release of the resource referenced by b. The absence of cycles in the graphs entails deadlock freedom. When programs have infinite states, in order to ensure termination, current algorithms use finite approximate models that are excerpted from the dependency graphs. The cases that are particularly critical are those of programs that create networks with an arbitrary number of nodes. In this paper we develop a technique to enable the deadlock analysis of processes with arbitrary networks of nodes. Instead of reasoning on finite approximations of such processes, we associate them with terms of a basic recursive model, called lam (*deadLock Analysis Model*), which collects dependencies and features recursion and dynamic name creation [bibliografia].

4.2 The language of lams

4.2.1 Syntax

We use an infinite set $\mathscr{A}$ of (level) *names*, ranged over a,b,c,$\cdots$. A relation on a set A of names, denoted R,R'.$\cdots$,is an elemento of $\mathscr{P}(\mathrm{A} \times \mathrm{A})$, $\mathscr{P}(\cdot)$ is the standard powerset operator and $\cdot \times \cdot$ is the cartesian product. Let

– R^+ be the *transitive closure* of R

– $\{\mathsf{R}_1, \cdots, \mathsf{R}_m\} \Subset \{\mathsf{R}'_1, \cdots, \mathsf{R}'_n\}$ if and only if, for all R_i, there is R'_j such that $\mathsf{R}_i \subseteq {\mathsf{R}'_j}^+$

– $(a_0, a_1), \cdots, (a_{n-1}, a_n) \in \{\mathsf{R}_1, \cdots, \mathsf{R}_m\}$ if and only if there is R_i such that $(a_0, a_1), \cdots, (a_{n-1}, a_n) \in \mathsf{R}_i$

– $\{\mathsf{R}_1, \cdots, \mathsf{R}_m\} \,\&\, \{\mathsf{R}'_1, \cdots, \mathsf{R}'_n\} \stackrel{def}{=} \{\mathsf{R}_i \cup \mathsf{R}'_j | 1 \leq i \leq m \text{ and } 1 \leq j \leq n\}$.

We use $\mathscr{RR}$,$\cdots$ to range over $\{\mathsf{R}_1, \cdots, \mathsf{R}_m\} \,\&\, \{\mathsf{R}'_1, \cdots, \mathsf{R}'_n\}$ which are element of $\mathscr{P}(\mathscr{P}(\mathrm{A} \times \mathrm{A}))$.

Definition 1 *A relation* R ***has a circularity*** *if* $(a,a) \in \mathsf{R}^+$ *for same a, A set of relations* $\mathscr{P}$ *has a circularity if there is* $\mathsf{R} \in \mathscr{R}$ *that has a circularity.*

For instance $\big\{\{(a,b),(b,c)\},\{(a,b),(c,b),(d,b),(b,c)\},\{(e,d),(d,c)\},\{(e,d)\}\big\}$ has a circularity because the second element of the set does.

In addition to the set of (level) *names*, we will use a *function names*, randeg over by $\mathtt{f}$,$\mathtt{g}$,$\cdots$. A sequence of names is denoted by $\widetilde{a}$ and, with an abuse of notation,we also use $\widetilde{a}$ to address the *set of names* in the sequence. A *lam program* is a pair ($\mathscr{L}$,$\mathtt{L}$), where $\mathscr{L}$ is a *finite set* of *function definitions* $\mathtt{f}(\widetilde{a}) = \mathtt{L}_\mathtt{f}$ with $\widetilde{a}$ and $\mathtt{L}_\mathtt{f}$ being the *formal parameters* and the *body* of $\mathtt{f}$, and $\mathtt{L}$ is the *main lam*. The syntax of the function bodies and the main lam is

$$\mathtt{L} \quad ::= \quad \mathtt{0} \;\mid\; (a,b) \;\mid\; \mathtt{f}(\widetilde{a}) \;\mid\; \mathtt{L}\&\mathtt{L} \;\mid\; \mathtt{L}\,;\mathtt{L} \;\mid\; [x=y]\mathtt{L} \;\mid\; [x\neq y]\mathtt{L}$$

The lam $\mathtt{0}$ enforces no dependency, the lam (a,b) enforces the dependency (a,b), while $\mathtt{f}(\widetilde{a})$ represents a function invocation. The composite lam $\mathtt{L}\,;\mathtt{L}'$ enforces the dependencies of $\mathtt{L}$ *and* of $\mathtt{L}'$, while $\mathtt{L}\&\mathtt{L}'$ nondeterministically enforces the dependencies of $\mathtt{L}$ *or* of $\mathtt{L}'$. Whenever parentheses are omitted, the operation "$\&$" has precedence over "$\,;\,$". We will shorten $\mathtt{L}_1\&\cdots\&\mathtt{L}_n$ into $\&_{i\in 1..n}\mathtt{L}_i$. Moreover, we use $\mathtt{T}$ to range over lams that do not contain function invocations.

Let $var(\mathtt{L})$ be the set of names in $\mathtt{L}$. In a function definition $\mathtt{f}(\widetilde{x}) = \mathtt{L}$, $\widetilde{x}$ are the *formal parameters* and the occurrences of names $x \in \widetilde{x}$ in $\mathtt{L}$ are *bound*; the names $var(\mathtt{L}) \setminus \widetilde{x}$ are *free*.

In the syntax of $\mathtt{L}$, the operations "$\&$" and "$\,;\,$" are associative, commutative with $\mathtt{0}$ being the identity on $\&$. Additionally, when $\mathtt{T}$ ranges over lams that do not contain function invocations, the following axioms hold:

$$\mathtt{T}\&\mathtt{T} = \mathtt{T} \qquad \mathtt{T}\,;\mathtt{T} = \mathtt{T} \qquad \mathtt{T}\&(\mathtt{L}'\,;\mathtt{L}'') = \mathtt{T}\&\mathtt{L}'\,;\mathtt{T}\&\mathtt{L}''$$

The axioms permit to rewrite a lam without function invocations as a *collection* (operation $;$) of *relations* (elements of a relation are gathered by

the operation $\&$). Let $\equiv$ be the least conseguence containing the above axioms.

Definition 2 *A lam* $\mathtt{T}$ *is in* ***normal form****, denoted* $\mathtt{nf}(\mathtt{T})$*, if* $\mathtt{T} = (\mathtt{T}_1 \fatsemi \cdots \fatsemi \mathtt{T}_n)$*, where* $\mathtt{T}_1 \fatsemi \cdots \fatsemi \mathtt{T}_n$ *are dependencies only composed with* $\&$*.*

Proposition 1 *For every* $\mathtt{T}$*, there is* $\mathtt{nf}(\mathtt{T})$ *such that* $\mathtt{T} \equiv \mathtt{nf}(\mathtt{T})$*.*

In the rest of the work, we will never distinguish between equal lams. Moreover, we alaways assume lam programs ($\mathscr{L}$,$\mathtt{L}$) to be *well formed*.

Remark 1 *The axioms (1) are restructed to tern* $\mathtt{T}$ *that do not contain function invocations. In fact,* $\mathtt{f}(\widetilde{d})\&((a,b) \fatsemi (b,c)) \neq (\mathtt{f}(\widetilde{d})\&(a,b)) \fatsemi (\mathtt{f}(\widetilde{d})\&(b,c))$ *because the evaluation of the two lams may produce terms with different name.*

Definition 3 *A lam program* ($\mathscr{L}$,$\mathtt{L}$) *is* ***well formed*** *if (1) function definitions in* $\mathscr{L}$ *have pairwise different function names adn all function names occurring in the function bodies and* $\mathtt{L}$ *are defined: (2) the arity of function invocations occurring anywhere in the program matches the arity of the corresponding function definition; (3) every function definition in* $\mathscr{L}$ *has shape* $\mathtt{f}(\widetilde{a}) = \mathtt{L}_f$

4.2.2 Operational semantics

Let a *lam context*, noted $\mathfrak{L}[\,]$, be a term derived by the following syntax:

$$\mathfrak{L}[\,] \quad ::= \quad [\,] \quad | \quad \mathtt{L}\&\mathfrak{L}[\,] \quad | \quad \mathtt{L} \fatsemi \mathfrak{L}[\,]$$

As usual $\mathfrak{L}[\mathtt{L}]$ is the lam where the hole of $\mathfrak{L}[\,]$ is replaced by $\mathtt{L}$. The operational semantics of a program ($\mathscr{L}$, $\mathtt{L}$) is a transition system whose *states* are lams, the *transition relation* is the least one satisfying the rule:

$$\text{(RED)}\quad \frac{\mathtt{f}(\widetilde{x}) = \mathtt{L} \qquad \widetilde{w} \text{ are fresh} \qquad \mathtt{L}[\widetilde{w}/\widetilde{x}] = \mathtt{L}'_{\mathtt{f}} \qquad \mathtt{nf}(\mathtt{L}'_{\mathtt{f}}) = \mathtt{L}''_{\mathtt{f}}}{\mathfrak{L}[\mathtt{f}(\widetilde{u})] \longrightarrow \mathfrak{L}[\mathtt{L}'']}$$

By (RED), a lam $\mathtt{L}$ is evaluated by successively replacing function invocations with the corresponding lam instances. Name creation is handled with a mechanism similar to that of mutations. For example, if $\mathtt{f}(a) = (a,c)\&\mathtt{f}(c)$ and $\mathtt{f}(a')$ occurs in the main lam, then $\mathtt{f}(a')$ is replaced by $(a',c')\&\mathtt{f}(c')$, where c' is a fresh name.

Appendix A

Runtime Rules and Subject Reduction

A.1 Runtime Rules

In order to type the configurations we use a *runtime type system*. To this aim we extend the syntax of contracts in Figure 3.1 and define *extended futures* F and *behavioural type* $\mathbb{k}$ as follows:

$$F ::= f \mid \mathbb{1}_f$$

$$\mathbb{b} ::= \textit{as in Figure 3.1} \mid f\&(\alpha, \alpha')$$

$$\mathbb{k} ::= \mathbf{0} \mid \mathbb{b} \mid \langle \mathbb{b} \rangle_f \mid \mathbb{k} \fatsemi \mathbb{k} \mid \mathbb{k}\&\mathbb{k}$$

As regards F, they are introduced for distinguishing two kind of future names: i) f that has been used in the contract inference system as a *static time* representation of a future, but is now used as its *runtime* representation; ii) $\mathbb{1}_f$ now replacing f in its role of *static time* future (it's typically used to reference a future that isn't created yet).

As regards $\mathbb{b}$ and $\mathbb{k}$, the extensions are motivated by the fact that, at runtime, the informations about contracts are scattered in all the configuration. However, when we plug all the parts to type the whole configuration, we can merge the different informations to get back a simple contract $\mathbb{b}$. This merging is done using a set of rewriting rules $\Rightarrow$ defined in Figure A.1 that let one replace the occurrences of runtime futures in runtime contracts $\mathbb{k}$ with the corresponding contract of the future. We write $f \in \mathit{names}(\mathbb{k})$ whenever f occurs in $\mathbb{k}$.

The substitution $\mathbb{k}[^{\mathbb{b}}/_f]$ replace the occurrences of f in contracts $\mathbb{b}$ of $\mathbb{k}$ (by definition of our configurations, in these cases f can never occur as index in $\mathbb{k}$).

It is easy to demonstrate that the merging process always terminates and, in the following, we let $(\!|\mathbb{k}|\!)$ be the *normal form* of $\mathbb{k}$ with respect to $\Rightarrow$.

$$\mathbb{k}_1\&\cdots\&\langle\mathbb{b}\rangle_f\&\cdots\&\mathbb{k}_n \Rightarrow (\mathbb{k}_1\&\cdots\&\mathbb{k}_n)[^{\mathbb{b}}/_f] \qquad \text{if } f \in \mathit{names}(\mathbb{k}_1\&\cdots\&\mathbb{k}_n)$$

$$\mathbb{k}_1\&\cdots\&\langle\mathbb{b}\rangle_f\&\cdots\&\mathbb{k}_n \Rightarrow \mathbb{k}_1\&\cdots\&\mathbb{k}_n\&\mathbb{b} \qquad \text{if } f \notin \mathit{names}(\mathbb{k}_1\&\cdots\&\mathbb{k}_n)$$

Figure A.1: Definition of $\Rightarrow$

The typing rules for the runtime configuration are in Figure A.2, A.3, A.4, A.5. Additionally, the typing judgments are identical to the corresponding one in the typing system, with two minor differences:

– the typing environment, that now contains a reference to the behavioural type class table and mappings object names to pairs $(\mathtt{C}, \mathbb{r})$, is called Δ; as for Γ in Chapter 3, we have that $\Delta[x \mapsto v]$ is the function such that $(\Delta[x \mapsto v])(x) = v$ and $(\Delta[x \mapsto v])(y) = \Delta(y)$, when $y \neq x$.

– the $rt\_unsync(\cdot)$ function on environments Δ is similar to $unsync(\cdot)$ in Chapter 3, except that it grabs values of static time futures $\imath_f$ rather than those on futures f (which are runtime now). More precisely

$$rt\_unsync(\Delta) \stackrel{def}{=} \mathbb{b}_1 \& \cdots \& \mathbb{b}_n$$

where $\{\mathbb{b}_1, \cdots, \mathbb{b}_n\} = \{\mathbb{b}' \mid \text{there are } \imath_f, \mathbb{r} : \ \Delta(\imath_f) = (\mathbb{r}, \mathbb{b}')\}$.

$\widehat{\sigma}$ is an environment such that $\mathrm{dom}(\sigma) = \mathrm{dom}(\widehat{\sigma})$ and

$$\text{(TR-Sigma)}\quad \frac{\widehat{\sigma}(o) = [act : \alpha, \overline{x : \mathbb{r}}] \qquad \Delta + \widehat{\sigma} \vdash^{\alpha}_{R} \overline{v : \mathbb{r}}}{\Delta + \widehat{\sigma} \vdash^{\alpha}_{R} o : [act : \alpha, \overline{x : \mathbb{r}}]}$$

for every o $\in \mathrm{dom}(\sigma)$

Figure A.2: Runtime typing rules for $\widehat{\sigma}$

The typing rules for runtime configurations

$$\text{(TR-Future)}\quad \frac{\Delta(f) = (\alpha \rightsquigarrow \mathbb{r}, \mathbb{b}) \qquad val \neq \bot \Rightarrow \Delta(val) = \mathbb{r}}{\Delta \vdash_R fut(f, val) : \mathtt{0}}$$

$$\text{(TR-Process)}\quad \frac{\Delta(o) = [act : \alpha, \overline{x : \mathbb{r}}] \qquad \Delta \vdash_R \overline{val : \mathbb{x}} \qquad \Delta' = \Delta[\mathrm{destiny} \mapsto f, \mathrm{this} \mapsto o, \overline{x \mapsto \mathbb{x}}] \qquad \Delta' \vdash_R s : \mathbb{b} \mid \Delta''}{\Delta \vdash_R \{\mathrm{destiny} \mapsto f, \mathrm{this} \mapsto o, \ \overline{x \mapsto val} \mid s\} : \langle \mathbb{b} \& rt\_unsync(\Delta'') \rangle_f}$$

$$\text{(TR-Idle)}\quad \Delta \vdash_R \mathtt{idle} : \mathtt{0}$$

$$\text{(TR-Object)}\quad \frac{\Delta + \widehat{\sigma} \vdash^{\alpha}_{R} \sigma \qquad \Delta + \widehat{\sigma} \vdash_R p : \mathbb{k} \qquad \Delta + \widehat{\sigma} \vdash_R \overline{q} : \overline{\mathbb{k}'}}{\Delta \vdash_R ob(\alpha, o, \sigma, p, \overline{q}) : \mathbb{k} \& \overline{\mathbb{k}'}}$$

$$\text{(TR-Configurations)}\quad \frac{\Delta \vdash_R \overline{ob(\alpha, o, \sigma, p, q)} : \overline{\mathbb{k}_o} \qquad \Delta \vdash_R \overline{fut(f, val)} : \overline{\mathtt{0}}}{\Delta \vdash_R \overline{ob(\alpha, o, \sigma, p, q)} \ \overline{fut(f, val)} \ : (\!| \& \overline{\mathbb{k}_o} |\!)}$$

Figure A.3: Typing rules for runtime configurations

Runtime typing rules for expressions.

$$\text{(TR-Obj)}\ \frac{\Delta(o) = \mathbb{r}}{\Delta \vdash_R o : \mathtt{C}, \mathbb{r}} \qquad \text{(TR-Fut)}\ \frac{\Delta(F) = \mathbb{z}}{\Delta \vdash_R F : \mathbb{z}} \qquad \text{(TR-Var)}\ \frac{\Delta(x) = \mathbb{x}}{\Delta \vdash_R x : \mathbb{x}} \qquad \text{(TR-Field)}\ \frac{x \notin \mathrm{dom}(\Delta) \qquad \Delta(\mathtt{this}.x) = \mathbb{r}}{\Delta \vdash_R x : \mathbb{r}}$$

$$\text{(TR-Val)}\ \frac{e \quad \textit{primitive value}}{\Delta \vdash_R e : \_} \qquad \text{(TR-Pure)}\ \frac{\Delta \vdash_R e : \mathbb{r}}{\Delta \vdash_R e : \mathbb{r}, \mathtt{0} \mid \Delta}$$

Runtime typing rules for expressions with side-effects.

$$\text{(TR-Value)}\ \frac{\Delta \vdash_R x : \mathbb{1}_f \qquad \Delta \vdash_R \mathbb{1}_f : (\mathbb{r}, \mathbb{b}) \qquad \Delta \vdash_R \mathtt{this} : [act : \alpha, \overline{x : \mathbb{s}}] \qquad \Delta' = \Delta[\mathbb{1}_f \mapsto (\mathbb{r}, \mathtt{0})^{\checkmark}] \qquad \mathbb{r} = \alpha' \rightsquigarrow \mathbb{r}'}{\Delta \vdash_R x : \mathbb{r}', [\alpha \neq \alpha'](\mathbb{b} \& (\alpha, \alpha')) \mid \Delta'}$$

$$\text{(TR-Value-Runtime)}\ \frac{\Delta \vdash_R x : f \qquad \Delta \vdash_R f : (\mathbb{r}, \mathbb{b}) \qquad \Delta \vdash_R \mathtt{this} : [act : \alpha, \overline{x : \mathbb{s}}] \qquad \Delta' = \Delta[f \mapsto (\mathbb{r}, \mathtt{0})^{\checkmark}] \qquad \mathbb{r} = \alpha' \rightsquigarrow \mathbb{r}'}{\Delta \vdash_R x : \mathbb{r}', f \& (\alpha, \alpha') \mid \Delta'}$$

$$\text{(TR-Value-Tick)}\ \frac{\Delta \vdash_R x : F \qquad \Delta \vdash_R F : (\mathbb{r}, \mathbb{b})^{\checkmark} \qquad \mathbb{r} = \alpha \rightsquigarrow \mathbb{r}'}{\Delta \vdash_R x : \mathbb{r}', \mathtt{0} \mid \Delta} \qquad \text{(TR-Exp)}\ \frac{\Delta \vdash_R e : \mathbb{r}, \mathbb{b} \mid \Delta' \qquad \Delta' \vdash_R e' : \mathbb{r}, \mathbb{b}' \mid \Delta''}{\Delta \vdash_R e \oplus e' : \mathbb{r}, \mathbb{b} \& \mathbb{b}' \mid \Delta''}$$

$$\text{(TR-NewActive)}\ \frac{\left(\Delta_i \vdash_R e_i : \mathbb{r}_i, \mathbb{b}_i \mid \Delta_{i+1}\right)^{i=1\ldots n} \qquad \textit{fields}(\mathtt{C}) = \overline{T\ x} \qquad \alpha \text{ fresh}}{\Delta_1 \vdash_R \mathtt{newActive\ C}(e_1, \ldots, e_n) : [act{:}\alpha, x_1{:}\mathbb{r}_1, \ldots, x_n{:}\mathbb{r}_n], \mathbb{b}_1 \& \ldots \& \mathbb{b}_n \mid \Delta_{n+1}}$$

$$\text{(TR-New)}\ \frac{\left(\Delta_i \vdash_R e_i : \mathbb{r}_i, \mathbb{b}_i \mid \Delta_{i+1}\right)^{i=1\ldots n} \qquad \textit{fields}(\mathtt{C}) = \overline{T\ x} \qquad \Delta_1 \vdash_R \mathtt{this} : [act : \alpha, \overline{x : \mathbb{s}}]}{\Delta_1 \vdash_R \mathtt{new\ C}(e_1, \ldots, e_n) : [act{:}\alpha, x_1{:}\mathbb{r}_1, \ldots, x_n{:}\mathbb{r}_n], \mathbb{b}_1 \& \ldots \& \mathbb{b}_n \mid \Delta_{n+1}}$$

$$\text{(TR-Invk)}\ \frac{\begin{array}{c}\Delta_0 \vdash_R e : [act{:}\alpha', \overline{x{:}\mathbb{r}}], \mathbb{b}_0 \mid \Delta_1 \\ \Delta_0 \vdash_R \mathtt{this} : [act{:}\alpha, \overline{x{:}\mathbb{s}}] \qquad \left(\Delta_i \vdash_R e_i : \mathbb{s}_i, \mathbb{b}_i \mid \Delta_{i+1}\right)^{i=1\ldots n} \\ \textit{class}(\textit{types}(e)) = \mathtt{C} \qquad \textit{fields}(\mathtt{C}) = \overline{T\ x} \qquad \Delta_0(\mathtt{C.m}) = \mathbb{r}'(\overline{\mathbb{s}'})\{\mathbb{b}\}\mathbb{r}'' \\ \overline{\alpha'} = \textit{act\_names}(\mathbb{r}'') \setminus \textit{act\_names}(\mathbb{r}', \overline{\mathbb{s}'}) \qquad \overline{\alpha}, \mathbb{1}_f \text{ fresh} \qquad \mathbb{s}'' = \mathbb{r}''[\overline{\alpha}/\overline{\alpha'}][\mathbb{r}, \overline{\mathbb{s}}/\mathbb{r}', \overline{\mathbb{s}'}] \\ \Delta' = \Delta_{n+1}\left[\mathbb{1}_f \mapsto \Big(\alpha' \rightsquigarrow \mathbb{s}'', [\alpha \neq \alpha'](\mathtt{C.m}([act : \alpha', \overline{x{:}\mathbb{r}}], \overline{\mathbb{s}}) \to \mathbb{s}'')\Big)\right]\end{array}}{\Delta_0 \vdash_R e.\mathtt{m}(e1, \ldots, e_n) : \mathbb{1}_f, \mathbb{b}_0 \& \ldots \& \mathbb{b}_n \wr [\alpha = \alpha']\Big((\mathtt{C.m}([act : \alpha', \overline{x{:}\mathbb{r}}], \overline{\mathbb{s}}) \to \mathbb{s}'') \& \textit{rt\_unsync}(\Delta_{n+1})\Big) \mid \Delta'}$$

Figure A.4: Runtime typing rules for expressions and expressions with side-effects.

Runtime typing rules for statements.

(TR-Var-Record)
$$\frac{\Delta \vdash_R x : \mathbb{r} \qquad \Delta \vdash_R z : \mathbb{r}}{\Delta \vdash_R x = z : \mathtt{0} \mid \Delta[x \mapsto \mathbb{r}]}$$

(TR-Field-Record)
$$\frac{x \notin \mathrm{dom}(\Delta) \qquad \Delta(\mathtt{this}.x) = \mathbb{r} \qquad \Delta \vdash_R z : \mathbb{r}}{\Delta \vdash_R x = z : \mathtt{0} \mid \Delta}$$

(TR-Var-Future)
$$\frac{\Delta \vdash_R x : F}{\Delta \vdash_R x = f : \mathtt{0} \mid \Delta[x \mapsto f]}$$

(TR-Var-Expression)
$$\frac{\Delta \vdash_R e : \mathbb{r}, \mathbb{b} \mid \Delta' \ e \textit{ is not a variable}}{\Delta \vdash_R x = e : \mathbb{b} \mid \Delta'[x \mapsto \mathbb{r}]}$$

(TR-IF)
$$\frac{\begin{array}{c}\Delta \vdash_R e : \mathtt{Bool} \qquad \Delta \vdash_R s_1 : \mathbb{b}_1 \mid \Delta_1 \qquad \Delta \vdash_R s_2 : \mathbb{b}_2 \mid \Delta_2 \\ x \in \mathrm{dom}(\Delta) \Longrightarrow \Delta_1(x) = \Delta_2(x) \\ x \in \mathtt{Fut}(\Delta) \Longrightarrow \Delta_1(\Delta_1(x)) = \Delta_2(\Delta_2(x)) \\ \Delta' = \Delta_1 + (\Delta_2 \setminus (\mathrm{dom}(\Delta) \cup \{\Delta_2(x) \mid x \in \mathtt{Fut}(\Delta_2)\}))\end{array}}{\Delta \vdash_R \mathtt{if}\ e\ \{\, s_1 \,\}\ \mathtt{else}\ \{\, s_2 \,\} : \mathbb{b}_1 \& \mathbb{b}_2 \mid \Delta'}$$

(TR-Seq)
$$\frac{\Delta \vdash_R s_1 : \mathbb{b}_1 \mid \Delta_1 \qquad \Delta_1 \vdash_R s_2 : \mathbb{b}_2 \mid \Delta_2}{\Delta \vdash_R s_1; s_2 : \mathbb{b}_1 \fatsemi \mathbb{b}_2 \mid \Delta_2}$$

(TR-Return)
$$\frac{\Delta \vdash_R e : \mathbb{r} \qquad \Delta(\mathrm{destiny}) = f \qquad \Delta(f) = (c \rightsquigarrow \mathbb{r}, \mathbb{b})}{\Delta \vdash_R \mathtt{return}\ e : \mathtt{0} \mid \Delta}$$

Figure A.5: Runtime typing rules for statements.

A.2 Subject Reduction

Theorem 1 (Subject Reduction)

Let $\Delta \vdash_R cn : \mathbb{b}$ *and* $cn \to cn'$. *Then there exist* Δ', $\mathbb{b}'$, *and an injective renaming of activity names* $\imath$ *such that*

- $\Delta' \vdash_R cn' : \mathbb{b}'$ *and*
- $\imath(\mathbb{b}') \trianglelefteq \mathbb{b}$.

The proof is a case analysis on the reduction rule used in $cn \to cn'$ and we assume that the evaluation of an expression $[\![e]\!]_\sigma$ always terminates.

Definition 4 ($\trianglelefteq$) *Let* $\mathbb{b} \trianglelefteq \mathbb{b}'$, *say* $\mathbb{b}$ *is a later stage than* $\mathbb{b}'$, *be the least relation including the rules in Figure A.6. This later stage relation is a syntactic relationship between contracts whose basic law is that a method invocation is larger than the instantiation of the behavioral type of the method.*

Below for each rule of the operational semantics, which is present in Figure 2.3, in which we have that $cn \to cn'$, we will show how $\Delta \vdash_R cn : \mathbb{b}$ presenting the entire proof tree, then will be identified Δ' and after we show how $\Delta' \vdash_R cn' : \mathbb{b}'$. After we have defined the behavioral types $\mathbb{b}$ and $\mathbb{b}'$ we will show that $\imath(\mathbb{b}') \trianglelefteq \mathbb{b}$. In each proof tree we well identify with Φ,Φ',$\Phi''\ldots$ the branches of the proof tree that we assume that is demonstrated by hypothesis.

rules of the later stage relation

$$\text{(LS-Self)}\quad \mathbb{b} \trianglelefteq \mathbb{b} \qquad \text{(LS-Null)}\quad \mathtt{0} \trianglelefteq \mathbb{b} \qquad \text{(LS-RInvk)}\quad \frac{\mathbb{r} = [act : \alpha', \overline{x} : \overline{\mathbb{s}}] \quad \alpha \neq \alpha'}{\mathtt{C.m}\ \mathbb{r}(\overline{\mathbb{r}}) \to \mathbb{r}' \& (\alpha, \alpha') \trianglelefteq \mathtt{C.m}\ \mathbb{r}(\overline{\mathbb{r}}) \to \mathbb{r}'}$$

$$\text{(LS-Invk)}\quad \frac{\begin{array}{c}\textsc{bct}(\mathtt{C.m}) = \mathbb{r}(\overline{\mathbb{r}})\{\mathbb{b}\}\mathbb{r}' \\ \overline{\alpha} = act\_names(\mathbb{b}) \setminus act\_names(\mathbb{r}, \overline{\mathbb{r}}, \mathbb{r}') \\ \overline{\alpha'} \cap act\_names(\mathbb{s}, \overline{\mathbb{s}}, \mathbb{s}') = \varnothing\end{array}}{\mathbb{b}[\overline{\alpha'}/\overline{\alpha}][\mathbb{s}, \overline{\mathbb{s}}, \mathbb{s}'/\mathbb{r}, \overline{\mathbb{r}}, \mathbb{r}'] \trianglelefteq \mathtt{C.m}\ \mathbb{s}(\overline{\mathbb{s}}) \to \mathbb{s}'}$$

$$\text{(LS-Sinc)}\quad \frac{\mathbb{b} \trianglelefteq \mathbb{b}'}{\mathbb{b}\&(\alpha, \alpha') \trianglelefteq \mathbb{b}'\&(\alpha, \alpha')} \qquad \text{(LS-Trans)}\quad \frac{\mathbb{b}_1 \trianglelefteq \mathbb{b}_2 \quad \mathbb{b}_2 \trianglelefteq \mathbb{b}_3}{\mathbb{b}_1 \trianglelefteq \mathbb{b}_3}$$

$$\text{(LS-Seq)}\quad \frac{\mathbb{b}_1 \trianglelefteq \mathbb{b}_3 \quad \mathbb{b}_2 \trianglelefteq \mathbb{b}_4}{\mathbb{b}_1 \fatsemi \mathbb{b}_2 \trianglelefteq \mathbb{b}_3 \fatsemi \mathbb{b}_4} \qquad \text{(LS-Par)}\quad \frac{\mathbb{b}_1 \trianglelefteq \mathbb{b}_3 \quad \mathbb{b}_2 \trianglelefteq \mathbb{b}_4}{\mathbb{b}_1 \& \mathbb{b}_2 \trianglelefteq \mathbb{b}_3 \& \mathbb{b}_4}$$

Figure A.6: The later-stage relation rules.

Object Creation.

We analyze below the rule (New-Object) in Figure 2.3.

$$\text{(New-Object)}\quad \frac{\mathit{fields}(\mathtt{C}) = \overline{T\ x} \quad o' = \text{fresh}(\mathtt{C}) \quad [\![\overline{e}]\!]_{(\sigma+\ell)} = \overline{v} \quad \sigma' = \sigma \cup \{o' \mapsto [\overline{x = v}]\}}{ob(\alpha, o, \sigma, \{\ell \mid x = \mathtt{new}\ \mathtt{C}(\overline{e})\ ;\ s\ \}, q) \to ob(\alpha, o, \sigma', \{\ell \mid x = o'\ ;\ s\ \}, q)}$$

- Let $\Delta \vdash_R ob(\alpha, o, \sigma, \{\ell \mid x = \mathtt{new}\ \mathtt{C}(\overline{e})\ ;\ s\ \}, q) : \mathbb{b}$
- knowing that:
$$ob(\alpha, o, \sigma, \{\ell \mid x = \mathtt{new}\ \mathtt{C}(\overline{e})\ ;\ s\ \}, q) \to ob(\alpha, o, \sigma', \{\ell \mid x = o'\ ;\ s\ \}, q)$$
- we must show there exist:
$$\Delta' \vdash_R ob(\alpha, o, \sigma', \{\ell \mid x = o';\ s\ \}, q) : \mathbb{b}'$$
- and $\imath(\mathbb{b}') \trianglelefteq \mathbb{b}$.

We show below that $\Delta \vdash_R ob(\alpha, o, \sigma, \{\ell \mid x = \texttt{new C}(\overline{e})\ ;\ s\ \}, q) : \mathbb{b}$

$$\dfrac{\Delta + \widehat{\sigma} \vdash_R \sigma \qquad \dfrac{\Theta}{\Delta + \widehat{\sigma} \vdash_R \{\ell \mid x = \texttt{new C}(\overline{e})\ ;\ s\ \} : \langle \mathbb{b} \rangle_f}\ \text{(TR-Process)} \qquad \dfrac{\Phi}{\Delta + \widehat{\sigma} \vdash_R q : \mathbb{k}_q}\ \text{Hyp}}{\dfrac{\Delta \vdash_R ob(\alpha, o, \sigma, \{\ell \mid x = \texttt{new C}(\overline{e})\ ;\ s\ \}, q) : (\!|\mathbb{k}_p \& \mathbb{k}_q|\!)}{ob(\alpha, o, \sigma, \{\ell \mid x = \texttt{new C}(\overline{e})\ ;\ s\ \}, q)}\ \text{(TR-Configuration)}}\ \text{(TR-Object)}$$

Θ

$$\dfrac{\Delta + \widehat{\sigma}(o) = [act : \alpha, \overline{x : \mathbb{r}}] \qquad \Delta + \widehat{\sigma} \vdash_R \overline{val : \mathbb{x}} \qquad \Delta_1 = \Delta + \widehat{\sigma}[\text{destiny} \mapsto f, \text{this} \mapsto o', \overline{x \mapsto \mathbb{x}}] \qquad \Theta'}{\Delta + \widehat{\sigma} \vdash_R \{\text{destiny} \mapsto f, \text{this} \mapsto o', \overline{x \mapsto \mathbb{x}} \mid x = \texttt{new C}(\overline{e})\ ;\ s\ \} : \langle \mathbb{b} \& rt\_unsync(\Delta'') \rangle_f}\ \text{(TR-Process)}$$

Θ'

$$\dfrac{\dfrac{\dfrac{}{\Delta_1 \vdash_R \texttt{this} : [act : \alpha, \overline{x : \mathbb{s}}]} \qquad \dfrac{\dfrac{\Phi''}{(\Delta_i \vdash_R e_i : \mathbb{r}_i, \mathbb{b}_i \mid \Delta_{i+1})^{i=1\ldots n}}\ \text{Hyp} \qquad \mathit{fields}(\texttt{C}) = \overline{T\ x}}{\Delta_1 \vdash_R \texttt{new C}(e_1, \ldots, e_n) : [act{:}\alpha, x_1{:}\mathbb{r}_1, \ldots, x_n{:}\mathbb{r}_n], \mathbb{b}_1 \& \ldots \& \mathbb{b}_n \mid \Delta_{n+1}}\ \text{(TR-New)}}\ \text{(TR-Var-Expression)} \qquad \dfrac{\Phi'}{\Delta_{n+1} \vdash_R s : \mathbb{b}'' \mid \Delta''}\ \text{Hyp}}{\dfrac{\Delta_1 \vdash_R x = \texttt{new C}(\overline{e}) : \mathbb{b}' \mid \Delta_{n+1}[x \mapsto \mathbb{r}]}{\Delta_1 \vdash_R x = \texttt{new C}(\overline{e})\ ;\ s\ : \mathbb{b}' \fatsemi \mathbb{b}'' \mid \Delta''}\ \text{(TR-Seq)}}$$

With the demonstration above we can deduce that:
$\Delta \vdash_R ob(\alpha, o, \sigma, \{\ell \mid x = \texttt{new C}(\overline{e})\ ;\ s\ \}, q) : \left\langle \left((\mathbb{b}_1 \& \cdots \& \mathbb{b}_n) \fatsemi \mathbb{b}'' \right) \& rt\_unsync(\Delta'') \right\rangle_f \& \mathbb{k}_q$

Now we can proof that $\Delta' \vdash_R ob(\alpha, o, \sigma', \{\ell \mid x = o'\}, q) : \mathbb{b}'$ with $\Delta' = \Delta[x \mapsto o']$

$$\dfrac{\Delta' + \widehat{\sigma} \vdash_R \sigma' \qquad \dfrac{\Sigma}{\Delta' + \widehat{\sigma} \vdash_R \{\ell \mid x = o'\ ;\ s\ \} : \langle \mathbb{b} \rangle_f}\ \text{(TR-Process)} \qquad \dfrac{\Phi}{\Delta' + \widehat{\sigma} \vdash_R q : \mathbb{k}_q}\ \text{Hyp}}{\dfrac{\Delta' \vdash_R ob(\alpha, o, \sigma', \{\ell \mid x = o'\ ;\ s\ \}, q) : (\!|\mathbb{k}_p \& \mathbb{k}_q|\!)}{ob(\alpha, o, \sigma', \{\ell \mid x = o'\ ;\ s\ \}, q)}\ \text{(TR-Configuration)}}\ \text{(TR-Object)}$$

Σ

$$\frac{\Delta' + \widehat{\sigma}(o) = [act : \alpha, \overline{x : \mathbb{r}}] \qquad \Delta' + \widehat{\sigma} \vdash_R \overline{val : \mathbb{x}} \qquad \Delta_1 = \Delta' + \widehat{\sigma}[\text{destiny} \mapsto f, \text{this} \mapsto o', \overline{x \mapsto \mathbb{x}}] \qquad \Sigma'}{\Delta' + \widehat{\sigma} \vdash_R \{\text{destiny} \mapsto f, \text{this} \mapsto o', \overline{x \mapsto \mathbb{x}} \mid x = o' \ ; \ s \ \} : \langle \mathbb{b} \& rt\_unsync(\Delta'') \rangle_f} \text{ (TR-Process)}$$

Σ'

$$\dfrac{\dfrac{\dfrac{\dfrac{\Delta_1(o') = \mathbb{r}}{\Delta_1 \vdash_R o' : \mathtt{C}, \mathbb{r}} \text{ (TR-Obj)} \qquad \Delta_1 \vdash_R x : \mathbb{r}}{\Delta_1 \vdash_R x = o' : \mathtt{0} \mid \Delta_1[x \mapsto \mathbb{r}]} \text{ (TR-Var-Record)} \qquad \dfrac{\Phi'}{\Delta_1 \vdash_R s : \mathbb{b}' \mid \Delta''} \text{ Hyp}}{\Delta_1 \vdash_R x = o' \ ; \ s : \mathbb{b}' \fatsemi \mathbb{b}'' \mid \Delta''} \text{ (TR-Seq)}$$

With the demonstration above we can deduce that:

$$\Delta \vdash_R ob(\alpha, o, \sigma, \{\ell \mid x = o';\ s\ \}, q) : \left\langle \left(\mathtt{0} \fatsemi \mathbb{b}''\right) \& rt\_unsync(\Delta'') \right\rangle_f \& \mathbb{k}_q$$

Now we can show that:

$$\left\langle \left(\mathtt{0} \fatsemi \mathbb{b}''\right) \& rt\_unsync(\Delta'') \right\rangle_f \& \mathbb{k}_q \ \trianglelefteq \ \left\langle \left((\mathbb{b}_1 \& \cdots \& \mathbb{b}_n) \fatsemi \mathbb{b}''\right) \& rt\_unsync(\Delta'') \right\rangle_f \& \mathbb{k}_q.$$

$$\dfrac{\dfrac{\dfrac{\mathtt{0} \trianglelefteq \ \mathbb{b}_1 \& \cdots \& \mathbb{b}_n \qquad \mathbb{b}'' \trianglelefteq \ \mathbb{b}''}{\mathtt{0} \fatsemi \mathbb{b}'' \ \trianglelefteq \ (\mathbb{b}_1 \& \cdots \& \mathbb{b}_n) \fatsemi \mathbb{b}''} \text{ (LS-Seq)} \qquad rt\_unsync(\Delta'') \trianglelefteq \ rt\_unsync(\Delta'')}{\left\langle \left(\mathtt{0} \fatsemi \mathbb{b}''\right) \& rt\_unsync(\Delta'') \right\rangle_f \ \trianglelefteq \ \left\langle \left((\mathbb{b}_1 \& \cdots \& \mathbb{b}_n) \fatsemi \mathbb{b}''\right) \& rt\_unsync(\Delta'') \right\rangle_f} \text{ (LS-Par)} \qquad \mathbb{k}_q \ \trianglelefteq \ \mathbb{k}_q}{\left\langle \left(\mathtt{0} \fatsemi \mathbb{b}''\right) \& rt\_unsync(\Delta'') \right\rangle_f \& \mathbb{k}_q \ \trianglelefteq \ \left\langle \left((\mathbb{b}_1 \& \cdots \& \mathbb{b}_n) \fatsemi \mathbb{b}''\right) \& rt\_unsync(\Delta'') \right\rangle_f \& \mathbb{k}_q} \text{ (LS-Par)}$$

Active Object Creation.

$$\frac{\begin{array}{c}\textit{fields}(\mathtt{C}) = \overline{T\ x} \quad o' = \mathrm{fresh}(\mathtt{C}) \quad \gamma = \mathrm{fresh}(\ ) \\ \sigma' = \{o' \mapsto [\overline{x = v}], \mathrm{serialise}(\overline{v}, \sigma)\} \quad [\![\overline{e}]\!]_{(\sigma+\ell)} = \overline{v}\end{array}}{\begin{array}{c}ob(\alpha, o, \sigma, \{\ell \mid x = \mathtt{newActive}\ \ \mathtt{C}(\overline{e})\ ;\ s\ \}, q) \\ \rightarrow ob(\alpha, o, \sigma, \{\ell \mid x = o'\ ;\ s\ \}, q) \quad ob(\gamma, o', \sigma', \mathtt{idle}, \epsilon)\end{array}}\ (\textsc{New-Active})$$

- Let $\Delta \vdash_R ob(\alpha, o, \sigma, \{\ell \mid x = \mathtt{newActive}\ \ \mathtt{C}(\overline{e})\ ;\ s\ \}, q) : \mathbb{b}$
- knowing that:
  $ob(\alpha, o, \sigma, \{\ell \mid x = \mathtt{newActive}\ \ \mathtt{C}(\overline{e})\ ;\ s\ \}, q) \rightarrow ob(\alpha, o, \sigma, \{\ell \mid x = o'\}, q) \quad ob(\gamma, o', \sigma', \mathtt{idle}, \epsilon)$
- we must show there exist:
  $$\Delta' \vdash_R ob(\alpha, o, \sigma, \{\ell \mid x = o'\ ;\ s\ \}, q) \quad ob(\gamma, o', \sigma', \mathtt{idle}, \epsilon) : \mathbb{b}'$$
- and $\imath(\mathbb{b}') \trianglelefteq \mathbb{b}$.

We show below that $\Delta \vdash_R ob(\alpha, o, \sigma, \{\ell \mid x = \mathtt{newActive}\ \ \mathtt{C}(\overline{e})\ ;\ s\ \}, q) : \mathbb{b}$

$$\dfrac{\dfrac{\Delta + \widehat{\sigma} \vdash^{\alpha}_{R} \sigma \qquad \dfrac{\Theta}{\Delta + \widehat{\sigma} \vdash_R \{\ell \mid x = \mathtt{newActive}\ \ \mathtt{C}(\overline{e})\ ;\ s\ \} : \mathbb{b}_f}\ (\textsc{TR-Process}) \qquad \dfrac{\Phi}{\Delta + \widehat{\sigma} \vdash_R q : \mathbb{k}_q}\ \textsc{Hyp}}{\Delta \vdash_R ob(\alpha, o, \sigma, \{\ell \mid x = \mathtt{newActive}\ \ \mathtt{C}(\overline{e})\ ;\ s\ \}, q) : (\!|\mathbb{k}_p \& \mathbb{k}_q|\!)}\ (\textsc{TR-Object})}{ob(\alpha, o, \sigma, \{\ell \mid x = \mathtt{newActive}\ \ \mathtt{C}(\overline{e})\ ;\ s\ \}, q)}\ (\textsc{TR-Configuration})$$

Θ

$$\frac{\Delta + \widehat{\sigma}(o) = [act : \alpha, \overline{x : \mathbb{r}}] \qquad \Delta + \widehat{\sigma} \vdash_R \overline{val : \mathbb{x}} \qquad \Delta_1 = \Delta + \widehat{\sigma}[\mathrm{destiny} \mapsto f, \mathrm{this} \mapsto o', \overline{x \mapsto \mathbb{x}}] \qquad \Theta'}{\Delta' + \widehat{\sigma} \vdash_R \{\mathrm{destiny} \mapsto f, \mathrm{this} \mapsto o', \overline{x \mapsto \mathbb{x}} \mid x = \mathtt{newActive}\ \ \mathtt{C}(\overline{e})\ ;\ s\ \} : \langle \mathbb{b} \& rt\_unsync(\Delta'') \rangle_f}\ (\textsc{TR-Process})$$

Θ'

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\Delta_1 \vdash_R \texttt{this} : [act : \alpha, \overline{x : \mathbb{s}}]} \qquad \dfrac{\Phi''}{\left(\Delta_i \vdash_R e_i : \mathbb{r}_i, \mathbb{b}_i \mid \Delta_{i+1}\right)^{i=1\ldots n}}\,\text{Hyp} \qquad \mathit{fields}(\texttt{C}) = \overline{T\ x}}{\Delta_1 \vdash_R \texttt{newActive}\ \ \texttt{C}(e_1, \ldots, e_n) : [act{:}\alpha, x_1{:}\mathbb{r}_1, \ldots, x_n{:}\mathbb{r}_n], \mathbb{b}_1 \& \ldots \& \mathbb{b}_n \mid \Delta_{n+1}}\,\text{(TR-NewActive)} \qquad \dfrac{\Phi'}{\Delta_{n+1} \vdash_R s : \mathbb{b}'' \mid \Delta''}\,\text{Hyp}}{\Delta_1 \vdash_R x = \texttt{newActive}\ \ \texttt{C}(\overline{e}) : \mathbb{b}' \mid \Delta_{n+1}[x \mapsto \mathbb{r}]}\,\text{(TR-Var-Expression)}}{\Delta_1 \vdash_R x = \texttt{newActive}\ \ \texttt{C}(\overline{e})\ ;\ s\ : \mathbb{b}' \fatsemi \mathbb{b}'' \mid \Delta''}\,\text{(TR-Seq)}$$

With the demonstration above we can deduce that:
$\Delta \vdash_R ob(\alpha, o, \sigma, \{\ell \mid x = \texttt{newActive}\ \ \texttt{C}(\overline{e})\ ;\ s\ \}, q) : \left\langle \left((\mathbb{b}_1 \& \cdots \& \mathbb{b}_n) \fatsemi \mathbb{b}'' \right) \& rt\_unsync(\Delta'') \right\rangle_f \& \mathbb{k}_q$

Now we can show that $\Delta' \vdash_R ob(\alpha, o, \sigma', \{\ell \mid x = o'\}, q)\ ob(\gamma, o', \sigma', \texttt{idle}, \epsilon) : \mathbb{b}'$ with $\Delta' = \Delta[x \mapsto o']$

$$\dfrac{\dfrac{\dfrac{\Sigma}{\Delta' \vdash_R ob(\alpha, o, \sigma, \{\ell \mid x = o'\ ;\ s\ \}, q) : \mathbb{k}_p \& \mathbb{k}_q}\,\text{(TR-Object)} \qquad \dfrac{\Delta + \widehat{\sigma} \vdash_R \sigma' \qquad \Delta + \widehat{\sigma} \vdash_R \texttt{idle} : 0}{\Delta' \vdash_R ob(\gamma, o', \sigma', \texttt{idle}, \epsilon) : \mathbb{k}'_p}}{\Delta' \vdash_R ob(\alpha, o, \sigma, \{\ell \mid x = o'\ ;\ s\ \}, q)\ ob(\gamma, o', \sigma', \texttt{idle}, \epsilon) : (\!|\mathbb{k} \& \mathbb{k}'|\!)}}{ob(\alpha, o, \sigma, \{\ell \mid x = o'\ ;\ s\ \}, q)\ ob(\gamma, o', \sigma', \texttt{idle}, \epsilon)}$$

Σ

$$\dfrac{\Delta + \widehat{\sigma} \vdash_R \sigma \qquad \dfrac{\dfrac{\Sigma'}{\Delta + \widehat{\sigma} \vdash_R \{\ell \mid x = o'\ ;\ s\ \} : \langle \mathbb{b} \rangle_f}\,\text{(TR-Process)} \qquad \dfrac{\Phi}{\Delta + \widehat{\sigma} \vdash_R q : \mathbb{k}_q}\,\text{Hyp}}{\Delta' \vdash_R ob(\alpha, o, \sigma, \{\ell \mid x = o'\ ;\ s\ \}, q) : \mathbb{k}_p \& \mathbb{k}_q}\,\text{(TR-Object)}$$

Σ'

$$\dfrac{\Delta + \widehat{\sigma}(o) = [act : \alpha, \overline{x : \mathbb{r}}] \qquad \Delta + \widehat{\sigma} \vdash_R \overline{val : \mathbb{x}} \qquad \Delta_1 = \Delta + \widehat{\sigma}[\text{destiny} \mapsto f, \text{this} \mapsto o', \overline{x \mapsto \mathbb{x}}] \qquad \Sigma''}{\Delta + \widehat{\sigma} \vdash_R \{\text{destiny} \mapsto f, \text{this} \mapsto o', \overline{x \mapsto \mathbb{x}} \mid x = o'\ ;\ s\ \} : \langle \mathbb{b} \& rt\_unsync(\Delta'') \rangle_f}\,\text{(TR-Process)}$$

Σ''

$$\dfrac{\dfrac{\dfrac{}{\Delta_1 \vdash_R x : \mathbb{r}}\ \dfrac{\Delta_1(o') = \mathbb{r}}{\Delta_1 \vdash_R o' : \mathtt{C}, \mathbb{r}}\ \text{(TR-Obj)}}{\Delta_1 \vdash_R x = o' : \mathtt{0} \mid \Delta_1[x \mapsto \mathbb{r}]}\ \text{(TR-Var-Record)} \qquad \dfrac{\Phi'}{\Delta_1 \vdash_R s : \mathbb{b}_2 \mid \Delta''}\ \text{Hyp}}{\Delta_1 \vdash_R x = o' \ ; \ s : \mathbb{b}_1 \fatsemi \mathbb{b}_2 \mid \Delta''}\ \text{(TR-Seq)}$$

With the demonstration above we can deduce that:
$\Delta \vdash_R ob(\alpha, o, \sigma, \{\ell \mid x = o' \ ; \ s\ \}, q)\ ob(\gamma, o', \sigma', \mathtt{idle}, \epsilon) : \left\langle \left(\mathtt{0} \fatsemi \mathbb{b}''\right) \& rt\_unsync(\Delta'') \right\rangle_f \& \mathbb{k}_q$

Now we can show that:
$\left\langle \left(\mathtt{0} \fatsemi \mathbb{b}''\right) \& rt\_unsync(\Delta'') \right\rangle_f \& \mathbb{k}_q \ \trianglelefteq\ \left\langle \left((\mathbb{b}_1 \& \cdots \& \mathbb{b}_n) \fatsemi \mathbb{b}''\right) \& rt\_unsync(\Delta'') \right\rangle_f \& \mathbb{k}_q.$

$$\dfrac{\dfrac{\dfrac{\mathtt{0} \trianglelefteq\ \mathbb{b}_1 \& \cdots \& \mathbb{b}_n \qquad \mathbb{b}'' \trianglelefteq\ \mathbb{b}''}{\mathtt{0} \fatsemi \mathbb{b}''\ \trianglelefteq\ (\mathbb{b}_1 \& \cdots \& \mathbb{b}_n) \fatsemi \mathbb{b}''}\ \text{(LS-Seq)} \qquad rt\_unsync(\Delta'') \trianglelefteq\ rt\_unsync(\Delta'')}{\left\langle \left(\mathtt{0} \fatsemi \mathbb{b}''\right) \& rt\_unsync(\Delta'') \right\rangle_f\ \trianglelefteq\ \left\langle \left((\mathbb{b}_1 \& \cdots \& \mathbb{b}_n) \fatsemi \mathbb{b}''\right) \& rt\_unsync(\Delta'') \right\rangle_f}\ \text{(LS-Par)} \qquad \mathbb{k}_q\ \trianglelefteq\ \mathbb{k}_q}{\left\langle \left(\mathtt{0} \fatsemi \mathbb{b}''\right) \& rt\_unsync(\Delta'') \right\rangle_f \& \mathbb{k}_q\ \trianglelefteq\ \left\langle \left((\mathbb{b}_1 \& \cdots \& \mathbb{b}_n) \fatsemi \mathbb{b}''\right) \& rt\_unsync(\Delta'') \right\rangle_f \& \mathbb{k}_q}\ \text{(LS-Par)}$$

Asynchronous call.

$$\frac{\begin{array}{c}[\![e]\!]_{(\sigma+\ell)} = o' \quad [\![\overline{e}]\!]_{(\sigma+\ell)} = \overline{v} \quad f = \text{fresh}(\ ) \quad \text{dom}(\sigma) \cap \text{dom}(\sigma') = \varnothing \\ \text{bind}(o', m, \overline{v}, \text{class}(o')) = \{\ell'' \mid s'\} \\ \sigma'' = \sigma' \cup \text{serialize}(\overline{v}, \sigma) \quad \ell' = [\texttt{destiny} \mapsto f, this \mapsto o'] \cup \ell''\end{array}}{\begin{array}{c}ob(\alpha, o, \sigma, \{\ell \mid x = e.\mathtt{m}(\overline{e})\ ;\ s\}, q)\ ob(\beta, o', \sigma', p', q') \\ \rightarrow ob(\alpha, o, \sigma, \{\ell \mid x = f\ ;\ s\}, q)\ ob(\beta, o', \sigma'', p', q' :: \{\ell' \mid s'\})\ fut(f, \bot)\end{array}}\ (\textsc{Invk-Active})$$

- Let $\Delta \vdash_R ob(\alpha, o, \sigma, \{\ell \mid x = e.\mathtt{m}(\overline{e})\ ;\ s\}, q)\ ob(\beta, o', \sigma', p', q') : \mathbb{b}$
- knowing that:
$$ob(\alpha, o, \sigma, \{\ell \mid x = e.\mathtt{m}(\overline{e})\ ;\ s\}, q)\ ob(\beta, o', \sigma', p', q') \rightarrow ob(\alpha, o, \sigma, \{\ell \mid x = f\ ;\ s\}, q)\ ob(\beta, o', \sigma'', p', q' :: \{\ell' \mid s'\})\ fut(f, \bot)$$
- we must show there exist:
$$\Delta' \vdash_R ob(\alpha, o, \sigma, \{\ell \mid x = f\ ;\ s\}, q)\ ob(\beta, o', \sigma'', p', q' :: \{\ell' \mid s'\})\ fut(f, \bot) : \mathbb{b}'$$
- and $\imath(\mathbb{b}') \trianglelefteq \mathbb{b}$.

We show below that $\Delta \vdash_R ob(\alpha, o, \sigma, \{\ell \mid x = e.\mathtt{m}(\overline{e})\ ;\ s\ \}, q)\ ob(\beta, o', \sigma', p', q') : \mathbb{b}$

$$\dfrac{\dfrac{\dfrac{\Theta_1}{\Delta \vdash_R ob(\alpha, o, \sigma, \{\ell \mid x = e.\mathtt{m}(\overline{e})\ ;\ s\ \}, q) : \mathbb{k}'_p \& \mathbb{k}'_q}\ (\textsc{TR-Object}) \quad \dfrac{\Theta_2}{\Delta \vdash_R ob(\beta, o', \sigma', p', q') : \mathbb{k}''_p \& \mathbb{k}''_q}\ (\textsc{TR-Object})}{\Delta \vdash_R ob(\alpha, o, \sigma, \{\ell \mid x = e.\mathtt{m}(\overline{e})\ ;\ s\ \}, q)\ ob(\beta, o', \sigma', p', q') : (\!|\mathbb{k}' \& \mathbb{k}''|\!)}\ (\textsc{TR-Configuration})}{ob(\alpha, o, \sigma, \{\ell \mid x = e.\mathtt{m}(\overline{e})\ ;\ s\ \}, q)\ ob(\beta, o', \sigma', p', q')}$$

Θ_1

$$\dfrac{\Delta + \widehat{\sigma} \vdash_R \sigma \quad \dfrac{\Theta'_1}{\Delta + \widehat{\sigma} \vdash_R \{\ell \mid x = e.\mathtt{m}(\overline{e})\ ;\ s\ \} : \mathbb{b}_f}\ (\textsc{TR-Process}) \quad \dfrac{\Phi_2}{\Delta + \widehat{\sigma} \vdash_R q : \mathbb{k}'_q}\ \textsc{Hyp}}{\Delta \vdash_R ob(\alpha, o, \sigma, \{\ell \mid x = e.\mathtt{m}(\overline{e})\ ;\ s\ \}, q) : \mathbb{k}'_p \& \mathbb{k}'_q}\ (\textsc{TR-Object})$$

Θ_1'

$$\frac{\Delta + \widehat{\sigma}(o) = [act : \alpha, \overline{x : \mathtt{r}}] \qquad \Delta + \widehat{\sigma} \vdash_R \overline{val : \mathtt{x}} \qquad \Delta_0 = \Delta + \widehat{\sigma}[\text{destiny} \mapsto f, \text{this} \mapsto o, \overline{x \mapsto \mathtt{x}}] \qquad \Theta_1''}{\Delta + \widehat{\sigma} \vdash_R \{\text{destiny} \mapsto f, \text{this} \mapsto o, \overline{x \mapsto \mathtt{x}} \mid x = e.\mathtt{m}(\overline{e}) \ ; \ s \ \} : \langle \mathbb{b} \rangle_f}\ \text{(TR-Process)}$$

Θ_1''

$$\frac{\dfrac{\Theta_1'''}{\Delta_0 \vdash_R x = e.\mathtt{m}(\overline{e}) : \mathbb{b}' \mid \Delta'[x \mapsto f]}\ \text{(TR-Invk)} \qquad \dfrac{\Phi_3}{\Delta' \vdash_R s : \mathbb{b}'' \mid \Delta''}\ \text{Hyp}}{\Delta_0 \vdash_R x = e.\mathtt{m}(\overline{e}) \ ; \ s \ : \mathbb{b}' \fatsemi \mathbb{b}'' \mid \Delta''}\ \text{(TR-Seq)}$$

Θ_1'''

$$\frac{\begin{array}{c}\Delta_0 \vdash_R e : [act{:}\beta, \overline{x{:}\mathtt{r}}], \mathbb{b}_0 \mid \Delta_1 \quad \Delta_0 \vdash_R \mathtt{this} : [act{:}\alpha, \overline{x{:}\mathtt{s}}] \\ class(types(e)) = \mathtt{C} \quad fields(\mathtt{C}) = \overline{T\ x} \quad \Delta_0(\mathtt{C.m}) = \mathtt{r}'(\overline{\mathtt{s}'})\{\mathbb{b}\}\mathtt{r}'' \\ \overline{\alpha}, \mathbb{1}_f \text{ fresh} \quad \overline{\alpha'} = act\_names(\mathtt{r}'') \setminus act\_names(\mathtt{r}', \overline{\mathtt{s}'}) \quad \mathtt{s}'' = \mathtt{r}''[\overline{\alpha}/\overline{\alpha'}][\mathtt{r}, \overline{\mathtt{s}}/\mathtt{r}', \overline{\mathtt{s}'}] \\ \Delta' = \Delta_{n+1}[\mathbb{1}_f \mapsto (\beta \rightsquigarrow \mathtt{s}'', [\alpha \neq \beta]\mathtt{C.m}([act : \beta, \overline{x{:}\mathtt{r}}], \overline{\mathtt{s}}) \to \mathtt{s}'')]\end{array} \qquad \dfrac{\Phi_4}{\left(\Delta_i \vdash_R e_i : \mathtt{s}_i, \mathbb{b}_i \mid \Delta_{i+1}\right)^{i=1\ldots n}}\ \text{Hyp}}{\Delta_0 \vdash_R e.\mathtt{m}(e1, \ldots, e_n) : \mathbb{1}_f, \mathbb{b}_0 \& \ldots \& \mathbb{b}_n \fatsemi [\alpha = \beta]\Big(\Big(\mathtt{C.m}\big([act : \beta, \overline{x{:}\mathtt{r}}], \overline{\mathtt{s}}\big) \to \mathtt{s}''\Big) \& rt\_unsync(\Delta_{n+1})\Big) \mid \Delta'}\ \text{(TR-Invk)}$$

Θ_2

$$\frac{\Delta + \widehat{\sigma}' \vdash_R^{\alpha} \sigma' \qquad \dfrac{\Phi}{\Delta + \widehat{\sigma}' \vdash_R p' : \mathbb{k}_p''}\ \text{Hyp} \qquad \dfrac{\Phi_1}{\Delta + \widehat{\sigma}' \vdash_R q' : \mathbb{k}_q''}\ \text{Hyp}}{\Delta \vdash_R ob(\beta, o', \sigma', p', q') : \mathbb{k}_p'' \& \mathbb{k}_q''}\ \text{(TR-Object)}$$

With the demonstration above we can deduce that:

$$\Delta \vdash_R ob(\alpha, o, \sigma, \{\ell \mid x = e.\mathtt{m}(\overline{e}) \ ; \ s \ \}, q)\ ob(\beta, o', \sigma', p', q') : \Big(\big\langle\big((\mathbb{b}_1 \& \cdots \& \mathbb{b}_n) \fatsemi \mathbb{b}''\big) \& rt\_unsync(\Delta'')\big\rangle_f \& \mathbb{k}_q'\Big) \& (\mathbb{k}_p'' \& \mathbb{k}_q'')$$

Now we can proof that
$\Delta' \vdash_R ob(\alpha, o, \sigma, \{\ell \mid x = f \ ; \ s\}, q)\ ob(\beta, o', \sigma'', p', q' :: \{\ell' \mid s'\})\ fut(f, \bot) : \mathbb{b}'$ with
$\Delta' = \Delta[f \mapsto \Delta(\imath_f)]$

$$\dfrac{\dfrac{\Sigma_1}{\Delta \vdash_R ob(\alpha, o, \sigma, \{\ell \mid x = f \ ; \ s\}, q) : \mathbb{k}'_p \& \mathbb{k}'_q} \quad \dfrac{\Sigma_2}{\Delta \vdash_R ob(\beta, o', \sigma'', p', q' :: \{\ell' \mid s'\}) : \mathbb{k}''_p \& \mathbb{k}''_q} \quad \Delta \vdash_R fut(f, \bot) : \mathtt{0}}{\dfrac{\Delta \vdash_R ob(\alpha, o, \sigma, \{\ell \mid x = f \ ; \ s\}, q)\ ob(\beta, o', \sigma'', p', q' :: \{\ell' \mid s'\})\ fut(f, \bot) \ : (\!|\mathbb{k}' \ \& \ \mathbb{k}'' \& \mathtt{0}|\!)}{ob(\alpha, o, \sigma, \{\ell \mid x = f \ ; \ s\}, q)\ ob(\beta, o', \sigma'', p', q' :: \{\ell' \mid s'\})\ fut(f, \bot)}\ \text{(TR-Configuration)}}$$

Σ_1

$$\dfrac{\Delta + \widehat{\sigma} \vdash_R \sigma \qquad \dfrac{\dfrac{\Sigma'_1}{\Delta + \widehat{\sigma} \vdash_R \{\ell \mid x = f \ ; \ s\} : \langle \mathbb{b} \rangle_f}\ \text{(TR-Process)} \qquad \dfrac{\Phi_2}{\Delta + \widehat{\sigma} \vdash_R q : \mathbb{k}'_q}\ \text{Hyp}}{\Delta + \widehat{\sigma} \vdash_R \{\ell \mid x = f \ ; \ s\} : \langle \mathbb{b} \rangle_f}}{\Delta \vdash_R ob(\alpha, o, \sigma, \{\ell \mid x = f \ ; \ s\}, q) : \mathbb{k}'_p \& \mathbb{k}'_q}\ \text{(TR-Object)}$$

Σ'_1

$$\dfrac{\Delta + \widehat{\sigma}(o') = [act : \alpha, \overline{x : \mathbb{r}}] \qquad \Delta + \widehat{\sigma} \vdash_R \overline{val : \mathbb{x}} \qquad \Delta' = \Delta + \widehat{\sigma}[\text{destiny} \mapsto f, \text{this} \mapsto o, \overline{x \mapsto \mathbb{x}}] \qquad \Sigma''_1}{\Delta + \widehat{\sigma} \vdash_R \{\text{destiny} \mapsto f, \text{this} \mapsto o, \overline{x \mapsto \mathbb{x}} \mid x = f \ ; \ s\} : \langle \mathbb{b} \& rt\_unsync(\Delta'') \rangle_f}\ \text{(TR-Process)}$$

Σ''_1

$$\dfrac{\dfrac{\Delta' \vdash_R x : F}{\Delta' \vdash_R x = f : \mathtt{0} \mid \Delta'[x \mapsto f]}\ \text{(TR-Var-Future)} \qquad \dfrac{\Phi_3}{\Delta' \vdash_R s : \mathbb{b}'' \mid \Delta''}\ \text{Hyp}}{\Delta' \vdash_R x = f \ ; \ s : \mathbb{b}' \fatsemi \mathbb{b}'' \mid \Delta''}\ \text{(TR-Seq)}$$

Σ_2

$$\dfrac{\Delta + \widehat{\sigma}'' \vdash_R \sigma \qquad \dfrac{\Phi}{\Delta + \widehat{\sigma}'' \vdash_R p' : \mathbb{k}''_p}\,\text{Hyp} \qquad \dfrac{\Sigma'_2}{\Delta + \widehat{\sigma}'' \vdash_R q' :: \{\ell' \mid s'\} : \mathbb{k}''_q \& \langle \mathbb{b}' \rangle_f}}{\Delta \vdash_R ob(\beta, o', \sigma'', p', q' :: \{\ell' \mid s'\}) : \mathbb{k}''_p \& \mathbb{k}''_{q1}}\,\text{(TR-Object)}$$

Σ'_2

$$\dfrac{\dfrac{\Phi_1}{\Delta + \widehat{\sigma}'' \vdash_R q' : \mathbb{k}''_q}\,\text{Hyp} \qquad \dfrac{\Sigma''_2}{\Delta + \widehat{\sigma}'' \vdash_R \{\ell' \mid s'\} : \langle \mathbb{b}' \rangle_f}\,\text{(TR-Process)}}{\Delta + \widehat{\sigma}'' \vdash_R q' :: \{\ell' \mid s'\} : \mathbb{k}''_q \& \langle \mathbb{b}' \rangle_f}$$

Σ''_2

$$\dfrac{\Delta + \widehat{\sigma}''(o') = [act : \beta, \overline{x : \mathbb{r}}] \qquad \Delta + \widehat{\sigma}'' \vdash_R \overline{val : \mathbb{x}} \qquad \Delta' = \Delta + \widehat{\sigma}''[\text{destiny} \mapsto f, \text{this} \mapsto o', \overline{x \mapsto \mathbb{x}}] \qquad \Sigma'''_2}{\Delta + \widehat{\sigma}'' \vdash_R \{\text{destiny} \mapsto f, \text{this} \mapsto o', \overline{x \mapsto \mathbb{x}} \mid s\} : \langle \mathbb{b}' \& rt\_unsync(\Delta'') \rangle_f}\,\text{(TR-Process)}$$

Σ'''_2

$$\dfrac{\text{where } \mathbb{b}_s = \mathbb{b}[\overline{\alpha}/\overline{\alpha'}][[act : \alpha, \overline{\mathbb{r}}/\overline{\mathbb{r}'}][\overline{\mathbb{s}}/\overline{\mathbb{s}'}] \text{ and } \mathbb{b} \text{ is the behavioral type of the body } \Delta_0(\mathtt{C.m}) = \mathbb{r}'(\overline{\mathbb{s}'})\{\mathbb{b}\}\mathbb{r}'' \text{ from } \Theta'''_1}{\Delta' \vdash_R s' : \mathbb{b}_s \mid \Delta''}$$

With the demonstration above we can deduce that:

$$\Delta' \vdash_R ob(\alpha, o, \sigma, \{\ell \mid x = f\ ;\ s\}, q)\ ob(\beta, o', \sigma'', p', q' :: \{\ell' \mid s'\})\ fut(f, \bot) : \mathbb{b}'$$

with $\mathbb{b}' = \left(\left\langle \mathbb{b}'' \,\&\, rt\_unsync(\Delta'') \right\rangle_f \,\&\, \mathbb{k}'_q\right) \,\&\, \left(\mathbb{k}''_p \,\&\, \left(\mathbb{k}''_q \,\&\, \left\langle \mathbb{b}[\overline{\alpha}/\overline{\alpha'}][[act : \alpha, \overline{\mathbb{r}}/\overline{\mathbb{r}'}][\overline{\mathbb{s}}/\overline{\mathbb{s}'}] \,\&\, rt\_unsync(\Delta'') \right\rangle_f\right)\right)$

Now we can show that:

$\left(\left\langle \mathbb{b}'' \,\&\, rt\_unsync(\Delta'') \right\rangle_f \,\&\, \mathbb{k}'_q\right) \,\&\, \left(\mathbb{k}''_p \,\&\, \left(\mathbb{k}''_q \,\&\, \left\langle \mathbb{b}[\overline{\alpha}/\overline{\alpha'}][[act : \alpha, \overline{\mathbb{r}}/\overline{\mathbb{r}'}][\overline{\mathbb{s}}/\overline{\mathbb{s}'}] \,\&\, rt\_unsync(\Delta'') \right\rangle_f\right)\right) \trianglelefteq$

$\left(\left\langle ((\mathbb{b}_1 \& \cdots \& \mathbb{b}_n)\ ;\ \mathbb{b}'') \& rt\_unsync(\Delta'') \right\rangle_f \& \mathbb{k}'_q\right) \& (\mathbb{k}''_p \& \mathbb{k}''_q)$

Synchronous call.

$$\frac{\begin{array}{c}o \in \mathrm{dom}(\sigma) \quad [\![e]\!]_{(\sigma+\ell)} = o' \quad [\![\overline{e}]\!]_{(\sigma+\ell)} = \overline{v} \quad f = \mathrm{fresh}(\ )\\ \mathrm{bind}(o', m, \overline{v}, \mathrm{class}(o')) = \{\ell'' \mid s'\} \quad \ell' = [\mathtt{destiny} \mapsto f, this \mapsto o'] \cup \ell''\end{array}}{ob(\alpha, o, \sigma, \{\ell \mid x = e.\mathtt{m}(\overline{e}) \ ; \ s\ \}, q) \to ob(\alpha, o, \sigma, \{\ell' \mid s'\}, \{\ell \mid x = f \ ; \ s\ \} :: q)}\ (\textsc{Invk-Passive})$$

- Let $\Delta \vdash_R ob(\alpha, o, \sigma, \{\ell \mid x = e.\mathtt{m}(\overline{e}) \ ; \ s\}, q) : \mathbb{b}$
- knowing that:
$$ob(\alpha, o, \sigma, \{\ell \mid x = e.\mathtt{m}(\overline{e}) \ ; \ s\ \}, q) \to ob(\alpha, o, \sigma, \{\ell' \mid s'\}, \{\ell \mid x = f \ ; \ s\} :: q)$$
- we must show there exist:
$$\Delta' \vdash_R ob(\alpha, o, \sigma, \{\ell' \mid s'\}, \{\ell \mid x = f \ ; \ s\} :: q) : \mathbb{b}'$$
- and $\imath(\mathbb{b}') \trianglelefteq \mathbb{b}$.

We show below that $\Delta \vdash_R ob(\alpha, o, \sigma, \{\ell \mid x = e.\mathtt{m}(\overline{e}) \ ; \ s\ \}, q) : \mathbb{b}$

$$\dfrac{\dfrac{\dfrac{\Delta + \widehat{\sigma} \vdash_R \sigma \qquad \dfrac{\Theta_1'}{\Delta + \widehat{\sigma} \vdash_R \{\ell \mid x = e.\mathtt{m}(\overline{e}) \ ; \ s\ \} : \langle \mathbb{b} \rangle_f}\ (\textsc{TR-Process}) \qquad \dfrac{\Phi_2}{\Delta + \widehat{\sigma} \vdash_R q : \mathbb{k}_q'}\ \textsc{Hyp}}{\Delta \vdash_R ob(\alpha, o, \sigma, \{\ell \mid x = e.\mathtt{m}(\overline{e}) \ ; \ s\ \}, q) : (\!|\mathbb{k}_p' \& \mathbb{k}_q'|\!)}\ (\textsc{TR-Object})}{ob(\alpha, o, \sigma, \{\ell \mid x = e.\mathtt{m}(\overline{e}) \ ; \ s\ \}, q)}\ (\textsc{TR-Configuration})$$

Θ_1

$$\dfrac{\Delta + \widehat{\sigma}(o) = [act : \alpha, \overline{x : \mathbb{r}}] \qquad \Delta + \widehat{\sigma} \vdash_R \overline{val : \mathbb{x}} \qquad \Delta_0 = \Delta + \widehat{\sigma}[\mathrm{destiny} \mapsto f, \mathrm{this} \mapsto o, \overline{x \mapsto \mathbb{x}}] \qquad \Theta_1''}{\Delta + \widehat{\sigma} \vdash_R \{\mathrm{destiny} \mapsto f, \mathrm{this} \mapsto o, \overline{x \mapsto \mathbb{x}} \mid x = e.\mathtt{m}(\overline{e}) \ ; \ s\ \} : \langle \mathbb{b} \& rt\_unsync(\Delta'') \rangle_f}\ (\textsc{TR-Process})$$

Θ_1'

$$\dfrac{\dfrac{\Theta_1''}{\Delta_0 \vdash_R x = e.\mathtt{m}(\overline{e}) : \mathbb{b}' \mid \Delta'[x \mapsto f]}\,(\text{TR-Invk}) \qquad \dfrac{\Phi_3}{\Delta' \vdash_R s : \mathbb{b}'' \mid \Delta''}\,\text{Hyp}}{\Delta_0 \vdash_R x = e.\mathtt{m}(\overline{e})\ ;\ s\ : \mathbb{b}' \fatsemi \mathbb{b}'' \mid \Delta''}\,(\text{TR-Seq})$$

Θ_1''

$$\dfrac{\begin{array}{c}\Delta_0 \vdash_R e : [act{:}\beta, \overline{x{:}\mathtt{r}}], \mathbb{b}_0 \mid \Delta_1 \quad \Delta_0 \vdash_R \mathtt{this} : [act{:}\alpha, \overline{x{:}\mathtt{s}}] \\ \mathit{class}(\mathit{types}(e)) = \mathtt{C} \quad \mathit{fields}(\mathtt{C}) = \overline{T\ x} \quad \Delta_0(\mathtt{C.m}) = \mathtt{r}'(\overline{\mathtt{s}'})\{\mathbb{b}\}\mathtt{r}'' \\ \overline{\alpha}, \mathtt{1}_f \text{ fresh} \quad \overline{\alpha'} = \mathit{act\_names}(\mathtt{r}'') \setminus \mathit{act\_names}(\mathtt{r}', \overline{\mathtt{s}'}) \quad \mathtt{s}'' = \mathtt{r}''[\overline{\alpha}/\overline{\alpha'}][\mathtt{r}, \overline{\mathtt{s}}/\mathtt{r}', \overline{\mathtt{s}'}] \\ \Delta' = \Delta_{n+1}[\mathtt{1}_f \mapsto (\beta \rightsquigarrow \mathtt{s}'', [\alpha \neq \beta]\mathtt{C.m}([act : \beta, \overline{x{:}\mathtt{r}}], \overline{\mathtt{s}}) \to \mathtt{s}'')]\end{array}}{\Delta_0 \vdash_R e.\mathtt{m}(e1, \ldots, e_n) : \mathtt{1}_f, \mathbb{b}_0 \& \ldots \& \mathbb{b}_n \fatsemi [\alpha = \beta](\mathtt{C.m}([act : \beta, \overline{x{:}\mathtt{r}}], \overline{\mathtt{s}}) \to \mathtt{s}'') \& \mathit{rt\_unsync}(\Delta_{n+1})) \mid \Delta'}\,(\text{TR-Invk}) \qquad \dfrac{\Phi_4}{\left(\Delta_i \vdash_R e_i : \mathtt{s}_i, \mathbb{b}_i \mid \Delta_{i+1}\right)^{i=1\ldots n}}\,\text{Hyp}$$

With the demonstration above we can deduce that:
$\Delta \vdash_R ob(\alpha, o, \sigma, \{\ell \mid x = e.\mathtt{m}(\overline{e})\ ;\ s\ \}, q) : \mathbb{b}$

with $\mathbb{b} =$

$$\left(\left\langle\left(\left((\mathbb{b}_0 \& \ldots \& \mathbb{b}_n)\ \fatsemi\ [\alpha = \beta]\big(\big(\mathtt{C.m}\big([act : \beta, \overline{x{:}\mathtt{r}}], \overline{\mathtt{s}}\big) \to \mathtt{s}''\big)\ \& \mathit{rt\_unsync}(\Delta_{n+1})\big)\right) \fatsemi \mathbb{b}''\right) \& \mathit{rt\_unsync}(\Delta'')\right\rangle_f \& \mathbb{k}_q'\right)$$

Now we can proof that $\Delta' \vdash_R ob(\alpha, o, \sigma, \{\ell' \mid s'\}, \{\ell \mid x = f \ ; \ s\} :: q) : \mathbb{b}'$

$$\dfrac{\Delta' + \widehat{\sigma} \vdash_R \sigma \qquad \dfrac{\Sigma_1'}{\Delta' + \widehat{\sigma} \vdash_R \{\ell' \mid s'\} : \langle \mathbb{b} \rangle_f} \text{(TR-Process)} \qquad \dfrac{\dfrac{\Theta_1}{\Delta' + \widehat{\sigma} \vdash_R \{\ell \mid x = f \ ; \ s\} : \langle \mathbb{b}' \rangle_f} \qquad \dfrac{\Phi_2}{\Delta' + \widehat{\sigma} \vdash_R q : \mathbb{k}_q'} \text{Hyp}}{\Delta' + \widehat{\sigma} \vdash_R \{\ell \mid x = f \ ; \ s\} :: q : \langle \mathbb{b}' \rangle_f \& \mathbb{k}_q'}}{\dfrac{\Delta' \vdash_R ob(\alpha, o, \sigma, \{\ell' \mid s'\}, \{\ell \mid x = f \ ; \ s\} :: q) : (\!| \mathbb{k}_p' \& \mathbb{k}_{q_1}' |\!)}{ob(\alpha, o, \sigma, \{\ell' \mid s'\}, \{\ell \mid x = f \ ; \ s\} :: q)} \text{(TR-Configuration)}} \text{(TR-Object)}$$

Σ_2'

$$\dfrac{\Delta + \widehat{\sigma}''(o') = [act : \beta, \overline{x : \mathbb{r}}] \qquad \Delta + \widehat{\sigma}'' \vdash_R \overline{val : \mathbb{x}} \qquad \Delta' = \Delta + \widehat{\sigma}''[\text{destiny} \mapsto f, \text{this} \mapsto o', \overline{x \mapsto \mathbb{x}}] \qquad \Sigma_2''}{\Delta + \widehat{\sigma}'' \vdash_R \{\text{destiny} \mapsto f, \text{this} \mapsto o', \overline{x \mapsto \mathbb{x}} \mid s \ \} : \langle \mathbb{b} \& rt\_unsync(\Delta'') \rangle_f} \text{(TR-Process)}$$

Σ_2''

$$\dfrac{\text{where } \mathbb{b}_s = \mathbb{b}[\overline{\alpha}/\overline{\alpha'}][[act : \alpha, \overline{\mathbb{r}}/\overline{\mathbb{r}'}][\overline{\mathbb{s}}/\overline{\mathbb{s}'}] \text{ and } \mathbb{b} \text{ is the behavioral type of the body } \Delta_0(\mathtt{C.m}) = \mathbb{r}'(\overline{\mathbb{s}'})\{\mathbb{b}\}\mathbb{r}'' \text{ from } \Theta_1''}{\Delta' \vdash_R s' : \mathbb{b}_s \mid \Delta''}$$

With the demonstration above we can deduce that:
$\Delta' \vdash_R ob(\alpha, o, \sigma, \{\ell \mid x = f \ ; \ s \ \}, q :: \{\ell' \mid s'\}) : \mathbb{b}'$

with $\mathbb{b}' = \Big(\big\langle \mathbb{b}'' \ \& \ rt\_unsync(\Delta'') \big\rangle_f \ \& \ \Big(\mathbb{k}_q' \ \& \ \big\langle \mathbb{b}[\overline{\alpha}/\overline{\alpha'}][[act : \alpha, \overline{\mathbb{r}}/\overline{\mathbb{r}'}][\overline{\mathbb{s}}/\overline{\mathbb{s}'}] \ \& \ rt\_unsync(\Delta'') \big\rangle_f \Big)\Big)$

Now we can show that:
$\Big(\big\langle \mathbb{b}'' \ \& \ rt\_unsync(\Delta'') \big\rangle_f \ \& \ \Big(\mathbb{k}_q' \ \& \ \big\langle \mathbb{b}[\overline{\alpha}/\overline{\alpha'}][[act : \alpha, \overline{\mathbb{r}}/\overline{\mathbb{r}'}][\overline{\mathbb{s}}/\overline{\mathbb{s}'}] \ \& \ rt\_unsync(\Delta'') \big\rangle_f \Big)\Big) \trianglelefteq$
$\Big(\Big\langle \Big(\big((\mathbb{b}_0 \& \ldots \& \mathbb{b}_n) \fatsemi [\alpha = \beta] \big(\big(\mathtt{C.m} \big([act : \beta, \overline{x : \mathbb{r}}], \overline{\mathbb{s}} \big) \to \mathbb{s}'' \big) \ \& rt\_unsync(\Delta_{n+1}) \big) \big) \fatsemi \mathbb{b}'' \Big) \& rt\_unsync(\Delta'') \Big\rangle_f \ \& \mathbb{k}_q' \Big)$

Return from a Synchronous call.

$$\frac{v = [\![e]\!]_{(\sigma+\ell)} \quad f = \ell(\texttt{destiny})}{ob(\alpha, o, \sigma, \{\ell \mid \texttt{return}\ e\}, \{\ell' \mid x = f\ ;\ s\ \} :: q) \rightarrow ob(\alpha, o, \sigma, \{\ell' \mid x = v\ ;\ s\ \}, q)}\ (\textsc{ReturnLocal})$$

- Let $\Delta \vdash_R ob(\alpha, o, \sigma, \{\ell \mid \texttt{return}\ e\}, \{\ell' \mid x = f\ ;\ s\ \} :: q) : \mathbb{b}$
- knowing that:
  $$ob(\alpha, o, \sigma, \{\ell \mid \texttt{return}\ e\}, \{\ell' \mid x = f\ ;\ s\ \} :: q) \rightarrow ob(\alpha, o, \sigma, \{\ell' \mid x = v\ ;\ s\ \}, q)$$
- we must show there exist:
  $$\Delta' \vdash_R ob(\alpha, o, \sigma, \{\ell' \mid x = v\ ;\ s\ \}, q) : \mathbb{b}'$$
- and $\imath(\mathbb{b}') \trianglelefteq \mathbb{b}$.

We show below that $\Delta \vdash_R ob(\alpha, o, \sigma, \{\ell \mid \texttt{return}\ e\}, \{\ell' \mid x = f\ ;\ s\ \} :: q) : \mathbb{b}$

$$\dfrac{\dfrac{\Delta + \widehat{\sigma} \vdash_R \sigma \qquad \dfrac{\Theta}{\Delta + \widehat{\sigma} \vdash_R \{\ell \mid \texttt{return}\ e\} : \langle \mathbb{b} \rangle_f}\ (\textsc{TR-Process}) \qquad \dfrac{\dfrac{\Theta'}{\Delta + \widehat{\sigma} \vdash_R \{\ell' \mid x = f\ ;\ s\ \} : \langle \mathbb{b}' \rangle_f} \qquad \dfrac{\Phi}{\Delta + \widehat{\sigma} \vdash_R q : \mathbb{k}'_q}\ \text{Hyp}}{\Delta + \widehat{\sigma} \vdash_R \{\ell' \mid x = f\ ;\ s\ \} :: q : \langle \mathbb{b}' \rangle_f \& \mathbb{k}'_q}}{\Delta \vdash_R ob(\alpha, o, \sigma, \{\ell \mid \texttt{return}\ e\}, \{\ell' \mid x = f\ ;\ s\ \} :: q) : (\!|\mathbb{k}_p \& \mathbb{k}_q|\!)}\ (\textsc{TR-Object})}{ob(\alpha, o, \sigma, \{\ell \mid \texttt{return}\ e\}, \{\ell' \mid x = f\ ;\ s\ \} :: q)}\ (\textsc{TR-Configuration})$$

Θ

$$\dfrac{\Delta + \widehat{\sigma}(o) = [act : \alpha, \overline{x : \mathbb{r}}] \qquad \Delta + \widehat{\sigma} \vdash_R \overline{val : \mathbb{x}} \qquad \dfrac{\Delta_1 \vdash_R e : \mathbb{r} \quad \Delta_1(destiny) = f \quad \Delta_1(f) = (c \rightsquigarrow \mathbb{r}, \mathbb{b})}{\Delta_1 \vdash_R \texttt{return}\ e : \texttt{0} \mid \Delta_1}}{\Delta + \widehat{\sigma} \vdash_R \{\text{destiny} \mapsto f, \text{this} \mapsto o', \overline{x \mapsto \mathbb{x}} \mid \texttt{return}\ e\} : \langle \mathbb{b} \& rt\_unsync(\Delta_1) \rangle_f}\ (\textsc{TR-Process})$$

$\Delta_1 = \Delta + \widehat{\sigma}[\text{destiny} \mapsto f, \text{this} \mapsto o', \overline{x \mapsto \mathbb{x}}]$

Θ'

$$\dfrac{\dfrac{\Delta + \widehat{\sigma}(o) = [act : \alpha, \overline{x : \mathbb{r}}] \qquad \Delta + \widehat{\sigma} \vdash_R \overline{val : \mathbb{x}} \qquad \Delta_2 = \Delta + \widehat{\sigma}[\text{destiny} \mapsto f, \text{this} \mapsto o', \overline{x \mapsto \mathbb{x}}]}{\Delta + \widehat{\sigma} \vdash_R \{\text{destiny} \mapsto f, \text{this} \mapsto o, \overline{x \mapsto \mathbb{x}} \mid x = f \ ; \ s \ \} : \langle \mathbb{b}' \& rt\_unsync(\Delta_1) \rangle_f} \text{(TR-Process)} \qquad \dfrac{\dfrac{\Delta_2 \vdash_R x = f : \mathtt{0} \mid \Delta_2 \qquad \dfrac{\Phi_1}{\Delta_2 \vdash_R s : \mathbb{b}'' \mid \Delta''}}{\Delta_2 \vdash_R x = f \ ; \ s : \mathbb{b}' \fatsemi \mathbb{b}'' \mid \Delta''}}{}$$

With the demonstration above we can deduce that:
$\Delta \vdash_R ob(\alpha, o, \sigma, \{\ell \mid \mathtt{return}\ e\}, \{\ell' \mid x = f \ ; \ s \ \} :: q) : \left(\left\langle rt\_unsync(\Delta_1) \right\rangle_f \ \& \ \left(\langle \mathbb{b}'' \& rt\_unsync(\Delta'') \rangle_f \& \mathbb{k}'_q\right)\right)$

Now we can proof that $\Delta' \vdash_R ob(\alpha, o, \sigma, \{\ell' \mid x = v \ ; \ s \ \}, q) : \mathbb{b}'$

$$\dfrac{\dfrac{\dfrac{\Delta' + \widehat{\sigma}' \vdash_R \sigma' \qquad \dfrac{\Sigma}{\Delta' + \widehat{\sigma}' \vdash_R \{\ell' \mid x = v \ ; \ s \ \} : \langle \mathbb{b} \rangle_f} \text{(TR-Process)} \qquad \dfrac{\Phi_1}{\Delta' + \widehat{\sigma}' \vdash_R q : \mathbb{k}_q} \text{Hyp}}{\Delta' \vdash_R ob(\alpha, o, \sigma, \{\ell' \mid x = v \ ; \ s \ \}, q) : \mathbb{k}_p \& \mathbb{k}_q} \text{(TR-Object)}}{ob(\alpha, o, \sigma, \{\ell' \mid x = v \ ; \ s \ \}, q)}$$

Σ

$$\dfrac{\Delta' + \widehat{\sigma}(o) = [act : \alpha, \overline{x : \mathbb{r}}] \qquad \Delta' + \widehat{\sigma} \vdash_R \overline{val : \mathbb{x}} \qquad \Delta_1 = \Delta' + \widehat{\sigma}[\text{destiny} \mapsto f, \text{this} \mapsto o', \overline{x \mapsto \mathbb{x}}] \qquad \Sigma'}{\Delta' + \widehat{\sigma} \vdash_R \{\text{destiny} \mapsto f, \text{this} \mapsto o', \overline{x \mapsto \mathbb{x}} \mid x = v \ ; \ s \ \} : \langle \mathbb{b} \& rt\_unsync(\Delta'') \rangle_f} \text{(TR-Process)}$$

Σ'

$$\dfrac{\dfrac{\dfrac{\Delta_1 \vdash_R x : \mathbb{r} \qquad \Delta_1 \vdash_R v : \mathbb{r}}{\Delta_1 \vdash_R x = v : \mathtt{0} \mid \Delta_1[x \mapsto \mathbb{r}]} \text{(TR-Var-Record)} \qquad \dfrac{\Phi}{\Delta_1 \vdash_R s : \mathbb{b}'' \mid \Delta''} \text{Hyp}}{\Delta_1 \vdash_R x = v \ ; \ s : \mathbb{b}' \fatsemi \mathbb{b}'' \mid \Delta''} \text{(TR-Seq)}}{}$$

With the demonstration above we can deduce that:
$\Delta' \vdash_R ob(\alpha, o, \sigma, \{\ell' \mid x = v \ ; \ s \ \}, q) : \left(\left\langle \mathbb{b}'' \& rt\_unsync(\Delta_1) \right\rangle_f \ \& \ \mathbb{k}'_q\right)$

Now we can show that:
$\left(\left\langle \mathbb{b}'' \& rt\_unsync(\Delta_1) \right\rangle_f \ \& \ \mathbb{k}'_q\right) \ \trianglelefteq \ \left(\left\langle rt\_unsync(\Delta_1) \right\rangle_f \ \& \ \left(\langle \mathbb{b}'' \& rt\_unsync(\Delta'') \rangle_f \& \mathbb{k}'_q\right)\right)$

$$\dfrac{\mathsf{0} \trianglelefteq \left\langle rt\_unsync(\Delta_1)\right\rangle_f \qquad \dfrac{\dfrac{\mathbb{b}'' \trianglelefteq \mathbb{b}'' \qquad rt\_unsync(\Delta_1) \trianglelefteq rt\_unsync(\Delta'')}{\left\langle \mathbb{b}''\&rt\_unsync(\Delta_1)\right\rangle_f \trianglelefteq \left\langle \mathbb{b}''\&rt\_unsync(\Delta'')\right\rangle_f} \text{(LS-Par)} \qquad \mathbb{k}'_q \trianglelefteq \mathbb{k}'_q}{\left\langle \mathbb{b}''\&rt\_unsync(\Delta_1)\right\rangle_f \ \&\ \mathbb{k}'_q \ \trianglelefteq\ \left\langle \mathbb{b}''\&rt\_unsync(\Delta'')\right\rangle_f \ \&\ \mathbb{k}'_q} \text{(LS-Par)}}{\left(\left\langle \mathbb{b}''\&rt\_unsync(\Delta_1)\right\rangle_f \ \&\ \mathbb{k}'_q\right) \ \trianglelefteq\ \left(\left\langle rt\_unsync(\Delta_1)\right\rangle_f \ \&\ \left(\langle \mathbb{b}''\&rt\_unsync(\Delta'')\rangle_f \&\mathbb{k}'_q\right)\right)} \text{(LS-Par)}$$

Return from an Asynchronous call.

$$\frac{v = [\![e]\!]_{(\sigma+\ell)} \quad f = \ell(\texttt{destiny})}{ob(\alpha, o, \sigma, \{\ell \mid \texttt{return}\ e\}, q)\ fut(f, \bot) \rightarrow ob(\alpha, o, \sigma, \texttt{idle}, q)\ fut(f, (v, \text{serialize}(v, \sigma)))}\ (\textsc{Return})$$

- Let $\Delta \vdash_R ob(\alpha, o, \sigma, \{\ell \mid \texttt{return}\ e\}, q)\ fut(f, \bot) : \mathbb{b}$
- knowing that:
  $ob(\alpha, o, \sigma, \{\ell \mid \texttt{return}\ e\}, q)\ fut(f, \bot) \rightarrow ob(\alpha, o, \sigma, \texttt{idle}, q)\ fut(f, (v, \text{serialize}(v, \sigma)))$
- we must show there exist:
  $\Delta' \vdash_R ob(\alpha, o, \sigma, \texttt{idle}, q)\ fut(f, (v, \text{serialize}(v, \sigma))) : \mathbb{b}'$
- and $\imath(\mathbb{b}') \trianglelefteq \mathbb{b}$.

We show below that $\Delta \vdash_R ob(\alpha, o, \sigma, \{\ell \mid \texttt{return}\ e\}, q)\ fut(f, \bot) : \mathbb{b}$

$$\dfrac{\dfrac{\Delta + \widehat{\sigma} \vdash_R \sigma \qquad \overset{\Theta}{\Delta + \widehat{\sigma} \vdash_R \{\ell \mid \texttt{return}\ e\} : \langle \mathbb{b} \rangle_f}\ (\textsc{TR-Process}) \qquad \overset{\Phi}{\Delta + \widehat{\sigma} \vdash_R q : \mathbb{k}_q}\ \textsc{Hyp}}{\Delta \vdash_R ob(\alpha, o, \sigma, \{\ell \mid \texttt{return}\ e\}, q) : \mathbb{k}_p \& \mathbb{k}_q}\ (\textsc{TR-Object}) \qquad \Delta \vdash_R fut(f, \bot) : \mathtt{0}}{\dfrac{\Delta \vdash_R ob(\alpha, o, \sigma, \{\ell \mid \texttt{return}\ e\}, q)\ fut(f, \bot) : (\!|\mathbb{k}' \& \mathtt{0}|\!)}{ob(\alpha, o, \sigma, \{\ell \mid [\texttt{return}\ e\}, q)\ fut(f, \bot)}}\ (\textsc{TR-Configuration})$$

Θ

$$\dfrac{\Delta + \widehat{\sigma}(o) = [act : \alpha, \overline{x : \mathbb{r}}] \qquad \Delta + \widehat{\sigma} \vdash_R \overline{val : \mathbb{x}} \qquad \dfrac{\Delta_1 \vdash_R e : \mathbb{r} \quad \Delta_1(destiny) = f \quad \Delta_1(f) = (c \rightsquigarrow \mathbb{r}, \mathbb{b})}{\Delta_1 \vdash_R \texttt{return}\ e : \mathtt{0} \mid \Delta_1}}{\Delta_1 = \Delta_\sigma + \Delta[\text{destiny} \mapsto f, \text{this} \mapsto o', \overline{x \mapsto \mathbb{x}}]}\ (\textsc{TR-Process})$$

$$\Delta + \widehat{\sigma} \vdash_R \{\text{destiny} \mapsto f, \text{this} \mapsto o', \overline{x \mapsto \mathbb{x}} \mid \texttt{return}\ e\} : \langle \mathbb{b} \& rt\_unsync(\Delta_1) \rangle_f$$

With the demonstration above we can deduce that:
$\Delta \vdash_R ob(\alpha, o, \sigma, \{\ell \mid \texttt{return}\ e\}, q)\ fut(f, \bot) : \left(\langle rt\_unsync(\Delta_1) \rangle_f\ \&\ \mathbb{k}_q\right)$

Now we can proof that $\Delta' \vdash_R ob(\alpha, o, \sigma, \texttt{idle}, q)\ fut(f, (v, \text{serialize}(v, \sigma))) : \mathbb{b}'$ with $\Delta' = \Delta$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\Phi}\,\text{Hyp}}{\Delta + \widehat{\sigma} \vdash_R \sigma \quad \Delta + \widehat{\sigma} \vdash_R \texttt{idle} : \texttt{0} \quad \Delta + \widehat{\sigma} \vdash_R q : \mathbb{k}_q}\,(\text{TR-Object})}{\Delta \vdash_R ob(\alpha, o, \sigma, \texttt{idle}, q) : \mathbb{k}_p \& \mathbb{k}_q \qquad \Delta \vdash_R fut(f, (v, \text{serialize}(v, \sigma))) : \texttt{0}}}{\Delta \vdash_R ob(\alpha, o, \sigma, \texttt{idle}, q)\ fut(f, (v, \text{serialize}(v, \sigma))) : (\!|\mathbb{k} \& \texttt{0}|\!)}}{ob(\alpha, o, \sigma, \texttt{idle}, q)\ fut(f, (v, \text{serialize}(v, \sigma)))}$$

With the demonstration above we can deduce that:
$\Delta \vdash_R ob(\alpha, o, \sigma, \texttt{idle}, q)\ fut(f, (v, \text{serialize}(v, \sigma))) : \mathbb{k}_q$

Now we can show that: $\mathbb{k}_q \trianglelefteq \left(\left\langle rt\_unsync(\Delta_1)\right\rangle_f \,\&\, \mathbb{k}_q\right)$

$$\dfrac{\texttt{0} \trianglelefteq \left\langle rt\_unsync(\Delta_1)\right\rangle_f \qquad \mathbb{k}_q \trianglelefteq \mathbb{k}_q}{\mathbb{k}_q \trianglelefteq \left(\left\langle \mathbb{b}'' \& rt\_unsync(\Delta_1)\right\rangle_f \,\&\, \mathbb{k}'_q\right)}\,(\text{LS-Par})$$

Get Local.

$$\frac{\begin{array}{c}\ell(x) = f \quad \ell' = \ell[x \mapsto v] \\ \sigma'' = \sigma \cup \sigma' \quad \mathrm{dom}(\sigma) \cap \mathrm{dom}(\sigma') = \varnothing\end{array}}{\begin{array}{c}ob(\alpha, o, \sigma, \{\ell \mid s\}, q)\ fut(f, (v, \sigma')) \\ \rightarrow ob(\alpha, o, \sigma'', \{\ell' \mid s\}, q)\ fut(f, (v, \sigma'))\end{array}}\ (\textsc{GetLocal})$$

- Let $\Delta \vdash_R ob(\alpha, o, \sigma, \{\ell \mid s\}, q)\ fut(f, (v, \sigma')) : \mathbb{b}$
- knowing that:
$$ob(\alpha, o, \sigma, \{\ell \mid s\}, q)\ fut(f, (v, \sigma')) \rightarrow ob(\alpha, o, \sigma'', \{\ell' \mid s\}, q)\ fut(f, (v, \sigma'))$$
- we must show there exist:
$$\Delta' \vdash_R ob(\alpha, o, \sigma'', \{\ell' \mid s\}, q)\ fut(f, (v, \sigma')) : \mathbb{b}'$$
- and $\imath(\mathbb{b}') \trianglelefteq \mathbb{b}$.

We show below that $\Delta \vdash_R ob(\alpha, o, \sigma, \{\ell \mid s\}, q)\ fut(f, (v, \sigma')) : \mathbb{b}$

$$\dfrac{\dfrac{\dfrac{\Delta + \widehat{\sigma} \vdash_R \sigma \qquad \overset{\Theta}{\overbrace{\Delta + \widehat{\sigma} \vdash_R \{\ell \mid s\} : \langle \mathbb{b} \rangle_f}}\ (\textsc{TR-Process}) \qquad \overset{\Phi}{\overbrace{\Delta + \widehat{\sigma} \vdash_R q : \mathbb{k}_q}}\ \textsc{Hyp}}{\Delta \vdash_R ob(\alpha, o, \sigma, \{\ell \mid s\}, q) : \mathbb{k}_p \& \mathbb{k}_q}\ (\textsc{TR-Object}) \qquad \Delta \vdash_R fut(f, (v, \sigma')) : \mathtt{0}}{\Delta \vdash_R ob(\alpha, o, \sigma, \{\ell \mid s\}, q)\ fut(f, (v, \sigma')) : (\!|\mathbb{k}' \& \mathtt{0}|\!)}}{ob(\alpha, o, \sigma, \{\ell \mid s\}, q)\ fut(f, (v, \sigma'))}$$

Θ

$$\dfrac{\begin{array}{c}\Delta + \widehat{\sigma}(o) = [act : \alpha, \overline{x : \mathbb{r}}] \qquad \Delta + \widehat{\sigma} \vdash_R \overline{val : \mathbb{x}} \qquad \overset{\Phi_1}{\overbrace{\qquad\qquad}} \\ \Delta_1 = \Delta_\sigma + \Delta[\mathrm{destiny} \mapsto f, \mathrm{this} \mapsto o', \overline{x \mapsto \mathbb{x}}] \qquad \Delta_1 \vdash_R s : \mathbb{b} \mid \Delta_2\end{array}}{\Delta + \widehat{\sigma} \vdash_R \{\mathrm{destiny} \mapsto f, \mathrm{this} \mapsto o', \overline{x \mapsto \mathbb{x}} \mid s\} : \langle \mathbb{b} \& rt\_unsync(\Delta_2) \rangle_f}\ (\textsc{TR-Process})$$

With the demonstration above we can deduce that:
$\Delta \vdash_R ob(\alpha, o, \sigma'', \{\ell' \mid s\}, q)\ fut(f, (v, \sigma')) : \langle \mathbb{b} \& rt\_unsync(\Delta_2) \rangle_f \& \mathbb{k}_q$

Now we can proof that $\Delta' \vdash_R ob(\alpha, o, \sigma, \texttt{idle}, q)\ fut(f, (v, \text{serialize}(v, \sigma))) : \mathbb{b}'$ with $\Delta = \Delta'$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\Delta + \widehat{\sigma}'' \vdash_R \sigma''}\quad \dfrac{\dfrac{\Sigma}{\Delta + \widehat{\sigma}'' \vdash_R \{\ell' \mid s\} : \langle \mathbb{b} \rangle_f}\,\text{(TR-Process)} \quad \dfrac{\Phi}{\Delta + \widehat{\sigma}'' \vdash_R q : \mathbb{k}_q}\,\text{Hyp}}{\Delta \vdash_R ob(\alpha, o, \sigma'', \{\ell' \mid s\}, q) : \mathbb{k}_p \& \mathbb{k}_q}\,\text{(TR-Object)}}{\Delta \vdash_R ob(\alpha, o, \sigma'', \{\ell' \mid s\}, q)\ fut(f, (v, \sigma')) : (\!|\mathbb{k}' \& \texttt{0}|\!)}}{ob(\alpha, o, \sigma'', \{\ell' \mid s\}, q)\ fut(f, (v, \sigma'))}$$

Σ

$$\dfrac{\begin{array}{c}\Delta + \widehat{\sigma}''(o) = [act : \alpha, \overline{x : \mathbb{r}}] \qquad \Delta + \widehat{\sigma}'' \vdash_R \overline{val : \mathbb{x}} \qquad \dfrac{\Phi_1}{\Delta_1 \vdash_R s : \mathbb{b} \mid \Delta_2} \\ \Delta_1 = \Delta + \widehat{\sigma}''[\text{destiny} \mapsto f, \text{this} \mapsto o', \overline{x \mapsto \mathbb{x}}]\end{array}}{\Delta + \widehat{\sigma}'' \vdash_R \{\text{destiny} \mapsto f, \text{this} \mapsto o', \overline{x \mapsto \mathbb{x}} \mid s\} : \langle \mathbb{b} \& rt\_unsync(\Delta_2) \rangle_f}\,\text{(TR-Process)}$$

With the demonstration above we can deduce that:
$\Delta \vdash_R ob(\alpha, o, \sigma, \texttt{idle}, q)\ fut(f, (v, \text{serialize}(v, \sigma))) : \langle \mathbb{b} \& rt\_unsync(\Delta_2) \rangle_f \& \mathbb{k}_q$

Now we can show that: $\langle \mathbb{b} \& rt\_unsync(\Delta_2) \rangle_f \& \mathbb{k}_q \trianglelefteq \langle \mathbb{b} \& rt\_unsync(\Delta_2) \rangle_f \& \mathbb{k}_q$

Bibliography

[1] N. Kobayashi, "A new type system for deadlock-free processes," in *CONCUR*, ser. LNCS, vol. 4137. Springer, 2006, pp. 233–247.

[2] K. Suenaga and N. Kobayashi, "Type-based analysis of deadlock for a concurrent calculus with interrupts," in *ESOP*, ser. LNCS. Springer, 2007, vol. 4421, pp. 490–504.

[3] K. Suenaga, "Type-based deadlock-freedom verification for non-block-structured lock primitives and mutable references," in *APLAS*, ser. LNCS. Springer, 2008, vol. 5356, pp. 155–170.

[4] V. T. Vasconcelos, F. Martins, and T. Cogumbreiro, "Type inference for deadlock detection in a multithreaded polymorphic typed assembly language," in *PLACES*, ser. EPTCS, vol. 17, 2009, pp. 95–109.

[5] C. Boyapati, R. Lee, and M. Rinard, "Ownership types for safe program.: preventing data races and deadlocks," in *OOPSLA*. ACM, 2002, pp. 211–230.

[6] C. Flanagan and S. Qadeer, "A type and effect system for atomicity," in *PLDI*. ACM, 2003, pp. 338–349.

[7] M. Abadi, C. Flanagan, and S. N. Freund, "Types for safe locking: Static race detection for Java," *TOPLAS*, vol. 28, 2006.

[8] N. Kobayashi, "A partially deadlock-free typed process calculus," *TOPLAS*, vol. 20, no. 2, pp. 436–482, 1998.

[9] A. Igarashi and N. Kobayashi, "A generic type system for the pi-calculus," *Theor. Comput. Sci.*, vol. 311, no. 1-3, pp. 121–163, 2004.

[10] C. Laneve and L. Padovani, "The *must* preorder revisited," in *CONCUR*, ser. LNCS, vol. 4703. Springer, 2007, pp. 212–225.

[11] S. Parastatidis and J. Webber, *MEP SSDL Protocol Framework*, Apr. 2005, http://ssdl.org.

[12] S. D. Brookes, C. A. R. Hoare, and A. W. Roscoe, "A theory of communicating sequential processes," *J. ACM*, vol. 31, pp. 560–599, 1984.

[13] R. Milner, J. Parrow, and D. Walker, "A calculus of mobile processes, ii," *Inf. and Comput.*, vol. 100, pp. 41–77, 1992.

[14] H. R. Nielson and F. Nielson, "Higher-order concurrent programs with finite communication topology," in *POPL*. ACM, 1994, pp. 84–97.

[15] S. Chaki, S. K. Rajamani, and J. Rehof, "Types as models: model checking message-passing programs," *SIGPLAN Not.*, vol. 37, no. 1, pp. 45–57, 2002.

[16] R. Milner, *A Calculus of Communicating Systems*. Springer, 1982.

[17] S. P. Masticola, "Static detection of deadlocks in polynomial time," Ph.D. dissertation, 1993.

[18] R. Carlsson and H. Millroth, "On cyclic process dependencies and the verification of absence of deadlocks in reactive systems," 1997.

[19] E. Giachino and C. Laneve, "Deadlock and livelock analysis in concurrent objects with futures," www.cs.unibo.it/~laneve, 2012.

[20] E. B. Johnsen and O. Owe, "An asynchronous communication model for distributed concurrent objects," *Software and System Modeling*, vol. 6, no. 1, pp. 39–58, 2007.

[21] A. Welc, S. Jagannathan, and A. Hosking, "Safe futures for Java," in *OOPSLA*. New York, NY, USA: ACM, 2005, pp. 439–453.

[22] E. Giachino and C. Laneve, "A beginner's guide to the deadLock Analysis Model," in *TGC*, ser. LNCS. Springer-Verlag, 2013.

[23] E. Giachino, C. A. Grazia, C. Laneve, M. Lienhardt, and P. Y. H. Wong, "Deadlock analysis of concurrent objects: Theory and practice," 2013, submitted. Available at www.cs.unibo.it/~laneve.