A Calculus for Dynamic Reconfiguration with Low Priority Linking [1]

Davide Ancona [2] Sonia Fagorzi [3] Elena Zucca [4]

DISI
University of Genova
Genova, Italy

Abstract

Building on our previous work, we present a simple module calculus where execution steps of a module component can be interleaved with reconfiguration steps (that is, reductions at the module level), and where execution can partly control precedence between these reconfiguration steps. This is achieved by means of a *low priority link* operator which is only performed when a certain component, which has not been linked yet, is both available and really needed for execution to proceed, otherwise precedence is given to the outer operators. We illustrate the expressive power of this mechanism by a number of examples.

We ensure soundness by combining a static type system, which prevents errors in applying module operators, and a dynamic check which raises a linkage error if the running program needs a component which cannot be provided by reconfiguration steps. In particular no linkage errors can be raised if *all* components are potentially available.

Key words: Module calculi, dynamic linking.

1 Introduction

Traditional module calculi [6,16,13,2] are based on a *static* view of module manipulation, in the sense that open code fragments can be flexibly combined together, but all module operations can only be performed *before* starting execution of a program, that is, evaluation of a module component. On the contrary, modern programming environments such as those of Java and C#

[1] Partially supported by Dynamic Assembly, Reconfiguration and Type-checking - EC project IST-2001-33477, and APPSEM II - Thematic network IST-2001-38957i.
[2] Email: `davide@disi.unige.it`
[3] Email: `fagorzi@disi.unige.it`
[4] Email: `zucca@disi.unige.it`

support dynamic linking of software fragments, and, more in general, we can expect that in the future systems will support more and more forms of interleaving of *reconfiguration* and standard *execution* steps. Hence, the definition of clean and powerful module calculi providing foundations for these features is an important open problem.

In previous work, we have provided an example in this direction proposing CMS^{ℓ} [4], an extension of *CMS* (Calculus of Module Systems) [6] where module operators are executed on demand, that is, only when the executing program would otherwise get stuck in the current configuration. Hence, configuration steps do not need to be always completely performed, but those which will be performed are uniquely determined since the beginning, independently of the program execution. However, an important issue in practice is to allow a user's program to decide dynamically which configuration steps should be performed and in which order.

In this paper, we extend CMS^{ℓ} by defining a new calculus CMS^{ℓ,ℓ^-} enriched by a very simple but rather powerful mechanism for driving configuration steps from the running program. That is, we add a *low priority link* operator which is only performed when a certain component (which has not been linked yet) is both available and really needed for execution to proceed, otherwise precedence is given to the outer operators. In this way, control over precedence between configuration steps can be achieved by appropriately using variables in user's code. We illustrate the expressive power of this mechanism by a number of examples. Notably, with this more lazy link operator it is possible to model existing dynamic linking mechanisms in object-oriented contexts. For instance, in [12], we defined an encoding of a simple model of Java dynamic class loading (with multiple loaders) into the calculus. More in general, a model with different link operators represents an interesting framework which allows not only to better understand existing dynamic linking mechanisms, but also to study and compare possible variations of them.

A key issue in systems supporting dynamic reconfiguration is the balance between the opposite requirements of flexibility and capability of preventing errors (stuck reductions). In the calculus we propose, we are able to study this problem in a simple formal setting, where errors are either due to wrong applications of module operators, or to the fact that execution needs a module component which is neither currently available, nor can be provided by reconfiguration steps. Whereas the first kind of errors should, and easily can, be ruled out by a static type system, for the "missing component" error a purely static type system has the drawback that many safe reductions, which do not need some component, are ruled out. An improvement can be obtained by a type system based on dependencies, as that we developed for CMS^{ℓ} in [4], at the cost of an increasing complexity. For this reason, in this paper we prefer to explore a different solution by combining static type analyis with dynamic checks, in order to better support systems where configurations evolve in many ways which are hardly predictable at compile-time. More in detail, execution

of configurations with some missing components is also allowed, possibly raising the linkage error $\mathsf{err}(X : \tau, \pi)$, in case the component X with type τ is needed by the running program and none of the available definitions, specified by the type assignment π, can be associated with X by applying the given module operators.

2 An informal introduction to the calculus

In this section we illustrate $CMS^{\ell,\ell\text{-}}$ and its expressive power by means of some examples. For simplicity we omit type annotations in module components, which are not relevant here.

Terms of $CMS^{\ell,\ell\text{-}}$ are called *configurations* and are of two kinds: *non-executable* or *module expressions*, which are as in traditional module calculi based on static manipulation, and *executable*, which are, roughly speaking, module expressions paired with a running program (an expression in the underlying core language), such that steps of the program execution and reduction steps at the module level can be interleaved.

Module expressions

As in *CMS*, module expressions are constructed on top of *basic modules*, which have the form:

$$\left[x_i \mapsto X_i^{i\in 1..n};\ Y_j \mapsto e_j^{j\in 1..m};\ {x'}_k \mapsto {e'}_k^{k\in 1..p}\right]$$

where the three mappings correspond to *input*, *output* and *local* components of the module, respectively, X_i, Y_j are (*input* and *output*) *names*, used to refer to a component from outside the module, hence relevant in module composition, whereas x_i, x'_k are (*deferred* and *local*) *variables*, used to refer to components from inside the module, hence appearing in expressions e_j, e'_k but not relevant in module composition. Output and local components have an associated definition, which is an expression in the underlying core language ($CMS^{\ell,\ell\text{-}}$, as *CMS*, is a parametric calculus which can be instantiated on top of different core calculi for defining module components), whereas input components are declared but not yet defined.

For instance, denoting by $e[x_1, \ldots, x_n]$ an arbitrary core expression with free variables $x_1, \ldots, x_n$,

$$M_1 = [x \mapsto X;\ Y \mapsto e_1[x, y];\ y \mapsto e_2[x, y]]$$

is a basic module with one input, one output and one local component. Using some syntactic sugar, this could be written as follows.

```
module M1 is
 import X as x, export Y = e1[x,y], local  y = e2[x,y]
end M1
```

Operators for composing modules are *sum*, *reduct*, and *link*.

The sum of two modules can only be performed if they have no output components with the same name (that is, no conflicting definitions). In this case, the resulting module is obtained by putting together the components of the arguments: input components with the same name are shared, whereas conflicting deferred or local variables, if any, are solved by α-renaming.

For instance, let us define the module expression M_3 as the sum of M_1 above and another basic module

$$M_2 = [y \mapsto Y;\ X \mapsto e_3[x, y];\ x \mapsto e_4[x, y]]\,.$$

Then, the module expression $M_3 = M_1 + M_2$ reduces to

$$[x \mapsto X, y' \mapsto Y;\ Y \mapsto e_1[x, y], X \mapsto e_3[x', y'];\ y \mapsto e_2[x, y], x' \mapsto e_4[x', y']]\,.$$

The reduct operator performs a renaming of component names where input and output names are renamed independently. The input renaming is a mapping whose domain and codomain are old input names and new input names, respectively, whereas the output renaming goes in the opposite direction. For instance [5]

$$_{X\mapsto X, Y\mapsto X, \_\mapsto Z}|M_3|_{Y_1\mapsto Y, Y_2\mapsto Y}\,,$$

reduces to

$$[x \mapsto X, y' \mapsto X, z \mapsto Z;\ Y_1 \mapsto e_1[x, y], Y_2 \mapsto e_1[x, y];\ y \mapsto e_2[x, y], x' \mapsto e_4[x', y']]\,.$$

Note that a non-injective input renaming allows to merge two input names (in the example X and Y into X), whereas a non-surjective is used for adding dummy input names (Z in the example). A non-injective output renaming allows duplication of definitions (in the example the definition of Y is used as definition of both Y_1 and Y_2), whereas a non-surjective one is used for deleting output components (X in the example).

In the following we will use the abbreviation $M \setminus X$ to denote the module obtained by removing the output component X from M, that is, $_{\mathsf{id}}|M|_{\mathsf{em}}$ where id is the identity over input names, and em is the embedding of all output names but X into all output names, respectively.

The *link* operator connects input and output components having the same name inside a module, so that an input component becomes local.

For instance,

$$\mathsf{link}_{X\mapsto X}[x \mapsto X;\ X \mapsto e[x];\ ] \qquad \text{reduces to} \qquad [;\ X \mapsto e[x];\ x \mapsto e[x]]\,.$$

[5] The notation $\_ \mapsto Z$ means that Z is in the codomain of the mapping and no name is mapped in Z.

Configurations

Module expressions can be seen as a *configuration* language, in the sense that they model different ways in which software fragments can be composed together. However, eventually we want to get executable code and run it. This is modeled by *selection* of a module component. In traditional module calculi, like *CMS*, the selection operator, denoted $M.X$, can only be performed when M is a basic module with no input components (all configuration steps have been performed and the module is self-contained). For instance, $[;\, X \mapsto x + y;\, x \mapsto 2, y \mapsto 3]\,.X$ reduces to $2 + 3$, which then reduces to 5 by core reduction, whereas $[y \mapsto Y;\, X \mapsto x + y;\, x \mapsto 2]\,.X$ is stuck (and is prevented by the type system). We obtain a stuck module expression even when the defining expression of X does not depend on any deferred variable, e.g., $[y \mapsto Y;\, X \mapsto x + 3;\, x \mapsto 2]\,.X$ is stuck.

Note that in this way after selection the enclosing module structure disappears, hence no configuration steps are possible.

Here, we want to allow interleaving between evaluation of a module component and reconfiguration steps. Hence we take a rather different view of selection.

First, selection on a basic module does not return just the core expression which defines the selected component, but rather a *basic executable configuration*, that is, a pair consisting of the basic module itself and the core expression. This models an application running in the context of the components offered by the module. For instance, $[;\, X \mapsto x + y;\, x \mapsto 2, y \mapsto 3]\,.X$ reduces to

$$< [;\, X \mapsto x + y;\, x \mapsto 2, y \mapsto 3]\,,\; x + y >,$$

and then to

$$< [;\, X \mapsto x + y;\, x \mapsto 2, y \mapsto 3]\,,\; 2 + y >,$$

and so on. This simple change allows further configuration steps to be performed after selection, as will be shown below.

Moreover, selection can be performed even when there are still input components, since these missing components could be either never needed or become later available by performing configuration steps. An example of the first situation is shown by the following reduction steps:

$[y \mapsto Y;\, X \mapsto x + 3;\, x \mapsto 2]\,.X \longrightarrow$
$< [y \mapsto Y;\, X \mapsto x + 3;\, x \mapsto 2]\,,\; x + 3 > \longrightarrow$
$< [y \mapsto Y;\, X \mapsto x + 3;\, x \mapsto 2]\,,\; 2 + 3 > \longrightarrow$
$< [y \mapsto Y;\, X \mapsto x + 3;\, x \mapsto 2]\,,\; 5 >.$

An example of the second situation is illustrated below:

$\mathsf{link}_{Y \mapsto Y}([y \mapsto Y;\, X \mapsto x + y;\, x \mapsto 2]\,.X + [;\, Y \mapsto 3;\,]) \longrightarrow$
$\mathsf{link}_{Y \mapsto Y}(< [y \mapsto Y;\, X \mapsto x + y;\, x \mapsto 2]\,,\; x + y > + [;\, Y \mapsto 3;\,]) \longrightarrow$

$$\mathsf{link}_{Y\mapsto Y}(< [y \mapsto Y;\ X \mapsto x+y;\ x \mapsto 2]\,,\ 2+y > + [;\ Y \mapsto 3;\ ]) \longrightarrow$$
$$\mathsf{link}_{Y\mapsto Y} < [y \mapsto Y;\ X \mapsto x+y, Y \mapsto 3;\ x \mapsto 2]\,,\ 2+y > \longrightarrow$$
$$< [;\ X \mapsto x+y, Y \mapsto 3;\ x \mapsto 2, y \mapsto 3]\,,\ 2+y > \longrightarrow$$
$$< [;\ X \mapsto x+y, Y \mapsto 3;\ x \mapsto 2, y \mapsto 3]\,,\ 2+3 > \longrightarrow$$
$$< [;\ X \mapsto x+y, Y \mapsto 3;\ x \mapsto 2, y \mapsto 3]\,,\ 5 >.$$

This example also illustrates another feature of CMS^{ℓ,ℓ^-}: module operators we described above can be applied on top of basic executable configurations as well, giving *executable configurations* [6]; however, in this case operators have a *lazy* behavior, in the sense that they are performed *on demand*, only when program execution is stuck since it needs a deferred variable. For instance, in the case above, if the definition of X was $x+3$ instead (as before), then the sum and link operators would not be executed.

In other words, evolution of an executable configuration consists in program execution (applying the reduction rules at the core level which can manipulate local variables offered by the basic module), unless this execution requires to access a module component which is currently an input component. In this case, reconfiguration steps must be performed until this input component becomes available, that is, is imported from another module.

Note, however, that until now only a limited form of dynamic reconfiguration is allowed, since all reconfiguration steps are planned statically: the fact that they will be actually performed depends on the program execution, but it is not possible to perform *different* reconfiguration steps depending on the execution. To add a simple form of execution-driven reconfiguration, CMS^{ℓ,ℓ^-} includes also a variant link^- of the link operator, called *low priority link*, which can only be applied to executable configurations. Low priority link is an even more lazy form of link which is only performed when program execution would otherwise be stuck, as other operators, and, moreover, performing this operator will actually make continuation of the execution possible. This means that in the mapping specified in the low priority link there is an association from an input name to an output name which is executable (that is, both the names are present in the module) and whose application actually allows the program execution to continue (that is, the program needs exactly that input component). In this case, the link is performed incrementally, that is, *only* the needed component is resolved.

For instance, in

$$\mathsf{link}_{\substack{Y\mapsto Y,\\ Z\mapsto Z}}(\,\mathsf{link}^-_{\substack{X\mapsto X,\\ Y\mapsto W}}(< [x \mapsto X, y \mapsto Y;\ X \mapsto 2;\ ]\,,\ x+1 > + [;\ Y \mapsto 2;\ ])),$$

since execution needs component X, the $\mathsf{link}^-_{X\mapsto X}$ operator is executed and the

[6] However, sum is only allowed when at most one (conventionally the left) argument is executable, since we do not want to deal here with multiple threads.

configuration reduces in one step to

$$\mathsf{link}_{Y\mapsto Y,Z\mapsto Z}(\mathsf{link}^{-}_{Y\mapsto W}< [y\mapsto Y;\, X\mapsto 2;\, x\mapsto 2]\,,\; x+1>+[;\, Y\mapsto 2;\,]).$$

However, if the execution needs the component Y instead, e.g., in

$$\mathsf{link}_{\substack{Y\mapsto Y,\\ Z\mapsto Z}}(\,\mathsf{link}^{-}_{\substack{X\mapsto X,\\ Y\mapsto W}}(< [x\mapsto X, y\mapsto Y;\, X\mapsto 2;\,]\,,\; y+1>+[;\, Y\mapsto 2;\,])),$$

then the $\mathsf{link}^{-}_{X\mapsto X}$ is not performed, and outer operators are moved inside and performed instead, as shown below.

$\longrightarrow\ \mathsf{link}_{Y\mapsto Y,Z\mapsto Z}(\mathsf{link}^{-}_{X\mapsto X,Y\mapsto W}< [x\mapsto X, y\mapsto Y;\, X\mapsto 2, Y\mapsto 2;\,]\,,\; y+1>)$

$\longrightarrow\ \mathsf{link}^{-}_{X\mapsto X,Y\mapsto W}(\mathsf{link}_{Y\mapsto Y,Z\mapsto Z}< [x\mapsto X, y\mapsto Y;\, X\mapsto 2, Y\mapsto 2;\,]\,,\; y+1>)$

$\longrightarrow\ \mathsf{link}^{-}_{X\mapsto X,Y\mapsto W}< [x\mapsto X;\, X\mapsto 2, Y\mapsto 2;\, y\mapsto 2]\,,\; y+1>$

$\longrightarrow\ \ldots$

Note that we consider a slightly more liberal form of link w.r.t. *CMS* and CMS^{ℓ}, allowing associations for input names which are not present in the basic module (like $Z\mapsto Z$): these associations are simply ignored. On the other hand, associations from a present input name to an output name which is either missing or has the wrong type get stuck (and will be prevented by the type system). For the low priority link, instead, we also allow associations with missing or having wrong type output names (like $Y\mapsto W$): execution of these links will be delayed until both the two names will be present with the same type (thanks to the execution of some reconfiguration operator). Furthermore, in a low priority link, only the currently needed association is performed.

Expressive power

We show now some slightly more involved examples which illustrate how the simple mechanism offered by low priority link is powerful enough to model a variety of real-world situations.

Example 2.1 This configuration models a situation where a program can decide whether to link no components, only the component X_1 or only the component X_2:

$$\mathsf{link}^{-}_{X_2\mapsto X_2}(\ \mathsf{link}^{-}_{X_1\mapsto X_1}(\ <[x_1\mapsto X_1, x_2\mapsto X_2;\ ;\ z\mapsto e_z[x_i], x_0\mapsto e_0]\,, e[z,\ldots]> \\ +[\ldots;\, X_1\mapsto e_1,\, X_2\mapsto e_2;\, \ldots]\ )\ ).$$

The decision is coded in the definition of the control variable z, which in turn refers to x_i, with $i\in\{0,1,2\}$: if $i=0$, then no new component is linked; if $i=1$ (respectively, $i=2$), then only the component X_1 (respectively, X_2) is linked.

Example 2.2 This example models a a program using a library of software components X_i, $i \in 1..n$, for which there exist two versions e_i, e'_i, for instance different implementations of the same required functionality. In a first phase, if the program needs some component X_i, the initial version e_i is taken. For instance, the program uses the initial version for the first m components, for some $1 \leq m < n$. Then, the program can request for the following components the linking of the new version by means of the control variable z, as shown below. Here and in the following example we assume that $e[x_1, \ldots, x_n]$ is a core expression whose execution needs variables $x_1, \ldots, x_n$ in this order.

$$
\begin{array}{l}
\mathsf{link}^{-}_{Z \mapsto Z}(\\
\quad \mathsf{link}^{-}_{X_i \mapsto X_i^{i \in 1..n}}(< \left[z \mapsto Z, x_i \mapsto X_i^{i \in 1..n}; \; X_i \mapsto e_i^{i \in 1..n}; \; \right], \\
\qquad\qquad\qquad\qquad e[x_, \ldots, x_m, z, x_{m+1}, \ldots, x_n] > \\
\quad) \setminus X_i^{\,i \in 1..n} + \left[; \; X_i \mapsto e'^{\,i \in 1..n}_i, Z \mapsto e_Z; \; \right] \\
)
\end{array}
$$

Example 2.3 In this similar example

$$
\begin{array}{l}
\mathsf{link}^{-}_{X_2 \mapsto X_2}(\\
\quad \mathsf{link}^{-}_{X_1 \mapsto X_1}(\\
\qquad \mathsf{link}^{-}_{X \mapsto X}(< [x \mapsto X, x_1 \mapsto X_1, x_2 \mapsto X_2; \; X \mapsto e_0; \; z \mapsto e_z[x_i], x_0 \mapsto e_0] \, , \; e[z, x] > \\
\qquad) \setminus X + [; \; X \mapsto e_1, X_1 \mapsto e'_1; \;] \\
\quad) \setminus X + [; \; X \mapsto e_2, X_2 \mapsto e'_2; \;] \\
),
\end{array}
$$

the program uses a component internally referred to by x, for which there exist three versions: e_0 in the current execution context and e_1, e_2 in external modules. The decision about which version to use is coded in the definition of the control variable z which refers in turn to another control variable x_i. If $i = 0$, then the current version e_0 is used; if $i = 1$ (respectively, $i = 2$), then the version of X supplied by the first (resp. second) external module is linked.

3 Syntax and semantics

Notations

We write $f : A \to B$ to denote that f is a map with domain A, written $\mathsf{dom}(f)$, and codomain B, written $\mathsf{cod}(f)$.
We will use the following operators on maps:

- f, g is the union of two maps with disjoint domain.
- $f \cup g$ is the union of two compatible maps, that is, s.t. $f(x) = g(x)$ for all $x \in \mathsf{dom}(f) \cap \mathsf{dom}(g)$.
- $f|_C$ is the restriction of a map $f : A \to B$ to a set C (that is, $f|_C(x) = f(x)$ for $x \in A \cap C$).

- $f \backslash C$ is the removal from a map $f : A \to B$ of a set C (that is, $(f \backslash C)(x) = f(x)$ for $x \in A \setminus C$).
- $f \circ g$ denotes composition of two maps s.t. $\mathsf{cod}(g) \subseteq \mathsf{dom}(f)$.
- $f \subseteq g$ denotes map inclusion (that is, the usual subgraph relation).

The syntax of the calculus is given in Fig.1. We assume an infinite set Name of *names* X, an infinite set Var of *variables* x, and a set Exp of (core) expressions (the expressions of the underlying language used for defining module components). Indeed, as *CMS* and CMS^{ℓ}, CMS^{ℓ,ℓ^-} is a parametric calculus, which can be instantiated over different core calculi satisfying some (standard) assumptions specified in the sequel. In CMS^{ℓ,ℓ^-}, however, differently from *CMS*, module components cannot be modules.

Terms of the calculus are either *executable configurations* (configurations for short) or *non-executable configurations* (module expressions). An executable configuration can be constructed starting either from an *executable basic configuration* or from the selection of a component of a non executable configuration (module expression) M and by applying reconfiguration operators (*sum* with a module expression, *reduct*, *link* and *low priority link*).

An executable basic configuration is a pair $< [\iota; o; \rho]\,,\ e >$, consisting of a basic module and a core expression.

Basic modules are as in (typed) *CMS* apart that we adopt here a slightly different type decoration. They consist of three kinds of components: input assignment, representing the *input* interface of the module; output assignment, representing the *output* interface of the module; and local assignment, representing the *local* (that is, not visible outside the module) components.

The notation $x_i \overset{i\in I}{\mapsto} X_i : \tau_i$ (I possibly empty) is used for representing the unique surjective map ι such that $\mathsf{dom}(\iota) = \{x_i \mid i \in I\}$, $\mathsf{cod}(\iota) = \{X_i : \tau_i \mid i \in I\}$ and $\iota(x_i) = X_i : \tau_i$ for all $i \in I$. The expression is well-formed only if for any i_1 and i_2 in I, with $i_1 \neq i_2$, we have that $x_{i_1} \neq x_{i_2}$. We identify all expressions representing the same map. Moreover, a well-formed input assignment must satisfy a type coherence requirement, that is, for any i_1 and i_2 in I, if $X_{i_1} = X_{i_2}$, then $\tau_{i_1} = \tau_{i_2}$.

Given $\iota = x_i \overset{i\in I}{\mapsto} X_i : \tau_i$, we denote by ι_{Name} and ι_{Type} the maps which associate to each x_i the name X_i and the type τ_i, respectively.

Similar notations and assumptions are used for the other kinds of assignments. The notation $X_i \overset{i\in I}{\mapsto} Y_i, Y_j{}^{j\in J}$ (I or J possibly empty) is used for representing the unique map σ such that $\mathsf{dom}(\sigma) = \{X_i \mid i \in I\}$, $\mathsf{cod}(\sigma) = \{Y_i \mid i \in I \cup J\}$ and $\sigma(X_i) = Y_i$ for all $i \in I$.

Basic (both executable and non-executable) configurations are well-formed only if the sets of deferred and local variables are disjoint.

Note that for the low priority link operator we require the renaming σ to be not empty. Indeed, this operator is performed on demand, that is, an association $X \mapsto Y$ in σ is performed only when X is needed by the running

$C \in \mathsf{Conf}$	$::=$		**executable configuration**
		$< [\iota;\, o;\, \rho]\,,\ e >,$ with	
		$\mathsf{dom}(\iota) \cap \mathsf{dom}(\rho) = \emptyset$	executable basic configuration
	$\mid$	$C + M$	sum
	$\mid$	${}_{\sigma^\iota}\lvert C \rvert_{\sigma^o}$	reduct
	$\mid$	$\mathsf{link}_\sigma C$	link
	$\mid$	$\mathsf{link}^-_\sigma C,$ with $\sigma \neq \emptyset$	low priority link
	$\mid$	$M.X$	selection
$M \in \mathsf{Mod}$	$::=$		**non-executable configuration**
	$\mid$	$[\iota;\, o;\, \rho]\,,$ with	
		$\mathsf{dom}(\iota) \cap \mathsf{dom}(\rho) = \emptyset$	non-executable basic configuration
	$\mid$	$M + M$	sum
	$\mid$	${}_{\sigma^\iota}\lvert M \rvert_{\sigma^o}$	reduct
	$\mid$	$\mathsf{link}_\sigma M$	link
$\iota := x_i \overset{i \in I}{\mapsto} X_i : \tau_i$			**input assignment**
$o := X_i \overset{i \in I}{\mapsto} e_i : \tau_i$			**output assignment**
$\rho := x_i \overset{i \in I}{\mapsto} e_i$			**local assignment**
$\sigma := X_i \overset{i \in I}{\mapsto} Y_i, {Y_j}^{j \in J}$			**renaming**
$e \in \mathsf{Exp}$	$::=$	$x \mid \ldots$	**(core) expression**
$\tau \in \mathsf{Type}$	$::=$	$\ldots$	**(core) type**

Fig. 1. Syntax

program (and Y is available with the proper type), so specifying an empty renaming would make no sense.

We will explain module operators in more detail when introducing reduction rules.

Expressions of the core language are not specified; we only assume that they contain variables.

Reduction rules for sum, link and reduct on non-executable configurations, given in Fig.2, are exactly as in *CMS* and CMS^ℓ (apart from the treatment of type decorations), hence for their explanation we refer to the examples of the previous section and to [6,4] for more details. For sake of clarity, we write also some side conditions (labeled "implicit") which are redundant since implied

$$\mathcal{M} \in \mathcal{M}\mathsf{Ctx} ::= \Box \mid \mathcal{M} + M \mid [\iota;\, o;\, \rho] + \mathcal{M} \mid \mathsf{link}_\sigma \mathcal{M} \mid {}_{\sigma^\iota}|\mathcal{M}|_{\sigma^o}$$

$$(\mathcal{M}\text{-ctx}) \quad \frac{R^M \longrightarrow M}{\mathcal{M}\,[R^M] \longrightarrow \mathcal{M}\,[M]}$$

$$(M\text{-sum}) \quad \frac{}{[\iota_1;\, o_1;\, \rho_1] + [\iota_2;\, o_2;\, \rho_2] \longrightarrow [\iota_1,\, \iota_2;\, o_1,\, o_2;\, \rho_1,\, \rho_2]}$$

if $\mathsf{dom}(\iota_1, \rho_1) \cap \mathsf{FV}([\iota_2;\, o_2;\, \rho_2]) = \mathsf{dom}(\iota_2, \rho_2) \cap \mathsf{FV}([\iota_1;\, o_1;\, \rho_1]) = \emptyset$

$\mathsf{dom}(\iota_1, \rho_1) \cap \mathsf{dom}(\iota_2, \rho_2) = \emptyset$ (implicit)

$\mathsf{dom}(o_1) \cap \mathsf{dom}(o_2) = \emptyset$ (implicit)

$X : \tau_1 \in \mathsf{cod}(\iota_1) \wedge X : \tau_2 \in \mathsf{cod}(\iota_2) \Rightarrow \tau_1 = \tau_2$ (implicit)

$$(M\text{-reduct}) \quad \frac{}{{}_{\sigma^\iota}|[\iota, \iota';\, o;\, \rho]|_{\sigma^o} \longrightarrow [\sigma^\iota \circ_{\mathsf{Name}} \iota, \iota';\, o \circ \sigma^o;\, \rho]}$$

if $\mathsf{cod}(\iota'_{\mathsf{Name}}) \cap \mathsf{dom}(\sigma^\iota) = \emptyset$

$\mathsf{cod}(\iota_{\mathsf{Name}}) \subseteq \mathsf{dom}(\sigma^\iota)$ (implicit)

$\mathsf{cod}(\sigma^o) \subseteq \mathsf{dom}(o)$ (implicit)

$X_1 : \tau_1 \in \mathsf{cod}(\iota) \wedge \sigma^\iota(X_1) = X_2 \wedge X_2 : \tau_2 \in \mathsf{cod}(\iota') \Rightarrow \tau_1 = \tau_2$ (implicit)

$$(M\text{-link}) \quad \frac{}{\mathsf{link}_\sigma \left[x_i \overset{i \in I}{\mapsto} X_i : \tau_i, \iota;\, o;\, \rho\right] \longrightarrow \left[\iota;\, o;\, \rho, x_i \overset{i \in I}{\mapsto} o_{\mathsf{Exp}}(\sigma(X_i))\right]}$$

if $\mathsf{cod}(\iota_{\mathsf{Name}}) \cap \mathsf{dom}(\sigma) = \emptyset$

$\{X_i \mid i \in I\} \subseteq \mathsf{dom}(\sigma)$ (implicit)

$\{\sigma(X_i) \mid i \in I\} \subseteq \mathsf{dom}(o)$ (implicit)

Fig. 2. Reduction rules for module expressions

by the fact that terms must be well-formed.

In rule ($\mathcal{M}$-ctx) the metavariable R^M ranges over *module redexes*, that is, left-hand sides of other rules for module expressions in Fig.2. We write $\mathcal{M}\,[M]$ the expression obtained by syntactically replacing the hole in $\mathcal{M}$ with the expression M (*without* any variable renaming).

In rule (M-sum), $\mathsf{FV}(M)$denotes the set of the free variables in M, respectively, defined in the obvious way.

In rule (M-reduct), if $\sigma = X_h \overset{h \in H}{\mapsto} Y_h, Y_k \overset{k \in K}{}$ and $\iota = x_i \overset{i \in I}{\mapsto} X_i : \tau_i$, with $I \subseteq H$, then $\sigma \circ_{\mathsf{Name}} \iota = x_i \overset{i \in I}{\mapsto} Y_i : \tau_i, x_k \overset{k \in K}{\mapsto} Y_k : \tau_k$, where for all $k \in K$, x_k is a fresh variable and τ_k is an arbitrary type.

In Fig.3 and in Fig.4 we give reduction rules for executable configurations.

$$\mathcal{C} \in \mathcal{C}\mathsf{Ctx} \quad ::= \Box \mid \mathcal{C} + M \mid \mathsf{link}_\sigma \mathcal{C} \mid {}_{\sigma^\iota}|\mathcal{C}|_{\sigma^o} \mid \mathsf{link}^-_\sigma \mathcal{C}$$

$$\mathcal{CM} \in \mathcal{CM}\mathsf{Ctx} ::= \mathcal{M}.X \mid \mathcal{CM} + M \mid \mathsf{link}_\sigma \mathcal{CM} \mid {}_{\sigma^\iota}|\mathcal{CM}|_{\sigma^o} \mid \mathsf{link}^-_\sigma \mathcal{CM}$$

$$\mathcal{E} \in \mathcal{E}\mathsf{Ctx} \quad ::= \Box \mid \ldots$$

Contextual closure

$$(\mathcal{C}\text{-ctx}) \ \frac{R^C \longrightarrow C}{\mathcal{C}\left[R^C\right] \longrightarrow \mathcal{C}\left[C\right]} \qquad (\mathcal{CM}\text{-ctx}) \ \frac{R^M \longrightarrow M}{\mathcal{CM}\left[R^M\right] \longrightarrow \mathcal{CM}\left[M\right]}$$

Selection rule

$$(\text{sel}) \ \frac{}{[\iota;\, o;\, \rho].X \longrightarrow \ < [\iota;\, o;\, \rho]\,,\ o(X) >} \ X \in \mathsf{dom}(o) \ \text{(implicit)}$$

Program evaluation rules

$$(\text{core}) \ \frac{e \underset{core}{\longrightarrow} e'}{< [\iota;\, o;\, \rho]\,,\ e > \ \longrightarrow \ < [\iota;\, o;\, \rho]\,,\ e' >}$$

$$(\text{var}) \ \frac{}{< [\iota;\, o;\, \rho]\,,\ \mathcal{E}\left[x\right] > \ \longrightarrow \ < [\iota;\, o;\, \rho]\,,\ \mathcal{E}\{\rho(x)\} >} \begin{array}{l} x \in \mathsf{dom}(\rho) \ \text{(implicit)} \\ x \notin \mathsf{HB}\left(\mathcal{E}\right) \end{array}$$

Error rules

$$(\text{var/err}) \ \frac{}{< [\iota;\, o;\, \rho]\,,\ \mathcal{E}\left[x\right] > \ \longrightarrow \mathsf{err}(X : \tau, \pi)} \begin{array}{l} x \notin \mathsf{HB}\left(\mathcal{E}\right) \\ X : \tau = \iota(x) \\ \pi = o_{\mathsf{Type}} \end{array}$$

$$(\text{link}^-/\text{err}) \ \frac{C \longrightarrow \mathsf{err}(X : \tau, \pi)}{\mathsf{link}^-_\sigma C \longrightarrow \mathsf{err}(X : \tau, \pi)} \begin{array}{l} X \notin \mathsf{dom}(\sigma) \vee \\ \sigma(X) \notin \mathsf{dom}(\pi) \vee \\ \pi(\sigma(X)) \neq \tau \end{array}$$

Fig. 3. Reduction rules for executable configurations I

The intuition is that the execution of a configuration starts with the evaluation of the program running inside it, possibly obtained by selecting a component from a module expression (rule (sel) in Fig.3) and this evaluation proceeds by standard execution steps possibly accessing local variables offered by the basic module (program evaluation rules in Fig.3) until a deferred variable is encountered; in this case, reconfiguration steps are needed (error rules in Fig.3) and they are performed (reconfiguration rules in Fig.4) until the variable becomes local and rule (var) can be applied.

Contextual closure rule

There are two kinds of evaluation contexts and, correspondingly, two contextual closure rules for executable configurations: $\mathcal{C}$ is a context with hole requiring $C \in \mathsf{Conf}$, s.t. $\mathcal{C}[C] \in \mathsf{Conf}$, while $\mathcal{CM}$ is a context with hole requiring $M \in \mathsf{Mod}$, s.t. $\mathcal{CM}[M] \in \mathsf{Conf}$.

In rule ($\mathcal{C}$-ctx) the metavariable R^C ranges over configuration redexes, that is, left-hand side of other rules for configurations in Fig.3 and Fig.4.

Selection rule

Rule (sel) takes the non executable basic configuration $[\iota; o; \rho]$ and makes it executable by selecting the core expression $o(X)$ as program.

Program evaluation rules

Rule (core) models an execution step which is an evaluation step of the core expression in the basic executable configuration (we denote by $\xrightarrow[core]{}$ the reduction relation of the core calculus).

Rule (var) models the situation where the evaluation of the core expression needs a variable which has a corresponding definition in the current basic module (that is, is local). In this case, the evaluation can proceed by simply replacing the variable by its defining expression. We denote by $\mathcal{E}$ the core evaluation contexts, and by HB the function associating with each core evaluation context the set of binders around its hole, defined in the obvious way. Here and in the following rules, the side condition $x \notin \mathsf{HB}(\mathcal{E})$ expresses the fact that the occurrence of the variable x in the position denoted by the hole of the core context $\mathcal{E}$ is free (that is, not captured by any binder around the hole). Finally, we denote by $\mathcal{E}\{e\}$ the capture avoiding substitution, with the expression e, of the hole of the context $\mathcal{E}$.

Error rules

Rule (var/err) models the situation where the evaluation of the core expression needs a variable which has no corresponding definition in the current basic module (that is, is deferred). In this case, a reconfiguration step is triggered by raising the error $\mathsf{err}(X : \tau, \pi)$, which can be captured by outer operators as described in the paragraph on reconfiguration rules.

Rule (link^-/err) deals with the case when the link^- operator cannot be performed, either because no link is specified for the name X needed for the computation to continue (side condition $X \notin \mathsf{dom}(\sigma)$), or the required definition is missing ($\sigma(X) \notin \mathsf{dom}(\pi)$) or does not have the proper type ($\pi(\sigma(X)) \neq \tau$). In this case, the error $\mathsf{err}(X : \tau, \pi)$ is propagated to the outer operator (if any), so that either X will be eventually linked with the proper type τ, or the whole computation will terminate with the linkage error $\mathsf{err}(X : \tau, \pi)$.

It is worth to note that errors are not propagated by contextual closure rules; hence, they are captured by surrounding contexts consisting in usual

Reconfiguration rules

$$\text{(sum/basic)}\ \frac{< [\iota_1;\, o_1;\, \rho_1]\,,\ e > \ \longrightarrow \mathsf{err}(X : \tau, \pi)}{< [\iota_1;\, o_1;\, \rho_1]\,,\ e > + [\iota_2;\, o_2;\, \rho_2] \ \longrightarrow\ < [\iota_1,\, \iota_2;\, o_1,\, o_2;\, \rho_1,\, \rho_2]\,,\ e >}$$

if $\mathsf{dom}(\iota_1, \rho_1) \cap \mathsf{FV}([\iota_2;\, o_2;\, \rho_2]) = \mathsf{dom}(\iota_2, \rho_2) \cap \mathsf{FV}([\iota_1;\, o_1;\, \rho_1]) = \emptyset$

$$\text{(sum-closure)}\ \frac{< [\iota;\, o;\, \rho]\,,\ e > \ \longrightarrow \mathsf{err}(X : \tau, \pi) \quad M \longrightarrow M'}{< [\iota;\, o;\, \rho]\,,\ e > + M \ \longrightarrow\ < [\iota;\, o;\, \rho]\,,\ e > + M'}$$

$$\text{(sum/link}^{-}\text{)}\ \frac{\mathsf{link}^{-}_{\sigma} C \longrightarrow \mathsf{err}(X : \tau, \pi)}{\mathsf{link}^{-}_{\sigma} C + M \longrightarrow \mathsf{link}^{-}_{\sigma}(C + M)}$$

$$\text{(reduct/basic)}\ \frac{< [\iota, \iota';\, o;\, \rho]\,,\ e > \ \longrightarrow \mathsf{err}(X : \tau, \pi)}{{}_{\sigma^\iota}|< [\iota, \iota';\, o;\, \rho]\,,\ e >|_{\sigma^o} \ \longrightarrow\ < [\sigma^\iota \circ_{\mathsf{Name}} \iota, \iota';\, o \circ \sigma^o;\, \rho]\,,\ e >}$$

if $\mathsf{cod}(\iota'_{\mathsf{Name}}) \cap \mathsf{dom}(\sigma^\iota) = \emptyset$

$$\text{(reduct/link}^{-}\text{)}\ \frac{\mathsf{link}^{-}_{\sigma} C \longrightarrow \mathsf{err}(X : \tau, \pi)}{{}_{\sigma^\iota}|\mathsf{link}^{-}_{\sigma} C|_{\sigma^o} \longrightarrow \mathsf{link}^{-}_{\sigma}({}_{\sigma^\iota}|C|_{\sigma^o})}$$

$$\text{(link/basic)}\ \frac{< \left[x_i \overset{i\in I}{\mapsto} X_i : \tau_i, \iota;\, o;\, \rho\right]\,,\ e > \ \longrightarrow \mathsf{err}(X : \tau, \pi)}{\mathsf{link}_{\sigma} < \left[x_i \overset{i\in I}{\mapsto} X_i : \tau_i, \iota;\, o;\, \rho\right]\,,\ e > \ \longrightarrow\ < \left[\iota;\, o;\, \rho, x_i \overset{i\in I}{\mapsto} o_{\mathsf{Exp}}(\sigma(X_i))\right]\,,\ e >}$$

if $\mathsf{cod}(\iota_{\mathsf{Name}}) \cap \mathsf{dom}(\sigma) = \emptyset$

$$\text{(link/link}^{-}\text{)}\ \frac{\mathsf{link}^{-}_{\sigma'} C \longrightarrow \mathsf{err}(X : \tau, \pi)}{\mathsf{link}_{\sigma}\left(\mathsf{link}^{-}_{\sigma'} C\right) \longrightarrow \mathsf{link}^{-}_{\sigma'}(\mathsf{link}_{\sigma} C)}$$

$$\text{(link}^{-}\text{)}\ \frac{C \longrightarrow \mathsf{err}(X : \tau, \pi)}{\mathsf{link}^{-}_{\sigma,\, X \mapsto Y} C \longrightarrow \mathsf{link}^{-}_{\sigma}(\mathsf{link}_{X \mapsto Y} C)}\ \pi(Y) = \tau$$

Fig. 4. Reduction rules for executable configurations II

module operators (see rules (sum/basic), (sum-closure), (reduct/basic) and (link/basic) below); whereas a low-priority link propagates an error if it cannot resolve it.

Hence, intuitively, $C \longrightarrow \mathsf{err}(X : \tau, \pi)$ holds if and only if the program evaluation in C needs an input component with name X and type τ, and the output components currently available are those specified by the type assignment π. The latter information is used, in case a low priority link for X is applied to C, to decide whether rule (link-) or (link-/err) is applicable.

Reconfiguration rules

Sum, reduct and link operators are performed on demand, whenever the evaluation of their enclosed configuration expression needs a deferred variable (and, hence, an error of the form $\mathsf{err}(X : \tau, \pi)$ was raised). Two different kinds of rules are needed for each operator. The former is applied when the enclosed configuration is basic, and in this case the operator is performed on the module expression inside the basic configuration. Note that the effect of module operators on the module inside a basic configuration is exactly as seen in Fig.2; all implicit side conditions (that for brevity are not reported here) still hold. The latter is applied when the enclosed configuration has a (surely either not needed or not applicable) link^- as outer operator; in this case, the operator is swapped with the link^-. By repeatedly applying this rule, the operator goes inside until it can be performed by the corresponding (_/basic) rule.

In rule (link^-), the link^- operator can be performed only if the required input name X can be safely linked to an output component with the proper type (side condition $\pi(Y) = \tau$). Note that the link^- is applied incrementally, in the sense that only the required input name will be linked. To this end, the application of link^- is split into a link^- and link application. Then, by repeatedly applying the (link/link^-) rule, the operator link will go inside any inner link^- operator until it can be performed by the (link/basic) rule. In this rule, we assume to identify $\mathsf{link}^-_\emptyset C$ with C (the link^- disappears when fully executed, that is, when $\sigma = \emptyset$).

Finally, for the sum operator we also need a further rule to force, when needed, the evaluation of the module expression in the second argument of the sum.

4 Type system

By the reduction rules given in the previous section, stuck reductions are either due to wrong applications of module operators, as already in *CMS* [7], or to the fact that program execution needs a module component which is neither currently available, nor can be provided by reconfiguration steps.

The first kind of errors can easily be ruled out by a static type system analogous to that originally designed for *CMS* in [6], where module types are pairs of *signatures* specifying input and output components with their types (for configurations it is enough to add the type of the running program). The only novelty is the low priority link operator. It is easy to see that this operator does not introduce any new typing error, since link^-_σ can be safely applied regardless of the type of the argument C (indeed, if it is not applicable it has simply no effect). On the other hand, it is not so clear what should be the type

[7] That is, incompatible input or conflicting output components in a sum, renaming of a missing output name, and linking to a missing output component.

of the resulting configuration: indeed, since an applicable low priority link for an input component, say X, is either performed or just ignored depending on whether the running program needs X, then $\mathsf{link}^{-}_{\sigma}C$ can have either the type obtained by linking X (that is, the type obtained from C by removing X from the input names), or the same type as C. However, only the latter solution (reflected by the formal rule (C-link^{-}) in Fig.7) is safe, since removing input names from a type does not introduce type unsafe application of module operators, whereas the converse does not hold (for instance, we might end up with a type unsafe sum involving incompatible input components).

For avoiding the "missing component" error, a simple solution which can be enforced by this static type system is to consider as statically correct only those configurations which are "closed", that is, those which have no input components. Indeed, this intuitively means that, whichever component the running program will ever need, this component can eventually become available by reconfiguration. However, this solution has the drawback that many safe reductions, which never need a non-available component, are ruled out. An improvement can be obtained by a more complex type system based on dependencies, as that we developed for CMS^{ℓ} in [4]. In this paper, however, we prefer to explore a different solution, since the expressive power gained by introducing the low priority link operator would be partly lost by the adoption of a too strict type discipline. Indeed, as said above, in practice this operator is not taken into account when computing the type of a term, whereas at run-time its application could supply a needed component. Hence, we prefer to combine the static type analysis for preventing unsafe module application with dynamic checks preventing stuck execution due to a missing component.

More in detail, configurations with input components are considered type safe as well, and execution can raise the linkage error $\mathsf{err}(X : \tau, \pi)$, in case the component X with type τ is needed by the running program and none of the available definitions, specified by the type assignment π, can be associated with X by applying the given module operators.

We give now the formal definition of the type system. Module types have the form $[\pi^{\iota}; \pi^{o}]$, where π^{ι}, π^{o} are *signatures*, that is, sequences ${X_i : \tau_i}^{i \in I}$ of pairs consisting of a component name and a type. In the following we will identify all signatures which represent the same set of pairs (that is, order and repetitions are immaterial). Intuitively, if a module M has type $\left[{X_i : \tau_i}^{i \in I}; {X_j : \tau_j}^{j \in J}\right]$, then $\{X_i \mid i \in I\}$ and $\{X_j \mid j \in J\}$ represent the sets of input and output components of M, respectively. The type annotation $X_i : \tau_i$ says that the input (resp. output) component X_i can be correctly bound to (resp. associated with) an expression of type τ_i.

A module type is well-formed if the two signatures π^{ι} and π^{o} turn out to be two maps from component names into well-formed types. This is formalized by the judgment $\vdash [\pi^{\iota}; \pi^{o}]$ defined by rules in Fig.5, where $\vdash_{core} \tau$ is the judgment for well-formed types at the core level.

In the following we will use on (well-formed) signatures the operators for

$$\frac{\vdash \pi^\iota \quad \vdash \pi^o}{\vdash [\pi^\iota; \pi^o]} \qquad \frac{\{\vdash_{core} \tau_i \mid i \in I\}}{\vdash X_i : \tau_i{}^{i \in I}} \ \forall h, k \in I. X_h = X_k \Rightarrow \tau_h = \tau_k$$

Fig. 5. Well-formed module types

$$(M\text{-basic}) \ \frac{\vdash \left[X_i : \tau_i{}^{i \in I}; \ X_j : \tau_j{}^{j \in O}\right] \quad \left\{x_h : \tau_h{}^{h \in I \cup L} \vdash_{core} e_k : \tau_k \mid k \in O \cup L\right\}}{\vdash_M \left[x_i \overset{i \in I}{\mapsto} X_i : \tau_i; \ X_j \overset{j \in O}{\mapsto} e_j : \tau_j; \ x_l \overset{l \in L}{\mapsto} e_l\right] : \left[X_i : \tau_i{}^{i \in I}; \ X_j : \tau_j{}^{j \in O}\right]}$$

$$(M\text{-sum}) \ \frac{\vdash_M M_1 : [\pi^\iota{}_1; \ \pi^o{}_1] \qquad \vdash_M M_2 : [\pi^\iota{}_2; \ \pi^o{}_2]}{\vdash_M M_1 + M_2 : [\pi^\iota{}_1 \cup \pi^\iota{}_2; \ \pi^o{}_1 \cup \pi^o{}_2]} \ \pi^o{}_1 \cap \pi^o{}_2 = \emptyset$$

$$(M\text{-reduct}) \ \frac{\vdash_M M : [\pi^\iota; \ \pi^o]}{\vdash_M \sigma^\iota |M|_{\sigma^o} : [\tilde{\pi}^\iota \cup \pi^\iota \backslash \mathsf{dom}(\sigma^\iota); \ \tilde{\pi}^o]} \ \begin{array}{l} \sigma^\iota|_{\mathsf{dom}(\pi^\iota)} : \pi^\iota|_{\mathsf{dom}(\sigma)} \to \tilde{\pi}^\iota \\ \sigma^o : \tilde{\pi}^o \to \pi^o \end{array}$$

$$(M\text{-link}) \ \frac{\vdash_M M : [\pi^\iota; \ \pi^o]}{\vdash_M \mathsf{link}_\sigma M : [\pi^\iota \backslash \mathsf{dom}(\sigma); \ \pi^o]} \ \sigma|_{\mathsf{dom}(\pi^\iota)} : \pi^\iota|_{\mathsf{dom}(\sigma)} \to \pi^o$$

Fig. 6. Typing rules for module expressions

maps (which are closed w.r.t. well-formed signatures).

The type system of the calculus is given in Fig.6 and Fig.7.

The type judgment for module expressions has form $\vdash_M M : [\pi^\iota; \pi^o]$, meaning that M is a well-formed module expression with type $[\pi^\iota; \pi^o]$.

Typing rules for module expressions are given in a slightly different form than in standard module calculi [6]. Indeed, in addition to the treatment of type decorations, they allow a more general form of application of the reduct and the link operators.

In rule (M-basic), we denote by $\Gamma \vdash_{core} e : \tau$ the typing judgment for core expressions, meaning that e is a well-formed expression of type τ in Γ, where Γ is a (core) context, that is, a map from variables to well-formed (core) types. Note that the module type must be well-formed.

The (M-sum) typing rule allows sharing of input components having the same name and type, whereas the side condition prevents output components from being shared. Recall that the expression $f \cup g$ denotes the union of two compatible maps f and g. So, it implicitly holds that the two resulting signatures are well-formed.

In rules (M-reduct) and (M-link) the side-conditions having the form $\sigma : \pi_1 \to \pi_2$ ensure that the renaming σ preserves types; formally, this means that $\sigma : \mathsf{dom}(\pi_1) \to \mathsf{dom}(\pi_2)$ and $\sigma(X) = Y \Rightarrow \pi_1(X) = \pi_2(Y)$.

In rule (M-reduct), differently from the original formulation [6], the domain of an input renaming can be any set of names: indeed, renaming of input names not present in the module is simply ignored (by considering the restriction of σ^ι to the domain of π^ι). Moreover, module input names which are not renamed by σ^ι are unaffected. For the output renaming, the codomain must be set of the output names in the module. The two side conditions, besides

$$(C\text{-basic})\ \frac{\vdash \left[X_i : {\tau_i}^{i\in I};\ X_j : {\tau_j}^{j\in O}\right] \quad x_h : {\tau_h}^{h\in I\cup L} \vdash_e e : \tau \quad \left\{x_h : {\tau_h}^{h\in I\cup L} \vdash_{core} e_k : \tau_k \mid k \in O \cup L\right\}}{\vdash_C < \left[x_i \overset{i\in I}{\mapsto} X_i : \tau_i;\ X_j \overset{j\in O}{\mapsto} e_j : \tau_j;\ x_l \overset{l\in L}{\mapsto} e_l\right],\ e >: \left(\left[X_i : {\tau_i}^{i\in I};\ X_j : {\tau_j}^{j\in O}\right], \tau\right)}$$

$$(C\text{-sum})\ \frac{\vdash_C C : \left([\pi^\iota{}_C;\ \pi^o{}_C], \tau\right) \qquad \vdash_M M : [\pi^\iota{}_M;\ \pi^o{}_M]}{\vdash_C C + M : \left([\pi^\iota{}_C \cup \pi^\iota{}_M;\ \pi^o{}_C \cup \pi^o{}_M], \tau\right)}\ \pi^o{}_C \cap \pi^o{}_M = \emptyset$$

$$(C\text{-reduct})\ \frac{\vdash_C C : \left([\pi^\iota;\ \pi^o], \tau\right)}{\vdash_C \sigma^\iota|C|_{\sigma^o} : \left([\tilde{\pi}^\iota \cup \pi^\iota \setminus \mathsf{dom}(\sigma^\iota);\ \tilde{\pi}^o], \tau\right)}\ \begin{array}{l}\sigma^\iota|_{\mathsf{dom}(\pi^\iota)} : \pi^\iota|_{\mathsf{dom}(\sigma^\iota)} \to \tilde{\pi}^\iota \\ \sigma^o : \tilde{\pi}^o \to \pi^o\end{array}$$

$$(C\text{-link})\ \frac{\vdash_C C : \left([\pi^\iota;\ \pi^o], \tau\right)}{\vdash_C \mathsf{link}_\sigma C : \left([\pi^\iota \setminus \mathsf{dom}(\sigma);\ \pi^o], \tau\right)}\ \sigma|_{\mathsf{dom}(\pi^\iota)} : \pi^\iota|_{\mathsf{dom}(\sigma)} \to \pi^o$$

$$(C\text{-link}^-)\ \frac{\vdash_C C : \left([\pi^\iota;\ \pi^o], \tau\right)}{\vdash_C \mathsf{link}^-_\sigma C : \left([\pi^\iota;\ \pi^o], \tau\right)}$$

$$(\text{sel})\ \frac{\vdash_M M : [\pi^\iota;\ \pi^o]}{\vdash_C M.X : \left([\pi^\iota;\ \pi^o], \pi^o(X)\right)}$$

Fig. 7. Typing rules for executable configurations

guarantee type preservation, determine the resulting signature of the module. Note that the resulting input signature is the union of the not affected and the new input signatures. These two signatures must be compatible (as implicitly required for well-formedness of the resulting type), since the domains of $\tilde{\pi}^\iota$ and $\pi^\iota \setminus \mathsf{dom}(\sigma^\iota)$ might be not disjoint.

As in (M-reduct), also in (M-link) we allow the renaming to be defined on any set of names (differently from the original formulation [6]). Input names which are not linked by σ remain in the resulting module, whereas, again, linking of input names not present in the module is ignored (by taking the restriction of σ to the domain of π^ι).

The typing judgment for executable configurations has form $\vdash_C C : \left([\pi^\iota;\ \pi^o], \tau\right)$, meaning that C is a well-formed executable configuration of type $\left([\pi^\iota;\ \pi^o], \tau\right)$.

The first component $[\pi^\iota;\ \pi^o]$ has the same meaning as for module expressions, while τ is the type of the running program.

In rule (C-basic), the first component of the type is computed as for basic module expressions. The second component in the configuration type corresponds to the type of the running program e in the context of all the (deferred and local) variables of the basic module (that is, $x_h : {\tau_h}^{h\in I\cup L}$).

Rules for sum, reduct and link are the obvious extension of those for non executable configurations, where the type τ of the running program is just propagated.

In rule (C-link^{-}), the application of a low priority link operator $\mathsf{link}^{-}_{\sigma}$ to a configuration C does not change its type. Indeed, as explained at the beginning of this section, during static analysis low priority links are not considered, that is, the type system returns the type one would get if no low priority link had been ever executed; this is safe since any application of a low priority link never lead to undefined applications of the other operators.

In rule (C-sel) the type τ of the expression to be executed coincides with the type of X.

5 Results

In this section we collect the technical results about the calculus. In particular, we state the determinacy, subject reduction and progress properties for the reduction relation. These results hold providing that the corresponding properties are verified at the core level as well.

We first introduce (Fig.8) the sets VMod and VConf of values for the terms of the calculus.

$$M^v \in \mathsf{VMod} ::= [\iota;\, o;\, \rho]$$

$$C^v \in \mathsf{VConf} ::= < [\iota;\, o;\, \rho]\,,\ e^v > \ \mid C^v + M \mid \mathsf{link}_{\sigma} C^v \mid \mathsf{link}^{-}_{\sigma} C^v \mid {}_{\sigma^\iota}|C^v|_{\sigma^o}$$

$$e^v \in \mathsf{VExp} \qquad \text{(core) values}$$

Fig. 8. Values

Core assumption 5.1 *We assume the core language to be such that:*

(i) *(Unique decomposition) Given $e \in \mathsf{Exp}$, at most one of the following cases holds:*
   (a) $e \in \mathsf{CVal}$,
   (b) *there exist unique a core evaluation context $\mathcal{E}$, a core rule (r) and a core redex R^e (instance of the left-hand side of (r)) such that $\mathcal{E}\,[R^e] = e$,*
   (c) *there exist unique a core evaluation context $\mathcal{E}$, and a variable x such that $\mathcal{E}\,[x] = e$ and $x \notin \mathsf{HB}\,(\mathcal{E})$.*

(ii) *(Progress) if $\Gamma \vdash_e e : \tau$, with $e \notin \mathsf{CVal}$, then either $e \equiv \mathcal{E}\,[R^e]$ for some R^e or $e \equiv \mathcal{E}\,[x]$ with $x \notin \mathsf{HB}\,(\mathcal{E})$ and $x \in \mathsf{dom}(\Gamma)$.*

(iii) *(Subject reduction) if $\Gamma \vdash_e e : \tau$ and $e \xrightarrow[e]{} e'$, then $\Gamma \vdash_e e' : \tau$.*

(iv) *(Substitution) if $\Gamma \vdash_e \mathcal{E}\,[x] : \tau$, with $x \notin \mathsf{HB}\,(\mathcal{E})$, and $\Gamma \vdash_e e' : \tau'$, then $\Gamma \vdash_e \mathcal{E}\{e'\} : \tau$.*

(v) *(Weakening) if $\Gamma \vdash_e e : \tau$, then, for all $\Gamma' \supseteq \Gamma$ we have that $\Gamma' \vdash_e e : \tau$.*

Note that the progress property in point (ii) with $\Gamma = \emptyset$ takes the usual form, that is, a well-typed e is either a value or performs a reduction step

$e \xrightarrow[e]{} e'$ (by reducing the redex R^e).The property also implicitly implies that a variable cannot be a redex. Moreover, in point (iv), the condition $x \notin \mathsf{HB}(\mathcal{E})$, which is needed to avoid substitution of bound variables, implies $x \in \mathsf{dom}(\Gamma)$. Hence, we allow the substituted term e' to in turn refer to the variable x.

Theorem 5.1 (Unique decomposition)

- *Given $M \in \mathsf{Mod}$, at most one of the following cases holds:*
  - *$M \in \mathsf{VMod}$,*
  - *there exist unique an evaluation context $\mathcal{M}$, a rule (r) and a redex R^M (instance of the left-hand side of rule (r)) such that $\mathcal{M}\left[R^M\right] = M$.*
- *Given $C \in \mathsf{Conf}$, at most one of the following cases holds:*
  - *$C \in \mathsf{VConf}$,*
  - *there exist unique an evaluation context $\mathcal{C}$, a rule (r) and a redex R^C (instance of the left-hand side of rule (r)) such that $\mathcal{C}\left[R^C\right] = C$,*
  - *there exist unique an evaluation context $\mathcal{C}$, and a variable x such that $\mathcal{C}[x] = C$ and $x \notin \mathsf{HB}(\mathcal{C})$.*

Proof. *By induction on the structure of M and C, respectively. We use the unique decomposition property (i) we assume for the core language.* □

Determinacy follows from this theorem as a corollary.

Corollary 5.2 (Determinacy)

- *Given M, there exists at most one $M' \in \mathsf{Mod}$ s.t. $M \longrightarrow M'$;*
- *given C, there exists at most one $C' \in \mathsf{Conf}$ s.t. $C \longrightarrow C'$.*

Theorem 5.3 (Progress)

(A) *If $\vdash_M M : [\pi^\iota; \pi^o]$ and $M \notin \mathsf{VMod}$, then there exists M' s.t. $M \longrightarrow M'$;*

(B) *if $\vdash_C C : ([\pi^\iota; \pi^o], \tau)$ and $C \notin \mathsf{VConf}$, then one of the following cases holds:*

- *there exists C' s.t. $C \longrightarrow C'$;*
- *$C \longrightarrow \mathsf{err}(X : \tau, \pi^o)$ with $X : \tau \in \pi^\iota$.*

Proof. *See the appendix.* □

Corollary 5.4 *If $\vdash_C C : ([\emptyset; \pi^o], \tau)$ and $C \notin \mathsf{VConf}$, then there exists C' s.t. $C \longrightarrow C'$.*

Theorem 5.5 (Subject reduction)

(A) *If $\vdash_M M : [\pi^\iota; \pi^o]$ and $M \longrightarrow M'$, then $\vdash_M M' : [\pi^\iota; \pi^o]$;*

(B) *if $\vdash_C C : ([\pi^\iota; \pi^o], \tau)$ and $C \longrightarrow C'$, then there exists $\pi^{\iota\prime} \subseteq \pi^\iota$ such that $\vdash_C C' : ([\pi^{\iota\prime}; \pi^o], \tau)$.*

Proof. *See the appendix.* □

6 Conclusion

We have extended the calculus with lazy module operators CMS^{ℓ} [4] by adding a lazier low priority link operator which allows the user to have some control on dynamic configuration steps directly in the code to be executed; for instance, by using control variables it is possible to decide which version of the code should be dynamically linked for a given component.

Soundness is ensured by a combination of a static type system, which prevents errors in applying module operators, and a dynamic check which raises a linkage error if the running program needs a component which cannot be provided by reconfiguration steps. In particular no linkage errors can be raised if *all* components are potentially available.

This work is part of a stream of research [3,4,5,11] whose aim is the development of foundational calculi providing an abstract framework for dynamic software reconfiguration. In particular, the possibility of extending module calculi with selection on open modules, interleaving of component evaluation with reconfiguration steps and a lazy strategy has been firstly explored in [4]. In [5] we have investigated how to increase flexibility in a different direction, that is, by introducing *virtual* module components and higher-order configurations. Fagorzi's thesis [11] provides a comprehensive presentation of our results.

One of the main motivation for $CMS^{\ell,\ell-}$ is the need for foundational calculi providing an abstract framework for dynamic reconfiguration (that is, interleaving of reconfiguration steps and execution steps). Indeed, though the area of unanticipated software evolution continues attracting large interest, with its foundations studied in, e.g., [15], there is a little amount of work at our knowledge going toward the development of abstract models for dynamic reconfiguration, analogous to those which exist for the static case, where the configuration phase always precedes execution [8,16,6]. Apart from the wide literature concerning concrete dynamic linking mechanisms in existing programming environments [9,10], we mention [7], which presents a simple calculus modeling dynamic software updating, where modules are just records, many versions of the same module may coexist and update is modeled by an external transition which can be enforced by an `update` primitive in code, [1], where dynamic linking is studied as the programming language counterpart to the axiom of choice, and the module system defined in [14], where static linking, dynamic linking and cross-computation communication are all defined in a uniform framework.

Further work includes the investigation on the expressive power of lazy module calculi, by showing which kind of real-world reconfiguration mechanisms can be modeled and which kind require a richer model, and the introduction of more powerful mechanisms allowing the running program to control reconfiguration in a more direct way .

References

[1] Abadi, M., G. Gonthier and B. Werner, *Choice in dynamic linking*, in: *FOSSACS'04 - Foundations of Software Science and Computation Structures 2004*, Lecture Notes in Computer Science (2004), pp. 12–26.

[2] Ancona, D., S. Fagorzi, E. Moggi and E. Zucca, *Mixin modules and computational effects*, in: J. C. M. Baeten et al., editors, *International Colloquium on Automata, Languages and Programming 2003*, number 2719 in Lecture Notes in Computer Science (2003), pp. 224–238.

[3] Ancona, D., S. Fagorzi and E. Zucca, *A calculus for dynamic linking*, in: C. Blundo and C. Laneve, editors, *Italian Conf. on Theoretical Computer Science 2003*, number 2841 in Lecture Notes in Computer Science, 2003, pp. 284–301.

[4] Ancona, D., S. Fagorzi and E. Zucca, *A calculus with lazy module operators*, in: J.-J. Levy, E. W. Mayr and J. C. Mitchell, editors, *TCS 2004 (IFIP Int. Conf. on Theoretical Computer Science)* (2004), pp. 423–436.

[5] Ancona, D., S. Fagorzi and E. Zucca, *Mixin modules for dynamic rebinding*, in: *TGC 2005 -Symposium on Trustworthy Global Computing*, Lecture Notes in Computer Science (2005), to appear.

[6] Ancona, D. and E. Zucca, *A calculus of module systems*, Journ. of Functional Programming **12** (2002), pp. 91–132.

[7] Bierman, G., M. Hicks, P. Sewell and G. Stoyle, *Formalizing dynamic software updating (Extended Abstract)*, in: *USE'03 - the Second International Workshop on Unanticipated Software Evolution*, 2003.

[8] Cardelli, L., *Program fragments, linking, and modularization*, in: *ACM Symp. on Principles of Programming Languages 1997* (1997), pp. 266–277.

[9] Drossopoulou, S., *Towards an abstract model of Java dynamic linking and verfication*, in: R. Harper, editor, *TIC'00 - Third Workshop on Types in Compilation (Selected Papers)*, Lecture Notes in Computer Science **2071** (2001), pp. 53–84.

[10] Drossopoulou, S., G. Lagorio and S. Eisenbach, *Flexible models for dynamic linking*, in: P. Degano, editor, *ESOP 2003 - European Symposium on Programming 2003*, 2003, pp. 38–53.

[11] Fagorzi, S., "Module Calculi for Dynamic Reconfiguration," Ph.D. thesis, Dipartimento di Informatica e Scienze dell'Informazione, Università di Genova (2005), to appear.

[12] Fagorzi, S. and E. Zucca, *A case-study in encoding configuration languages: Multiple class loaders*, Journ. of Object Technology **3** (2004), pp. 31–53.

[13] Hirschowitz, T. and X. Leroy, *Mixin modules in a call-by-value setting*, in: D. L. Métayer, editor, *ESOP 2002 - European Symposium on Programming 2002*, number 2305 in Lecture Notes in Computer Science (2002), pp. 6–20.

[14] Liu, Y. D. and S. F. Smith, *Modules with interfaces for dynamic linking and communication*, in: M. Odersky, editor, *ECOOP 2004 - Object-Oriented Programming*, number 3086 in Lecture Notes in Computer Science (2004), pp. 414–439.

[15] Mens, T. and G. Kniesel, *Workshop on foundations of unanticipated software evolution* (2004), eTAPS 2004, http://joint.org/fuse2004/.

[16] Wells, J. and R. Vestergaard, *Confluent equational reasoning for linking with first-class primitive modules*, in: *ESOP 2000 - European Symposium on Programming 2000*, number 1782 in Lecture Notes in Computer Science (2000), pp. 412–428.

A Results and proofs

In this section we collect the proofs of results stated in Sec.5.

Lemma A.1 *If* $C \longrightarrow \mathsf{err}(X : \tau, \pi)$*, then we have that:*

(i) *C has one of the following forms:*
- $C \equiv < [\iota;\, o;\, \rho]\,,\ \mathcal{E}\,[x] >$ *with* $x \notin \mathsf{HB}\,(\mathcal{E})$, $X : \tau = \iota(x)$ *and* $\pi = o_{\mathsf{Type}}$*;*
- $C \equiv \mathsf{link}^{-}_{\sigma} C'$ *with* $C' \longrightarrow \mathsf{err}(X : \tau, \pi)$ *and either* $X \notin \mathsf{dom}(\sigma)$, $\sigma(X) \notin \mathsf{dom}(\pi)$ *or* $\pi(\sigma(X)) \neq \tau$.

(ii) *if* $\vdash_C C : ([\pi^\iota;\, \pi^o]\,,\, \tau)$*, then*
- $X : \tau \in \pi^\iota$,
- $\pi = \pi^o$.

Proof.

(i) *Immediate from the definition of the reduction relation.*

(ii) *Easy induction on typing rules, with case analysis on the structure of C (exploiting the first point of this lemma).*

□

Theorem 5.3 (Progress)

(A) If $\vdash_M M : [\pi^\iota;\, \pi^o]$ and $M \notin \mathsf{VMod}$, then there exists M' s.t. $M \longrightarrow M'$;

(B) if $\vdash_C C : ([\pi^\iota;\, \pi^o]\,,\, \tau)$ and $C \notin \mathsf{VConf}$, then one of the following cases holds:
- there exists C' s.t. $C \longrightarrow C'$;
- $C \longrightarrow \mathsf{err}(X : \tau, \pi^o)$ with $X : \tau \in \pi^\iota$.

Proof.
We rewrite the progress property in the following form.

(A) If $\vdash_M M : [\pi^\iota;\, \pi^o]$ and $M \notin \mathsf{VMod}$, then $M \equiv \mathcal{M}\left[R^M\right]$ for some R^M and there exists M' s.t. $R^M \longrightarrow M'$;

(B) if $\vdash_C C : ([\pi^\iota; \pi^o], \tau)$ and $C \notin \mathsf{VConf}$, then one of the following cases holds:

- $C \equiv \mathcal{C}\left[R^C\right]$ for some R^C and there exists C' s.t. $R^C \longrightarrow C'$;
- $C \equiv \mathcal{C}\mathcal{M}\left[R^M\right]$ for some R^M and there exists M' s.t. $R^M \longrightarrow M'$;
- $C \longrightarrow \mathsf{err}(X : \tau, \pi)$.

We now separately prove the two facts.

(A) Induction on typing rules:

(M-basic): we do not consider this rule since in the conclusion we have $[\iota; o; \rho] \in \mathsf{VMod}$.

(M-sum): we derive $\vdash_M M_1 + M_2 : [\pi^\iota{}_1 \cup \pi^\iota{}_2; \pi^o{}_1 \cup \pi^o{}_2]$. There are two cases to be considered:

- $M_1 \notin \mathsf{VMod}$. In this case, by applying the inductive hypothesis to the first premise of the typing rule (M-sum), that is $\vdash_M M_1 : [\pi^\iota{}_1; \pi^o{}_1]$, we have that $M_1 \equiv \mathcal{M}\left[R^M\right]$ for some R^M and there exists M' s.t. $R^M \longrightarrow M'$. Hence, we can conclude by observing that $\mathcal{M} + M_2 \in \mathcal{M}\mathsf{Ctx}$;
- $M_1 \in \mathsf{VMod}$, that is $M_1 \equiv [\iota_1; o_1; \rho_1]$. There are two subcases:
  - $M_2 \notin \mathsf{VMod}$. In this case, by applying the inductive hypothesis to the second premise of the typing rule (M-sum), that is $\vdash_M M_2 : [\pi^\iota{}_2; \pi^o{}_2]$, we have that $M_2 \equiv \mathcal{M}\left[R^M\right]$ for some R^M and there exists M' s.t. $R^M \longrightarrow M'$. Hence, we can conclude by observing that $[\iota_1; o_1; \rho_1] + \mathcal{M} \in \mathcal{M}\mathsf{Ctx}$;
  - $M_2 \in \mathsf{VMod}$, that is $M_2 \equiv [\iota_2; o_2; \rho_2]$. In this case, we have: $[\iota_1; o_1; \rho_1] + [\iota_2; o_2; \rho_2] \xrightarrow{(M\text{-sum})} [\iota_1, \iota_2; o_1, o_2; \rho_1, \rho_2]$. Note that we can perform this reduction step since all (implicit and explicit) side-conditions are satisfied: surely $\mathsf{FV}([\iota_i; o_i; \rho_i]) = \emptyset$, $i \in \{1, 2\}$ (since for the two premises of typing rule (M-sum) the two basic modules are well-typed in the empty context); all assignments have disjoint domains (for the input and local assignments this can be obtained by α-conversion, while for the output assignment this is ensured by the side-condition of the typing rule (M-sum)); also the type coherence requirement on the input assignment is satisfied (from well-formedness of the input signature $\pi^\iota{}_1 \cup \pi^\iota{}_2$ in the resulting type of (M-sum)).

(M-reduct): we derive $\vdash_M {}_{\sigma^\iota}|M|_{\sigma^o} : [\tilde{\pi}^\iota \cup \pi^\iota \setminus \mathsf{dom}(\sigma^\iota); \tilde{\pi}^o]$. There are two cases to be considered:

- $M \notin \mathsf{VMod}$. In this case, by applying the inductive hypothesis to the premise of the typing rule (M-reduct), that is, $\vdash_M M : [\pi^\iota; \pi^o]$, we have that $M \equiv \mathcal{M}\left[R^M\right]$ for some R^M and there exists M' s.t. $R^M \longrightarrow M'$. Hence, we can conclude by observing that ${}_{\sigma^\iota}|\mathcal{M}|_{\sigma^o} \in \mathcal{M}\mathsf{Ctx}$;
- $M \in \mathsf{VMod}$, that is $M \equiv [\iota; o; \rho]$ and from the premise and the first side-

condition of the typing rule (M-reduct) we have that $\vdash_M [\iota; o; \rho] : [\pi^\iota; \pi^o]$ and $\sigma^\iota|_{\mathsf{dom}(\pi^\iota)} : \pi^\iota|_{\mathsf{dom}(\sigma)} \to \tilde{\pi}^\iota$. Choosing a partition of ι into ι', ι'' such that $\mathsf{cod}(\iota''_{\mathsf{Name}}) \cap \mathsf{dom}(\sigma^\iota) = \emptyset$ and $\mathsf{cod}(\iota'_{\mathsf{Name}}) \subseteq \mathsf{dom}(\sigma^\iota)$, we have that: ${}_{\sigma^\iota}|[\iota', \iota''; o; \rho]|_{\sigma^o} \xrightarrow{(M\text{-reduct})} [\sigma^\iota \circ_{\mathsf{Name}} \iota', \iota''; o \circ \sigma^o; \rho]$. Note that we can perform this step since all side-conditions are satisfied: first two conditions are obviously satisfied by the chosen partition ι', ι''; the third condition $\mathsf{cod}(\sigma^o) \subseteq \mathsf{dom}(o)$ is satisfied (from the second side-condition of the typing rule (M-reduct) we have that $\sigma^o : \tilde{\pi}^o \to \pi^o$); also the type coherence requirement on the input assignment is satisfied (from well-formedness of the input signature $\tilde{\pi}^\iota \cup \pi^\iota \backslash \mathsf{dom}(\sigma^\iota)$ in the resulting type of (M-reduct)).

(M-link): we derive $\vdash_M \mathsf{link}_\sigma M : [\pi^\iota \backslash \mathsf{dom}(\sigma); \pi^o]$. There are two cases to be considered:

- $M \notin \mathsf{VMod}$. In this case, by applying the inductive hypothesis to the premise of the typing rule (M-link), that is, $\vdash_M M : [\pi^\iota; \pi^o]$, we have that $M \equiv \mathcal{M}\left[R^M\right]$ for some R^M and there exists M' s.t. $R^M \longrightarrow M'$. Hence, we can conclude by observing that $\mathsf{link}_\sigma \mathcal{M} \in \mathcal{M}\mathsf{Ctx}$;
- $M \in \mathsf{VMod}$, that is $M \equiv [\iota; o; \rho]$. From the premise and the side-condition of the typing rule (M-link) we have that $\vdash_M [\iota; o; \rho] : [\pi^\iota; \pi^o]$ and $\sigma|_{\mathsf{dom}(\pi^\iota)} : \pi^\iota|_{\mathsf{dom}(\sigma)} \to \pi^o$. Choosing a partition of ι into ι', ι'', with $\iota' = x_i \overset{i\in I}{\mapsto} X_i : \tau_i$, such that $\mathsf{cod}(\iota''_{\mathsf{Name}}) \cap \mathsf{dom}(\sigma) = \emptyset$ and $\{X_i \mid i \in I\} \subseteq \mathsf{dom}(\sigma)$, we have that: $\mathsf{link}_\sigma\left[x_i \overset{i\in I}{\mapsto} X_i : \tau_i, \iota''; o; \rho\right] \xrightarrow{(M\text{-link})} \left[\iota''; o; \rho, x_i \overset{i\in I}{\mapsto} o_{\mathsf{Exp}}(\sigma(X_i))\right]$. Note that we can perform this step since all side-conditions are satisfied: first and second side-conditions are obviously satisfied by the chosen partition ι', ι''; and also the condition $\{\sigma(X_i) \mid i \in I\} \subseteq \mathsf{dom}(o)$ is satisfied (from the side-condition of the typing rule (M-link) we have that $\sigma|_{\mathsf{dom}(\pi^\iota)} : \pi^\iota|_{\mathsf{dom}(\sigma)} \to \pi^o$ and $\{X_i \mid i \in I\} \subseteq \mathsf{dom}(\sigma)$).

(B) Induction on typing rules; we use Lemma A.1 and the first part of this theorem:

(C-basic): we derive $\vdash_C < [\iota; o; \rho]\,,\ e\ >: ([\pi^\iota; \pi^o]\,, \tau)$. There are two cases to be considered:

- $e \in \mathsf{CVal}$: this case is impossible since for hypothesis $< [\iota; o; \rho]\,,\ e\ > \notin \mathsf{VConf}$;
- $e \notin \mathsf{CVal}$: from the premise $x_h : \tau_h{}^{h\in I\cup L} \vdash_e e : \tau$ of the typing rule (C-basic), with $\{x_h \mid h \in I\} = \mathsf{dom}(\iota)$ and $\{x_h \mid h \in L\} = \mathsf{dom}(\rho)$ and from the assumption 5.1 (ii) on the core language, that is, the progress property, we get that one of the following two cases holds:
  - $e \equiv \mathcal{E}\left[R^e\right]$ for some R^e and there exists e' s.t. $R^e \xrightarrow[e]{} e'$. In this case, we have that $< [\iota; o; \rho]\,,\ \mathcal{E}\left[R^e\right] > \xrightarrow{(\text{core})} < [\iota; o; \rho]\,,\ \mathcal{E}\left[e'\right] >$;

· $e \equiv \mathcal{E}\,[x]$ with $x \notin \mathsf{HB}\,(\mathcal{E})$ and $x \in \mathsf{dom}(\iota) \cup \mathsf{dom}(\rho)$:
  ◦ if $x \in \mathsf{dom}(\rho)$, then $< [\iota;\, o;\, \rho]\,,\ \mathcal{E}\,[x] > \xrightarrow{\text{(var)}} < [\iota;\, o;\, \rho]\,,\ \mathcal{E}\{\rho(x)\} >$;
  ◦ if $x \in \mathsf{dom}(\iota)$, then $< [\iota;\, o;\, \rho]\,,\ \mathcal{E}\,[x] > \xrightarrow{\text{(var/err)}} \mathsf{err}(\iota(x), o_{\mathsf{Type}})$.

(C-sum): we derive $\vdash_C C + M : ([\pi^\iota{}_C \cup \pi^\iota{}_M;\, \pi^o{}_C \cup \pi^o{}_M]\,,\, \tau)$. We suppose $C \notin \mathsf{VConf}$ (otherwise we would have $C + M \in \mathsf{VConf}$). Applying the inductive hypothesis to the first premise of the typing rule, that is $\vdash_C C : ([\pi^\iota{}_C;\, \pi^o{}_C]\,,\, \tau)$, we have that one of the following three cases holds:
- $C \equiv \mathcal{C}\left[R^C\right]$ for some R^C and there exists C' such that $R^C \longrightarrow C'$. In this case, we can conclude by observing that $\mathcal{C} + M \in \mathcal{C}\mathsf{Ctx}$;
- $C \equiv \mathcal{C}\mathcal{M}\left[R^M\right]$ for some R^M and there exists M' such that $R^M \longrightarrow M'$. In this case, we can conclude by observing that $\mathcal{C}\mathcal{M} + M \in \mathcal{C}\mathcal{M}\mathsf{Ctx}$;
- $C \longrightarrow \mathsf{err}(X : \tau, \pi)$. By applying Lemma A.1 we get that C has one of the following two forms:
  · if $C \equiv < [\iota_1;\, o_1;\, \rho_1]\,,\ \mathcal{E}\,[x] >$, then we have to consider the following two subcases:
    ◦ $M \notin \mathsf{VMod}$, then applying the first part of this theorem we have that $M \equiv \mathcal{M}\left[R^M\right]$ for some R^M and there exists M' s.t. $R^M \longrightarrow M'$. Hence, we have that $C + M \xrightarrow{\text{(sum-closure)}} C + \mathcal{M}\,[M']$;
    ◦ $M \in \mathsf{VMod}$, that is, $M \equiv [\iota_2;\, o_2;\, \rho_2]$, then $< [\iota_1;\, o_1;\, \rho_1]\,,\ \mathcal{E}\,[x] > + [\iota_2;\, o_2;\, \rho_2] \xrightarrow{\text{(sum/basic)}} < [\iota_1,\, \iota_2;\, o_1,\, o_2;\, \rho_1,\, \rho_2]\,,\ \mathcal{E}\,[x] >$. Note that we can perform this reduction step since all side-conditions are satisfied (similarly to the case (M-sum) seen before);
  · $C \equiv \mathsf{link}^-_\sigma C'$, then we have that $\mathsf{link}^-_\sigma C' + M \xrightarrow{\text{(sum/link}^-\text{)}} \mathsf{link}^-_\sigma (C' + M)$.

(C-reduct): we derive $\vdash_C {}_{\sigma^\iota}|C|_{\sigma^o} : ([\tilde{\pi}^\iota \cup \pi^\iota \backslash \mathsf{dom}(\sigma^\iota);\, \tilde{\pi}^o]\,,\, \tau)$. We suppose $C \notin \mathsf{VConf}$ (otherwise we would have ${}_{\sigma^\iota}|C|_{\sigma^o} \in \mathsf{VConf}$). Applying the inductive hypothesis to the premise of the typing rule (C-reduct), that is $\vdash_C C : ([\pi^\iota{}_1;\, \pi^o]\,,\, \tau)$, we have that one of the following three cases holds:
- $C \equiv \mathcal{C}\left[R^C\right]$ for some R^C and there exists C' s.t. $R^C \longrightarrow C'$. In this case, we can conclude by observing that ${}_{\sigma^\iota}|\mathcal{C}|_{\sigma^o} \in \mathcal{C}\mathsf{Ctx}$;
- $C \equiv \mathcal{C}\mathcal{M}\left[R^M\right]$ for some R^M and there exists M' s.t. $R^M \longrightarrow M'$. In this case, we can conclude by observing that ${}_{\sigma^\iota}|\mathcal{C}\mathcal{M}|_{\sigma^o} \in \mathcal{C}\mathcal{M}\mathsf{Ctx}$;
- $C \longrightarrow \mathsf{err}(X : \tau, \pi)$, then, using the Lemma A.1, we obtain that C has one of the following two forms:
  · $C \equiv < [\iota;\, o;\, \rho]\,,\ \mathcal{E}\,[x] >$, then
  ${}_{\sigma^\iota}|< [\iota;\, o;\, \rho]\,,\ \mathcal{E}\,[x] >|_{\sigma^o} \xrightarrow{\text{(reduct/basic)}} < [\sigma^\iota \circ_{\mathsf{Name}} \iota, \iota';\, o \circ \sigma^o;\, \rho]\,,\ \mathcal{E}\,[x] >$. Note that we can perform this reduction step since all side-conditions are

satisfied (similarly to the case (M-reduct) seen before).

· $C \equiv \mathsf{link}^-_\sigma C'$, then we have that ${}_{\sigma^\iota}|\mathsf{link}^-_\sigma C'|_{\sigma^o} \xrightarrow{(\text{reduct/link}^-)} \mathsf{link}^-_\sigma({}_{\sigma^\iota}|C'|_{\sigma^o})$.

(C-link): we derive $\vdash_C \mathsf{link}_\sigma C : ([\pi^\iota \backslash \mathsf{dom}(\sigma); \pi^o], \tau)$. We suppose $C \notin \mathsf{VConf}$ (otherwise we would have $\mathsf{link}_\sigma C \in \mathsf{VConf}$). Applying the inductive hypothesis to the premise of the typing rule (C-link), that is $\vdash_C C : ([\pi^\iota; \pi^o], \tau)$, we have that one of the following three cases holds:

- $C \equiv \mathcal{C}\left[R^C\right]$ for some R^C and there exists C' s.t. $R^C \longrightarrow C'$. In this case, we can conclude by observing that $\mathsf{link}_\sigma \mathcal{C} \in \mathcal{C}\mathsf{Ctx}$;
- $C \equiv \mathcal{C}\mathcal{M}\left[R^M\right]$ for some R^M and there exists M' s.t. $R^M \longrightarrow M'$. In this case, we can conclude by observing that $\mathsf{link}_\sigma \mathcal{C}\mathcal{M} \in \mathcal{C}\mathcal{M}\mathsf{Ctx}$;
- $C \longrightarrow \mathsf{err}(X : \tau, \pi)$, then, using the Lemma A.1, we have that C has one of the following two forms:
  · $C \equiv < [\iota; o; \rho], \mathcal{E}[x] >$, then
  $\mathsf{link}_\sigma < [\iota; o; \rho], \mathcal{E}[x] > \xrightarrow{(\text{link/basic})} < \left[\iota; o; \rho, x_i \overset{i \in I}{\mapsto} o_{\mathsf{Exp}}(\sigma(X_i))\right], \mathcal{E}[x] >$.
  Note that we can perform this reduction step since all side-conditions are satisfied (similarly to the case (M-link) above).
  · $C \equiv \mathsf{link}^-_{\sigma'} C'$, then we have that $\mathsf{link}_\sigma\left(\mathsf{link}^-_{\sigma'} C'\right) \xrightarrow{(\text{link/link}^-)} \mathsf{link}^-_{\sigma'}(\mathsf{link}_\sigma C')$.

(C-link$^-$): we derive $\vdash_C \mathsf{link}^-_\sigma C : ([\pi^\iota; \pi^o], \tau)$. Note that since $\mathsf{link}^-_\sigma C$ is well-formed we surely have that $\sigma \neq \emptyset$. We suppose $C \notin \mathsf{VConf}$, (otherwise we would have $\mathsf{link}^-_\sigma C \in \mathsf{VConf}$). Applying the inductive hypothesis to the premise of the typing rule (C-link$^-$), that is $\vdash_C C : ([\pi^\iota; \pi^o], \tau)$, we have that one of the following three cases holds:

- $C \equiv \mathcal{C}\left[R^C\right]$ for some R^C and there exists C' s.t. $R^C \longrightarrow C'$. In this case, we can conclude by observing that $\mathsf{link}^-_\sigma \mathcal{C} \in \mathcal{C}\mathsf{Ctx}$;
- $C \equiv \mathcal{C}\mathcal{M}\left[R^M\right]$ for some R^M and there exists M' s.t. $R^M \longrightarrow M'$. In this case, we can conclude by observing that $\mathsf{link}^-_\sigma \mathcal{C}\mathcal{M} \in \mathcal{C}\mathcal{M}\mathsf{Ctx}$;
- $C \longrightarrow \mathsf{err}(X : \tau, \pi)$, then, there are two cases to be considered:
  · if $X \in \mathsf{dom}(\sigma)$ and $\pi(\sigma(X)) = \tau$, writing $\mathsf{link}^-_\sigma C$ as $\mathsf{link}^-_{\sigma\backslash\{X\}, X \mapsto \sigma(X)} C$, we have that $\mathsf{link}^-_{\sigma\backslash\{X\}, X \mapsto \sigma(X)} C \xrightarrow{(\text{link}^-)} \mathsf{link}^-_{\sigma\backslash\{X\}}\left(\mathsf{link}_{X \mapsto \sigma(X)} C\right)$, where $\mathsf{link}^-_{\sigma\backslash\{X\}}\left(\mathsf{link}_{X \mapsto \sigma(X)} C\right)$ is identified with $\mathsf{link}_{X \mapsto \sigma(X)} C$ if $\sigma \backslash \{X\} = \emptyset$;
  · if $X \notin \mathsf{dom}(\sigma)$ or $\sigma(X) \notin \mathsf{dom}(\pi)$ or $\pi(\sigma(X)) \neq \tau$, then we have that $\mathsf{link}^-_\sigma C \xrightarrow{(\text{link}^-/\text{err})} \mathsf{err}(X : \tau, \pi)$.

(sel): we derive $\vdash_C M.X : ([\pi^\iota; \pi^o], \pi^o(X))$. There are two cases to be considered:

- $M \equiv \mathcal{M}\left[R^M\right]$ for some R^M and there exists M' s.t. $R^M \longrightarrow M'$. Hence,

we can conclude by observing that $\mathcal{M}.X \in \mathcal{CM}\mathsf{Ctx}$;

- $M \in \mathsf{VMod}$, that is $M \equiv [\iota; o; \rho]$. In this case we have that $[\iota; o; \rho].X \xrightarrow{(\text{sel})} < [\iota; o; \rho]\,,\, o(X) >$. Note that we surely have $X \in \mathsf{dom}(o)$ (from well-formedness of $\pi^o(X)$ in the conclusion of the typing rule (sel)).

□

Theorem 5.3 (Subject reduction)

(A) If $\vdash_M M : [\pi^\iota; \pi^o]$ and $M \longrightarrow M'$, then $\vdash_M M' : [\pi^\iota; \pi^o]$;

(B) if $\vdash_C C : ([\pi^\iota; \pi^o]\,,\, \tau)$ and $C \longrightarrow C'$, then there exist $\pi^{\iota\prime} \subseteq \pi^\iota$ such that $\vdash_C C' : ([\pi^{\iota\prime}; \pi^o]\,,\, \tau)$.

Proof. Both the two facts are proved by induction on reduction rules.

(A)

($\mathcal{M}$-ctx): we derive $\mathcal{M}\left[R^M\right] \longrightarrow \mathcal{M}\left[M\right]$. In this case we proceed by case analysis on the structure of $\mathcal{M}$ and in all cases we can conclude by applying the inductive hypothesis.

(M-sum): we derive $[\iota_1; o_1; \rho_1] + [\iota_2; o_2; \rho_2] \longrightarrow [\iota_1, \iota_2; o_1, o_2; \rho_1, \rho_2]$. We suppose $\vdash_M [\iota_1; o_1; \rho_1] + [\iota_1; o_1; \rho_1] : [\pi^\iota; \pi^o]$. This judgment can only be derived by using rule (M-sum), hence it must be:

- $[\pi^\iota; \pi^o] = [\pi^\iota{}_1 \cup \pi^\iota{}_2; \pi^o{}_1 \cup \pi^o{}_2]$;
- $\vdash_M [\iota_1; o_1; \rho_1] : [\pi^\iota{}_1; \pi^o{}_1]$; (1)
- $\vdash_M [\iota_2; o_2; \rho_2] : [\pi^\iota{}_2; \pi^o{}_2]$; (2)

Both judgment (1) and (2) can only be derived by using rule (M-basic). Hence, from (1) we have that it must be:

- $[\pi^\iota{}_1; \pi^o{}_1] = \left[X_i : \tau_i{}^{i\in I_1}; X_j : \tau_j{}^{j\in O_1}\right]$;
- $[\iota_1; o_1; \rho_1] = \left[x_i \overset{i\in I_1}{\mapsto} X_i : \tau_i; X_j \overset{j\in O_1}{\mapsto} e_j : \tau_j; x_l \overset{l\in L_1}{\mapsto} e_l\right]$;
- $\left\{x_h : \tau_h{}^{h\in I_1\cup L_1} \vdash_e e_k : \tau_k \mid k \in O_1 \cup L_1\right\}$; (1a)
- $\vdash \left[X_i : \tau_i{}^{i\in I_1}; X_j : \tau_j{}^{j\in O_1}\right]$. (1b)

And similarly for (2):

- $[\pi^\iota{}_2; \pi^o{}_2] = \left[X_i : \tau_i{}^{i\in I_2}; X_j : \tau_j{}^{j\in O_2}\right]$;
- $[\iota_2; o_2; \rho_2] = \left[x_i \overset{i\in I_2}{\mapsto} X_i : \tau_i; X_j \overset{j\in O_2}{\mapsto} e_j : \tau_j; x_l \overset{l\in L_2}{\mapsto} e_l\right]$;
- $\left\{x_h : \tau_h{}^{h\in I_2\cup L_2} \vdash_e e_k : \tau_k \mid k \in O_2 \cup L_2\right\}$; (2a)
- $\vdash \left[X_i : \tau_i{}^{i\in I_2}; X_j : \tau_j{}^{j\in O_2}\right]$. (2b)

Hence, we get that:

- $[{\pi^\iota}_1 \cup {\pi^\iota}_2;\ {\pi^o}_1 \cup {\pi^o}_2] = \left[X_i : {\tau_i}^{i\in I_1\cup I_2};\ X_j : {\tau_j}^{j\in O_1\cup O_2}\right]$ and from (1b), (2b) and well-formedness of the two (compatible) unions in π^ι and π^o we have $\vdash \left[X_i : {\tau_i}^{i\in I_1\cup I_2};\ X_j : {\tau_j}^{j\in O_1\cup O_2}\right]$. (5)
- $[\iota_1, \iota_2;\ o_1, o_2;\ \rho_1, \rho_2] = \left[x_i \overset{i\in I_1\cup I_2}{\mapsto} X_i : \tau_i;\ X_j \overset{j\in O_1\cup O_2}{\mapsto} e_j : \tau_j;\ x_l \overset{l\in L_1\cup L_2}{\mapsto} e_l\right]$.

By applying to all judgments in (1a) and (2a) the core assumption 5.1 (v), that is, the weakening property, we obtain:
$\left\{x_h : {\tau_h}^{h\in I_1\cup L_1\cup I_2\cup L_2} \vdash_e e_k : \tau_k \mid k \in O_1 \cup L_1 \cup O_2 \cup L_2\right\}$. (6)
Note that $x_h : {\tau_h}^{h\in I_1\cup L_1\cup I_2\cup L_2}$ is well-formed since if we perform the step (M-sum) it implicitly holds that $\mathsf{dom}(\iota_1,\ \rho_1) \cap \mathsf{dom}(\iota_2,\ \rho_2) = \emptyset$.
We can obtain the following derivation:

$$\frac{(5) \quad (6)}{\vdash_M [\iota_1, \iota_2;\ o_1, o_2;\ \rho_1, \rho_2] : [{\pi^\iota}_1 \cup {\pi^\iota}_2;\ {\pi^o}_1 \cup {\pi^o}_2]}\ (M\text{-basic})$$

(M-reduct): we derive ${}_{\sigma^\iota}|[\iota, \iota';\ o;\ \rho]|_{\sigma^o} \longrightarrow [\sigma^\iota \circ_{\mathsf{Name}} \iota, \iota';\ o \circ \sigma^o;\ \rho]$. We suppose $\vdash_M {}_{\sigma^\iota}|[\iota, \iota';\ o;\ \rho]|_{\sigma^o} : [{\pi^\iota}_M;\ {\pi^o}_M]$. This judgment can only be derived by using rule (M-reduct), hence it must be:

- $[{\pi^\iota}_M;\ {\pi^o}_M] = [\tilde{\pi}^\iota \cup \pi^\iota \backslash \mathsf{dom}(\sigma^\iota);\ \tilde{\pi}^o]$;
- $\vdash_M [\iota, \iota';\ o;\ \rho] : [\pi^\iota;\ \pi^o]$; (1)
- $\sigma^\iota|_{\mathsf{dom}(\pi^\iota)} : \pi^\iota|_{\mathsf{dom}(\sigma)} \to \tilde{\pi}^\iota$; (2)
- $\sigma^o : \tilde{\pi}^o \to \pi^o$; (3)

Judgment (1) can only be derived by using rule (M-basic), so, it must be:

- $[\iota, \iota';\ o;\ \rho] = \left[x_i \overset{i\in I}{\mapsto} X_i : \tau_i;\ X_j \overset{j\in O}{\mapsto} e_j : \tau_j;\ x_l \overset{l\in L}{\mapsto} e_l\right]$;
- $[\pi^\iota;\ \pi^o] = \left[X_i : {\tau_i}^{i\in I};\ X_j : {\tau_j}^{j\in O}\right]$;
- $\left\{x_h : {\tau_h}^{h\in I\cup L} \vdash_e e_k : \tau_k \mid k \in O \cup L\right\}$; (1a)
- $\vdash \left[X_i : {\tau_i}^{i\in I};\ X_j : {\tau_j}^{j\in O}\right]$. (1b)

We split I in into I_1 and I_2 such that $\iota = x_i \overset{i\in I_1}{\mapsto} X_i : \tau_i$ and $\iota' = x_i \overset{i\in I_2}{\mapsto} X_i : \tau_i$. Hence, we have that:

- $X_i : {\tau_i}^{i\in I_1} = \pi^\iota|_{\mathsf{dom}(\sigma^\iota)}$, from the (implicit) side-condition $\mathsf{cod}(\iota_{\mathsf{Name}}) \subseteq \mathsf{dom}(\sigma^\iota)$ of reduction rule (M-reduct);
- $X_i : {\tau_i}^{i\in I_2} = \pi^\iota \backslash \mathsf{dom}(\sigma^\iota)$, from the side condition $\mathsf{cod}({\iota'}_{\mathsf{Name}}) \cap \mathsf{dom}(\sigma^\iota) = \emptyset$ of reduction rule (M-reduct).

Moreover, we observe that in $\tilde{\pi}^\iota$ and $\tilde{\pi}^o$ have the following forms:

- $\tilde{\pi}^\iota = \sigma^\iota(X_i) : {\tau_i}^{i\in I_1},\ X_i : {\tau_i}^{i\in F}$, where $\{X_i \mid i \in F\} = \mathsf{cod}(\sigma^\iota) \backslash \{\sigma^\iota(X_i) \mid i \in I_1\}$, which intuitively corresponds to the new names added to the input signature.
- $\tilde{\pi}^o = X_j : {\tau_j}^{j\in \tilde{O}}$, where $\{\sigma^o(X_j) \mid j \in \tilde{O}\} = \mathsf{cod}(\sigma^o)$, with $\tilde{O} \subseteq O$ since for the (implicit) side-condition of reduction rule (M-reduct) we have that $\mathsf{cod}(\sigma^o) \subseteq \mathsf{dom}(o)$.

Hence, we have that:

- $[\tilde{\pi}^\iota \cup \pi^\iota \backslash \mathsf{dom}(\sigma^\iota);\ \tilde{\pi}^o] = \left[\left(\sigma^\iota(X_i) : {\tau_i}^{i\in I_1},\ X_i : {\tau_i}^{i\in F}\right) \cup X_i : {\tau_i}^{i\in I_2};\ X_j : {\tau_j}^{j\in \tilde{O}}\right]$ and from (1b), the

properties (2) and (3) that $\sigma^\iota|_{\mathsf{dom}(\pi^\iota)}$ and σ^o preserve types and well-formedness of the (compatible) union in ${\pi^\iota}_M$, we have that $\vdash \left[\left(\sigma^\iota(X_i) : {\tau_i}^{i\in I_1},\, X_i : {\tau_i}^{i\in F}\right) \cup X_i : {\tau_i}^{i\in I_2};\, X_j : {\tau_j}^{j\in\tilde{O}}\right]$. (1c')

- $[\sigma^\iota \circ_{\mathsf{Name}} \iota,\, \iota';\, o \circ \sigma^o;\, \rho] =$
  $\left[x_i \overset{i\in I_1}{\mapsto} \sigma^\iota(X_i) : \tau_i, x_i^{\mathsf{f}} \overset{i\in F}{\mapsto} X_i : \tau_i, x_i \overset{i\in I_2}{\mapsto} X_i : \tau_i;\, X_j \overset{j\in\tilde{O}}{\mapsto} e_j : \tau_j;\, x_l \overset{l\in L}{\mapsto} e_l\right]$,
  where for all $i \in F$, x_i^{f} is a fresh variable.

We select from (1a) the following subset of judgments:
$\left\{x_h : {\tau_h}^{h\in I\cup L} \vdash_e e_k : \tau_k \mid k \in \tilde{O} \ \vee \ k \in L\right\}$. (1a')
We can now obtain the following derivation:

$$\dfrac{(1c') \qquad \dfrac{(1a')}{\left\{x_h : {\tau_h}^{h\in I\cup L\cup F} \vdash_e e_k : \tau_k \mid k \in \tilde{O} \ \vee \ k \in L\right\}}\,(\text{Weakening})}{\vdash_M [\sigma^\iota \circ_{\mathsf{Name}} \iota, \iota';\, o \circ \sigma^o;\, \rho] : [\tilde{\pi}^\iota \cup \pi^\iota \backslash \mathsf{dom}(\sigma^\iota);\, \tilde{\pi}^o]}\,(M\text{-basic})$$

(M-link): we derive $\mathsf{link}_\sigma\left[x_i \overset{i\in I_1}{\mapsto} X_i : \tau_i, \iota;\, o;\, \rho\right] \longrightarrow \left[\iota;\, o;\, \rho, x_i \overset{i\in I_1}{\mapsto} o_{\mathsf{Exp}}(\sigma(X_i))\right]$. We suppose $\vdash_M \mathsf{link}_\sigma\left[x_i \overset{i\in I_1}{\mapsto} X_i : \tau_i,\, \iota;\, o;\, \rho\right] : [{\pi^\iota}_M;\, {\pi^o}_M]$. This judgment can only be derived by using rule (M-link), hence it must be:

- $[{\pi^\iota}_M;\, {\pi^o}_M] = [\pi^\iota \backslash \mathsf{dom}(\sigma);\, \pi^o]$
- $\vdash_M \left[x_i \overset{i\in I_1}{\mapsto} X_i : \tau_i, \iota;\, o;\, \rho\right] : [\pi^\iota;\, \pi^o]$; (1)
- $\sigma|_{\mathsf{dom}(\pi^\iota)} : \pi^\iota|_{\mathsf{dom}(\sigma)} \to \pi^o$. (2)

Judgment (1) can only be derived by using rule (M-basic), so, it must be:

- $\left[x_i \overset{i\in I_1}{\mapsto} X_i : \tau_i, \iota;\, o;\, \rho\right] = \left[x_i \overset{i\in I}{\mapsto} X_i : \tau_i;\, X_j \overset{j\in O}{\mapsto} e_j : \tau_j;\, x_l \overset{l\in L}{\mapsto} e_l\right]$;
- $[\pi^\iota;\, \pi^o] = \left[X_i : {\tau_i}^{i\in I};\, X_j : {\tau_j}^{j\in O}\right]$
- $\left\{x_h : {\tau_h}^{h\in I\cup L} \vdash_e e_k : \tau_k \mid k \in O \cup L\right\}$; (1a)
- $\vdash \left[X_i : {\tau_i}^{i\in I};\, X_j : {\tau_j}^{j\in O}\right]$. (1b)

where ι has the form $x_i \overset{i\in I_2}{\mapsto} X_i : \tau_i$, with $I = I_1 \cup I_2$.
We observe that:

- $X_i : {\tau_i}^{i\in I_1} = \pi^\iota|_{\mathsf{dom}(\sigma)}$, from the (implicit) side-condition $\{X_i \mid i \in I_1\} \subseteq \mathsf{dom}(\sigma)$ of reduction rule (M-link);
- $X_i : {\tau_i}^{i\in I_2} = \pi^\iota \backslash \mathsf{dom}(\sigma)$, from the side-condition $\mathsf{cod}(\iota_{\mathsf{Name}}) \cap \mathsf{dom}(\sigma) = \emptyset$ of reduction rule (M-link).

Hence, we get:

- $[\pi^\iota \backslash \mathsf{dom}(\sigma);\, \pi^o] = \left[X_i : {\tau_i}^{i\in I_2};\, X_j : {\tau_j}^{j\in O}\right]$ and from (1b) we have that $\vdash \left[X_i : {\tau_i}^{i\in I_2};\, X_j : {\tau_j}^{j\in O}\right]$; (1c')
- $\left[\iota;\, o;\, \rho, x_i \overset{i\in I_1}{\mapsto} o_{\mathsf{Exp}}(\sigma(X_i))\right] =$
  $\left[x_i \overset{i\in I_2}{\mapsto} X_i : \tau_i;\, X_j \overset{j\in O}{\mapsto} e_j : \tau_j;\, x_l \overset{l\in L}{\mapsto} e_l, x_i \overset{i\in I_1}{\mapsto} o_{\mathsf{Exp}}(\sigma(X_i))\right]$.

We can now obtain the following derivation:

$$\frac{\text{(1c')} \quad \text{(1a)}}{\vdash_M \left[\iota;\, o;\, \rho, x_i \overset{i\in I_1}{\mapsto} o_{\mathsf{Exp}}(\sigma(X_i))\right] : [\pi^\iota \backslash \mathsf{dom}(\sigma);\, \pi^o]}\ (M\text{-basic})$$

(B)

($\mathcal{C}$-ctx): we derive $\mathcal{C}\left[R^C\right] \longrightarrow \mathcal{C}\left[C\right]$. In this case we proceed by case analysis on the structure of $\mathcal{C}$ and in all cases we can conclude by applying the inductive hypothesis, using the premise of the reduction rule ($\mathcal{C}$-ctx). In particular, we illustrate the case $\mathcal{C} = \mathsf{link}_\sigma \mathcal{C}'$. For hypothesis we have $\vdash_C \mathsf{link}_\sigma \mathcal{C}'\left[R^C\right] : [{\pi^\iota}_C;\, {\pi^o}_C]$. This judgment can only be derived by using rule (C-link), hence it must be:

- $[{\pi^\iota}_C;\, {\pi^o}_C] = [\pi^\iota \backslash \mathsf{dom}(\sigma);\, \pi^o]$
- $\vdash_C \mathcal{C}'\left[R^C\right] : [\pi^\iota;\, \pi^o]$; (1)
- $\sigma|_{\mathsf{dom}(\pi^\iota)} : \pi^\iota|_{\mathsf{dom}(\sigma)} \to \pi^o$. (2)

By applying the inductive hypothesis to $\mathcal{C}'\left[R^C\right] \longrightarrow \mathcal{C}'\left[C\right]$ (derived from the premise of reduction rule ($\mathcal{C}$-ctx), that is, $R^C \longrightarrow C$, by using ($\mathcal{C}$-ctx) with evaluation context $\mathcal{C}'$) with (1) we obtain that there exist $\pi^{\iota\prime} \subseteq \pi^\iota$ such that $\vdash_C \mathcal{C}'\left[C\right] : [\pi^{\iota\prime};\, \pi^o]$. (3)

We can now obtain the following derivation:

$$\frac{\text{(3)}}{\vdash_C \mathsf{link}_\sigma \mathcal{C}'\left[C\right] : \left[\pi^{\iota\prime} \backslash \mathsf{dom}(\sigma);\, \pi^o\right]}\ (C\text{-link})$$

Note that we can apply this rule since from $\pi^{\iota\prime} \subseteq \pi^\iota$ we get that $\sigma|_{\mathsf{dom}(\pi^{\iota\prime})} : {\pi^{\iota\prime}}|_{\mathsf{dom}(\sigma)} \to \pi^o$.

($\mathcal{CM}$-ctx): we derive $\mathcal{CM}\left[R^M\right] \longrightarrow \mathcal{CM}\left[M\right]$. In this case we proceed by case analysis on the structure of $\mathcal{CM}$ and in all cases we can conclude by applying the first point of this theorem.

(sel): we derive $[\iota;\, o;\, \rho].X \longrightarrow\ <\, [\iota;\, o;\, \rho]\,,\, o(X)\, >$. We suppose $\vdash_C [\iota;\, o;\, \rho].X : ([\pi^\iota;\, \pi^o]\,,\, \tau)$. This judgment can only be derived by using rule (sel), so it must be:

- $\vdash_M [\iota;\, o;\, \rho] : [\pi^\iota;\, \pi^o]$; (1)
- $\tau = \pi^o(X)$. (2)

Judgment (1) can only be derived by using rule (M-basic), so it must be:

- $[\iota;\, o;\, \rho] = \left[x_i \overset{i\in I}{\mapsto} X_i : \tau_i;\, X_j \overset{j\in O}{\mapsto} e_j : \tau_j;\, x_l \overset{l\in L}{\mapsto} e_l\right]$;
- $[\pi^\iota;\, \pi^o] = \left[X_i : {\tau_i}^{i\in I};\, X_j : {\tau_j}^{j\in O}\right]$;
- $\vdash \left[X_i : {\tau_i}^{i\in I};\, X_j : {\tau_j}^{j\in O}\right]$; (1a)
- $\left\{x_h : {\tau_h}^{h\in I\cup L} \vdash_e e_k : \tau_k \mid k \in O \cup L\right\}$. (1b)

Since $X \in \mathsf{dom}(o)$ (from the (implicit) side-condition of reduction rule (sel)), we have that there exists $p \in O$ such that $X = X_p$, $o(X) = e_p$ and for (2)

$\pi^o(X) = \tau_p$; hence, from (1b) we get
$x_h : \tau_h{}^{h \in I \cup L} \vdash_e o(X) : \tau$. (1c)
We can now obtain the following derivation:

$$\frac{(1a) \quad (1b) \quad (1c)}{\vdash_C < [\iota;\, o;\, \rho]\,,\ o(X) >: ([\pi^\iota;\, \pi^o]\,,\, \pi^o(X))}\ (C\text{-basic})$$

(core): we derive that $< [\iota;\, o;\, \rho]\,,\ e > \longrightarrow < [\iota;\, o;\, \rho]\,,\ e' >$.

We suppose $\vdash_C < [\iota;\, o;\, \rho]\,,\ e >: ([\pi^\iota;\, \pi^o]\,,\, \tau)$. This judgment can only be derived by using rule (C-basic), so it must be:

- $[\iota;\, o;\, \rho] = \left[x_i \overset{i \in I}{\mapsto} X_i : \tau_i;\ X_j \overset{j \in O}{\mapsto} e_j : \tau_j;\ x_l \overset{l \in L}{\mapsto} e_l\right]$;
- $[\pi^\iota;\, \pi^o] = \left[X_i : \tau_i{}^{i \in I};\ X_j : \tau_j{}^{j \in O}\right]$;
- $\vdash \left[X_i : \tau_i{}^{i \in I};\ X_j : \tau_j{}^{j \in O}\right]$; (1)
- $\left\{x_h : \tau_h{}^{h \in I \cup L} \vdash_e e_k : \tau_k \mid k \in O \cup L\right\}$; (2)
- $x_h : \tau_h{}^{h \in I \cup L} \vdash_e e : \tau$. (3)

From (3) and from the premise of the rule (core), that is, $e \underset{e}{\longrightarrow} e'$, by applying the core assumption 5.1 (iii), that is, the subject reduction property, we get $x_h : \tau_h{}^{h \in I \cup L} \vdash_e e' : \tau$. (4)
We can now obtain the following derivation:

$$\frac{(1) \quad (2) \quad (4)}{\vdash_C < [\iota;\, o;\, \rho]\,,\ e' >: ([\pi^\iota;\, \pi^o]\,,\, \tau)}\ (C\text{-basic})$$

(var): we derive $< [\iota;\, o;\, \rho]\,,\ \mathcal{E}\,[x] > \longrightarrow < [\iota;\, o;\, \rho]\,,\ \mathcal{E}\{\rho(x)\} >$. We suppose $\vdash_C < [\iota;\, o;\, \rho]\,,\ \mathcal{E}\,[x] >: ([\pi^\iota;\, \pi^o]\,,\, \tau)$. This judgment can only be derived by using rule (C-basic), so it must be:

- $[\iota;\, o;\, \rho] = \left[x_i \overset{i \in I}{\mapsto} X_i : \tau_i;\ X_j \overset{j \in O}{\mapsto} e_j : \tau_j;\ x_l \overset{l \in L}{\mapsto} e_l\right]$;
- $[\pi^\iota;\, \pi^o] = \left[X_i : \tau_i{}^{i \in I};\ X_j : \tau_j{}^{j \in O}\right]$;
- $\vdash \left[X_i : \tau_i{}^{i \in I};\ X_j : \tau_j{}^{j \in O}\right]$; (1)
- $\left\{x_h : \tau_h{}^{h \in I \cup L} \vdash_e e_k : \tau_k \mid k \in O \cup L\right\}$; (2)
- $x_h : \tau_h{}^{h \in I \cup L} \vdash_e \mathcal{E}\,[x] : \tau$. (3)

Since for the side-condition of the reduction rule (var) we have that $x \in \mathsf{dom}(\rho)$, then there exists $p \in L$ such that $x = x_p$ and so $\rho(x) = e_p$ and (from (2)) $x_h : \tau_h{}^{h \in I \cup L} \vdash_e \rho(x) : \tau_p$. (4)
From (3) and (4), since for the (implicit) side-condition or reduction rule (var) we have that $x \notin \mathsf{HB}\,(\mathcal{E})$, by applying the core assumption 5.1 (iv), that is, the Substitution Lemma, we get:
$x_h : \tau_h{}^{h \in I \cup L} \vdash_e \mathcal{E}\{\rho(x)\} : \tau$. (5)
We can now obtain the following derivation:

$$\frac{(1) \quad (2) \quad (5)}{\vdash_C < [\iota;\, o;\, \rho]\,,\ \mathcal{E}\{\rho(x)\} >: ([\pi^\iota;\, \pi^o]\,,\, \tau)}\ (C\text{-basic})$$

(var/err) and (link⁻/err): we do not consider this rules since they reduce a configuration into an error.

(sum/basic): we derive

$< [\iota_1; o_1; \rho_1], e > + [\iota_2; o_2; \rho_2] \longrightarrow < [\iota_1, \iota_2; o_1, o_2; \rho_1, \rho_2], e >$.

We suppose $\vdash_C < [\iota_1; o_1; \rho_1], e > + [\iota_2; o_2; \rho_2] : ([\pi^\iota; \pi^o], \tau)$. This judgment can only be derived by using rule (C-sum), so it must be:

- $[\pi^\iota; \pi^o] = [\pi^\iota{}_1 \cup \pi^\iota{}_2; \pi^o{}_1 \cup \pi^o{}_2]$;
- $\vdash_C < [\iota_1; o_1; \rho_1], e > : ([\pi^\iota{}_1; \pi^o{}_1], \tau)$; (1)
- $\vdash_M [\iota_2; o_2; \rho_2] : [\pi^\iota{}_2; \pi^o{}_2]$; (2)

Judgment (1) can only be derived by using rule (C-basic), so, it must be:

- $[\pi^\iota{}_1; \pi^o{}_1] = \left[X_i : \tau_i{}^{i \in I_1}; X_j : \tau_j{}^{j \in O_1}\right]$;
- $[\iota_1; o_1; \rho_1] = \left[x_i \overset{i \in I_1}{\mapsto} X_i : \tau_i; X_j \overset{j \in O_1}{\mapsto} e_j : \tau_j; x_l \overset{l \in L_1}{\mapsto} e_l\right]$;
- $\left\{x_h : \tau_h{}^{h \in I_1 \cup L_1} \vdash_e e_k : \tau_k \mid k \in O_1 \cup L_1\right\}$; (1a)
- $\vdash \left[X_i : \tau_i{}^{i \in I_1}; X_j : \tau_j{}^{j \in O_1}\right]$; (1b)
- $x_h : \tau_h{}^{h \in I_1 \cup L_1} \vdash_e e : \tau$. (1c)

Similarly, judgment (2) can only be derived by using rule (M-basic), so, it must be:

- $[\pi^\iota{}_2; \pi^o{}_2] = \left[X_i : \tau_i{}^{i \in I_2}; X_j : \tau_j{}^{j \in O_2}\right]$;
- $[\iota_2; o_2; \rho_2] = \left[x_i \overset{i \in I_2}{\mapsto} X_i : \tau_i; X_j \overset{j \in O_2}{\mapsto} e_j : \tau_j; x_l \overset{l \in L_2}{\mapsto} e_l\right]$;
- $\left\{x_h : \tau_h{}^{h \in I_2 \cup L_2} \vdash_e e_k : \tau_k \mid k \in O_2 \cup L_2\right\}$; (2a)
- $\vdash \left[X_i : \tau_i{}^{i \in I_2}; X_j : \tau_j{}^{j \in O_2}\right]$. (2b)

By applying the core assumption 5.1 (v), that is, the weakening property, to (1c) we get $x_h : \tau_h{}^{h \in I_1 \cup L_1 \cup I_2 \cup L_2} \vdash_e e : \tau$; hence, in a similar way to what seen for the case (M-sum), we can derive from (1a), (2a), (1b) and (2b) the judgment $\vdash_C < [\iota_1, \iota_2; o_1, o_2; \rho_1, \rho_2], e > : ([\pi^\iota{}_1 \cup \pi^\iota{}_2; \pi^o{}_1 \cup \pi^o{}_2], \tau)$.

(sum-closure): we derive

$< [\iota; o; \rho], e > + M \longrightarrow < [\iota; o; \rho], e > + M'$. We suppose $\vdash_C < [\iota; o; \rho], e > + M : ([\pi^\iota; \pi^o], \tau)$. This judgment can only be derived by using rule (C-sum), so it must be:

- $[\pi^\iota; \pi^o] = [\pi^\iota{}_1 \cup \pi^\iota{}_2; \pi^o{}_1 \cup \pi^o{}_2]$;
- $\vdash_C < [\iota; o; \rho], e > : ([\pi^\iota{}_1; \pi^o{}_1], \tau)$; (1)
- $\vdash_M M : [\pi^\iota{}_2; \pi^o{}_2]$; (2)
- $\pi^o{}_1 \cap \pi^o{}_2 = \emptyset$; (3)

By applying the first point of this theorem to the premise of the reduction rule (sum-closure), that is, $M \longrightarrow M'$, and to (3), we get:

$\vdash_M M' : [\pi^\iota{}_2; \pi^o{}_2]$ (4)

We can now obtain the following derivation:

$$\frac{(1) \quad (4)}{\vdash_C < [\iota; o; \rho], e > + M' : ([\pi^\iota{}_1 \cup \pi^\iota{}_2; \pi^o{}_1 \cup \pi^o{}_2], \tau)} (C\text{-sum}) \text{ using } (3)$$

(sum/link$^-$): we derive $\mathsf{link}^-_\sigma C + M \longrightarrow \mathsf{link}^-_\sigma(C + M)$.

We suppose $\vdash_C \mathsf{link}^-_\sigma C + M : ([\pi^\iota; \pi^o], \tau)$. This judgment can only be derived by using (C-sum), so it must be:
- $[\pi^\iota; \pi^o] = [{\pi^\iota}_1 \cup {\pi^\iota}_2; {\pi^o}_1 \cup {\pi^o}_2]$;
- $\vdash_C \mathsf{link}^-_\sigma C : ([{\pi^\iota}_1; {\pi^o}_1], \tau)$; (1)
- $\vdash_M M : [{\pi^\iota}_2; {\pi^o}_2]$. (2)

Judgment (1) can only be derived by using rule (C-link$^-$), so, it must be:
$\vdash_C C : ([{\pi^\iota}_1; {\pi^o}_1], \tau)$. (1a)
We can now obtain the following derivation:

$$\dfrac{\dfrac{\text{(1a)} \quad \text{(2)}}{\vdash_C C + M : ([{\pi^\iota}_1 \cup {\pi^\iota}_2; {\pi^o}_1 \cup {\pi^o}_2], \tau)}\,(C\text{-sum})}{\vdash_C \mathsf{link}^-_\sigma(C + M) : ([{\pi^\iota}_1 \cup {\pi^\iota}_2; {\pi^o}_1 \cup {\pi^o}_2], \tau)}\,(C\text{-link}^-)$$

(reduct/basic): in this case the thesis follows similarly to what seen for the case (M-reduct), by applying rule (C-basic).

(reduct/link$^-$): we derive that ${}_{\sigma^\iota}|\mathsf{link}^-_\sigma C|_{\sigma^o} \longrightarrow \mathsf{link}^-_\sigma({}_{\sigma^\iota}|C|_{\sigma^o})$. We suppose $\vdash_C {}_{\sigma^\iota}|\mathsf{link}^-_\sigma C|_{\sigma^o} : ([{\pi^\iota}_M; {\pi^o}_M], \tau)$. This judgment can only be derived by using rule (C-reduct), so it must be:
- $[{\pi^\iota}_M; {\pi^o}_M] = [\tilde{\pi}^\iota \cup \pi^\iota \backslash \mathsf{dom}(\sigma^\iota); \tilde{\pi}^o]$;
- $\vdash_C \mathsf{link}^-_{\sigma'} C : ([\pi^\iota; \pi^o], \tau)$; (1)
- $\sigma^\iota|_{\mathsf{dom}(\pi^\iota)} : \pi^\iota|_{\mathsf{dom}(\sigma)} \to \tilde{\pi}^\iota$; (2)
- $\sigma^o : \tilde{\pi}^o \to \pi^o$. (3)

Judgment (1) can only be derived by using rule (C-link$^-$), so it must be:
$\vdash_C C : ([\pi^\iota; \pi^o], \tau)$. (1a)
We can now obtain the following derivation:

$$\dfrac{\dfrac{\text{(1a)}}{\vdash_C {}_{\sigma^\iota}|C|_{\sigma^o} : ([\tilde{\pi}^\iota \cup \pi^\iota \backslash \mathsf{dom}(\sigma^\iota); \tilde{\pi}^o], \tau)}\,(C\text{-reduct}) \text{ with (2) and (3)}}{\vdash_C \mathsf{link}^-_\sigma({}_{\sigma^\iota}|C|_{\sigma^o}) : ([\tilde{\pi}^\iota \cup \pi^\iota \backslash \mathsf{dom}(\sigma^\iota); \tilde{\pi}^o], \tau)}\,(C\text{-link}^-)$$

(link/basic): in this case the thesis follows similarly to what seen for the case (M-link), by applying rule (C-basic).

(link/link$^-$): we derive that $\mathsf{link}_\sigma\big(\mathsf{link}^-_{\sigma'} C\big) \longrightarrow \mathsf{link}^-_{\sigma'}(\mathsf{link}_\sigma C)$. We suppose $\vdash_C \mathsf{link}_\sigma\big(\mathsf{link}^-_{\sigma'} C\big) : ([{\pi^\iota}_M; {\pi^o}_M], \tau)$. This judgment can only be derived by using rule (C-link), so it must be:
- $[{\pi^\iota}_M; {\pi^o}_M] = [\pi^\iota \backslash \mathsf{dom}(\sigma); \pi^o]$;
- $\vdash_C \mathsf{link}^-_{\sigma'} C : ([\pi^\iota; \pi^o], \tau)$; (1)
- $\sigma|_{\mathsf{dom}(\pi^\iota)} : \pi^\iota|_{\mathsf{dom}(\sigma)} \to \pi^o$. (2)

Judgment (1) can only be derived by using rule (C-link$^-$), so it must be:
$\vdash_C C : ([\pi^\iota; \pi^o], \tau)$. (1a)

We can now obtain the following derivation:

$$\dfrac{\dfrac{(1a)}{\vdash_C \mathsf{link}_\sigma C : ([\pi^\iota \backslash \mathsf{dom}(\sigma);\, \pi^o]\,,\, \tau)}\,(C\text{-link}) \text{ with } (2)}{\vdash_C \mathsf{link}^-_{\sigma'}(\mathsf{link}_\sigma C) : ([\pi^\iota \backslash \mathsf{dom}(\sigma);\, \pi^o]\,,\, \tau)}\,(C\text{-link}^-)$$

(link$^-$): we derive $\mathsf{link}^-_{\sigma, X\mapsto Y} C \longrightarrow \mathsf{link}^-_\sigma(\mathsf{link}_{X\mapsto Y} C)$. We suppose $\vdash_C \mathsf{link}^-_{\sigma,\, X\mapsto Y} C : ([{\pi^\iota}_M;\, {\pi^o}_M]\,,\, \tau)$. This judgment can only be derived by using rule (C-link$^-$), so it must be $\vdash_C C : ([{\pi^\iota}_M;\, {\pi^o}_M]\,,\, \tau)$. (1)
Since for the premise of the reduction rule (link$^-$) we have that $C \longrightarrow \mathsf{err}(X : \tau, \pi)$, with $\pi(Y) = \tau$, from (1), by applying the second point of Lemma A.1, we obtain that $X : \tau \in {\pi^\iota}_M$ and $\pi = {\pi^o}_M$. Hence, ${\pi^\iota}_M(X) = {\pi^o}_M(Y)$. (2)
We can now obtain the following derivation:

$$\dfrac{\dfrac{(1)}{\vdash_C \mathsf{link}_{X\mapsto Y} C : ([{\pi^\iota}_M \backslash \{X\};\, {\pi^o}_M]\,,\, \tau)}\,(C\text{-link}) \text{ with } (2)}{\vdash_C \mathsf{link}^-_{\sigma'}(\mathsf{link}_\sigma C) : ([{\pi^\iota}_M \backslash \{X\};\, {\pi^o}_M]\,,\, \tau)}\,(C\text{-link}^-)$$

Note that ${\pi^\iota}_M \backslash \{X\} \subseteq {\pi^\iota}_M$.

□