Lazy Modules
A lazy evaluation strategy for more recursive initialization patterns

Keiko Nakata

CNAM/INRIA

Abstract. Modern programming languages have mechanisms for structuring programs, such as object systems and the ML module system. The nature of these mechanisms vary with respect to expressiveness, safety and efficiency, influenced by their evaluation strategies. In this paper we formalize and examine an evaluation strategy for modules by using a lazy evaluation mechanism in a disciplined way. On the one hand, the use of laziness facilitates the introduction of recursion between modules. On the other hand, its controlled use keeps evaluation safer and the evaluation order more explicit than unconstrained uses, at the cost of some expressiveness. We formalize the strategy we propse by translating the source language for modules into the target language, which is an adaptation of the call-by-need letrec lambda calculus of Ariola and Felleisen.

1 Introduction

Modern programming languages have mechanisms for promoting modular programming. For instance, objects[1] encapsulate methods which operate on object states. F# [23] has a module system on top of its object system for better support of large program structuring. The ML module system [20, 14] is well-known for its strong support for modular programming. OCaml [15] and Moby [11] incorporate both the ML module system and object systems in a single language. In the case of Scala [7], the object system is expressive enough that it can serve as both an object system and a ML-like module system.

Common to all above mentioned languages is support of nestable structure, namely the ability to decompose a program into hierarchy. Nesting is a powerful way to organize program codes and namespaces that should be simple and intuitive for any programmer. An abbreviation mechanism, or the ability to make aliases for modules, is also common. For instance, it is useful to give a module a meaningful name, like Hashtable. At the same time, it is often convenient to locally refer to the module with an abbreviation, like H, when the referred module is clear from the context.

[1] In this paper, we stick to the terminology "objects" and "object systems" even where "classes" may be suitable.

```
type t = Int of int | Add of t * t
let three = Add(Int 1, Add(Int 2, Int 0))
module F = functor(X : sig val eval : t → int end) →
  struct
    let eval x = match x with | Int i → i | Add(a1, a2) → (X.eval a1) + (X.eval a2)
    let _ = print_int (eval three)
  end
module rec Eval = F(Eval)
```

Fig. 1. An addition language

Support for features beyond nesting and abbreviations vary and some features are closely tied to the evaluation strategies of the underlying languages. Below we compare two opposites.

Object systems support flexible recursion between objects. Not only methods can call each other recursively across object boundaries, but also object fields can be initialized recursively among objects. Yet this flexibility costs. At creation time, object fields contain default values, such as the infamous null-pointers, which are supposed to be initialized to meaningful values afterward, for instance through constructor calls. There is no guarantee that those default values are properly initialized eventually. A null-pointer exception can be raised at any point during program execution. The programmer has to carefully determine a correct initialization order, which does not access object fields that are not yet initialized.

The ML module system comprises structures and functors. A structure encloses type and value definitions, which we call core fields of the structure, and possibly sub-structures, which we call module fields of the structure[2]. Functors are functions on structures. They facilitate code reuse in a modular way. Traditionally ML structures are initialized eagerly. In other words, all core fields of a structure, including those of sub-structures, are evaluated immediately following the definition order in which they textually appear in the structure. Since ML does not have recursively defined structures or functors and since recursive value bindings are syntactically restricted, the initialization is always successful without possibilities of observing meaningless default values. This enhances both safety and efficiency, since references to fields of structures are necessarily proper values, hence, for instance, no null-check is required. ML enjoys this property at the cost of difficulties in extending the module system with recursion or dynamic loading of modules.

In this paper, we propose and examine an alternative evaluation strategy for ML-style modules supporting recursion, stimulated by evaluation strategies of object systems. Concretely evaluation of a module is delayed until the module is accessed, but all its core fields, not including those of sub-modules, are evaluated at once when some field of the module is accessed for the first time. The evaluation proceeds following the definition order in which the core fields appear within the enclosing module.

[2] Precisely a structure may contain functors and functors may be higher-order.

Our evaluation strategy inherits both merits and demerits from evaluation strategies of object systems. By introducing a lazy mechanism to module evaluation, we check the absence of unsafe recursion between modules at runtime. This allows us to support recursive initialization patterns that would not be available with the eager evaluation strategy or in the context where any potential of unsafe recursion is statically eliminated. At the same time, this introduces the possibilities of runtime errors since there is no guarantee that module evaluation does not involve unsafe recursion. Yet, as mentioned above, our evaluation strategy is stricter, thus differs from those of object systems in the following points: 1) Backward references to core fields are necessarily proper values, but not suspended evaluation or do not cause runtime errors. 2) A runtime error can only be raised during the initialization phase. Hence, once a module is fully evaluated, all its core fields contain proper values. As a result, we give up some recursive initialization patterns that object systems can handle. In Fig. 1, we give an example, written in the OCaml syntax[3], that condenses scenarios where we believe recursive modules are useful; to take the fix-points of functors. If we assume the ML core language, modules on the recursion may contain side-effects. The flexibility in taking the fix-points is compelling, since it allows separate and extensible development of recursively defined modules [4, 21, 18, 16].

The main contribution of the paper is the complete formalization of a lazy evaluation strategy for recursive ML-style modules, by exploring the design space and examining issues involved. To the best of our knowledge, no previous work has examined a lazy evaluation strategy for ML-style modules supporting recursion in such depth. Inspired by the formalization of call-by-name and call-by-need mixin calculi [24, 8], we formalize the strategy by translating the source language for recursive modules into the target language, which is an adaptation of Ariola and Felleisen's call-by-need letrec lambda calculus [1]

We would like to pose a question to ML programmers whether module evaluation can be lazy, and stimulate those who are not familiar with ML modules to recognize and ponder the vast design space of evaluation strategies for this advanced module system. We also hope our formalization shall serve as a witness of the flexibility of Ariola and Felleisen's calculus, by demonstrating how it can be adapted to express a more controlled call-by-need evaluation strategy.

2 The source language *Osan* for recursive modules

In Fig. 2, we define the source language, named *Osan*. We use *Osan* to introduce the evaluation strategy we propose in the paper. We use the following metavariable conventions: m ranges over core names; M over module names; X over module variables; n over positive integers. A module expression is either a *recursive structure*, *module path*, *functor* or *functor application*. A recursive structure, simply called structure hereafter, $\{(X)\ \overline{D}\}$ is a sequence of definitions with a declarations of a "self variable" X. A definition either binds a core name

[3] With a signature of `Eval` explicitly given, the example type checks in OCaml. The evaluation signals a runtime error.

Definitions	$D ::= M = E \mid m = e$
Module expressions	$E ::= \{(X)\ \overline{D}\} \mid p \mid \Lambda X.E \mid E_1(E_2)$
Paths	$p\ ::= X \mid p.n$
Core expressions	$e\ ::= p.n \mid \ldots$

Fig. 2. Syntax for *Osan*

to a core expression or a module name to a module expression. X is bound to the structure $\{(X)\ \overline{D}\}$. Any expression within $\overline{D}$ can refer to each other recursively through X, using paths. We use the overlined notation to denote possibly empty sequences punctuated with semicolons. Moreover we may use subscripts to specify the length. For instance $\overline{D_i}$ is a shorthand for an empty sequence ϵ when $i = 0$ and $D_1; D_2; \ldots; D_i$ when $i > 0$. We use i, j, k as a metavariable for zero and positive integers.

A module path is a reference to a module. It is either a module variable X or qualified identifier $p.n$, which accesses the n-th field of the structure that p refers to. In this paper we only consider well-typed programs where the type system ensures that the n-th field binds module expression, not a core expression. A functor $\Lambda X.E$ is a function on modules, where X is the parameter and E is the body. $E_1(E_2)$ denotes functor application.

Our formalization of an evaluation strategy for modules is largely independent of the core language. We only assume the core language includes *value paths* $p.n$, which accesses the n-th field of the structure that p refers to. Again we assume the type system ensures that the n-th field binds a core expression.

In the paper, we do not address typing issues. To the best of our knowledge, most type systems for ML modules [20, 14], including those which consider recursive module extensions [4, 21, 6, 18], are independent of whether modules are evaluated eagerly or lazily. Hence we can benefit from those previous work.

2.1 Remark on the possible surface language

For the surface language to be used by the programmer, we would prefer more programmer-friendly syntax for paths. That is, we use sequences of field names, instead of indexes, punctuated by the dots for paths. Then we insert an elaboration phase, prior to runtime, to translate name-based paths into index-based paths. For the translation we only need to know the definition order in which core and module names are bound in the enclosing structure to map names to indexes. Usual type checkers for ML modules collect enough type information for this purpose; in fact translation from name-based paths into index-based paths is a well-known technique in compiling ML modules.

2.2 The overview of the evaluation strategy

In this subsection, we introduce our evaluation strategy through a series of examples. As memtioned above, we use name-based paths throughout the examples.

To better explain the underlying design principles, we assume a simple call-by-value functional language for the core language in examples. In particular, we use a side-effecting function "`print` e, which prints the resulting value of evaluating e, which is always an integer in examples, then returns the value, to visualize the evaluation order.

The evaluation strategy follows the intuition that modules are evaluated lazily, where the evaluation is triggered on a per-module basis when they are first accessed. Some of its implications will become clearer later. For the sake of the first example, it is understood as 1) evaluation of a module is suspended until some field of the module is first accessed, and 2) when a module is accessed, all its core fields, but not those of sub-modules, are evaluated at once following the textual definition order.

Let us look at the following program:

```
{(X)
  M1 = {(X1)
    m1 = print 2;
    M2 = { m1 = 3;  m2 = print 4 };
    M3 = { m1 = print 5 };
    m2 = print 6;
    M4 = { m1 = print 7;  m2 = X1.M2.m1 };
    m3 = print 8; };
  main = print X.M1.M4.m2 }
```

For the explanatory purpose, we assume a program is a structure containing a single core field named main and possibly several sub-modules. Execution of the program starts by evaluating main. We omit declarations of self variables when they are not used. Executing the above program prints "2 6 8 7 4 3" in this order. Here is why:

1. Execution starts by evaluating X.M1.M4.m2.
2. The value path X.M1.M4.m2 accesses the module M1, triggering evaluation of all M1's core fields following the definition order. This prints 2 6 8.
3. Then the module M4 is evaluated, since X.M1.M4.m2 accesses M4. Evaluation of M4 first prints 7, then forces evaluation of M2 to fetch a value from the field M2.m1. After printing 4, evaluation of M2 is completed and so is evaluation of M4. Now M4.m2 is bound to 3.
4. Finally evaluation of print X.M1.M4.m2 is completed by printing 3 and so is the entire execution.
5. Since the module M3 is not accessed, it is not evaluated.

Functors are simply lazy functions on modules. For instance, executing the following program prints "1 3 2" in this order:

```
{(X)
  M1 = ΛX1.{ m1 = print 1; m2 = print X1.m1 };
  M2 = { m1 = 2; m2 = print 3 };
  M3 = X.M1(X.M2);
  main = X.M3.m1 }
```

Interaction with recursion The top-down evaluation order of core fields within structures has an impact on the possible recursion between modules. We examine it below. Here we are also interested in recursion that our evaluation strategy does not support; by being so, we intend to explain the differences of our recursive structures from recursive records with lazy fields, e.g, let rec x = {x1 = lazy (force (x.x2)); x2 = lazy 3 } written in the OCaml syntax, and to show our controlled use of a lazy mechanism.

First we detail the underlying design principles.

1. All preceding core fields of the same and enclosing structures must have been evaluated first. Enclosing structures may contain other sub-modules, whose core fields need not be evaluated first.
2. Evaluation proceeds on a per-module basis. Or,
   (a) Once a field of a structure is accessed, all its core fields are eventually evaluated. Hence a structure is not left with some of its core fields being evaluated and others not.
   (b) Evaluation does not alternate between modules, yet follows stack-like discipline in the sense that when evaluation of a module triggers evaluation of another module, then the triggered evaluation should be completed first before the triggering evaluation is resumed.

Delayed dependencies A core field can refer to a subsequent field as long as evaluation of the referring field need not evaluate the referred field. Typical cases are where the forward references are under abstraction. For instance, we support both of intra-module recursion such as

```
{(X)
 M = {(X1)  m1 = fun x → X1.m2 (x+1);  m2 = fun x → X1.m1 (x+2) };
 main = X.M.m1 0 }
```

and inter-module recursion such as

```
{(X)
 M1 = { m = fun x → X.M2.m (x+1) };  M2 = { m = fun x → X.M1.m (x+2) };
 main = X.M1.m 0}
```

The first principle The following programs cause runtime errors[4].

```
{(X)  M = {(X1)  m1 = X1.m2 + 3;  m2 = 2 };  main = X.M.m1}
```

or

```
{(X)
 M = {(X1)  m = (fun x → x+1) X1.M1.m;  M1 = { m = 0 } };
 main = X.M.m}
```

Evaluation of the latter program fails since we cannot complete evaluating the field M.m before starting to evaluate the field M1.m; in respect of M1.m, M.m is a preceding field of the enclosing structure. The latter program could be made

[4] We could have allowed subsequent fields to be evaluated earlier against the order in some cases, e.g. when the bound expressions to the fields are immediate values, to support more recursive initialization patterns. Related discussion is found in Java's class initialization policy [12]. Yet we do not explore this design space in the paper.

safe by enclosing the field M.m into another sub-module as:

```
{(X)
  M = {(X1)  M2 = { m = (fun x → x+1) X1.M1.m };  M1 = { m = 0 } };
  main = X.M.M2.m}
```

Now evaluation of the above program succeeds since M2 is not the same or enclosing structure of the field M1.m.

The second principle The following program fails:

```
{(X)
  M1 = {(X1) m1 = X.M2.m1;  m2 = X1.m1 + 2;  m3 = X.M2.m3 + X1.m2 };
  M2 = {(X2) m1 = 3;  m2 = X.M1.m2;  m3 = X2.m1 + X2.m2 };
  main = X.M1.m3 }
```

To be successful, evaluation of the fields of M1 and M2 needs to be interleaved. This is against the second principle; evaluation does not alternate between modules. A right way to do is to explicitly break the evaluation order via nesting:

```
{(X)
  M1 = {(X1)
    m1 = X.M2.m1;
    M1' = {(X1')  m2 = X1.m1 + 2;  M1'' = { m3 = X.M2.M2'.M2''.m3 + X1'.m2 } } };
  M2 = {(X2)
    m1 = 3;
    M2' = {(X2')  m2 = X.M1.M1'.m2;  M2'' = { m3 = X2.m1 + X2'.m2 } } };
  main = X.M1.M1'.M1''.m3}
```

The last example demonstrates that programmers can control the evaluation order in an explicit way using lazily evaluated sub-modules, when the default evaluation order is too restrictive. For the programmer's convenience, we could provide syntactic sugar in the surface language for decomposing a sequence of definitions into a cascade of nested structures to mitigate the verbosity.

Motivation We conclude this section by reviewing our design choices and explaining the background. Recursive initialization patterns we support are not as expressive as those available in prominent object systems like Java, or our evaluation strategy will not result in as simple an evaluation order as the traditional eager evaluation strategy of ML modules. Indeed we have been and still are in our quest for design choices which keep a good tension between safety, simplicity, expressivity and efficiency.

Syme's work on initialization graphs [22] motivated us to introduce a lazy mechanism in a controlled way, by checking initialization safety at runtime, to support more recursive initialization patterns than those available under the traditional eager evaluation strategy. In particular, we followed his design principle by evaluating all core fields of a structure at once when the structure is first accessed. As Syme noted, this enhances safety in the sense that runtime errors due to unsafe recursion are necessarily coming from modules that are currently under evaluation and that no error is left due to initialization of modules, once the initialization phase is completed.

$$
\begin{array}{lll}
\textit{Expressions} & a & ::= x \mid \lambda x.a \mid a_1 a_2 \mid a.n \mid \langle a \mid \overline{d} \rangle \mid \{\overline{a}\} \mid \lfloor \overline{s} \rfloor \\
\textit{Stables} & s & ::= v \mid x \\
\textit{Values} & v & ::= \lambda x.a \mid \lfloor \overline{s} \rfloor \\
\textit{Definition} & d & ::= x = a \\
\textit{Answers} & A & ::= v \mid \langle A \mid \overline{d} \rangle
\end{array}
$$

$$
\begin{array}{lll}
E[] & ::= & [] \mid E[]a \mid E[].n \mid \{\overline{v}; E[]; \overline{a}\} \mid \langle E[] \mid d \rangle \\
 & \mid & \langle E[x] \mid x = E[], \overline{d} \rangle \mid \langle E[x] \mid d[x, x_n], x_n = E[], \overline{d} \rangle \\
d[x, x_n] & ::= & x = E[x_1], x_1 = E[x_2], \ldots, x_{n-1} = E[x_n]
\end{array}
$$

Fig. 3. Syntax for $\lambda_{\{\}}$

The last example of this section in fact describes an encoding of an alternative evaluation strategy which evaluates core fields of a structure as much as necessary. This alternative could result in a more unexpected evaluation order than the proposed strategy. For instance, let us consider the following program:

```
{(X)
 M1 = { (X1)
   m1 = print 1;
   M2 = { m1 = print 2; m2 = X1.M3.m1; m3 = print 3 };
   m2 = print 4;
   M3 = { m1 = print 5 }
   m3 = print 6 }
 main = X.M1.M2.m3}
```

With the as-much-as-necessary strategy, the execution will print "1 2 4 5 3", whereas with our strategy it will print "1 4 6 2 5 3". It seems for us the latter is simpler to predict than the former; in particular, we found the occurrence of 4 between 2 and 5 in the former rather abrupt. This is one reason why we have not preferred this alternative.

Our insist on the top-down evaluation order within a structure is not only for the compatibility with the tradition of ML modules, but also for efficiency. The introduction of a lazy mechanism will unavoidably cause a runtime penalty of tag check to determine whether evaluation is still suspended or has been forced. However, backward references to core fields are necessarily bound to proper values, but not to suspensions or do not cause runtime errors. This implies that we can eliminate the tag check for backward references to core fields of the same and enclosing structures. We hope this could open a possibility that we can keep the same efficiency as eagerly evaluated ML modules for non-recursive programs that do not access other sub-modules.

Admittedly we need more practical experience to have stronger confidence in our design choices.

3 Semantics

Inspired by the previous formalization of call-by-name and call-by-need mixin calculi [24, 8], we adapt the call-by-need letrec lambda calculus of Ariola and

$$(1)\quad (\lambda x.a_1)a_2 \to \langle a_1 \mid x = a_2\rangle \qquad (2)\quad \lfloor \overline{s_{n-1}}; s; \overline{s}\rfloor.n \to s \qquad (3)\quad \{\overline{v}\} \to \lfloor \overline{v}\rfloor$$

lift

$$(4)\quad \langle A \mid \overline{d}\rangle a \to \langle Aa \mid \overline{d}\rangle \qquad (5)\quad \langle A \mid \overline{d}\rangle.n \to \langle A.n \mid \overline{d}\rangle$$

deref

$$(6)\quad \langle E[x] \mid x = v, \overline{d}\rangle \to \langle E[v] \mid x = v, \overline{d}\rangle$$
$$(7)\quad \langle E[x] \mid d[x, x_n], x_n = v, \overline{d}\rangle \to \langle E[x] \mid d[x, v], x_n = v, \overline{d}\rangle$$
$$(8)\quad \langle E[x] \mid d[x, x_n], \overline{d}\rangle \to \langle E[x] \mid d[x, \lfloor \overline{v}\rfloor], \overline{d}\rangle \quad \textit{when } x_n = \{\overline{v}; a; \overline{a}\} \in d[x, x_n] \textit{ and } a \textit{ is not a value.}$$

assoc

$$(9)\quad \{\overline{v}; \langle A \mid \overline{d}\rangle; \overline{a}\} \to \langle \{\overline{v}; A; \overline{a}\} \mid \overline{d}\rangle$$
$$(10)\quad \langle E[x] \mid x = \langle A \mid \overline{d}\rangle, \overline{d}_1\rangle \to \langle E[x] \mid x = A, \overline{d}, \overline{d}_1\rangle$$
$$(11)\quad \langle E[x] \mid d[x, x_n], x_n = \langle A \mid \overline{d}\rangle, \overline{d}_1\rangle \to \langle E[x] \mid d[x, x_n], x_n = A, \overline{d}, \overline{d}_1\rangle$$
$$(12)\quad \langle E[x] \mid d[x, x_n], \overline{d}_1\rangle \to \langle E[x] \mid d[x, x_n]\backslash x_n, x_n = a, \overline{d}, \overline{d}_1\rangle \quad \textit{when } x_n = \langle a \mid \overline{d}\rangle \in d[x, x_n]$$

context

$$E[a] \to E[a'] \textit{ iff } a \to a'$$

Fig. 4. Reduction rules

Felleisen [1] as the target language of *Osan*[5]. The target language, named $\lambda_{\{\}}$, is defined in Fig. 3, and its reduction rules in Fig. 4. We use x as a metavariable for variables of $\lambda_{\{\}}$. The notation $\overline{d}$ denotes a possibly empty sequence of d's punctuated with commas; we abuse the notation without ambiguity. No ordering among the bindings in $\overline{d}$ is assumed. $\overline{d}$ must not bind the same variable twice. We identify terms up to α-renaming of bound variables. Thus re-association of bindings in rules (10), (11) and (12) is done by taking appropriate α-equivalent terms, avoiding name clash. The reader can find the rigorous treatment of α-renaming issues in previous work [24, 8], which considers similar but more expressive calculi than $\lambda_{\{\}}$. We assume the reader has familiarity with the previous formalization [1, 2], which gently introduces the call-by-need letrec lambda calculus.

Notable constructs of $\lambda_{\{\}}$ for implementing *Osan*'s evaluation strategy are two sorts of records, namely *unstable* records $\{\overline{a}\}$ and *stable* records $\lfloor \overline{s}\rfloor$. Whereas unstable records may contain any expressions as fields, stable records may only variables and values, and we regard stable records as values. The motivation is that we emulate a "pointer" to x by $\lfloor x\rfloor$. We will use this ability of emulating pointers to encode lazy sub-modules and to tame the interaction between module abbreviations and module recursion. Note that only $\{x\}$, but not $\lfloor x\rfloor$, can be decomposed into a evaluation context and a redex for evaluating x. this is the reason for the syntactic distinction between unstable and stable records. It should be emphasized that the defining expression of x, or the right-hand side of x, in a letrec is evaluated at most once and the evaluation is shared among all references to x. Thus we can regard $\lfloor x\rfloor$ as a value and pass it around by substitution

[5] To be precise, an anonymous reviewer of PPDP '08 directed me to those previous work; I am grateful for the reviewer's comment.

without introducing possibilities of evaluating the same expression more than once.

There are two particularities of the evaluation strategy of *Osan*, which require care to be encoded into $\lambda_{\{\}}$. We examine them below.

Core fields of a structure are evaluated following the definition order, whereas module fields are evaluated lazily. The top-down evaluation order is expressed simply by restricting the evaluation context for unstable records to $\{\overline{v}; E[]; \overline{a}\}$, where the preceding fields to $E[]$ are all values. As we have hinted above, we implement lazy sub-modules by translating a module field E of a structure in *Osan* into a field $\lfloor x \rfloor$ of a record in $\lambda_{\{\}}$, where $\lfloor x \rfloor$ is surrounded by a letrec to bind x to the result of translating E. In this way, evaluation of E is suspended until the variable is actually used via projection. The translation will be defined later in Section 3.1.

Fields of a structure are accessible immediately after they are evaluated, while subsequent core fields may be still to be evaluated consecutively. We express this strategy by adding a special rule (8) to pull out a value from an unstable record by taking the longest prefix of values. The notation $x_n = \{\overline{v}; a; \overline{a}\} \in d[x, x_n]$ denotes that the binding $x_n = \{\overline{v}; a; \overline{a}\}$ appears in $d[x, x_n]$. The rule deals with inter-record recursion during evaluation of recursively defined records, or inter-module recursion during the initialization of recursive modules in the view of *Osan*. The side-condition $x_n = \{\overline{v}; a; \overline{a}\} \in d[x, x_n]$ indicates inter-record recursion during evaluation, since evaluation of the right-hand side of x_{n-1} (or x when $n = 1$) dereferences x_n, whose right-hand side is still under evaluation. Indeed the reference to x_n can be a self-reference, i.e. x_n and x_{n-1} may be identical. Rule (12) is added to effectively exercise rule (8), by pulling out expressions from letrec's on demand. The notation $d[x, x_n]\backslash x_{n'}$ denotes the sequence of bindings obtained from $d[x, x_n]$ by removing the binding for $x_{n'}$. It is undefined if $d[x, x_n]$ does not contain such a binding. Thus, for instance, if $d[x, x_n]$ consists of a single binding then $d[x, x_n]\backslash x$ is an empty sequence. Note that a value is extracted from an unstable record only when the dereference is recursive. Otherwise the rules (6) and (7) are only applicable, where the dereference returns if and only if the dereferenced variable is bound to a value. In this way, "the stack-like discipline", as described in Section 2.2, is expressed.

Ill-typedness, unsafe recursion and an attempt to evaluate subsequent fields of a record against the field order are sources of getting stuck. To relate $\lambda_{\{\}}$to the call-by-name strategy is future work. We are also interested in examining correspondences to an operational semantics in the natural semantics style; neither of previous work [1, 2, 24, 8] examined this issue in the presence of recursion.

3.1 Translation

In Fig. 5, we define a function $\mathcal{T}(\_)_\rho$ for translating module and core expressions of *Osan* to expressions of $\lambda_{\{\}}$. ρ ranges over mappings from module variables of *Osan* to variables of $\lambda_{\{\}}$. The notation $\rho[X \mapsto x]$ denotes mapping extension. Translation of the outer-most expression starts with an empty mapping. We assume given a translation $\mathcal{T}(\_)_\rho$ for core expressions other than value paths. It

$$\mathcal{T}(\{(X)\ \overline{D_i}\})_\rho = \langle x \mid x = \lfloor x' \rfloor, x' = \{\overline{\mathcal{T}(D_i)_{\rho[X \mapsto x]}}\}\rangle \quad (x, x' \text{ fresh})$$
$$\mathcal{T}(m = e)_\rho = \{\mathcal{T}(e_j)_\rho\}$$
$$\mathcal{T}(M = E)_\rho = \langle \lfloor x \rfloor \mid x = \mathcal{T}(E)_\rho \rangle \ (x \text{ fresh})$$
$$\mathcal{T}(X)_\rho = \rho(X)$$
$$\mathcal{T}(p.n)_\rho = \mathcal{T}(p)_\rho.1.n.1$$
$$\mathcal{T}(\Lambda X.E)_\rho = \lambda x.\mathcal{T}(E)_{\rho[X \mapsto x]} \quad (x \text{ fresh})$$
$$\mathcal{T}(E_1(E_2))_\rho = \mathcal{T}(E_1)_\rho \mathcal{T}(E_2)_\rho$$

Fig. 5. Translation from *Osan* to $\lambda_{\{\}}$

is likely that $\lambda_{\{\}}$ is not expressive enough to encode interesting core languages such as the call-by-value ML core language. Extending $\lambda_{\{\}}$ to accommodate call-by-value is future work.

Having outlined how to express particularities of *Osan*'s evaluation strategy, the translation is mostly as expected. A structure is translated into an unstable record. A core field e_j becomes $\{\mathcal{T}(e_j)_{\rho[X \mapsto x]}\}$. To uniformly access module and core fields, $\mathcal{T}(e_j)_{\rho[X \mapsto x]}$ is wrapped inside a record. A module field E_j becomes a value $\lfloor x_j \rfloor$, surrounded by a letrec to bind x_j to $\mathcal{T}(E_j)_{\rho[X \mapsto x]}$. The mediation by $\lfloor x_j \rfloor$ is for suspending evaluation of sub-modules, as explained above. It is important that the field order is preserved. The translation introduces an extra layer of a pointer to reference the unstable record. Precisely, the body of the letrec, namely x, is bound to $\lfloor x' \rfloor$, a pointer to x', and x' is to the unstable record $\{\overline{a_i}\}$. This is a trick to tame the potential interaction between module abbreviations and module recursion, which we will explain in Section 3.1. Correspondingly, translation of the dot notation inserts two extra projections; one for dereferencing the structure and one for dereferencing the field. For instance, a structure $\{(\mathsf{X})\ 2;\ \{(\mathsf{X'})\ 3;\ \mathsf{X}.2.1\}\}$ of *Osan* is translated into $\langle x \mid x = \lfloor x_1 \rfloor, x_1 = \{\{2\}; \langle \lfloor x_2 \rfloor \mid x_2 = \langle x' \mid x' = \lfloor x_3 \rfloor, x_3 = \{\{3\}; \{x.1.2.1.1.1.1\}\}\rangle\}\rangle$. Functors and functor applications in *Osan* respectively become functions and function applications in $\lambda_{\{\}}$.

In the separately attached document are found the results of the translation, as well as reductions, of selected examples from the paper. All examples are considered in [16].

Abbreviation Module abbreviations potentially interact with module recursion. Below is an example:

$$\{(\mathsf{X})\ \ \mathsf{M1} = \mathsf{X.M2};\ \ \mathsf{M2} = \{\ \mathsf{m1} = 0;\ \ \mathsf{m2} = \mathsf{X.M1.m1} + 1\ \};\ \ \mathsf{main} = \mathsf{X.M1.m2}\ \}$$

The core field M2.m2 refers to the preceding field M2.m1 via the forward referencing abbreviation M1 = X.M2. We want to successfully evaluate the above program, since we want to interpret abbreviations just as renaming for notational convenience; referring to a field through abbreviations should not make evaluation fail if the evaluation would succeed without abbreviations.

The desire to support flexible uses of abbreviations is the reason to have introduced an extra layer of a pointer to reference the record representing a

Signature	$T ::= \{\overline{S}\} \mid T_1 \rightarrow T_2$
Declarations	$S ::= m \mid M : T$
Definitions	$D ::= M = E \mid m = e$
Module expressions	$E ::= \{(X)\ \overline{D}\} \mid p \mid \Lambda X.E \mid E_1(E_2) \mid (E : T :> T')$
Paths	$p ::= X \mid p.n$
Core expressions	$e ::= p.n \mid \ldots$

Fig. 6. Syntax extended with signature constraints

structure. Then we can use the pointer to achieve the sharing of the same structure between abbreviations even during evaluating the fields of the structure.

4 Signature constraint

So far we have not considered an important feature of ML modules, namely the ability to constrain the interface of a module with a signature. Although we do not address typing issues in the paper, we are interested in considering signature constraint, since it effectively simulates a separate compilation scenario, where only public signatures are available for separately compiled modules. Then our challenge is how to access fields of a structure by indexes, when the public signature only exposes part of structure fields in a possibly different order from the actual definition order in the structure.

To examine the problem, we extend *Osan* with signature constraints as in Fig. 6. A *signature* is either a *structure type* or *functor type*. A structure type $\{\overline{S}\}$ is a sequence of declarations, specifying the "public" fields of a structure. A declaration either mentions a core name or publicizes a sub-module along with its signature. Since we do not deal with typing in the paper, a declaration of a core name does not carry its type. A functor type $T_1 \rightarrow T_2$ denotes an interface of functors which takes a module of interface T_1 as argument, then returns a module of interface T_2. A new construct, namely signature constraints $(E : T :> T')$, constrains the interface of E to T', where T is the signature of E. For the surface language, T would be omitted. In other words, if type inference is available for the core language, then T can be inferered.

4.1 Translation into $\lambda_{\{\}}$

For translating signature constraints into $\lambda_{\{\}}$, we use a similar technique to *module thinning*. Module thinning is a well-known technique of prunning and reordering fields of a structure according to the constraining signature. Some care is necessary to adapt the techunique for our lazy evaluation setting and to avoid introducing undesirable interaction between the top-down evaluation order and module recursion. We explain the two issues we are concerned about, using the following example:

```
{(X)
  M = ({ m1 = 1;  m2 = X.M.m1 + 1;  m3 = X.M.m2 + 1 } : { m2;  m1;  m3});
  main = X.M.m2 }
```

$\mathcal{T}((E : \{\overline{S}\} :> \{\overline{S'}_i\}))_\rho = \langle x \mid x = \lfloor x' \rfloor, x' = \{\overline{a}_i; \{x''.1.1.1\}\}, x'' = \mathcal{T}(E)_\rho \rangle$
where $\forall j \in \{1, \ldots, i\}$,
$a_j = \langle \lfloor x_j \rfloor \mid x_j = x''.1.n.1 \rangle$ when $S'_i = m$ and m is the n-th declaration in $\{\overline{S}\}$,
and $a_i = \langle \lfloor x_j \rfloor \mid x_j = \mathcal{T}((x''.1.n.1 : T :> T'))_\rho \rangle$ when $S'_i = M : T'$
and $M : T$ is the n-th declaration in $\{\overline{S}\}$ (x_j fresh)
$\mathcal{T}((E : T_1 \to T_2 :> T'_1 \to T'_2))_\rho =$
$\langle \lambda x'.\mathcal{T}((X(X' : T'_1 :> T_1) : T_2 :> T'_2))_{\rho[X \mapsto x][X' \mapsto x']} \mid x = \mathcal{T}(E)_\rho \rangle$ (x, x' fresh)

Fig. 7. Translation from *Osan* to $\lambda_{\{\}}$ with signature constraints

The above program is almost translated into the program below by the module thinning operation adapted for our lazy setting.

```
{(X)
  N = { m1 = 1;  m2 = X.M'.m1 + 1;  m3 = X.M'.m2 + 1 };
  M' = { m2 = X.N.m2; m1 = X.N.m1; m3 = X.N.m3};
  main = X.M'.m2 }
```

Above the module M' is a wrapper for the newly introduced module N, which contains the implementation. For explanatory purpose, we use the name M' for the wrapper module, but we could have used the same name M so as not to affect the rest of the program, i.e., then we need not modify the definition of main. Now accessing the first field of M' is delegated to the second field of N. Thus access to the field of M' by name m2 can be correctly translated into access by index 1, which can be done based on the constraining signature of M.

The first thing we observed is that to successfuly evaluate the translated program, we cannot enforce the top-down evaluation order within M'. To evaluate the field N.m2, the field M'.m1 should be accessible against the definition order in M', since the field M'.m2 cannot be evaluated yet. Indeed, it is enough to enforce the top-down evaluation order once in N, thus we need not enforce the top-down evaluation order in M'. In other words, any field of M' should be evaluated on a per-field basis when the field is projected. This is easy to express in $\lambda_{\{\}}$.

The second issue is more subtle. We cannot let evaluation of the field M'.m2 trigger evaluation of the module N, which implies M'.m2 will not be accessible until all the field of N have been evaluated. For the evaluation of the above program to be successful, the field M'.m2 should be accessible when evaluation N.m3. To address the second issue, we will include in M' an anonymous field _ = N.m1 which are to be evaluated before any field of M' when M' is first accessed. This is a bit of a hack, but is also easy to implement in $\lambda_{\{\}}$.

Finally we define translation from *Osan* into $\lambda_{\{\}}$ in Fig 7. We do not modify the previous translation of Fig. 5, which is not repeated here, but need to add two cases for translating signature constraints. The first case is for constraints by structure types. Continuing the above example, x' corresponds to the wrapper structure M' and x'' the implementation module N. All but the last fields of $\{\overline{a}_i; \{x''.1.1.1\}\}$ are pointers $\lfloor x_j \rfloor$, thus evaluation of x_j is suspended until x_j is projected. The surrounding letrec of $\lfloor x_j \rfloor$ binds x_j to a value path refering to the

corresponding core field of the constraiend module, or or the result of translating $(x''.1.n.1 : T :> T')$, where $x''.1.n.1$ refers to the corresponding module field and the constraint is pushed into the sub-module. The last field $\{x''.1.1.1\}$ implements the trick for the second issue, corresponding to the anonymous field $\_ = \mathsf{N.m1}$ in case of the above example. The second case is for constraints by functor types, The constraint is pushed into the functor argument contra-variantly, and into the body covariantly.

5 Related work

To the best of our knowledge, no previous work has formalized and examined a lazy evaluation strategy for ML-style modules supporting recursion. Our design principles resemble Syme's initialization graphs, which we first examine below. Much work has been devoted to statically guarantee initialization safety of Java-like languages and recursive modules with the traditional eager evaluation strategy. The objective of the paper is not about static guarantees of initialization safety. However we believe the techniques introduced in the previous work on this subject are potentially beneficial to ours to enhance safety by finding potential runtime errors at compile time. Hence we overview it in respect of its applicability to *Osan*. We end this section by reviewing OCaml's recursive modules, which are the starting point of our work.

Initialization graphs Syme proposed initialization graphs [22] to facilitate safer uses of unrestricted recursion for the ML core language. We explained in Section 2 how his work motivated our design principles. The technical development is different. We consider the module language, treating and examining an evaluation strategy for records (i.e., modules) in a special way, whereas Syme does the core language and no special attention is paid to records.

Initialization safety in object systems The problem of safe initialization of recursively defined records, or objects, has long been addressed in the object system community. Notably type systems for Java-like languages have been proposed to statically ensure the absence of null pointer dereferences [9, 10]. The situation they confront is different from ours at least in the following two points. On the one hand, restricted capability of generics (parameterized classes) in these languages, in comparison to higher-order functors, makes it possible to statically detect, thus reject, cyclic inheritance, whereas unrestricted combination of recursive modules and functors makes the detection of cyclically defined modules undecidable [17] and we check the absence of cycles at runtime. On the other hand, these object-oriented languages allow more flexible initialization patterns than *Osan*. Objects are more like records with lazy fields, with possibly safe default values. Thus, for instance, initialization can alternate between several objects recursively. Regardless of these differences, we are optimistic about adapting their ideas in null-pointer analysis to *Osan*. We are developing an algorithm for statically ensuring the absence of cyclic module definitions for programs which satisfy a certain criterion [17]; exploring the combination of the null-pointer analysis and the algorithm looks interesting future work.

Recursive modules and mixin modules Considerable effort has been made to investigate extensions with recursion of the ML module system and mixin modules, which can be seen as a generalization of recursive ML-style modules. Two difficulties involved are type checking and initialization. Below we review previous work on the latter; type checking is not in the scope of the paper, hence we refer overviews of the former to other papers [4, 21, 6, 19, 18].

Boudol [3], Hirschowitz & Leroy [13], and Dreyer [5] have proposed type systems which ensure initialization safety of recursively defined modules. They adopt the traditional eager evaluation strategy of the ML module system, where for the safe initialization a recursively defined module variable must not be dereferenced during evaluation of the bound module expression to the variable. As a result, initialization patterns they consider are much more limited than ours. Yet, these type systems are potentially beneficial to *Osan* in that programs that they deem well-typed are also error-free in respect of our strategy.

OCaml's recursive modules The experimental implementation of recursive modules in OCaml [15] was our starting point. In the family of the ML module system, OCaml's modules are evaluated eagerly. Initialization safety is not checked at compile time but at runtime like *Osan*, by throwing an exception. The compiler places a certain restriction on possible signatures of recursive modules, where the restriction allows the compiler to avoid introducing a lazy evaluation mechanism for module initialization. We have experimented with OCaml to have more experience in programming with recursive modules. They gave us a useful insight about what we can do with recursive modules and let us examine some corner cases that we need to deal with. One thing we found uncomfortable is the potential that a field of a recursively defined structure is left improperly initialized and we trigger a runtime error on attempting to access the field after the initialization phase has been completed. OCaml's less flexible handling of module and value abbreviations was sometimes inconvenient, too. These are one reason why we insist on the absence of runtime errors after the initialization and why we took substantial care of module abbreviations.

6 Conclusion

We have examined and formalized an evaluation strategy for a ML-style module system supporting recursion, by introducing a lazy evaluation mechanism in a disciplined way. On the one hand, the use of the laziness allows flexible handling of recursion between modules. On the other hand, its disciplined use keeps evaluation safer and the evaluation order more explicit than unconstrained uses. We formalized the strategy by translating the source language for recursive modules into the target language, which is an adaptation of the call-by-need letrec lambda calculus of Ariola and Felleisen. We believe the proposed strategy keeps a good tension between flexibility, simplicity and safety, and potential for optimization.

Acknowledgment I am grateful to Xavier Leroy for the valuable advice throughout the development of this work. I thank Andrew Tolmach and Jacques Gar-

rigue for fruitfull discussions, and reviewers of the previous drafts for constructive comments.

References

1. Z. Ariola and M. Felleisen. The Call-by-Need Lambda Calculus. *Journal of Functional Programming*, 7(3), 1997.
2. Z. Ariola, M. Felleisen, J. Maraist, M. Odersky, and P. Wadler. A Call-by-Need Lambda Calculus. In *Proc. POPL*, 1995.
3. G. Boudol. The recursive record semantics of objects revisited. *Journal of Functional Programming*, 14:263–315, 2004.
4. K. Crary, R. Harper, and S. Puri. What is a recursive module? In *Proc. PLDI*, pages 50–63, 1999.
5. D. Dreyer. A type system for well-founded recursion. In *Proc. POPL*, 2004.
6. D. Dreyer. Recursive Type Generativity. In *Proc. ICFP'05*. ACM Press, 2005.
7. M. Odersky et al. The Scala Programming Language. Software and documentation available on the Web, http://www.scala-lang.org/.
8. S. Fagorzi and E. Zucca. A Calculus of Open Modules: Call-by-need Strategy and Confluence,. *Mathematical Structures in Computer Science*, 17, 2007.
9. M. Fähndrich and R. Leino. Declaring and checking non-null types in an object-oriented language. In *Proc. OOPLSA*. ACM Press, 2003.
10. M. Fähndrich and S. Xia. Establishing Object Invariants with Delayed Types. In *Proc. OOPLSA*. ACM Press, 2007.
11. K. Fisher and J. Reppy. The Moby Programming Language. Software and documentation available on the Web, http://moby.cs.uchicago.edu/.
12. J. Gosling, B. Joy, G. Steele, and G. Bracha. *The Java Language Specification, Third Edition*, chapter 12.4. Prentice Hall, 2005.
13. T. Hirschowitz and X. Leroy. Mixin modules in a call-by-value setting. In *Proc. ESOP*, volume 2305, pages 6–20. Springer, 2002.
14. X. Leroy. A modular module system. *Journal of Functional Programming*, 10(3):269–303, 2000.
15. X. Leroy, D. Doligez, J. Garrigue, D. Rémy, and J. Vouillon. The Objective Caml system, release 3.10. Software and documentation available on the Web, http://caml.inria.fr/, 2007.
16. K. Nakata. Lazy modules: A lazy evaluation strategy for more recursive initialization patterns. Longer version available at http://gallium.inria.fr/ nakata/Osanlong.pdf, 2008.
17. K. Nakata and J. Garrigue. Path resolution for nested recursive modules. Submitted. Draft available on the Web, http://gallium.inria.fr/ nakata/pathresolve.pdf.
18. K. Nakata and J. Garrigue. Recursive Modules for Programming. In *Proc. ICFP*. ACM Press, 2006.
19. S. Owens and M. Flatt. From Structures and Functors to Modules and Units. In *Proc. ICFP*, 2006.
20. R.Milner, M. Tofte, R. Harper, and D. MacQueen. *The Definition of Standard ML - Revised*. The MIT Press, 1997.
21. C. Russo. Recursive Structures for Standard ML. In *Proc. ICFP'01*, pages 50–61. ACM Press, 2001.
22. D. Syme. Initializing mutually referential abstract objects: The value recursion challenge. In *Proc. Workshop on ML*, volume 148, pages 3–25, 2005.

23. D. Syme and J. Margetson. The F# Programming Language. Software and documentation available on the Web, http://research.microsoft.com/fsharp/fsharp.aspx.
24. J. Wells and R. Vestergaard. Equational reasoning for linking with first-class primitive modules. In *Proc. ESOP*, 2000.

7 Appendix

This appendix contains the results of the translation and the reductions of the selected examples from the paper. We use x, m, a, b, c, ... as metavariables for variables of $\lambda_{\{\}}$. The notation $\rightarrow^*$ denotes the transitive closure of $\rightarrow$. We use subscripts to indicate the rule(s) used for the reduction(s). To trigger evaluation of a program, we translate the top-level structure of a program using $\mathcal{P}(\_)$ defied by:

$$\mathcal{P}(\{(X)\ \overline{D}_i\}) = \langle x.1.i.1 \mid x = \lfloor x' \rfloor, x' = \{\overline{a}_i\}\rangle\ (x, x'\ \text{ fresh})$$
$$\text{where } \forall j \in \{1,\ldots,i-1\},\ \ a_j = \langle \lfloor x_j \rfloor \mid x_j = \mathcal{T}(D_j)_{[X\mapsto x]}\rangle\ (x_j \text{ fresh})$$
$$\text{and } a_i = \{\mathcal{T}(D_i)_{[X\mapsto x]}\}$$

Recall that a program is a structure containing a single core field named main and possibly several sub-modules. We further assume main is the last field of the structure, which is the case with all the examples in the paper.

7.1 The example from the introduction

We consider the following simpler variant of the example from the introduction:

```
{(X)
  M1 = Λx1.{(X2)
    m1 = fun x → if x = 0 then 0 else x1.1.1.1 (x-1)
    m2 = print (X2.1.1.1 3)}
  M2 = X.M1(X.M2)
  main = X.M2.m2 }
```

Execution of the above program succeeds. Below are the reductions.

```
⟨a.1.3.1 | a = ⌊a1⌋
  a1 = {
    ⟨⌊m1⌋ | m1 = λx1.⟨b | b = ⌊b1⌋, b1 = {{fun x → if x = 0 then 0 else x1.1.1.1 (x-1)};{print (b.1.1.1 3)}}⟩⟩;
    ⟨⌊m2⌋ | m2 = a.1.1.1 a.1.2.1⟩; {a.1.2.1.1.2.1}}⟩
→*6,2
⟨a1.3.1 | a = ⌊a1⌋
  a1 = {
    ⟨⌊m1⌋ | m1 = λx1.⟨b | b = ⌊b1⌋, b1 = {{fun x → if x = 0 then 0 else x1.1.1.1 (x-1)};{print (b.1.1.1 3)}}⟩⟩;
    ⟨⌊m2⌋ | m2 = a.1.1.1 a.1.2.1⟩; {a.1.2.1.1.2.1}}⟩
→*9,9
⟨a1.3.1 | a = ⌊a1⌋
  a1 = ⟨⟨{⌊m1⌋;⌊m2⌋;{a.1.2.1.1.2.1}} |
    m1 = λx1.⟨b | b = ⌊b1⌋, b1 = {{fun x → if x = 0 then 0 else x1.1.1.1 (x-1)};{print (b.1.1.1 3)}}⟩⟩ |
    m2 = a.1.1.1 a.1.2.1⟩⟩
→*7,2
```

⟨a1.3.1 | a = ⌊a1⌋
  a1 = ⟨⟨{⌊m1⌋;⌊m2⌋;{a1.2.1.1.2.1}} |
    m1 = λx1.⟨b | b = ⌊b1⌋, b1 = {{fun x → if x = 0 then 0 else x1.1.1.1 (x-1)};{print (b.1.1.1 3)}}⟩⟩ |
    m2 = a.1.1.1 a.1.2.1⟩⟩
$\longrightarrow^{*}_{12,12}$
⟨a1.3.1 | a = ⌊a1⌋
  a1 = {⌊m1⌋;⌊m2⌋;{a1.2.1.1.2.1}},
  m1 = λx1.⟨b | b = ⌊b1⌋, b1 = {{fun x → if x = 0 then 0 else x1.1.1.1 (x-1)};{print (b.1.1.1 3)}}⟩,
  m2 = a.1.1.1 a.1.2.1⟩
$\longrightarrow^{*}_{8,2,2}$
⟨a1.3.1 | a = ⌊a1⌋
  a1 = {⌊m1⌋;⌊m2⌋;{m2.1.2.1}},
  m1 = λx1.⟨b | b = ⌊b1⌋, b1 = {{fun x → if x = 0 then 0 else x1.1.1.1 (x-1)};{print (b.1.1.1 3)}}⟩,
  m2 = a.1.1.1 a.1.2.1⟩
$\longrightarrow^{*}_{7,2,8,2,2}$
⟨a1.3.1 | a = ⌊a1⌋
  a1 = {⌊m1⌋;⌊m2⌋;{m2.1.2.1}},
  m1 = λx1.⟨b | b = ⌊b1⌋, b1 = {{fun x → if x = 0 then 0 else x1.1.1.1 (x-1)};{print (b.1.1.1 3)}}⟩,
  m2 = m1 a.1.2.1⟩
$\longrightarrow_{7}$
⟨a1.3.1 | a = ⌊a1⌋
  a1 = {⌊m1⌋;⌊m2⌋;{m2.1.2.1}},
  m1 = λx1.⟨b | b = ⌊b1⌋, b1 = {{fun x → if x = 0 then 0 else x1.1.1.1 (x-1)};{print (b.1.1.1 3)}}⟩,
  m2 = λx1.⟨b | b = ⌊b1⌋, b1 = {{fun x → if x = 0 then 0 else x1.1.1.1 (x-1)};{print (b.1.1.1 3)}}⟩ a.1.2.1⟩
$\longrightarrow_{1}$
⟨a1.3.1 | a = ⌊a1⌋
  a1 = {⌊m1⌋;⌊m2⌋;{m2.1.2.1}},
  m1 = λx1.⟨b | b = ⌊b1⌋, b1 = {{fun x → if x = 0 then 0 else x1.1.1.1 (x-1)};{print (b.1.1.1 3)}}⟩,
  m2 = ⟨⟨b | b = ⌊b1⌋, b1 = {{fun x → if x = 0 then 0 else x1.1.1.1 (x-1)};{print (b.1.1.1 3)}}⟩ | x1 = a.1.2.1⟩⟩
$\longrightarrow^{*}_{6,11,11}$
⟨a1.3.1 | a = ⌊a1⌋
  a1 = {⌊m1⌋;⌊m2⌋;{m2.1.2.1}},
  m1 = λx1.⟨b | b = ⌊b1⌋, b1 = {{fun x → if x = 0 then 0 else x1.1.1.1 (x-1)};{print (b.1.1.1 3)}}⟩,
  m2 = ⌊b1⌋, b = ⌊b1⌋, b1 = {{fun x → if x = 0 then 0 else x1.1.1.1 (x-1)};{print (b.1.1.1 3)}}, x1 = a.1.2.1⟩
$\longrightarrow^{*}_{7,2}$
⟨a1.3.1 | a = ⌊a1⌋
  a1 = {⌊m1⌋;⌊m2⌋;{b1.2.1}},
  m1 = λx1.⟨b | b = ⌊b1⌋, b1 = {{fun x → if x = 0 then 0 else x1.1.1.1 (x-1)};{print (b.1.1.1 3)}}⟩,
  m2 = ⌊b1⌋, b = ⌊b1⌋, b1 = {{fun x → if x = 0 then 0 else x1.1.1.1 (x-1)};{print (b.1.1.1 3)}}, x1 = a.1.2.1⟩
$\longrightarrow_{3}$
⟨a1.3.1 | a = ⌊a1⌋
  a1 = {⌊m1⌋;⌊m2⌋;{b1.2.1}},
  m1 = λx1.⟨b | b = ⌊b1⌋, b1 = {{fun x → if x = 0 then 0 else x1.1.1.1 (x-1)};{print (b.1.1.1 3)}}⟩,
  m2 = ⌊b1⌋, b = ⌊b1⌋, b1 = {⌊fun x → if x = 0 then 0 else x1.1.1.1 (x-1)⌋;{print (b.1.1.1 3)}}, x1 = a.1.2.1⟩
$\longrightarrow^{*}_{7,2,8,2,2}$
⟨a1.3.1 | a = ⌊a1⌋
  a1 = {⌊m1⌋;⌊m2⌋;{b1.2.1}},
  m1 = λx1.⟨b | b = ⌊b1⌋, b1 = {{fun x → if x = 0 then 0 else x1.1.1.1 (x-1)};{print (b.1.1.1 3)}}⟩,
  m2 = ⌊b1⌋, b = ⌊b1⌋, b1 = {⌊fun x → if x = 0 then 0 else x1.1.1.1 (x-1)⌋;{print (x1.1.1.1 2)}}, x1 = a.1.2.1⟩
$\longrightarrow^{*}_{7,2,8,2,2}$
⟨a1.3.1 | a = ⌊a1⌋
  a1 = {⌊m1⌋;⌊m2⌋;{b1.2.1}},
  m1 = λx1.⟨b | b = ⌊b1⌋, b1 = {{fun x → if x = 0 then 0 else x1.1.1.1 (x-1)};{print (b.1.1.1 3)}}⟩,
  m2 = ⌊b1⌋, b = ⌊b1⌋, b1 = {⌊fun x → if x = 0 then 0 else x1.1.1.1 (x-1)⌋;{print (x1.1.1.1 2)}}, x1 = m2⟩
$\longrightarrow_{6}$

⟨a1.3.1 | a = ⌊a1⌋
  a1 = {⌊m1⌋;⌊m2⌋;{b1.2.1}},
  m1 = λx1.⟨b | b = ⌊b1⌋, b1 = {{fun x → if x = 0 then 0 else x1.1.1.1 (x-1)};{print (b.1.1.1 3)}}⟩,
  m2 = ⌊b1⌋, b = ⌊b1⌋, b1 = {⌊fun x → if x = 0 then 0 else x1.1.1.1 (x-1)⌋;{print (x1.1.1.1 2)}}, x1 = ⌊b1⌋⟩
$\rightarrow^*_{7,2,8,2,2}$
⟨a1.3.1 | a = ⌊a1⌋
  a1 = {⌊m1⌋;⌊m2⌋;{b1.2.1}},
  m1 = λx1.⟨b | b = ⌊b1⌋, b1 = {{fun x → if x = 0 then 0 else x1.1.1.1 (x-1)};{print (b.1.1.1 3)}}⟩,
  m2 = ⌊b1⌋, b = ⌊b1⌋, b1 = {⌊fun x → if x = 0 then 0 else x1.1.1.1 (x-1)⌋;{print (x1.1.1.1 1)}}, x1 = ⌊b1⌋⟩
$\rightarrow^*_{7,2,8,2,2}$
⟨a1.3.1 | a = ⌊a1⌋
  a1 = {⌊m1⌋;⌊m2⌋;{b1.2.1}},
  m1 = λx1.⟨b | b = ⌊b1⌋, b1 = {{fun x → if x = 0 then 0 else x1.1.1.1 (x-1)};{print (b.1.1.1 3)}}⟩,
  m2 = ⌊b1⌋, b = ⌊b1⌋, b1 = {⌊fun x → if x = 0 then 0 else x1.1.1.1 (x-1)⌋;{print (x1.1.1.1 0)}}, x1 = ⌊b1⌋⟩
$\rightarrow^*_{7,2,8,2,2}$
⟨a1.3.1 | a = ⌊a1⌋
  a1 = {⌊m1⌋;⌊m2⌋;{b1.2.1}},
  m1 = λx1.⟨b | b = ⌊b1⌋, b1 = {{fun x → if x = 0 then 0 else x1.1.1.1 (x-1)};{print (b.1.1.1 3)}}⟩,
  m2 = ⌊b1⌋, b = ⌊b1⌋, b1 = {⌊fun x → if x = 0 then 0 else x1.1.1.1 (x-1)⌋;{print 0}}, x1 = ⌊b1⌋⟩
→
0 is printed
⟨a1.3.1 | a = ⌊a1⌋
  a1 = {⌊m1⌋;⌊m2⌋;{b1.2.1}},
  m1 = λx1.⟨b | b = ⌊b1⌋, b1 = {{fun x → if x = 0 then 0 else x1.1.1.1 (x-1)};{print (b.1.1.1 3)}}⟩,
  m2 = ⌊b1⌋, b = ⌊b1⌋, b1 = {⌊fun x → if x = 0 then 0 else x1.1.1.1 (x-1)⌋;{0}}, x1 = ⌊b1⌋⟩
$\rightarrow^*_{3,3}$
⟨a1.3.1 | a = ⌊a1⌋
  a1 = {⌊m1⌋;⌊m2⌋;{b1.2.1}},
  m1 = λx1.⟨b | b = ⌊b1⌋, b1 = {{fun x → if x = 0 then 0 else x1.1.1.1 (x-1)};{print (b.1.1.1 3)}}⟩,
  m2 = ⌊b1⌋, b = ⌊b1⌋, b1 = ⌊⌊fun x → if x = 0 then 0 else x1.1.1.1 (x-1)⌋;⌊0⌋⌋, x1 = ⌊b1⌋⟩
$\rightarrow^*_{7,2,2}$
⟨a1.3.1 | a = ⌊a1⌋
  a1 = {⌊m1⌋;⌊m2⌋;{0}},
  m1 = λx1.⟨b | b = ⌊b1⌋, b1 = {{fun x → if x = 0 then 0 else x1.1.1.1 (x-1)};{print (b.1.1.1 3)}}⟩,
  m2 = ⌊b1⌋, b = ⌊b1⌋, b1 = ⌊⌊fun x → if x = 0 then 0 else x1.1.1.1 (x-1)⌋;⌊0⌋⌋, x1 = ⌊b1⌋⟩
$\rightarrow^*_{3,3}$
⟨a1.3.1 | a = ⌊a1⌋
  a1 = ⌊⌊m1⌋;⌊m2⌋;⌊0⌋⌋,
  m1 = λx1.⟨b | b = ⌊b1⌋, b1 = {{fun x → if x = 0 then 0 else x1.1.1.1 (x-1)};{print (b.1.1.1 3)}}⟩,
  m2 = ⌊b1⌋, b = ⌊b1⌋, b1 = ⌊⌊fun x → if x = 0 then 0 else x1.1.1.1 (x-1)⌋;⌊0⌋⌋, x1 = ⌊b1⌋⟩
$\rightarrow^*_{6,2,2}$
⟨0 | a = ⌊a1⌋
  a1 = ⌊⌊m1⌋;⌊m2⌋;⌊0⌋⌋,
  m1 = λx1.⟨b | b = ⌊b1⌋, b1 = {{fun x → if x = 0 then 0 else x1.1.1.1 (x-1)};{print (b.1.1.1 3)}}⟩,
  m2 = ⌊b1⌋, b = ⌊b1⌋, b1 = ⌊⌊fun x → if x = 0 then 0 else x1.1.1.1 (x-1)⌋;⌊0⌋⌋, x1 = ⌊b1⌋⟩

7.2 The first example for overview of the strategy (page 5)

The original program written in the syntax of *Osan* is:

```
{(X)
  M1 = {(X1)
    m1 = print 2;
    M2 = { m1 = 3;  m2 = print 4; };
    M3 = { m1 = print 5; };
    m2 = print 6;
    M4 = { m1 = print 7;  m2 = X1.M2.m1; };
    m3 = print 8; };
  main = print X.M1.M4.m2; }
```

Execution of the above program succeeds and prints "2 6 8 7 4 3" in this order. Below are the reductions.

$\langle$a.1.2.1 | a = $\lfloor$a1$\rfloor$,
  a1 = {$\langle\lfloor$m1$\rfloor$ | m1 = $\langle$b | b = $\lfloor$b1$\rfloor$, b1 = {{print 2};
    $\langle\lfloor$m2$\rfloor$ | m2 = $\langle$c | c = $\lfloor$c1$\rfloor$, c1 = {{3}; {print 4}}$\rangle\rangle$;
    $\langle\lfloor$m3$\rfloor$ | m3 = $\langle$d | d = $\lfloor$d1$\rfloor$, d1 = {{print 5}}$\rangle\rangle$; {print 6};
    $\langle\lfloor$m4$\rfloor$ | m4 = $\langle$e | e = $\lfloor$e1$\rfloor$, e1 = {{print 7}; {b.1.2.1.1.1.1}}$\rangle\rangle$; {print 8}}$\rangle\rangle$; {print a.1.1.1.1.5.1.1.2.1}}$\rangle$
$\rightarrow^{*}_{6,2}$
$\langle$a1.2.1 | a = $\lfloor$a1$\rfloor$,
  a1 = {$\langle\lfloor$m1$\rfloor$ | m1 = $\langle$b | b = $\lfloor$b1$\rfloor$, b1 = {{print 2};
    $\langle\lfloor$m2$\rfloor$ | m2 = $\langle$c | c = $\lfloor$c1$\rfloor$, c1 = {{3}; {print 4}}$\rangle\rangle$;
    $\langle\lfloor$m3$\rfloor$ | m3 = $\langle$d | d = $\lfloor$d1$\rfloor$, d1 = {{print 5}}$\rangle\rangle$; {print 6};
    $\langle\lfloor$m4$\rfloor$ | m4 = $\langle$e | e = $\lfloor$e1$\rfloor$, e1 = {{print 7}; {b.1.2.1.1.1.1}}$\rangle\rangle$; {print 8}}$\rangle\rangle$; {print a.1.1.1.1.5.1.1.2.1}}$\rangle$
$\rightarrow_{9}$
$\langle$a1.2.1 | a = $\lfloor$a1$\rfloor$,
  a1 = $\langle${$\lfloor$m1$\rfloor$;{print a.1.1.1.1.5.1.1.2.1}} | m1 = $\langle$b | b = $\lfloor$b1$\rfloor$, b1 = {{print 2};
    $\langle\lfloor$m2$\rfloor$ | m2 = $\langle$c | c = $\lfloor$c1$\rfloor$, c1 = {{3}; {print 4}}$\rangle\rangle$;
    $\langle\lfloor$m3$\rfloor$ | m3 = $\langle$d | d = $\lfloor$d1$\rfloor$, d1 = {{print 5}}$\rangle\rangle$; {print 6};
    $\langle\lfloor$m4$\rfloor$ | m4 = $\langle$e | e = $\lfloor$e1$\rfloor$, e1 = {{print 7}; {b.1.2.1.1.1.1}}$\rangle\rangle$; {print 8}}$\rangle\rangle\rangle$
$\rightarrow^{*}_{7,2}$
$\langle$a1.2.1 | a = $\lfloor$a1$\rfloor$,
  a1 = $\langle${$\lfloor$m1$\rfloor$;{print a1.1.1.1.5.1.1.2.1}} | m1 = $\langle$b | b = $\lfloor$b1$\rfloor$, b1 = {{print 2};
    $\langle\lfloor$m2$\rfloor$ | m2 = $\langle$c | c = $\lfloor$c1$\rfloor$, c1 = {{3}; {print 4}}$\rangle\rangle$;
    $\langle\lfloor$m3$\rfloor$ | m3 = $\langle$d | d = $\lfloor$d1$\rfloor$, d1 = {{print 5}}$\rangle\rangle$; {print 6};
    $\langle\lfloor$m4$\rfloor$ | m4 = $\langle$e | e = $\lfloor$e1$\rfloor$, e1 = {{print 7}; {b.1.2.1.1.1.1}}$\rangle\rangle$; {print 8}}$\rangle\rangle\rangle$
$\rightarrow_{12}$
$\langle$a1.2.1 | a = $\lfloor$a1$\rfloor$,
  a1 = {$\lfloor$m1$\rfloor$;{print a1.1.1.1.5.1.1.2.1}}, m1 = $\langle$b | b = $\lfloor$b1$\rfloor$, b1 = {{print 2};
    $\langle\lfloor$m2$\rfloor$ | m2 = $\langle$c | c = $\lfloor$c1$\rfloor$, c1 = {{3}; {print 4}}$\rangle\rangle$;
    $\langle\lfloor$m3$\rfloor$ | m3 = $\langle$d | d = $\lfloor$d1$\rfloor$, d1 = {{print 5}}$\rangle\rangle$; {print 6};
    $\langle\lfloor$m4$\rfloor$ | m4 = $\langle$e | e = $\lfloor$e1$\rfloor$, e1 = {{print 7}; {b.1.2.1.1.1.1}}$\rangle\rangle$; {print 8}}$\rangle\rangle$
$\rightarrow^{*}_{8,2}$
$\langle$a1.2.1 | a = $\lfloor$a1$\rfloor$,
  a1 = {$\lfloor$m1$\rfloor$;{print m1.1.5.1.1.2.1}}, m1 = $\langle$b | b = $\lfloor$b1$\rfloor$, b1 = {{print 2};
    $\langle\lfloor$m2$\rfloor$ | m2 = $\langle$c | c = $\lfloor$c1$\rfloor$, c1 = {{3}; {print 4}}$\rangle\rangle$;
    $\langle\lfloor$m3$\rfloor$ | m3 = $\langle$d | d = $\lfloor$d1$\rfloor$, d1 = {{print 5}}$\rangle\rangle$; {print 6};
    $\langle\lfloor$m4$\rfloor$ | m4 = $\langle$e | e = $\lfloor$e1$\rfloor$, e1 = {{print 7}; {b.1.2.1.1.1.1}}$\rangle\rangle$; {print 8}}$\rangle\rangle$
$\rightarrow^{*}_{6,11}$
$\langle$a1.2.1 | a = $\lfloor$a1$\rfloor$,
  a1 = {$\lfloor$m1$\rfloor$;{print m1.1.5.1.1.2.1}}, m1 = $\lfloor$b1$\rfloor$, b = $\lfloor$b1$\rfloor$,
  b1 = {{print 2}; $\langle\lfloor$m2$\rfloor$ | m2 = $\langle$c | c = $\lfloor$c1$\rfloor$, c1 = {{3}; {print 4}}$\rangle\rangle$;
    $\langle\lfloor$m3$\rfloor$ | m3 = $\langle$d | d = $\lfloor$d1$\rfloor$, d1 = {{print 5}}$\rangle\rangle$; {print 6};
    $\langle\lfloor$m4$\rfloor$ | m4 = $\langle$e | e = $\lfloor$e1$\rfloor$, e1 = {{print 7}; {b.1.2.1.1.1.1}}$\rangle\rangle$; {print 8}}$\rangle$
$\rightarrow^{*}_{7,2}$

⟨a1.2.1 | a = ⌊a1⌋,
  a1 = {⌊m1⌋;{print b1.5.1.1.2.1}}, m1 = ⌊b1⌋, b = ⌊b1⌋,
  b1 = {{print 2}; ⟨⌊m2⌋ | m2 = ⟨c | c = ⌊c1⌋, c1 = {{3}; {print 4}}⟩⟩;
    ⟨⌊m3⌋ | m3 = ⟨d | d = ⌊d1⌋, d1 = {{print 5}}⟩⟩; {print 6};
    ⟨⌊m4⌋ | m4 = ⟨e | e = ⌊e1⌋, e1 = {{print 7}; {b.1.2.1.1.1.1}}⟩⟩; {print 8}}⟩
→

2 is printed

⟨a1.2.1 | a = ⌊a1⌋,
  a1 = {⌊m1⌋;{print b1.5.1.1.2.1}}, m1 = ⌊b1⌋, b = ⌊b1⌋,
  b1 = {{2}; ⟨⌊m2⌋ | m2 = ⟨c | c = ⌊c1⌋, c1 = {{3}; {print 4}}⟩⟩;
    ⟨⌊m3⌋ | m3 = ⟨d | d = ⌊d1⌋, d1 = {{print 5}}⟩⟩; {print 6};
    ⟨⌊m4⌋ | m4 = ⟨e | e = ⌊e1⌋, e1 = {{print 7}; {b.1.2.1.1.1.1}}⟩⟩; {print 8}}⟩
$\rightarrow_{3}$
⟨a1.2.1 | a = ⌊a1⌋,
  a1 = {⌊m1⌋;{print b1.5.1.1.2.1}}, m1 = ⌊b1⌋, b = ⌊b1⌋,
  b1 = {⌊2⌋; ⟨⌊m2⌋ | m2 = ⟨c | c = ⌊c1⌋, c1 = {{3}; {print 4}}⟩⟩;
    ⟨⌊m3⌋ | m3 = ⟨d | d = ⌊d1⌋, d1 = {{print 5}}⟩⟩; {print 6};
    ⟨⌊m4⌋ | m4 = ⟨e | e = ⌊e1⌋, e1 = {{print 7}; {b.1.2.1.1.1.1}}⟩⟩; {print 8}}⟩
$\rightarrow^{*}_{9,9}$
⟨a1.2.1 | a = ⌊a1⌋,
  a1 = {⌊m1⌋;{print b1.5.1.1.2.1}}, m1 = ⌊b1⌋, b = ⌊b1⌋,
  b1 = ⟨⟨{⌊2⌋; ⌊m2⌋; ⌊m3⌋;{print 6};
    ⟨⌊m4⌋ | m4 = ⟨e | e = ⌊e1⌋, e1 = {{print 7}; {b.1.2.1.1.1.1}}⟩⟩; {print 8}} |
    m2 = ⟨c | c = ⌊c1⌋, c1 = {{3}; {print 4}}⟩⟩ | m3 = ⟨d | d = ⌊d1⌋, d1 = {{print 5}}⟩⟩⟩

6 is printed.

$\rightarrow_{3}$
⟨a1.2.1 | a = ⌊a1⌋,
  a1 = {⌊m1⌋;{print b1.5.1.1.2.1}}, m1 = ⌊b1⌋, b = ⌊b1⌋,
  b1 = ⟨⟨{⌊2⌋; ⌊m2⌋; ⌊m3⌋;⌊6⌋;
    ⟨⌊m4⌋ | m4 = ⟨e | e = ⌊e1⌋, e1 = {{print 7}; {b.1.2.1.1.1.1}}⟩⟩; {print 8}} |
    m2 = ⟨c | c = ⌊c1⌋, c1 = {{3}; {print 4}}⟩⟩ | m3 = ⟨d | d = ⌊d1⌋, d1 = {{print 5}}⟩⟩⟩

8 is printed.

$\rightarrow^{*}_{9,3,3}$
⟨a1.2.1 | a = ⌊a1⌋,
  a1 = {⌊m1⌋;{print b1.5.1.1.2.1}}, m1 = ⌊b1⌋, b = ⌊b1⌋,
  b1 = ⟨⟨⟨⌊⌊2⌋; ⌊m2⌋; ⌊m3⌋;⌊6⌋; ⌊m4⌋; ⌊8⌋⌋ |
    m2 = ⟨c | c = ⌊c1⌋, c1 = {{3}; {print 4}}⟩⟩ | m3 = ⟨d | d = ⌊d1⌋, d1 = {{print 5}}⟩⟩ |
    m4 = ⟨e | e = ⌊e1⌋, e1 = {{print 7}; {b.1.2.1.1.1.1}}⟩⟩⟩
$\rightarrow^{*}_{11,11,11}$
⟨a1.2.1 | a = ⌊a1⌋,
  a1 = {⌊m1⌋;{print b1.5.1.1.2.1}}, m1 = ⌊b1⌋, b = ⌊b1⌋,
  b1 = ⌊⌊2⌋; ⌊m2⌋; ⌊m3⌋; ⌊6⌋; ⌊m4⌋; ⌊8⌋⌋,
  m2 = ⟨c | c = ⌊c1⌋, c1 = {{3}; {print 4}}⟩, m3 = ⟨d | d = ⌊d1⌋, d1 = {{print 5}}⟩,
  m4 = ⟨e | e = ⌊e1⌋, e1 = {{print 7}; {b.1.2.1.1.1.1}}⟩⟩
$\rightarrow^{*}_{7,2,2}$
⟨a1.2.1 | a = ⌊a1⌋,
  a1 = {⌊m1⌋;{print m4.1.2.1}}, m1 = ⌊b1⌋, b = ⌊b1⌋,
  b1 = ⌊⌊2⌋; ⌊m2⌋; ⌊m3⌋; ⌊6⌋; ⌊m4⌋; ⌊8⌋⌋,
  m2 = ⟨c | c = ⌊c1⌋, c1 = {{3}; {print 4}}⟩, m3 = ⟨d | d = ⌊d1⌋, d1 = {{print 5}}⟩,
  m4 = ⟨e | e = ⌊e1⌋, e1 = {{print 7}; {b.1.2.1.1.1.1}}⟩⟩
$\rightarrow^{*}_{6,11}$

⟨a1.2.1 | a = ⌊a1⌋,
a1 = {⌊m1⌋;{print m4.1.2.1}}, m1 = ⌊b1⌋, b = ⌊b1⌋,
b1 = ⌊⌊2⌋; ⌊m2⌋; ⌊m3⌋; ⌊6⌋; ⌊m4⌋; ⌊8⌋⌋,
m2 = ⟨c | c = ⌊c1⌋, c1 = {{3}; {print 4}}⟩, m3 = ⟨d | d = ⌊d1⌋, d1 = {{print 5}}⟩,
m4 = ⌊e1⌋, e = ⌊e1⌋, e1 = {{print 7}; {b.1.2.1.1.1.1}}⟩

$\rightarrow^*_{7,2}$

⟨a1.2.1 | a = ⌊a1⌋,
a1 = {⌊m1⌋;{print e1.2.1}}, m1 = ⌊b1⌋, b = ⌊b1⌋,
b1 = ⌊⌊2⌋; ⌊m2⌋; ⌊m3⌋; ⌊6⌋; ⌊m4⌋; ⌊8⌋⌋,
m2 = ⟨c | c = ⌊c1⌋, c1 = {{3}; {print 4}}⟩, m3 = ⟨d | d = ⌊d1⌋, d1 = {{print 5}}⟩,
m4 = ⌊e1⌋, e = ⌊e1⌋, e1 = {{print 7}; {b.1.2.1.1.1.1}}⟩

7 is printed.

$\rightarrow_3$

⟨a1.2.1 | a = ⌊a1⌋,
a1 = {⌊m1⌋;{print e1.2.1}}, m1 = ⌊b1⌋, b = ⌊b1⌋,
b1 = ⌊⌊2⌋; ⌊m2⌋; ⌊m3⌋; ⌊6⌋; ⌊m4⌋; ⌊8⌋⌋,
m2 = ⟨c | c = ⌊c1⌋, c1 = {{3}; {print 4}}⟩, m3 = ⟨d | d = ⌊d1⌋, d1 = {{print 5}}⟩,
m4 = ⌊e1⌋, e = ⌊e1⌋, e1 = {⌊7⌋; {b.1.2.1.1.1.1}}⟩

$\rightarrow^*_{7,2,7,2,2}$

⟨a1.2.1 | a = ⌊a1⌋,
a1 = {⌊m1⌋;{print e1.2.1}}, m1 = ⌊b1⌋, b = ⌊b1⌋,
b1 = ⌊⌊2⌋; ⌊m2⌋; ⌊m3⌋; ⌊6⌋; ⌊m4⌋; ⌊8⌋⌋,
m2 = ⟨c | c = ⌊c1⌋, c1 = {{3}; {print 4}}⟩, m3 = ⟨d | d = ⌊d1⌋, d1 = {{print 5}}⟩,
m4 = ⌊e1⌋, e = ⌊e1⌋, e1 = {⌊7⌋; {m2.1.1.1}}⟩

$\rightarrow^*_{6,11}$

⟨a1.2.1 | a = ⌊a1⌋,
a1 = {⌊m1⌋;{print e1.2.1}}, m1 = ⌊b1⌋, b = ⌊b1⌋,
b1 = ⌊⌊2⌋; ⌊m2⌋; ⌊m3⌋; ⌊6⌋; ⌊m4⌋; ⌊8⌋⌋,
m2 = ⌊c1⌋, c = ⌊c1⌋, c1 = {{3}; {print 4}}, m3 = ⟨d | d = ⌊d1⌋, d1 = {{print 5}}⟩,
m4 = ⌊e1⌋, e = ⌊e1⌋, e1 = {⌊7⌋; {m2.1.1.1}}⟩

$\rightarrow^*_{7,2}$

⟨a1.2.1 | a = ⌊a1⌋,
a1 = {⌊m1⌋;{print e1.2.1}}, m1 = ⌊b1⌋, b = ⌊b1⌋,
b1 = ⌊⌊2⌋; ⌊m2⌋; ⌊m3⌋; ⌊6⌋; ⌊m4⌋; ⌊8⌋⌋,
m2 = ⌊c1⌋, c = ⌊c1⌋, c1 = {{3}; {print 4}}, m3 = ⟨d | d = ⌊d1⌋, d1 = {{print 5}}⟩,
m4 = ⌊e1⌋, e = ⌊e1⌋, e1 = {⌊7⌋; {c1.1}}⟩

4 is printed.

$\rightarrow^*_{3,3,3}$

⟨a1.2.1 | a = ⌊a1⌋,
a1 = {⌊m1⌋;{print e1.2.1}}, m1 = ⌊b1⌋, b = ⌊b1⌋,
b1 = ⌊⌊2⌋; ⌊m2⌋; ⌊m3⌋; ⌊6⌋; ⌊m4⌋; ⌊8⌋⌋,
m2 = ⌊c1⌋, c = ⌊c1⌋, c1 = ⌊⌊3⌋; ⌊4⌋⌋, m3 = ⟨d | d = ⌊d1⌋, d1 = {{print 5}}⟩,
m4 = ⌊e1⌋, e = ⌊e1⌋, e1 = {⌊7⌋; {c1.1}}⟩

$\rightarrow^*_{7,2}$

⟨a1.2.1 | a = ⌊a1⌋,
a1 = {⌊m1⌋;{print e1.2.1}}, m1 = ⌊b1⌋, b = ⌊b1⌋,
b1 = ⌊⌊2⌋; ⌊m2⌋; ⌊m3⌋; ⌊6⌋; ⌊m4⌋; ⌊8⌋⌋,
m2 = ⌊c1⌋, c = ⌊c1⌋, c1 = ⌊⌊3⌋; ⌊4⌋⌋, m3 = ⟨d | d = ⌊d1⌋, d1 = {{print 5}}⟩,
m4 = ⌊e1⌋, e = ⌊e1⌋, e1 = {⌊7⌋; {3}}⟩

$\rightarrow^*_{3,3}$

⟨a1.2.1 | a = ⌊a1⌋,
  a1 = {⌊m1⌋;{print e1.2.1}}, m1 = ⌊b1⌋, b = ⌊b1⌋,
  b1 = ⌊[2]; ⌊m2⌋; ⌊m3⌋; ⌊6⌋; ⌊m4⌋; ⌊8⌋⌋,
  m2 = ⌊c1⌋, c = ⌊c1⌋, c1 = ⌊⌊3⌋; ⌊4⌋⌋, m3 = ⟨d | d = ⌊d1⌋, d1 = {{print 5}}⟩,
  m4 = ⌊e1⌋, e = ⌊e1⌋, e1 = ⌊⌊7⌋; ⌊3⌋⌋⟩
$\rightarrow^*_{7,2,2}$
⟨a1.2.1 | a = ⌊a1⌋,
  a1 = {⌊m1⌋;{print 3}}, m1 = ⌊b1⌋, b = ⌊b1⌋,
  b1 = ⌊[2]; ⌊m2⌋; ⌊m3⌋; ⌊6⌋; ⌊m4⌋; ⌊8⌋⌋,
  m2 = ⌊c1⌋, c = ⌊c1⌋, c1 = ⌊⌊3⌋; ⌊4⌋⌋, m3 = ⟨d | d = ⌊d1⌋, d1 = {{print 5}}⟩,
  m4 = ⌊e1⌋, e = ⌊e1⌋, e1 = ⌊⌊7⌋; ⌊3⌋⌋⟩
3 is printed.
$\rightarrow^*_{3,3}$
⟨a1.2.1 | a = ⌊a1⌋,
  a1 = ⌊⌊m1⌋;⌊3⌋⌋, m1 = ⌊b1⌋, b = ⌊b1⌋,
  b1 = ⌊[2]; ⌊m2⌋; ⌊m3⌋; ⌊6⌋; ⌊m4⌋; ⌊8⌋⌋,
  m2 = ⌊c1⌋, c = ⌊c1⌋, c1 = ⌊⌊3⌋; ⌊4⌋⌋, m3 = ⟨d | d = ⌊d1⌋, d1 = {{print 5}}⟩,
  m4 = ⌊e1⌋, e = ⌊e1⌋, e1 = ⌊⌊7⌋; ⌊3⌋⌋⟩
$\rightarrow^*_{6,2,2}$
⟨3 | a = ⌊a1⌋,
  a1 = ⌊⌊m1⌋;⌊3⌋⌋, m1 = ⌊b1⌋, b = ⌊b1⌋,
  b1 = ⌊[2]; ⌊m2⌋; ⌊m3⌋; ⌊6⌋; ⌊m4⌋; ⌊8⌋⌋,
  m2 = ⌊c1⌋, c = ⌊c1⌋, c1 = ⌊⌊3⌋; ⌊4⌋⌋, m3 = ⟨d | d = ⌊d1⌋, d1 = {{print 5}}⟩,
  m4 = ⌊e1⌋, e = ⌊e1⌋, e1 = ⌊⌊7⌋; ⌊3⌋⌋⟩

7.3 The example for lazy functors (page 6 above)

The original program written in the syntax of *Osan* is:

```
{(X)
  M1 = ΛX1.{ m1 = print 1; m2 = print X1.m1 };
  M2 = { m1 = 2; m2 = print 3 };
  M3 = X.M1(X.M2);
  main = X.M3.m1 }
```

Execution of the above program succeeds and prints “1 3 2” in this order. Below are the reductions.

⟨a.1.4.1 | a = ⌊a1⌋,
  a1 = {
    ⟨⌊m1⌋ | m1 = λx.⟨b | b = ⌊b1⌋, b1 = {{print 1}; {print x.1.1.1}}⟩⟩;
    ⟨⌊m2⌋ | m2 = ⟨c | c = ⌊c1⌋, c1 = {{2}; {print 3}}⟩⟩;
    ⟨⌊m3⌋ | m3 = a.1.1.1 a.1.2.1⟩; {a.1.3.1.1.1.1}}⟩
$\rightarrow_6$
⟨⌊a1⌋.1.4.1 | a = ⌊a1⌋,
  a1 = {
    ⟨⌊m1⌋ | m1 = λx.⟨b | b = ⌊b1⌋, b1 = {{print 1}; {print x.1.1.1}}⟩⟩;
    ⟨⌊m2⌋ | m2 = ⟨c | c = ⌊c1⌋, c1 = {{2}; {print 3}}⟩⟩;
    ⟨⌊m3⌋ | m3 = a.1.1.1 a.1.2.1⟩; {a.1.3.1.1.1.1}}⟩
$\rightarrow_2$
⟨a1.4.1 | a = ⌊a1⌋,
  a1 = {
    ⟨⌊m1⌋ | m1 = λx.⟨b | b = ⌊b1⌋, b1 = {{print 1}; {print x.1.1.1}}⟩⟩;
    ⟨⌊m2⌋ | m2 = ⟨c | c = ⌊c1⌋, c1 = {{2}; {print 3}}⟩⟩;
    ⟨⌊m3⌋ | m3 = a.1.1.1 a.1.2.1⟩; {a.1.3.1.1.1.1}}⟩
$\rightarrow_9$

⟨a1.4.1 | a = ⌊a1⌋,
  a1 = ⟨{⌊m1⌋;
    ⟨⌊m2⌋ | m2 = ⟨c | c = ⌊c1⌋, c1 = {{2}; {print 3}}⟩⟩;
    ⟨⌊m3⌋ | m3 = a.1.1.1 a.1.2.1⟩; {a.1.3.1.1.1.1}} | m1 = λx.⟨b | b = ⌊b1⌋, b1 = {{print 1}; {print x.1.1.1}}⟩⟩⟩
$\longrightarrow_{9}$
⟨a1.4.1 | a = ⌊a1⌋,
  a1 = ⟨⟨{⌊m1⌋; ⌊m2⌋; ⟨⌊m3⌋ | m3 = a.1.1.1 a.1.2.1⟩; {a.1.3.1.1.1.1}} |
    m2 = ⟨c | c = ⌊c1⌋, c1 = {{2}; {print 3}}⟩⟩ | m1 = λx.⟨b | b = ⌊b1⌋, b1 = {{print 1}; {print x.1.1.1}}⟩⟩⟩
$\longrightarrow_{9}$
⟨a1.4.1 | a = ⌊a1⌋,
  a1 = ⟨⟨⟨{⌊m1⌋; ⌊m2⌋; ⌊m3⌋; {a.1.3.1.1.1.1}} | m3 = a.1.1.1 a.1.2.1⟩ |
    m2 = ⟨c | c = ⌊c1⌋, c1 = {{2}; {print 3}}⟩⟩ | m1 = λx.⟨b | b = ⌊b1⌋, b1 = {{print 1}; {print x.1.1.1}}⟩⟩⟩
$\longrightarrow_{7}$
⟨a1.4.1 | a = ⌊a1⌋,
  a1 = ⟨⟨⟨{⌊m1⌋; ⌊m2⌋; ⌊m3⌋; {⌊a1⌋.1.3.1.1.1.1}} | m3 = a.1.1.1 a.1.2.1⟩ |
    m2 = ⟨c | c = ⌊c1⌋, c1 = {{2}; {print 3}}⟩⟩ | m1 = λx.⟨b | b = ⌊b1⌋, b1 = {{print 1}; {print x.1.1.1}}⟩⟩⟩
$\longrightarrow_{2}$
⟨a1.4.1 | a = ⌊a1⌋,
  a1 = ⟨⟨⟨{⌊m1⌋; ⌊m2⌋; ⌊m3⌋; {a1.3.1.1.1.1}} | m3 = a.1.1.1 a.1.2.1⟩ |
    m2 = ⟨c | c = ⌊c1⌋, c1 = {{2}; {print 3}}⟩⟩ | m1 = λx.⟨b | b = ⌊b1⌋, b1 = {{print 1}; {print x.1.1.1}}⟩⟩⟩
$\longrightarrow_{12}$
⟨a1.4.1 | a = ⌊a1⌋,
  a1 = ⟨⟨{⌊m1⌋; ⌊m2⌋; ⌊m3⌋; {a1.3.1.1.1.1}} | m3 = a.1.1.1 a.1.2.1⟩ | m2 = ⟨c | c = ⌊c1⌋, c1 = {{2}; {print 3}}⟩⟩,
  m1 = λx.⟨b | b = ⌊b1⌋, b1 = {{print 1}; {print x.1.1.1}}⟩⟩
$\longrightarrow_{12}$
⟨a1.4.1 | a = ⌊a1⌋,
  a1 = ⟨{⌊m1⌋; ⌊m2⌋; ⌊m3⌋; {a1.3.1.1.1.1}} | m3 = a.1.1.1 a.1.2.1⟩,
  m2 = ⟨c | c = ⌊c1⌋, c1 = {{2}; {print 3}}⟩,
  m1 = λx.⟨b | b = ⌊b1⌋, b1 = {{print 1}; {print x.1.1.1}}⟩⟩
$\longrightarrow_{12}$
⟨a1.4.1 | a = ⌊a1⌋,
  a1 = {⌊m1⌋; ⌊m2⌋; ⌊m3⌋; {a1.3.1.1.1.1}},
  m3 = a.1.1.1 a.1.2.1, m2 = ⟨c | c = ⌊c1⌋, c1 = {{2}; {print 3}}⟩,
  m1 = λx.⟨b | b = ⌊b1⌋, b1 = {{print 1}; {print x.1.1.1}}⟩⟩
$\longrightarrow_{8}$
⟨a1.4.1 | a = ⌊a1⌋,
  a1 = {⌊m1⌋; ⌊m2⌋; ⌊m3⌋; {⌊⌊m1⌋; ⌊m2⌋; ⌊m3⌋⌋.3.1.1.1.1}},
  m3 = a.1.1.1 a.1.2.1,
  m2 = ⟨c | c = ⌊c1⌋, c1 = {{2}; {print 3}}⟩, m1 = λx.⟨b | b = ⌊b1⌋, b1 = {{print 1}; {print x.1.1.1}}⟩⟩
$\longrightarrow_{2}$
⟨a1.4.1 | a = ⌊a1⌋,
  a1 = {⌊m1⌋; ⌊m2⌋; ⌊m3⌋; {⌊m3⌋.1.1.1.1}},
  m3 = a.1.1.1 a.1.2.1, m2 = ⟨c | c = ⌊c1⌋, c1 = {{2}; {print 3}}⟩,
  m1 = λx.⟨b | b = ⌊b1⌋, b1 = {{print 1}; {print x.1.1.1}}⟩⟩
$\longrightarrow_{2}$
⟨a1.4.1 | a = ⌊a1⌋,
  a1 = {⌊m1⌋; ⌊m2⌋; ⌊m3⌋; {m3.1.1.1}},
  m3 = a.1.1.1 a.1.2.1, m2 = ⟨c | c = ⌊c1⌋, c1 = {{2}; {print 3}}⟩,
  m1 = λx.⟨b | b = ⌊b1⌋, b1 = {{print 1}; {print x.1.1.1}}⟩⟩
$\longrightarrow^{*}_{7,2}$
⟨a1.4.1 | a = ⌊a1⌋,
  a1 = {⌊m1⌋; ⌊m2⌋; ⌊m3⌋; {m3.1.1.1}},
  m3 = a1.1.1 a.1.2.1, m2 = ⟨c | c = ⌊c1⌋, c1 = {{2}; {print 3}}⟩,
  m1 = λx.⟨b | b = ⌊b1⌋, b1 = {{print 1}; {print x.1.1.1}}⟩⟩
$\longrightarrow_{8}$
⟨a1.4.1 | a = ⌊a1⌋,
  a1 = {⌊m1⌋; ⌊m2⌋; ⌊m3⌋; {m3.1.1.1}},
  m3 = ⌊⌊m1⌋; ⌊m2⌋; ⌊m3⌋⌋.1.1 a.1.2.1, m2 = ⟨c | c = ⌊c1⌋, c1 = {{2}; {print 3}}⟩,
  m1 = λx.⟨b | b = ⌊b1⌋, b1 = {{print 1}; {print x.1.1.1}}⟩⟩

$\longrightarrow^{*}_{2,2}$
⟨a1.4.1 | a = ⌊a1⌋,
  a1 = {⌊m1⌋; ⌊m2⌋; ⌊m3⌋; {m3.1.1.1}},
  m3 = m1 a.1.2.1, m2 = ⟨c | c = ⌊c1⌋, c1 = {{2}; {print 3}}⟩,
  m1 = λx.⟨b | b = ⌊b1⌋, b1 = {{print 1}; {print x.1.1.1}}⟩⟩
$\longrightarrow_{7}$
⟨a1.4.1 | a = ⌊a1⌋,
  a1 = {⌊m1⌋; ⌊m2⌋; ⌊m3⌋; {m3.1.1.1}},
  m3 = (λx.⟨b | b = ⌊b1⌋, b1 = {{print 1}; {print x.1.1.1}}⟩) a.1.2.1,
  m2 = ⟨c | c = ⌊c1⌋, c1 = {{2}; {print 3}}⟩, m1 = λx.⟨b | b = ⌊b1⌋, b1 = {{print 1}; {print x.1.1.1}}⟩⟩
$\longrightarrow_{1}$
⟨a1.4.1 | a = ⌊a1⌋,
  a1 = {⌊m1⌋; ⌊m2⌋; ⌊m3⌋; {m3.1.1.1}},
  m3 = ⟨⟨b | b = ⌊b1⌋, b1 = {{print 1}; {print x.1.1.1}}⟩ | x = a.1.2.1⟩,
  m2 = ⟨c | c = ⌊c1⌋, c1 = {{2}; {print 3}}⟩, m1 = λx.⟨b | b = ⌊b1⌋, b1 = {{print 1}; {print x.1.1.1}}⟩⟩
$\longrightarrow_{6}$
⟨a1.4.1 | a = ⌊a1⌋,
  a1 = {⌊m1⌋; ⌊m2⌋; ⌊m3⌋; {m3.1.1.1}},
  m3 = ⟨⟨⌊b1⌋ | b = ⌊b1⌋, b1 = {{print 1}; {print x.1.1.1}}⟩ | x = a.1.2.1⟩,
  m2 = ⟨c | c = ⌊c1⌋, c1 = {{2}; {print 3}}⟩, m1 = λx.⟨b | b = ⌊b1⌋, b1 = {{print 1}; {print x.1.1.1}}⟩⟩
$\longrightarrow_{11}$
⟨a1.4.1 | a = ⌊a1⌋,
  a1 = {⌊m1⌋; ⌊m2⌋; ⌊m3⌋; {m3.1.1.1}},
  m3 = ⟨⌊b1⌋ | b = ⌊b1⌋, b1 = {{print 1}; {print x.1.1.1}}⟩, x = a.1.2.1,
  m2 = ⟨c | c = ⌊c1⌋, c1 = {{2}; {print 3}}⟩, m1 = λx.⟨b | b = ⌊b1⌋, b1 = {{print 1}; {print x.1.1.1}}⟩⟩
$\longrightarrow_{11}$
⟨a1.4.1 | a = ⌊a1⌋,
  a1 = {⌊m1⌋; ⌊m2⌋; ⌊m3⌋; {m3.1.1.1}},
  m3 = ⌊b1⌋, b = ⌊b1⌋, b1 = {{print 1}; {print x.1.1.1}}, x = a.1.2.1,
  m2 = ⟨c | c = ⌊c1⌋, c1 = {{2}; {print 3}}⟩, m1 = λx.⟨b | b = ⌊b1⌋, b1 = {{print 1}; {print x.1.1.1}}⟩⟩
$\longrightarrow^{*}_{7,2}$
⟨a1.4.1 | a = ⌊a1⌋,
  a1 = {⌊m1⌋; ⌊m2⌋; ⌊m3⌋; {b1.1.1}},
  m3 = ⌊b1⌋, b = ⌊b1⌋, b1 = {{print 1}; {print x.1.1.1}}, x = a.1.2.1,
  m2 = ⟨c | c = ⌊c1⌋, c1 = {{2}; {print 3}}⟩, m1 = λx.⟨b | b = ⌊b1⌋, b1 = {{print 1}; {print x.1.1.1}}⟩⟩
$\longrightarrow$
1 is printed
⟨a1.4.1 | a = ⌊a1⌋,
  a1 = {⌊m1⌋; ⌊m2⌋; ⌊m3⌋; {b1.1.1}},
  m3 = ⌊b1⌋, b = ⌊b1⌋, b1 = {{1}; {print x.1.1.1}}, x = a.1.2.1,
  m2 = ⟨c | c = ⌊c1⌋, c1 = {{2}; {print 3}}⟩, m1 = λx.⟨b | b = ⌊b1⌋, b1 = {{print 1}; {print x.1.1.1}}⟩⟩
$\longrightarrow_{3}$
⟨a1.4.1 | a = ⌊a1⌋,
  a1 = {⌊m1⌋; ⌊m2⌋; ⌊m3⌋; {b1.1.1}},
  m3 = ⌊b1⌋, b = ⌊b1⌋, b1 = {⌊1⌋; {print x.1.1.1}}, x = a.1.2.1,
  m2 = ⟨c | c = ⌊c1⌋, c1 = {{2}; {print 3}}⟩, m1 = λx.⟨b | b = ⌊b1⌋, b1 = {{print 1}; {print x.1.1.1}}⟩⟩
$\longrightarrow^{*}_{7,2,8,2,2}$
⟨a1.4.1 | a = ⌊a1⌋,
  a1 = {⌊m1⌋; ⌊m2⌋; ⌊m3⌋; {b1.1.1}},
  m3 = ⌊b1⌋, b = ⌊b1⌋, b1 = {⌊1⌋; {print x.1.1.1}}, x = m2,
  m2 = ⟨c | c = ⌊c1⌋, c1 = {{2}; {print 3}}⟩, m1 = λx.⟨b | b = ⌊b1⌋, b1 = {{print 1}; {print x.1.1.1}}⟩⟩
$\longrightarrow^{*}_{6,11}$
⟨a1.4.1 | a = ⌊a1⌋,
  a1 = {⌊m1⌋; ⌊m2⌋; ⌊m3⌋; {b1.1.1}},
  m3 = ⌊b1⌋, b = ⌊b1⌋, b1 = {⌊1⌋; {print x.1.1.1}}, x = m2,
  m2 = ⌊c1⌋, c = ⌊c1⌋, c1 = {{2}; {print 3}}, m1 = λx.⟨b | b = ⌊b1⌋, b1 = {{print 1}; {print x.1.1.1}}⟩⟩
$\longrightarrow_{7}$

⟨a1.4.1 | a = ⌊a1⌋,
  a1 = {⌊m1⌋; ⌊m2⌋; ⌊m3⌋; {b1.1.1}},
  m3 = ⌊b1⌋, b = ⌊b1⌋, b1 = {⌊1⌋; {print x.1.1.1}}, x = ⌊c1⌋,
  m2 = ⌊c1⌋, c = ⌊c1⌋, c1 = {{2}; {print 3}}, m1 = λx.⟨b | b = ⌊b1⌋, b1 = {{print 1}; {print x.1.1.1}}⟩⟩
$\longrightarrow^{*}_{7,2}$
⟨a1.4.1 | a = ⌊a1⌋,
  a1 = {⌊m1⌋; ⌊m2⌋; ⌊m3⌋; {b1.1.1}},
  m3 = ⌊b1⌋, b = ⌊b1⌋, b1 = {⌊1⌋; {print c1.1.1}}, x = ⌊c1⌋,
  m2 = ⌊c1⌋, c = ⌊c1⌋, c1 = {{2}; {print 3}}, m1 = λx.⟨b | b = ⌊b1⌋, b1 = {{print 1}; {print x.1.1.1}}⟩⟩
$\longrightarrow_{3}$
⟨a1.4.1 | a = ⌊a1⌋,
  a1 = {⌊m1⌋; ⌊m2⌋; ⌊m3⌋; {b1.1.1}},
  m3 = ⌊b1⌋, b = ⌊b1⌋, b1 = {⌊1⌋; {print c1.1.1}}, x = ⌊c1⌋,
  m2 = ⌊c1⌋, c = ⌊c1⌋, c1 = {⌊2⌋; {print 3}}, m1 = λx.⟨b | b = ⌊b1⌋, b1 = {{print 1}; {print x.1.1.1}}⟩⟩
$\longrightarrow$

3 is printed

⟨a1.4.1 | a = ⌊a1⌋,
  a1 = {⌊m1⌋; ⌊m2⌋; ⌊m3⌋; {b1.1.1}},
  m3 = ⌊b1⌋, b = ⌊b1⌋, b1 = {⌊1⌋; {print c1.1.1}}, x = ⌊c1⌋,
  m2 = ⌊c1⌋, c = ⌊c1⌋, c1 = {⌊2⌋; {3}}, m1 = λx.⟨b | b = ⌊b1⌋, b1 = {{print 1}; {print x.1.1.1}}⟩⟩
$\longrightarrow_{3}$
⟨a1.4.1 | a = ⌊a1⌋,
  a1 = {⌊m1⌋; ⌊m2⌋; ⌊m3⌋; {b1.1.1}},
  m3 = ⌊b1⌋, b = ⌊b1⌋, b1 = {⌊1⌋; {print c1.1.1}}, x = ⌊c1⌋,
  m2 = ⌊c1⌋, c = ⌊c1⌋, c1 = {⌊2⌋; ⌊3⌋}, m1 = λx.⟨b | b = ⌊b1⌋, b1 = {{print 1}; {print x.1.1.1}}⟩⟩
$\longrightarrow_{3}$
⟨a1.4.1 | a = ⌊a1⌋,
  a1 = {⌊m1⌋; ⌊m2⌋; ⌊m3⌋; {b1.1.1}},
  m3 = ⌊b1⌋, b = ⌊b1⌋, b1 = {⌊1⌋; {print c1.1.1}}, x = ⌊c1⌋,
  m2 = ⌊c1⌋, c = ⌊c1⌋, c1 = ⌊⌊2⌋; ⌊3⌋⌋, m1 = λx.⟨b | b = ⌊b1⌋, b1 = {{print 1}; {print x.1.1.1}}⟩⟩
$\longrightarrow^{*}_{7,2,2}$
⟨a1.4.1 | a = ⌊a1⌋,
  a1 = {⌊m1⌋; ⌊m2⌋; ⌊m3⌋; {b1.1.1}},
  m3 = ⌊b1⌋, b = ⌊b1⌋, b1 = {⌊1⌋; {print 2}}, x = ⌊c1⌋,
  m2 = ⌊c1⌋, c = ⌊c1⌋, c1 = ⌊⌊2⌋; ⌊3⌋⌋, m1 = λx.⟨b | b = ⌊b1⌋, b1 = {{print 1}; {print x.1.1.1}}⟩⟩
$\longrightarrow$

2 is printed

⟨a1.4.1 | a = ⌊a1⌋,
  a1 = {⌊m1⌋; ⌊m2⌋; ⌊m3⌋; {b1.1.1}},
  m3 = ⌊b1⌋, b = ⌊b1⌋, b1 = {⌊1⌋; {2}}, x = ⌊c1⌋,
  m2 = ⌊c1⌋, c = ⌊c1⌋, c1 = ⌊⌊2⌋; ⌊3⌋⌋, m1 = λx.⟨b | b = ⌊b1⌋, b1 = {{print 1}; {print x.1.1.1}}⟩⟩
$\longrightarrow_{3}$
⟨a1.4.1 | a = ⌊a1⌋,
  a1 = {⌊m1⌋; ⌊m2⌋; ⌊m3⌋; {b1.1.1}},
  m3 = ⌊b1⌋, b = ⌊b1⌋, b1 = {⌊1⌋; ⌊2⌋}, x = ⌊c1⌋,
  m2 = ⌊c1⌋, c = ⌊c1⌋, c1 = ⌊⌊2⌋; ⌊3⌋⌋, m1 = λx.⟨b | b = ⌊b1⌋, b1 = {{print 1}; {print x.1.1.1}}⟩⟩
$\longrightarrow_{3}$
⟨a1.4.1 | a = ⌊a1⌋,
  a1 = {⌊m1⌋; ⌊m2⌋; ⌊m3⌋; {b1.1.1}},
  m3 = ⌊b1⌋, b = ⌊b1⌋, b1 = ⌊⌊1⌋; ⌊2⌋⌋, x = ⌊c1⌋,
  m2 = ⌊c1⌋, c = ⌊c1⌋, c1 = ⌊⌊2⌋; ⌊3⌋⌋, m1 = λx.⟨b | b = ⌊b1⌋, b1 = {{print 1}; {print x.1.1.1}}⟩⟩
$\longrightarrow^{*}_{7,2,2}$
⟨a1.4.1 | a = ⌊a1⌋,
  a1 = {⌊m1⌋; ⌊m2⌋; ⌊m3⌋; {1}},
  m3 = ⌊b1⌋, b = ⌊b1⌋, b1 = ⌊⌊1⌋; ⌊2⌋⌋, x = ⌊c1⌋,
  m2 = ⌊c1⌋, c = ⌊c1⌋, c1 = ⌊⌊2⌋; ⌊3⌋⌋, m1 = λx.⟨b | b = ⌊b1⌋, b1 = {{print 1}; {print x.1.1.1}}⟩⟩
$\longrightarrow_{3}$

$\langle$a1.4.1 | a = $\lfloor$a1$\rfloor$,
  a1 = {$\lfloor$m1$\rfloor$; $\lfloor$m2$\rfloor$; $\lfloor$m3$\rfloor$; $\lfloor$1$\rfloor$},
  m3 = $\lfloor$b1$\rfloor$, b = $\lfloor$b1$\rfloor$, b1 = $\lfloor\lfloor$1$\rfloor$; $\lfloor$2$\rfloor\rfloor$, x = $\lfloor$c1$\rfloor$,
  m2 = $\lfloor$c1$\rfloor$, c = $\lfloor$c1$\rfloor$, c1 = $\lfloor\lfloor$2$\rfloor$; $\lfloor$3$\rfloor\rfloor$, m1 = λx.$\langle$b | b = $\lfloor$b1$\rfloor$, b1 = {{print 1}; {print x.1.1.1}}$\rangle\rangle$
$\rightarrow_{3}$
$\langle$a1.4.1 | a = $\lfloor$a1$\rfloor$,
  a1 = $\lfloor\lfloor$m1$\rfloor$; $\lfloor$m2$\rfloor$; $\lfloor$m3$\rfloor$; $\lfloor$1$\rfloor\rfloor$,
  m3 = $\lfloor$b1$\rfloor$, b = $\lfloor$b1$\rfloor$, b1 = $\lfloor\lfloor$1$\rfloor$; $\lfloor$2$\rfloor\rfloor$, x = $\lfloor$c1$\rfloor$,
  m2 = $\lfloor$c1$\rfloor$, c = $\lfloor$c1$\rfloor$, c1 = $\lfloor\lfloor$2$\rfloor$; $\lfloor$3$\rfloor\rfloor$, m1 = λx.$\langle$b | b = $\lfloor$b1$\rfloor$, b1 = {{print 1}; {print x.1.1.1}}$\rangle\rangle$
$\rightarrow^{*}_{6,3,3}$
$\langle$1 | a = $\lfloor$a1$\rfloor$,
  a1 = $\lfloor\lfloor$m1$\rfloor$; $\lfloor$m2$\rfloor$; $\lfloor$m3$\rfloor$; $\lfloor$1$\rfloor\rfloor$,
  m3 = $\lfloor$b1$\rfloor$, b = $\lfloor$b1$\rfloor$, b1 = $\lfloor\lfloor$1$\rfloor$; $\lfloor$2$\rfloor\rfloor$, x = $\lfloor$c1$\rfloor$,
  m2 = $\lfloor$c1$\rfloor$, c = $\lfloor$c1$\rfloor$, c1 = $\lfloor\lfloor$2$\rfloor$; $\lfloor$3$\rfloor\rfloor$, m1 = λx.$\langle$b | b = $\lfloor$b1$\rfloor$, b1 = {{print 1}; {print x.1.1.1}}$\rangle\rangle$

7.4 The example of successful intra-module recursion (page 6 middle)

The original program written in the syntax of *Osan* is:

```
{(X)
 M = {(X1)  m1 = fun x → X1.m2 (x+1);  m2 = fun x → X1.m1 (x+2) };
 main = X.M.m1 0 }
```

Execution of the above program diverges. Below are the reductions.

$\langle$a.1.2.1 | a = $\lfloor$a1$\rfloor$, a1 = {$\langle\lfloor$m1$\rfloor$ |
  m1 = $\langle$b | b = $\lfloor$b1$\rfloor$, b1 = {{fun x → b.1.2.1 (x+1)}; {fun x → b.1.1.1 (x+2)}}$\rangle$}; {a.1.1.1.1.1.1 0}}$\rangle$
$\rightarrow^{*}_{6,2}$
$\langle$a1.2.1 | a = $\lfloor$a1$\rfloor$, a1 = {$\langle\lfloor$m1$\rfloor$ |
  m1 = $\langle$b | b = $\lfloor$b1$\rfloor$, b1 = {{fun x → b.1.2.1 (x+1)}; {fun x → b.1.1.1 (x+2)}}$\rangle$}; {a.1.1.1.1.1.1 0}}$\rangle$
$\rightarrow_{9}$
$\langle$a1.2.1 | a = $\lfloor$a1$\rfloor$, a1 = $\langle${$\lfloor$m1$\rfloor$; {a.1.1.1.1.1.1 0}} |
  m1 = $\langle$b | b = $\lfloor$b1$\rfloor$, b1 = {{fun x → b.1.2.1 (x+1)}; {fun x → b.1.1.1 (x+2)}}$\rangle\rangle$
$\rightarrow^{*}_{7,2}$
$\langle$a1.2.1 | a = $\lfloor$a1$\rfloor$, a1 = $\langle${$\lfloor$m1$\rfloor$; {a1.1.1.1.1.1 0}} |
  m1 = $\langle$b | b = $\lfloor$b1$\rfloor$, b1 = {{fun x → b.1.2.1 (x+1)}; {fun x → b.1.1.1 (x+2)}}$\rangle\rangle$
$\rightarrow_{12}$
$\langle$a1.2.1 | a = $\lfloor$a1$\rfloor$, a1 = {$\lfloor$m1$\rfloor$; {a1.1.1.1.1.1 0}},
  m1 = $\langle$b | b = $\lfloor$b1$\rfloor$, b1 = {{fun x → b.1.2.1 (x+1)}; {fun x → b.1.1.1 (x+2)}}$\rangle$
$\rightarrow^{*}_{8,2,2}$
$\langle$a1.2.1 | a = $\lfloor$a1$\rfloor$, a1 = {$\lfloor$m1$\rfloor$; {m1.1.1.1 0}},
  m1 = $\langle$b | b = $\lfloor$b1$\rfloor$, b1 = {{fun x → b.1.2.1 (x+1)}; {fun x → b.1.1.1 (x+2)}}$\rangle$
$\rightarrow_{6}$
$\langle$a1.2.1 | a = $\lfloor$a1$\rfloor$, a1 = {$\lfloor$m1$\rfloor$; {m1.1.1.1 0}},
  m1 = $\langle\lfloor$b1$\rfloor$ | b = $\lfloor$b1$\rfloor$, b1 = {{fun x → b.1.2.1 (x+1)}; {fun x → b.1.1.1 (x+2)}}$\rangle$
$\rightarrow_{11}$
$\langle$a1.2.1 | a = $\lfloor$a1$\rfloor$, a1 = {$\lfloor$m1$\rfloor$; {m1.1.1.1 0}},
  m1 = $\lfloor$b1$\rfloor$, b = $\lfloor$b1$\rfloor$, b1 = {{fun x → b.1.2.1 (x+1)}; {fun x → b.1.1.1 (x+2)}}$\rangle$
$\rightarrow^{*}_{7,2}$
$\langle$a1.2.1 | a = $\lfloor$a1$\rfloor$, a1 = {$\lfloor$m1$\rfloor$; {b1.1.1 0}},
  m1 = $\lfloor$b1$\rfloor$, b = $\lfloor$b1$\rfloor$, b1 = {{fun x → b.1.2.1 (x+1)}; {fun x → b.1.1.1 (x+2)}}$\rangle$
$\rightarrow^{*}_{3,3,3}$
$\langle$a1.2.1 | a = $\lfloor$a1$\rfloor$, a1 = {$\lfloor$m1$\rfloor$; {b1.1.1 0}},
  m1 = $\lfloor$b1$\rfloor$, b = $\lfloor$b1$\rfloor$, b1 = $\lfloor\lfloor$fun x → b.1.2.1 (x+1)$\rfloor$; $\lfloor$fun x → b.1.1.1 (x+2)$\rfloor\rfloor\rangle$
$\rightarrow^{*}_{7,2,2}$

⟨a1.2.1 | a = ⌊a1⌋, a1 = {⌊m1⌋; {(fun x → b.1.2.1 (x+1)) 0}},
  m1 = ⌊b1⌋, b = ⌊b1⌋, b1 = ⌊⌊fun x → b.1.2.1 (x+1)⌋; ⌊fun x → b.1.1.1 (x+2)⌋⌋⟩
→*
⟨a1.2.1 | a = ⌊a1⌋, a1 = {⌊m1⌋; {b.1.2.1 1}},
  m1 = ⌊b1⌋, b = ⌊b1⌋, b1 = ⌊⌊fun x → b.1.2.1 (x+1)⌋; ⌊fun x → b.1.1.1 (x+2)⌋⌋⟩
→*7,2,7,2,2
⟨a1.2.1 | a = ⌊a1⌋, a1 = {⌊m1⌋; {(fun x → b.1.1.1 (x+2)) 1}},
  m1 = ⌊b1⌋, b = ⌊b1⌋, b1 = ⌊⌊fun x → b.1.2.1 (x+1)⌋; ⌊fun x → b.1.1.1 (x+2)⌋⌋⟩
→*
⟨a1.2.1 | a = ⌊a1⌋, a1 = {⌊m1⌋; {b.1.1.1 3}},
  m1 = ⌊b1⌋, b = ⌊b1⌋, b1 = ⌊⌊fun x → b.1.2.1 (x+1)⌋; ⌊fun x → b.1.1.1 (x+2)⌋⌋⟩
. . .

The example for successful inter-module recursion (page 6 last)

The original program written in the syntax of *Osan* is:

```
{(X)
 M1 = { m = fun x → X.M2.m (x+1); };  M2 = { m = fun x → X.M1.m (x+2); };
 main = X.M1.m 0;}
```

Execution of the above program diverges. Below are the reductions.

⟨a.1.3.1 | a = ⌊a1⌋,
  a1 = {
    ⟨⌊m1⌋ | m1 = ⟨b | b = ⌊b1⌋, b1 = {{fun x → a.1.2.1.1.1.1 (x+1)}}⟩⟩;
    ⟨⌊m2⌋ | m2 = ⟨c | c = ⌊c1⌋, c1 = {{fun x → a.1.1.1.1.1.1 (x+2)}}⟩⟩;
    {a.1.1.1.1.1.1 0}}⟩
→*6,2
⟨a1.3.1 | a = ⌊a1⌋,
  a1 = {
    ⟨⌊m1⌋ | m1 = ⟨b | b = ⌊b1⌋, b1 = {{fun x → a.1.2.1.1.1.1 (x+1)}}⟩⟩;
    ⟨⌊m2⌋ | m2 = ⟨c | c = ⌊c1⌋, c1 = {{fun x → a.1.1.1.1.1.1 (x+2)}}⟩⟩;
    {a.1.1.1.1.1.1 0}}⟩
→9
⟨a1.3.1 | a = ⌊a1⌋,
  a1 = ⟨{⌊m1⌋;
    ⟨⌊m2⌋ | m2 = ⟨c | c = ⌊c1⌋, c1 = {{fun x → a.1.1.1.1.1.1 (x+2)}}⟩⟩; {a.1.1.1.1.1.1 0}}|
      m1 = ⟨b | b = ⌊b1⌋, b1 = {{fun x → a.1.2.1.1.1.1 (x+1)}}⟩⟩⟩
→9
⟨a1.3.1 | a = ⌊a1⌋,
  a1 = ⟨⟨{⌊m1⌋; ⌊m2⌋; {a.1.1.1.1.1.1 0}} | m2 = ⟨c | c = ⌊c1⌋, c1 = {{fun x → a.1.1.1.1.1.1 (x+2)}}⟩⟩|
    m1 = ⟨b | b = ⌊b1⌋, b1 = {{fun x → a.1.2.1.1.1.1 (x+1)}}⟩⟩⟩
→*7,2
⟨a1.3.1 | a = ⌊a1⌋,
  a1 = ⟨⟨{⌊m1⌋; ⌊m2⌋; {a1.1.1.1.1.1 0}} | m2 = ⟨c | c = ⌊c1⌋, c1 = {{fun x → a.1.1.1.1.1.1 (x+2)}}⟩⟩|
    m1 = ⟨b | b = ⌊b1⌋, b1 = {{fun x → a.1.2.1.1.1.1 (x+1)}}⟩⟩⟩
→*12,12
⟨a1.3.1 | a = ⌊a1⌋,
  a1 = {⌊m1⌋; ⌊m2⌋; {a1.1.1.1.1.1 0}}, m2 = ⟨c | c = ⌊c1⌋, c1 = {{fun x → a.1.1.1.1.1.1 (x+2)}}⟩,
  m1 = ⟨b | b = ⌊b1⌋, b1 = {{fun x → a.1.2.1.1.1.1 (x+1)}}⟩⟩
→*8,2,2
⟨a1.3.1 | a = ⌊a1⌋,
  a1 = {⌊m1⌋; ⌊m2⌋; {m1.1.1.1 0}}, m2 = ⟨c | c = ⌊c1⌋, c1 = {{fun x → a.1.1.1.1.1.1 (x+2)}}⟩,
  m1 = ⟨b | b = ⌊b1⌋, b1 = {{fun x → a.1.2.1.1.1.1 (x+1)}}⟩⟩

$\longrightarrow^*_{6,11}$
⟨a1.3.1 | a = ⌊a1⌋,
  a1 = {⌊m1⌋; ⌊m2⌋; {m1.1.1.1 0}}, m2 = ⟨c | c = ⌊c1⌋, c1 = {fun x → a.1.1.1.1.1.1 (x+2)}⟩,
  m1 = ⌊b1⌋, b = ⌊b1⌋, b1 = {{fun x → a.1.2.1.1.1.1 (x+1)}}⟩
$\longrightarrow^*_{7,2}$
⟨a1.3.1 | a = ⌊a1⌋,
  a1 = {⌊m1⌋; ⌊m2⌋; {b1.1.1 0}}, m2 = ⟨c | c = ⌊c1⌋, c1 = {{fun x → a.1.1.1.1.1.1 (x+2)}}⟩,
  m1 = ⌊b1⌋, b = ⌊b1⌋, b1 = {{fun x → a.1.2.1.1.1.1 (x+1)}}⟩
$\longrightarrow^*_{3,3}$
⟨a1.3.1 | a = ⌊a1⌋,
  a1 = {⌊m1⌋; ⌊m2⌋; {b1.1.1 0}}, m2 = ⟨c | c = ⌊c1⌋, c1 = {fun x → a.1.1.1.1.1.1 (x+2)}⟩,
  m1 = ⌊b1⌋, b = ⌊b1⌋, b1 = ⌊⌊fun x → a.1.2.1.1.1.1 (x+1)⌋⌋⟩
$\longrightarrow^*_{7,2,2}$
⟨a1.3.1 | a = ⌊a1⌋,
  a1 = {⌊m1⌋; ⌊m2⌋; {(fun x → a.1.2.1.1.1.1 (x+1)) 0}}, m2 = ⟨c | c = ⌊c1⌋, c1 = {{fun x → a.1.1.1.1.1.1 (x+2)}}⟩,
  m1 = ⌊b1⌋, b = ⌊b1⌋, b1 = ⌊⌊fun x → a.1.2.1.1.1.1 (x+1)⌋⌋⟩
$\longrightarrow^*$
⟨a1.3.1 | a = ⌊a1⌋,
  a1 = {⌊m1⌋; ⌊m2⌋; {a.1.2.1.1.1.1 1}}, m2 = ⟨c | c = ⌊c1⌋, c1 = {{fun x → a.1.1.1.1.1.1 (x+2)}}⟩,
  m1 = ⌊b1⌋, b = ⌊b1⌋, b1 = ⌊⌊fun x → a.1.2.1.1.1.1 (x+1)⌋⌋⟩
$\longrightarrow^*_{7,2,8,2,2}$
⟨a1.3.1 | a = ⌊a1⌋,
  a1 = {⌊m1⌋; ⌊m2⌋; {m2.1.1.1 1}}, m2 = ⟨c | c = ⌊c1⌋, c1 = {{fun x → a.1.1.1.1.1.1 (x+2)}}⟩,
  m1 = ⌊b1⌋, b = ⌊b1⌋, b1 = ⌊⌊fun x → a.1.2.1.1.1.1 (x+1)⌋⌋⟩
$\longrightarrow^*_{6,11}$
⟨a1.3.1 | a = ⌊a1⌋,
  a1 = {⌊m1⌋; ⌊m2⌋; {m2.1.1.1 1}}, m2 = ⌊c1⌋, c = ⌊c1⌋, c1 = {{fun x → a.1.1.1.1.1.1 (x+2)}},
  m1 = ⌊b1⌋, b = ⌊b1⌋, b1 = ⌊⌊fun x → a.1.2.1.1.1.1 (x+1)⌋⌋⟩
$\longrightarrow^*_{7,2}$
⟨a1.3.1 | a = ⌊a1⌋,
  a1 = {⌊m1⌋; ⌊m2⌋; {c1.1.1 1}}, m2 = ⌊c1⌋, c = ⌊c1⌋, c1 = {{fun x → a.1.1.1.1.1.1 (x+2)}},
  m1 = ⌊b1⌋, b = ⌊b1⌋, b1 = ⌊⌊fun x → a.1.2.1.1.1.1 (x+1)⌋⌋⟩
$\longrightarrow^*_{3,3}$
⟨a1.3.1 | a = ⌊a1⌋,
  a1 = {⌊m1⌋; ⌊m2⌋; {c1.1.1 1}}, m2 = ⌊c1⌋, c = ⌊c1⌋, c1 = ⌊⌊fun x → a.1.1.1.1.1.1 (x+2)⌋⌋,
  m1 = ⌊b1⌋, b = ⌊b1⌋, b1 = ⌊⌊fun x → a.1.2.1.1.1.1 (x+1)⌋⌋⟩
$\longrightarrow^*_{7,2,2}$
⟨a1.3.1 | a = ⌊a1⌋,
  a1 = {⌊m1⌋; ⌊m2⌋; {(fun x → a.1.1.1.1.1.1 (x+2)) 1}}, m2 = ⌊c1⌋, c = ⌊c1⌋, c1 = ⌊⌊fun x → a.1.1.1.1.1.1 (x+2)⌋⌋,
  m1 = ⌊b1⌋, b = ⌊b1⌋, b1 = ⌊⌊fun x → a.1.2.1.1.1.1 (x+1)⌋⌋⟩
$\longrightarrow^*$
⟨a1.3.1 | a = ⌊a1⌋,
  a1 = {⌊m1⌋; ⌊m2⌋; {a.1.1.1.1.1.1 3}}, m2 = ⌊c1⌋, c = ⌊c1⌋, c1 = ⌊⌊fun x → a.1.1.1.1.1.1 (x+2)⌋⌋,
  m1 = ⌊b1⌋, b = ⌊b1⌋, b1 = ⌊⌊fun x → a.1.2.1.1.1.1 (x+1)⌋⌋⟩
. . .

The first example for the first principle (page 7 first) The original program written in the syntax of *Osan* is:

{(X) M = {(X1) m1 = X1.m2 + 3; m2 = 2; }; main = X.M.m1;}

Execution of the above program gets stuck, since M.m1 accesses the subsequent field M.m2. Below are the reductions.

$\langle$a.1.2.1 | a = $\lfloor$a1$\rfloor$, a1 = {$\langle\lfloor$m$\rfloor$ | m = $\langle$b |b = $\lfloor$b1$\rfloor$, b1 = {{b.1.2.1+3}; {2}}$\rangle$; {a.1.1.1.1.1.1}}$\rangle$
$\rightarrow^{*}_{6,2}$
$\langle$a1.2.1 | a = $\lfloor$a1$\rfloor$, a1 = {$\langle\lfloor$m$\rfloor$ | m = $\langle$b | b = $\lfloor$b1$\rfloor$, b1 = {{b.1.2.1+3}; {2}}$\rangle\rangle$; {a.1.1.1.1.1.1}}$\rangle$
$\rightarrow_{9}$
$\langle$a1.2.1 | a = $\lfloor$a1$\rfloor$, a1 = $\langle${$\lfloor$m$\rfloor$;{a.1.1.1.1.1.1}} | m = $\langle$b | b = $\lfloor$b1$\rfloor$, b1 = {{b.1.2.1+3}; {2}}$\rangle\rangle\rangle$
$\rightarrow^{*}_{7,2}$
$\langle$a1.2.1 | a = $\lfloor$a1$\rfloor$, a1 = $\langle${$\lfloor$m$\rfloor$;{a1.1.1.1.1.1}} | m = $\langle$b | b = $\lfloor$b1$\rfloor$, b1 = {{b.1.2.1+3}; {2}}$\rangle\rangle\rangle$
$\rightarrow_{12}$
$\langle$a1.2.1 | a = $\lfloor$a1$\rfloor$, a1 = {$\lfloor$m$\rfloor$;{a1.1.1.1.1.1}}, m = $\langle$b | b = $\lfloor$b1$\rfloor$, b1 = {{b.1.2.1+3}; {2}}$\rangle\rangle$
$\rightarrow_{8,2,2}$
$\langle$a1.2.1 | a = $\lfloor$a1$\rfloor$, a1 = {$\lfloor$m$\rfloor$;{m.1.1.1}}, m = $\langle$b | b = $\lfloor$b1$\rfloor$, b1 = {{b.1.2.1+3}; {2}}$\rangle\rangle$
$\rightarrow^{*}_{6,11}$
$\langle$a1.2.1 | a = $\lfloor$a1$\rfloor$, a1 = {$\lfloor$m$\rfloor$;{m.1.1.1}}, m = $\lfloor$b1$\rfloor$, b = $\lfloor$b1$\rfloor$, b1 = {{b.1.2.1+3}; {2}}$\rangle$
$\rightarrow^{*}_{7,2}$
$\langle$a1.2.1 | a = $\lfloor$a1$\rfloor$, a1 = {$\lfloor$m$\rfloor$;{b1.1.1}}, m = $\lfloor$b1$\rfloor$, b = $\lfloor$b1$\rfloor$, b1 = {{b.1.2.1+3}; {2}}$\rangle$
$\rightarrow^{*}_{7,2}$
$\langle$a1.2.1 | a = $\lfloor$a1$\rfloor$, a1 = {$\lfloor$m$\rfloor$;{b1.1.1}}, m = $\lfloor$b1$\rfloor$, b = $\lfloor$b1$\rfloor$, b1 = {{b1.2.1+3}; {2}}$\rangle$
$\rightarrow_{8}$
$\langle$a1.2.1 | a = $\lfloor$a1$\rfloor$, a1 = {$\lfloor$m$\rfloor$;{b1.1.1}}, m = $\lfloor$b1$\rfloor$, b = $\lfloor$b1$\rfloor$, b1 = {{$\lfloor\rfloor$.2.1+3}; {2}}$\rangle$

$\lfloor\rfloor$.2 cannot be reduced, thus the reduction gets stuck.

7.5 The second example for the first principle (the second example in page 7)

The original program written in the syntax of *Osan* is:

```
{(X)
  M = {(X1)  m = (fun x → x+1) X1.M1.m;  M1 = { m = 0 } };
  main = X.M.m }
```

Execution of the above program gets stuck, since M.m accesses a field of the subsequent sub-module M1. Below are the reductions.

$\langle$a.1.2.1 | a = $\lfloor$a1$\rfloor$,
  a1 = {
    $\langle\lfloor$m$\rfloor$ | m = $\langle$b | b = $\lfloor$b1$\rfloor$,
      b1 = {{(fun x → x+1) b.1.2.1.1.1.1}; $\langle\lfloor$m1$\rfloor$ | m1 = $\langle$c | c = $\lfloor$c1$\rfloor$, c1 = {{0}}$\rangle\rangle\rangle$;
    {a.1.1.1.1.1.1}}$\rangle$
$\rightarrow^{*}_{6,2}$
$\langle$a1.2.1 | a = $\lfloor$a1$\rfloor$,
  a1 = {
    $\langle\lfloor$m$\rfloor$ | m = $\langle$b | b = $\lfloor$b1$\rfloor$,
      b1 = {{(fun x → x+1) b.1.2.1.1.1.1}; $\langle\lfloor$m1$\rfloor$ | m1 = $\langle$c | c = $\lfloor$c1$\rfloor$, c1 = {{0}}$\rangle\rangle\rangle$;
    {a.1.1.1.1.1.1}}$\rangle$
$\rightarrow_{9}$
$\langle$a1.2.1 | a = $\lfloor$a1$\rfloor$,
  a1 = $\langle${$\lfloor$m$\rfloor$; {a.1.1.1.1.1.1}} |
    m = $\langle$b | b = $\lfloor$b1$\rfloor$, b1 = {{(fun x → x+1) b.1.2.1.1.1.1}; $\langle\lfloor$m1$\rfloor$ | m1 = $\langle$c | c = $\lfloor$c1$\rfloor$, c1 = {{0}}$\rangle\rangle\rangle\rangle$
$\rightarrow^{*}_{7,2}$

⟨a1.2.1 | a = ⌊a1⌋,
  a1 = ⟨{⌊m⌋; {a1.1.1.1.1.1}} |
    m = ⟨b | b = ⌊b1⌋, b1 = {{(fun x → x+1) b.1.2.1.1.1.1}; ⟨⌊m1⌋ | m1 = ⟨c | c = ⌊c1⌋, c1 = {{0}}⟩⟩⟩⟩⟩
→12
⟨a1.2.1 | a = ⌊a1⌋, a1 = {⌊m⌋; {a1.1.1.1.1.1}},
  m = ⟨b | b = ⌊b1⌋, b1 = {{(fun x → x+1) b.1.2.1.1.1.1}; ⟨⌊m1⌋ | m1 = ⟨c | c = ⌊c1⌋, c1 = {{0}}⟩⟩⟩⟩
→*8,2
⟨a1.2.1 | a = ⌊a1⌋, a1 = {⌊m⌋; {m.1.1.1}},
  m = ⟨b | b = ⌊b1⌋, b1 = {{(fun x → x+1) b.1.2.1.1.1.1}; ⟨⌊m1⌋ | m1 = ⟨c | c = ⌊c1⌋, c1 = {{0}}⟩⟩⟩⟩
→6
⟨a1.2.1 | a = ⌊a1⌋, a1 = {⌊m⌋; {m.1.1.1}},
  m = ⟨⌊b1⌋ | b = ⌊b1⌋, b1 = {{(fun x → x+1) b.1.2.1.1.1.1}; ⟨⌊m1⌋ | m1 = ⟨c | c = ⌊c1⌋, c1 = {{0}}⟩⟩⟩⟩
→11
⟨a1.2.1 | a = ⌊a1⌋, a1 = {⌊m⌋; {m.1.1.1}},
  m = ⌊b1⌋, b = ⌊b1⌋, b1 = {{(fun x → x+1) b.1.2.1.1.1.1}; ⟨⌊m1⌋ | m1 = ⟨c | c = ⌊c1⌋, c1 = {{0}}⟩⟩⟩⟩
→*7,2
⟨a1.2.1 | a = ⌊a1⌋, a1 = {⌊m⌋; {b1.1.1}},
  m = ⌊b1⌋, b = ⌊b1⌋, b1 = {{(fun x → x+1) b.1.2.1.1.1.1}; ⟨⌊m1⌋ | m1 = ⟨c | c = ⌊c1⌋, c1 = {{0}}⟩⟩⟩⟩
→*7,2
⟨a1.2.1 | a = ⌊a1⌋, a1 = {⌊m⌋; {b1.1.1}},
  m = ⌊b1⌋, b = ⌊b1⌋, b1 = {{(fun x → x+1) b1.2.1.1.1.1}; ⟨⌊m1⌋ | m1 = ⟨c | c = ⌊c1⌋, c1 = {{0}}⟩⟩⟩⟩
→8
⟨a1.2.1 | a = ⌊a1⌋, a1 = {⌊m⌋; {b1.1.1}},
  m = ⌊b1⌋, b = ⌊b1⌋, b1 = {{(fun x → x+1) ⌊⌋.2.1.1.1.1}; ⟨⌊m1⌋ | m1 = ⟨c | c = ⌊c1⌋, c1 = {{0}}⟩⟩⟩⟩

⌊⌋.2 cannot be reduced, thus the reduction gets stuck.

7.6 The third example for the first principle (page 7 third)

The original program written in the syntax of *Osan* is:

```
{(X)
  M = {(X1)  M2 = { m = (fun x → x+1) X1.M1.m; };  M1 = { m = 0; }; };
  main = X.M.M2.m;}
```

Execution of the above program succeeds. Below are reductions.

⟨a.1.2.1 | a = ⌊a1⌋,
  a1 = {⟨⌊m⌋ | m = ⟨b | b = ⌊b1⌋,
    b1 = {⟨⌊m2⌋ | m2 = ⟨d | d = ⌊d1⌋, d1 = {{(fun x → x+1) b.1.2.1.1.1.1}}⟩⟩;
      ⟨⌊m1⌋ | m1 = ⟨c | c = ⌊c1⌋, c1 = {{0}}⟩⟩⟩⟩⟩; {a.1.1.1.1.1.1.1.1.1}}⟩
→*6,2
⟨a1.2.1 | a = ⌊a1⌋,
  a1 = {⟨⌊m⌋ | m = ⟨b | b = ⌊b1⌋,
    b1 = {⟨⌊m2⌋ | m2 = ⟨d | d = ⌊d1⌋, d1 = {{(fun x → x+1) b.1.2.1.1.1.1}}⟩⟩;
      ⟨⌊m1⌋ | m1 = ⟨c | c = ⌊c1⌋, c1 = {{0}}⟩⟩⟩⟩⟩; {a.1.1.1.1.1.1.1.1.1}}⟩
→*9
⟨a1.2.1 | a = ⌊a1⌋,
  a1 = ⟨{⌊m⌋;{a.1.1.1.1.1.1.1.1.1}} | m = ⟨b | b = ⌊b1⌋,
    b1 = {⟨⌊m2⌋ | m2 = ⟨d | d = ⌊d1⌋, d1 = {{(fun x → x+1) b.1.2.1.1.1.1}}⟩⟩;
      ⟨⌊m1⌋ | m1 = ⟨c | c = ⌊c1⌋, c1 = {{0}}⟩⟩⟩⟩⟩⟩
→*7,2
⟨a1.2.1 | a = ⌊a1⌋,
  a1 = ⟨{⌊m⌋;{a1.1.1.1.1.1.1.1.1.1}} | m = ⟨b | b = ⌊b1⌋,
    b1 = {⟨⌊m2⌋ | m2 = ⟨d | d = ⌊d1⌋, d1 = {{(fun x → x+1) b.1.2.1.1.1.1}}⟩⟩;
      ⟨⌊m1⌋ | m1 = ⟨c | c = ⌊c1⌋, c1 = {{0}}⟩⟩⟩⟩⟩⟩
→12

⟨a1.2.1 | a = ⌊a1⌋,
  a1 = {⌊m⌋;{a1.1.1.1.1.1.1.1.1}},
  m = ⟨b | b = ⌊b1⌋, b1 = {⟨⌊m2⌋ | m2 = ⟨d | d = ⌊d1⌋, d1 = {{(fun x → x+1) b.1.2.1.1.1.1}}⟩⟩;
    ⟨⌊m1⌋ | m1 = ⟨c | c = ⌊c1⌋, c1 = {{0}}⟩⟩}⟩⟩
→*8,2,2
⟨a1.2.1 | a = ⌊a1⌋,
  a1 = {⌊m⌋;{m.1.1.1.1.1.1.1}},
  m = ⟨b | b = ⌊b1⌋, b1 = {⟨⌊m2⌋ | m2 = ⟨d | d = ⌊d1⌋, d1 = {{(fun x → x+1) b.1.2.1.1.1.1}}⟩⟩;
    ⟨⌊m1⌋ | m1 = ⟨c | c = ⌊c1⌋, c1 = {{0}}⟩⟩}⟩⟩
→*6,11
⟨a1.2.1 | a = ⌊a1⌋,
  a1 = {⌊m⌋;{m.1.1.1.1.1.1.1}},
  m = ⌊b1⌋, b = ⌊b1⌋, b1 = {⟨⌊m2⌋ | m2 = ⟨d | d = ⌊d1⌋, d1 = {{(fun x → x+1) b.1.2.1.1.1.1}}⟩⟩;
    ⟨⌊m1⌋ | m1 = ⟨c | c = ⌊c1⌋, c1 = {{0}}⟩⟩}⟩
→*7,2
⟨a1.2.1 | a = ⌊a1⌋,
  a1 = {⌊m⌋;{b1.1.1.1.1.1.1}},
  m = ⌊b1⌋, b = ⌊b1⌋, b1 = {⟨⌊m2⌋ | m2 = ⟨d | d = ⌊d1⌋, d1 = {{(fun x → x+1) b.1.2.1.1.1.1}}⟩⟩;
    ⟨⌊m1⌋ | m1 = ⟨c | c = ⌊c1⌋, c1 = {{0}}⟩⟩}⟩
→9
⟨a1.2.1 | a = ⌊a1⌋,
  a1 = {⌊m⌋;{b1.1.1.1.1.1.1}}, m = ⌊b1⌋, b = ⌊b1⌋,
  b1 = ⟨{⌊m2⌋;⟨⌊m1⌋ | m1 = ⟨c | c = ⌊c1⌋, c1 = {{0}}⟩⟩} | m2 = ⟨d | d = ⌊d1⌋, d1 = {{(fun x → x+1) b.1.2.1.1.1.1}}⟩⟩⟩
→9
⟨a1.2.1 | a = ⌊a1⌋,
  a1 = {⌊m⌋;{b1.1.1.1.1.1.1}}, m = ⌊b1⌋, b = ⌊b1⌋,
  b1 = ⟨⟨{⌊m2⌋;⌊m1⌋} | m1 = ⟨c | c = ⌊c1⌋, c1 = {{0}}⟩⟩ | m2 = ⟨d | d = ⌊d1⌋, d1 = {{(fun x → x+1) b.1.2.1.1.1.1}}⟩⟩⟩
→*11,11
⟨a1.2.1 | a = ⌊a1⌋,
  a1 = {⌊m⌋;{b1.1.1.1.1.1.1}}, m = ⌊b1⌋, b = ⌊b1⌋,
  b1 = {⌊m2⌋;⌊m1⌋}, m1 = ⟨c | c = ⌊c1⌋, c1 = {{0}}⟩, m2 = ⟨d | d = ⌊d1⌋, d1 = {{(fun x → x+1) b.1.2.1.1.1.1}}⟩⟩
→3
⟨a1.2.1 | a = ⌊a1⌋,
  a1 = {⌊m⌋;{b1.1.1.1.1.1.1}}, m = ⌊b1⌋, b = ⌊b1⌋,
  b1 = ⌊⌊m2⌋;⌊m1⌋⌋, m1 = ⟨c | c = ⌊c1⌋, c1 = {{0}}⟩, m2 = ⟨d | d = ⌊d1⌋, d1 = {{(fun x → x+1) b.1.2.1.1.1.1}}⟩⟩
→*7,2,2
⟨a1.2.1 | a = ⌊a1⌋,
  a1 = {⌊m⌋;{m2.1.1.1}}, m = ⌊b1⌋, b = ⌊b1⌋,
  b1 = ⌊⌊m2⌋;⌊m1⌋⌋, m1 = ⟨c | c = ⌊c1⌋, c1 = {{0}}⟩, m2 = ⟨d | d = ⌊d1⌋, d1 = {{(fun x → x+1) b.1.2.1.1.1.1}}⟩⟩
→*6,11
⟨a1.2.1 | a = ⌊a1⌋,
  a1 = {⌊m⌋;{m2.1.1.1}}, m = ⌊b1⌋, b = ⌊b1⌋,
  b1 = ⌊⌊m2⌋;⌊m1⌋⌋, m1 = ⟨c | c = ⌊c1⌋, c1 = {{0}}⟩, m2 = ⌊d1⌋, d = ⌊d1⌋, d1 = {{(fun x → x+1) b.1.2.1.1.1.1}}⟩
→*7,2
⟨a1.2.1 | a = ⌊a1⌋,
  a1 = {⌊m⌋;{d1.1.1}}, m = ⌊b1⌋, b = ⌊b1⌋,
  b1 = ⌊⌊m2⌋;⌊m1⌋⌋, m1 = ⟨c | c = ⌊c1⌋, c1 = {{0}}⟩, m2 = ⌊d1⌋, d = ⌊d1⌋, d1 = {{(fun x → x+1) b.1.2.1.1.1.1}}⟩
→*7,2,7,2,2
⟨a1.2.1 | a = ⌊a1⌋,
  a1 = {⌊m⌋;{d1.1.1}}, m = ⌊b1⌋, b = ⌊b1⌋,
  b1 = ⌊⌊m2⌋;⌊m1⌋⌋, m1 = ⟨c | c = ⌊c1⌋, c1 = {{0}}⟩, m2 = ⌊d1⌋, d = ⌊d1⌋, d1 = {{(fun x → x+1) m1.1.1.1}}⟩
→*6,11
⟨a1.2.1 | a = ⌊a1⌋,
  a1 = {⌊m⌋;{d1.1.1}}, m = ⌊b1⌋, b = ⌊b1⌋,
  b1 = ⌊⌊m2⌋;⌊m1⌋⌋, m1 = ⌊c1⌋, c = ⌊c1⌋, c1 = {{0}}, m2 = ⌊d1⌋, d = ⌊d1⌋, d1 = {{(fun x → x+1) m1.1.1.1}}⟩

$\rightarrow^*_{7,2}$
⟨a1.2.1 | a = ⌊a1⌋,
a1 = {⌊m⌋;{d1.1.1}}, m = ⌊b1⌋, b = ⌊b1⌋,
b1 = ⌊⌊m2⌋;⌊m1⌋⌋, m1 = ⌊c1⌋, c = ⌊c1⌋, c1 = {{0}}, m2 = ⌊d1⌋, d = ⌊d1⌋, d1 = {{(fun x → x+1) c1.1.1}}⟩
$\rightarrow^*_{3,3,7,2,2}$
⟨a1.2.1 | a = ⌊a1⌋,
a1 = {⌊m⌋;{d1.1.1}}, m = ⌊b1⌋, b = ⌊b1⌋,
b1 = ⌊⌊m2⌋;⌊m1⌋⌋, m1 = ⌊c1⌋, c = ⌊c1⌋, c1 = ⌊⌊0⌋⌋, m2 = ⌊d1⌋, d = ⌊d1⌋, d1 = {{(fun x → x+1) 0}}⟩
$\rightarrow^*_{3,3}$
⟨a1.2.1 | a = ⌊a1⌋,
a1 = {⌊m⌋;{d1.1.1}}, m = ⌊b1⌋, b = ⌊b1⌋,
b1 = ⌊⌊m2⌋;⌊m1⌋⌋, m1 = ⌊c1⌋, c = ⌊c1⌋, c1 = ⌊⌊0⌋⌋, m2 = ⌊d1⌋, d = ⌊d1⌋, d1 = ⌊⌊1⌋⌋⟩
$\rightarrow^*_{7,2,2}$
⟨a1.2.1 | a = ⌊a1⌋,
a1 = {⌊m⌋;{1}}, m = ⌊b1⌋, b = ⌊b1⌋,
b1 = ⌊⌊m2⌋;⌊m1⌋⌋, m1 = ⌊c1⌋, c = ⌊c1⌋, c1 = ⌊⌊0⌋⌋, m2 = ⌊d1⌋, d = ⌊d1⌋, d1 = ⌊⌊1⌋⌋⟩
$\rightarrow^*_{3,3}$
⟨a1.2.1 | a = ⌊a1⌋,
a1 = ⌊⌊m⌋;⌊1⌋⌋, m = ⌊b1⌋, b = ⌊b1⌋,
b1 = ⌊⌊m2⌋;⌊m1⌋⌋, m1 = ⌊c1⌋, c = ⌊c1⌋, c1 = ⌊⌊0⌋⌋, m2 = ⌊d1⌋, d = ⌊d1⌋, d1 = ⌊⌊1⌋⌋⟩
$\rightarrow^*_{6,2,2}$
⟨1 | a = ⌊a1⌋,
a1 = ⌊⌊m⌋;⌊1⌋⌋, m = ⌊b1⌋, b = ⌊b1⌋,
b1 = ⌊⌊m2⌋;⌊m1⌋⌋, m1 = ⌊c1⌋, c = ⌊c1⌋, c1 = ⌊⌊0⌋⌋, m2 = ⌊d1⌋, d = ⌊d1⌋, d1 = ⌊⌊1⌋⌋⟩

7.7 The first example for the second principle (the 4th example in page 7)

The original program written in the syntax of *Osan* is:

{(X)
M1 = {(X1) m1 = X.M2.m1; m2 = X1.m1 + 2; m3 = X.M2.m3 + X1.m2 };
M2 = {(X2) m1 = 3; m2 = X.M1.m2; m3 = X2.m1 + X2.m2 };
main = X.M1.m3 }

Execution of the above program gets stuck, when M2.m2 attempts to access M1.m2. Below are reductions.

⟨a.1.3.1 | a = ⌊a1⌋, a1 = {
⟨⌊m1⌋ | m1 = ⟨b | b = ⌊b1⌋, b1 = {{a.1.2.1.1.1.1}; {b.1.1.1+2}; {a.1.2.1.1.3.1+b.1.2.1}}⟩⟩;
⟨⌊m2⌋ | m2 = ⟨c | c = ⌊c1⌋, c1 = {{3}; {a.1.1.1.1.2.1}; {c.1.1.1+c.1.2.1}}⟩⟩;
{a.1.1.1.1.3.1}}⟩
$\rightarrow^*_{6,2}$
⟨a1.3.1 | a = ⌊a1⌋, a1 = {
⟨⌊m1⌋ | m1 = ⟨b | b = ⌊b1⌋, b1 = {{a.1.2.1.1.1.1}; {b.1.1.1+2}; {a.1.2.1.1.3.1+b.1.2.1}}⟩⟩;
⟨⌊m2⌋ | m2 = ⟨c | c = ⌊c1⌋, c1 = {{3}; {a.1.1.1.1.2.1}; {c.1.1.1+c.1.2.1}}⟩⟩;
{a.1.1.1.1.3.1}}⟩
$\rightarrow^*_{9,9}$
⟨a1.3.1 | a = ⌊a1⌋, a1 = ⟨⟨{⌊m1⌋; ⌊m2⌋; {a.1.1.1.1.3.1}} |
m1 = ⟨b | b = ⌊b1⌋, b1 = {{a.1.2.1.1.1.1}; {b.1.1.1+2}; {a.1.2.1.1.3.1+b.1.2.1}}⟩⟩ |
m2 = ⟨c | c = ⌊c1⌋, c1 = {{3}; {a.1.1.1.1.2.1}; {c.1.1.1+c.1.2.1}}⟩⟩⟩
$\rightarrow^*_{7,2}$

⟨a1.3.1 | a = ⌊a1⌋, a1 = ⟨⟨{⌊m1⌋; ⌊m2⌋; {a1.1.1.1.3.1}} |
m1 = ⟨b | b = ⌊b1⌋, b1 = {{a.1.2.1.1.1.1}; {b.1.1.1+2}; {a.1.2.1.1.3.1+b.1.2.1}}⟩⟩ |
m2 = ⟨c | c = ⌊c1⌋, c1 = {{3}; {a.1.1.1.1.2.1}; {c.1.1.1+c.1.2.1}}⟩⟩⟩
$\rightarrow^*_{12,12}$
⟨a1.3.1 | a = ⌊a1⌋, a1 = {⌊m1⌋; ⌊m2⌋; {a1.1.1.1.3.1}},
m1 = ⟨b | b = ⌊b1⌋, b1 = {{a.1.2.1.1.1.1}; {b.1.1.1+2}; {a.1.2.1.1.3.1+b.1.2.1}}⟩
m2 = ⟨c | c = ⌊c1⌋, c1 = {{3}; {a.1.1.1.1.2.1}; {c.1.1.1+c.1.2.1}}⟩⟩
$\rightarrow^*_{8,2,2}$
⟨a1.3.1 | a = ⌊a1⌋, a1 = {⌊m1⌋; ⌊m2⌋; {m1.1.3.1}},
m1 = ⟨b | b = ⌊b1⌋, b1 = {{a.1.2.1.1.1.1}; {b.1.1.1+2}; {a.1.2.1.1.3.1+b.1.2.1}}⟩
m2 = ⟨c | c = ⌊c1⌋, c1 = {{3}; {a.1.1.1.1.2.1}; {c.1.1.1+c.1.2.1}}⟩⟩
$\rightarrow^*_{6,11}$
⟨a1.3.1 | a = ⌊a1⌋, a1 = {⌊m1⌋; ⌊m2⌋; {m1.1.3.1}},
m1 = ⌊b1⌋, b = ⌊b1⌋, b1 = {{a.1.2.1.1.1.1}; {b.1.1.1+2}; {a.1.2.1.1.3.1+b.1.2.1}}
m2 = ⟨c | c = ⌊c1⌋, c1 = {{3}; {a.1.1.1.1.2.1}; {c.1.1.1+c.1.2.1}}⟩⟩
$\rightarrow^*_{7,2}$
⟨a1.3.1 | a = ⌊a1⌋, a1 = {⌊m1⌋; ⌊m2⌋; {b1.3.1}},
m1 = ⌊b1⌋, b = ⌊b1⌋, b1 = {{a.1.2.1.1.1.1}; {b.1.1.1+2}; {a.1.2.1.1.3.1+b.1.2.1}}
m2 = ⟨c | c = ⌊c1⌋, c1 = {{3}; {a.1.1.1.1.2.1}; {c.1.1.1+c.1.2.1}}⟩⟩
$\rightarrow^*_{7,2,8,2,2}$
⟨a1.3.1 | a = ⌊a1⌋, a1 = {⌊m1⌋; ⌊m2⌋; {b1.3.1}},
m1 = ⌊b1⌋, b = ⌊b1⌋, b1 = {{m2.1.1.1}; {b.1.1.1+2}; {a.1.2.1.1.3.1+b.1.2.1}}
m2 = ⟨c | c = ⌊c1⌋, c1 = {{3}; {a.1.1.1.1.2.1}; {c.1.1.1+c.1.2.1}}⟩⟩
$\rightarrow^*_{6,11}$
⟨a1.3.1 | a = ⌊a1⌋, a1 = {⌊m1⌋; ⌊m2⌋; {b1.3.1}},
m1 = ⌊b1⌋, b = ⌊b1⌋, b1 = {{m2.1.1.1}; {b.1.1.1+2}; {a.1.2.1.1.3.1+b.1.2.1}}
m2 = ⌊c1⌋, c = ⌊c1⌋, c1 = {{3}; {a.1.1.1.1.2.1}; {c.1.1.1+c.1.2.1}}⟩
$\rightarrow^*_{7,2}$
⟨a1.3.1 | a = ⌊a1⌋, a1 = {⌊m1⌋; ⌊m2⌋; {b1.3.1}},
m1 = ⌊b1⌋, b = ⌊b1⌋, b1 = {{c1.1.1}; {b.1.1.1+2}; {a.1.2.1.1.3.1+b.1.2.1}}
m2 = ⌊c1⌋, c = ⌊c1⌋, c1 = {{3}; {a.1.1.1.1.2.1}; {c.1.1.1+c.1.2.1}}⟩
$\rightarrow_3$
⟨a1.3.1 | a = ⌊a1⌋, a1 = {⌊m1⌋; ⌊m2⌋; {b1.3.1}},
m1 = ⌊b1⌋, b = ⌊b1⌋, b1 = {{c1.1.1}; {b.1.1.1+2}; {a.1.2.1.1.3.1+b.1.2.1}}
m2 = ⌊c1⌋, c = ⌊c1⌋, c1 = {⌊3⌋; {a.1.1.1.1.2.1}; {c.1.1.1+c.1.2.1}}⟩
$\rightarrow^*_{7,2,8,2,2}$
⟨a1.3.1 | a = ⌊a1⌋, a1 = {⌊m1⌋; ⌊m2⌋; {b1.3.1}},
m1 = ⌊b1⌋, b = ⌊b1⌋, b1 = {{c1.1.1}; {b.1.1.1+2}; {a.1.2.1.1.3.1+b.1.2.1}}
m2 = ⌊c1⌋, c = ⌊c1⌋, c1 = {⌊3⌋; {m1.1.2.1}; {c.1.1.1+c.1.2.1}}⟩
$\rightarrow^*_{7,2}$
⟨a1.3.1 | a = ⌊a1⌋, a1 = {⌊m1⌋; ⌊m2⌋; {b1.3.1}},
m1 = ⌊b1⌋, b = ⌊b1⌋, b1 = {{c1.1.1}; {b.1.1.1+2}; {a.1.2.1.1.3.1+b.1.2.1}}
m2 = ⌊c1⌋, c = ⌊c1⌋, c1 = {⌊3⌋; {b1.2.1}; {c.1.1.1+c.1.2.1}}⟩
$\rightarrow_8$
⟨a1.3.1 | a = ⌊a1⌋, a1 = {⌊m1⌋; ⌊m2⌋; {b1.3.1}},
m1 = ⌊b1⌋, b = ⌊b1⌋, b1 = {{c1.1.1}; {b.1.1.1+2}; {a.1.2.1.1.3.1+b.1.2.1}}
m2 = ⌊c1⌋, c = ⌊c1⌋, c1 = {⌊3⌋; {⌊⌋.2.1}; {c.1.1.1+c.1.2.1}}⟩

⌊⌋.2 cannot be reduced. Thus the reduction gets stuck.

7.8 The second example for the second principle (the last example in page 7)

The original program written in the syntax of *Osan* is:

```
{(X)
 M1 = {(X1)
   m1 = X.M2.m1;
   M1' = {(X1')  m2 = X1.m1 + 2;  M1'' = { m3 = X.M2.M2'.M2''.m3 + X1'.m2 } } };
 M2 = {(X2)
   m1 = 3;
   M2' = {(X2')  m2 = X.M1.M1'.m2;  M2'' = { m3 = X2.m1 + X2'.m2 } } };
 main = X.M1.M1'.M1''.m3}
```

Execution of the above program succeeds. Below are the reductions.

$$
\begin{aligned}
&\langle \text{a.1.3.1} \mid \text{a} = \lfloor \text{a1} \rfloor,\ \text{a1} = \{ \\
&\quad \langle \lfloor \text{m1} \rfloor \mid \text{m1} = \langle \text{b} \mid \text{b} = \lfloor \text{b1} \rfloor,\ \text{b1} = \{\{\text{a.1.2.1.1.1.1}\};\ \langle \lfloor \text{m1'} \rfloor \mid \\
&\qquad \text{m1'} = \langle \text{c} \mid \text{c} = \lfloor \text{c1} \rfloor,\ \text{c1} = \{\{\text{b.1.1.1+2}\};\ \langle \lfloor \text{m1''} \rfloor \mid \\
&\qquad\quad \text{m1''} = \langle \text{d} \mid \text{d} = \lfloor \text{d1} \rfloor,\ \text{d1} = \{\{\text{a.1.2.1.1.2.1.1.2.1.1.1.1+c.1.1.1}\}\}\rangle\}\rangle\}\rangle\}\rangle; \\
&\quad \langle \lfloor \text{m2} \rfloor \mid \text{m2} = \langle \text{e} \mid \text{e} = \lfloor \text{e1} \rfloor,\ \text{e1} = \{\{\text{3}\};\ \langle \lfloor \text{m2'} \rfloor \mid \\
&\qquad \text{m2'} = \langle \text{f} \mid \text{f} = \lfloor \text{f1} \rfloor,\ \text{f1} = \{\{\text{a.1.1.1.1.2.1.1.1.1}\};\ \langle \lfloor \text{m2''} \rfloor \mid \\
&\qquad\quad \text{m2''} = \langle \text{g} \mid \text{g} = \lfloor \text{g1} \rfloor,\ \text{g1} = \{\{\text{e.1.1.1+f.1.1.1}\}\}\rangle\}\rangle\}\rangle\}\rangle; \\
&\quad \{\text{a.1.1.1.1.2.1.1.2.1.1.1.1}\}\}\rangle \\
&\rightarrow^{*}_{6,2} \\
&\langle \text{a1.3.1} \mid \text{a} = \lfloor \text{a1} \rfloor,\ \text{a1} = \{ \\
&\quad \langle \lfloor \text{m1} \rfloor \mid \text{m1} = \langle \text{b} \mid \text{b} = \lfloor \text{b1} \rfloor,\ \text{b1} = \{\{\text{a.1.2.1.1.1.1}\};\ \langle \lfloor \text{m1'} \rfloor \mid \\
&\qquad \text{m1'} = \langle \text{c} \mid \text{c} = \lfloor \text{c1} \rfloor,\ \text{c1} = \{\{\text{b.1.1.1+2}\};\ \langle \lfloor \text{m1''} \rfloor \mid \\
&\qquad\quad \text{m1''} = \langle \text{d} \mid \text{d} = \lfloor \text{d1} \rfloor,\ \text{d1} = \{\{\text{a.1.2.1.1.2.1.1.2.1.1.1.1+c.1.1.1}\}\}\rangle\}\rangle\}\rangle\}\rangle; \\
&\quad \langle \lfloor \text{m2} \rfloor \mid \text{m2} = \langle \text{e} \mid \text{e} = \lfloor \text{e1} \rfloor,\ \text{e1} = \{\{\text{3}\};\ \langle \lfloor \text{m2'} \rfloor \mid \\
&\qquad \text{m2'} = \langle \text{f} \mid \text{f} = \lfloor \text{f1} \rfloor,\ \text{f1} = \{\{\text{a.1.1.1.1.2.1.1.1.1}\};\ \langle \lfloor \text{m2''} \rfloor \mid \\
&\qquad\quad \text{m2''} = \langle \text{g} \mid \text{g} = \lfloor \text{g1} \rfloor,\ \text{g1} = \{\{\text{e.1.1.1+f.1.1.1}\}\}\rangle\}\rangle\}\rangle\}\rangle; \\
&\quad \{\text{a.1.1.1.1.2.1.1.2.1.1.1.1}\}\}\rangle \\
&\rightarrow^{*}_{9,9} \\
&\langle \text{a1.3.1} \mid \text{a} = \lfloor \text{a1} \rfloor,\ \text{a1} = \langle\langle\{\lfloor \text{m1} \rfloor;\ \lfloor \text{m2} \rfloor;\ \{\text{a.1.1.1.1.2.1.1.2.1.1.1.1}\}\} \mid \\
&\quad \text{m1} = \langle \text{b} \mid \text{b} = \lfloor \text{b1} \rfloor,\ \text{b1} = \{\{\text{a.1.2.1.1.1.1}\};\ \langle \lfloor \text{m1'} \rfloor \mid \\
&\qquad \text{m1'} = \langle \text{c} \mid \text{c} = \lfloor \text{c1} \rfloor,\ \text{c1} = \{\{\text{b.1.1.1+2}\};\ \langle \lfloor \text{m1''} \rfloor \mid \\
&\qquad\quad \text{m1''} = \langle \text{d} \mid \text{d} = \lfloor \text{d1} \rfloor,\ \text{d1} = \{\{\text{a.1.2.1.1.2.1.1.2.1.1.1.1+c.1.1.1}\}\}\rangle\}\rangle\}\rangle\}\rangle\rangle \mid \\
&\quad \text{m2} = \langle \text{e} \mid \text{e} = \lfloor \text{e1} \rfloor,\ \text{e1} = \{\{\text{3}\};\ \langle \lfloor \text{m2'} \rfloor \mid \\
&\qquad \text{m2'} = \langle \text{f} \mid \text{f} = \lfloor \text{f1} \rfloor,\ \text{f1} = \{\{\text{a.1.1.1.1.2.1.1.1.1}\};\ \langle \lfloor \text{m2''} \rfloor \mid \\
&\qquad\quad \text{m2''} = \langle \text{g} \mid \text{g} = \lfloor \text{g1} \rfloor,\ \text{g1} = \{\{\text{e.1.1.1+f.1.1.1}\}\}\rangle\}\rangle\}\rangle\}\rangle\rangle \\
&\rightarrow^{*}_{7,2} \\
&\langle \text{a1.3.1} \mid \text{a} = \lfloor \text{a1} \rfloor,\ \text{a1} = \langle\langle\{\lfloor \text{m1} \rfloor;\ \lfloor \text{m2} \rfloor;\ \{\text{a1.1.1.1.2.1.1.2.1.1.1.1}\}\} \mid \\
&\quad \text{m1} = \langle \text{b} \mid \text{b} = \lfloor \text{b1} \rfloor,\ \text{b1} = \{\{\text{a.1.2.1.1.1.1}\};\ \langle \lfloor \text{m1'} \rfloor \mid \\
&\qquad \text{m1'} = \langle \text{c} \mid \text{c} = \lfloor \text{c1} \rfloor,\ \text{c1} = \{\{\text{b.1.1.1+2}\};\ \langle \lfloor \text{m1''} \rfloor \mid \\
&\qquad\quad \text{m1''} = \langle \text{d} \mid \text{d} = \lfloor \text{d1} \rfloor,\ \text{d1} = \{\{\text{a.1.2.1.1.2.1.1.2.1.1.1.1+c.1.1.1}\}\}\rangle\}\rangle\}\rangle\}\rangle\rangle \mid \\
&\quad \text{m2} = \langle \text{e} \mid \text{e} = \lfloor \text{e1} \rfloor,\ \text{e1} = \{\{\text{3}\};\ \langle \lfloor \text{m2'} \rfloor \mid \\
&\qquad \text{m2'} = \langle \text{f} \mid \text{f} = \lfloor \text{f1} \rfloor,\ \text{f1} = \{\{\text{a.1.1.1.1.2.1.1.1.1}\};\ \langle \lfloor \text{m2''} \rfloor \mid \\
&\qquad\quad \text{m2''} = \langle \text{g} \mid \text{g} = \lfloor \text{g1} \rfloor,\ \text{g1} = \{\{\text{e.1.1.1+f.1.1.1}\}\}\rangle\}\rangle\}\rangle\}\rangle\rangle \\
&\rightarrow^{*}_{12,12} \\
&\langle \text{a1.3.1} \mid \text{a} = \lfloor \text{a1} \rfloor,\ \text{a1} = \{\lfloor \text{m1} \rfloor;\ \lfloor \text{m2} \rfloor;\ \{\text{a1.1.1.1.2.1.1.2.1.1.1.1}\}\} \\
&\quad \text{m1} = \langle \text{b} \mid \text{b} = \lfloor \text{b1} \rfloor,\ \text{b1} = \{\{\text{a.1.2.1.1.1.1}\};\ \langle \lfloor \text{m1'} \rfloor \mid \\
&\qquad \text{m1'} = \langle \text{c} \mid \text{c} = \lfloor \text{c1} \rfloor,\ \text{c1} = \{\{\text{b.1.1.1+2}\};\ \langle \lfloor \text{m1''} \rfloor \mid \\
&\qquad\quad \text{m1''} = \langle \text{d} \mid \text{d} = \lfloor \text{d1} \rfloor,\ \text{d1} = \{\{\text{a.1.2.1.1.2.1.1.2.1.1.1.1+c.1.1.1}\}\}\rangle\}\rangle\}\rangle\}\rangle\} \\
&\quad \text{m2} = \langle \text{e} \mid \text{e} = \lfloor \text{e1} \rfloor,\ \text{e1} = \{\{\text{3}\};\ \langle \lfloor \text{m2'} \rfloor \mid \\
&\qquad \text{m2'} = \langle \text{f} \mid \text{f} = \lfloor \text{f1} \rfloor,\ \text{f1} = \{\{\text{a.1.1.1.1.2.1.1.1.1}\};\ \langle \lfloor \text{m2''} \rfloor \mid \\
&\qquad\quad \text{m2''} = \langle \text{g} \mid \text{g} = \lfloor \text{g1} \rfloor,\ \text{g1} = \{\{\text{e.1.1.1+f.1.1.1}\}\}\rangle\}\rangle\}\rangle\}\rangle\rangle \\
&\rightarrow^{*}_{8,2,2}
\end{aligned}
$$

⟨a1.3.1 | a = ⌊a1⌋, a1 = {⌊m1⌋; ⌊m2⌋; {m1.1.2.1.1.2.1.1.1.1}}
  m1 = ⟨b | b = ⌊b1⌋, b1 = {{a.1.2.1.1.1.1}; ⟨⌊m1'⌋ |
    m1' = ⟨c | c = ⌊c1⌋, c1 = {{b.1.1.1+2}; ⟨⌊m1"⌋ |
      m1" = ⟨d | d = ⌊d1⌋, d1 = {{a.1.2.1.1.2.1.1.2.1.1.1.1+c.1.1.1}}⟩⟩⟩⟩⟩⟩
  m2 = ⟨e | e = ⌊e1⌋, e1 = {{3}; ⟨⌊m2'⌋ |
    m2' = ⟨f | f = ⌊f1⌋, f1 = {{a.1.1.1.1.2.1.1.1.1}; ⟨⌊m2"⌋ |
      m2" = ⟨g | g = ⌊g1⌋, g1 = {{e.1.1.1+f.1.1.1}}⟩⟩⟩⟩⟩⟩
$\rightarrow^*_{6,11}$
⟨a1.3.1 | a = ⌊a1⌋, a1 = {⌊m1⌋; ⌊m2⌋; {m1.1.2.1.1.2.1.1.1.1}}
  m1 = ⌊b1⌋, b = ⌊b1⌋, b1 = {{a.1.2.1.1.1.1}; ⟨⌊m1'⌋ |
    m1' = ⟨c | c = ⌊c1⌋, c1 = {{b.1.1.1+2}; ⟨⌊m1"⌋ |
      m1" = ⟨d | d = ⌊d1⌋, d1 = {{a.1.2.1.1.2.1.1.2.1.1.1.1+c.1.1.1}}⟩⟩⟩⟩⟩}
  m2 = ⟨e | e = ⌊e1⌋, e1 = {{3}; ⟨⌊m2'⌋ |
    m2' = ⟨f | f = ⌊f1⌋, f1 = {{a.1.1.1.1.2.1.1.1.1}; ⟨⌊m2"⌋ |
      m2" = ⟨g | g = ⌊g1⌋, g1 = {{e.1.1.1+f.1.1.1}}⟩⟩⟩⟩⟩⟩
$\rightarrow^*_{7,2}$
⟨a1.3.1 | a = ⌊a1⌋, a1 = {⌊m1⌋; ⌊m2⌋; {b1.2.1.1.2.1.1.1.1}}
  m1 = ⌊b1⌋, b = ⌊b1⌋, b1 = {{a.1.2.1.1.1.1}; ⟨⌊m1'⌋ |
    m1' = ⟨c | c = ⌊c1⌋, c1 = {{b.1.1.1+2}; ⟨⌊m1"⌋ |
      m1" = ⟨d | d = ⌊d1⌋, d1 = {{a.1.2.1.1.2.1.1.2.1.1.1.1+c.1.1.1}}⟩⟩⟩⟩}
  m2 = ⟨e | e = ⌊e1⌋, e1 = {{3}; ⟨⌊m2'⌋ |
    m2' = ⟨f | f = ⌊f1⌋, f1 = {{a.1.1.1.1.2.1.1.1.1}; ⟨⌊m2"⌋ |
      m2" = ⟨g | g = ⌊g1⌋, g1 = {{e.1.1.1+f.1.1.1}}⟩⟩⟩⟩⟩⟩
$\rightarrow^*_{7,2,8,2,2}$
⟨a1.3.1 | a = ⌊a1⌋, a1 = {⌊m1⌋; ⌊m2⌋; {b1.2.1.1.2.1.1.1.1}}
  m1 = ⌊b1⌋, b = ⌊b1⌋, b1 = {{m2.1.1.1}; ⟨⌊m1'⌋ |
    m1' = ⟨c | c = ⌊c1⌋, c1 = {{b.1.1.1+2}; ⟨⌊m1"⌋ |
      m1" = ⟨d | d = ⌊d1⌋, d1 = {{a.1.2.1.1.2.1.1.2.1.1.1.1+c.1.1.1}}⟩⟩⟩⟩}
  m2 = ⟨e | e = ⌊e1⌋, e1 = {{3}; ⟨⌊m2'⌋ |
    m2' = ⟨f | f = ⌊f1⌋, f1 = {{a.1.1.1.1.2.1.1.1.1}; ⟨⌊m2"⌋ |
      m2" = ⟨g | g = ⌊g1⌋, g1 = {{e.1.1.1+f.1.1.1}}⟩⟩⟩⟩⟩⟩
$\rightarrow^*_{6,11}$
⟨a1.3.1 | a = ⌊a1⌋, a1 = {⌊m1⌋; ⌊m2⌋; {b1.2.1.1.2.1.1.1.1}}
  m1 = ⌊b1⌋, b = ⌊b1⌋, b1 = {{m2.1.1.1}; ⟨⌊m1'⌋ |
    m1' = ⟨c | c = ⌊c1⌋, c1 = {{b.1.1.1+2}; ⟨⌊m1"⌋ |
      m1" = ⟨d | d = ⌊d1⌋, d1 = {{a.1.2.1.1.2.1.1.2.1.1.1.1+c.1.1.1}}⟩⟩⟩⟩⟩}
  m2 = ⌊e1⌋, e = ⌊e1⌋, e1 = {{3}; ⟨⌊m2'⌋ |
    m2' = ⟨f | f = ⌊f1⌋, f1 = {{a.1.1.1.1.2.1.1.1.1}; ⟨⌊m2"⌋ |
      m2" = ⟨g | g = ⌊g1⌋, g1 = {{e.1.1.1+f.1.1.1}}⟩⟩⟩⟩⟩⟩
$\rightarrow^*_{7,2}$
⟨a1.3.1 | a = ⌊a1⌋, a1 = {⌊m1⌋; ⌊m2⌋; {b1.2.1.1.2.1.1.1.1}}
  m1 = ⌊b1⌋, b = ⌊b1⌋, b1 = {{e1.1.1}; ⟨⌊m1'⌋ |
    m1' = ⟨c | c = ⌊c1⌋, c1 = {{b.1.1.1+2}; ⟨⌊m1"⌋ |
      m1" = ⟨d | d = ⌊d1⌋, d1 = {{a.1.2.1.1.2.1.1.2.1.1.1.1+c.1.1.1}}⟩⟩⟩⟩}
  m2 = ⌊e1⌋, e = ⌊e1⌋, e1 = {{3}; ⟨⌊m2'⌋ |
    m2' = ⟨f | f = ⌊f1⌋, f1 = {{a.1.1.1.1.2.1.1.1.1}; ⟨⌊m2"⌋ |
      m2" = ⟨g | g = ⌊g1⌋, g1 = {{e.1.1.1+f.1.1.1}}⟩⟩⟩⟩⟩
$\rightarrow^*_{3,9,3}$
⟨a1.3.1 | a = ⌊a1⌋, a1 = {⌊m1⌋; ⌊m2⌋; {b1.2.1.1.2.1.1.1.1}}
  m1 = ⌊b1⌋, b = ⌊b1⌋, b1 = {{e1.1.1}; ⟨⌊m1'⌋ |
    m1' = ⟨c | c = ⌊c1⌋, c1 = {{b.1.1.1+2}; ⟨⌊m1"⌋ |
      m1" = ⟨d | d = ⌊d1⌋, d1 = {{a.1.2.1.1.2.1.1.2.1.1.1.1+c.1.1.1}}⟩⟩⟩⟩}
  m2 = ⟨e1⌋, e = ⌊e1⌋, e1 = ⟨⌊⌊3⌋; ⌊m2'⌋⌋ |
    m2' = ⟨f | f = ⌊f1⌋, f1 = {{a.1.1.1.1.2.1.1.1.1}; ⟨⌊m2"⌋ |
      m2" = ⟨g | g = ⌊g1⌋, g1 = {{e.1.1.1+f.1.1.1}}⟩⟩⟩⟩

$\rightarrow_{11}$

⟨a1.3.1 | a = ⌊a1⌋, a1 = {⌊m1⌋; ⌊m2⌋; {b1.2.1.1.2.1.1.1.1}}
  m1 = ⌊b1⌋, b = ⌊b1⌋, b1 = {{e1.1.1}; ⟨⌊m1'⌋ |
    m1' = ⟨c | c = ⌊c1⌋, c1 = {{b.1.1.1+2}; ⟨⌊m1"⌋ |
      m1" = ⟨d | d = ⌊d1⌋, d1 = {{a.1.2.1.1.2.1.1.2.1.1.1.1+c.1.1.1}}⟩⟩}⟩⟩}
  m2 = ⌊e1⌋, e = ⌊e1⌋, e1 = ⌊⌊3⌋; ⌊m2'⌋⌋, m2' = ⟨f | f = ⌊f1⌋, f1 = {{a.1.1.1.1.2.1.1.1.1}; ⟨⌊m2"⌋ |
    m2" = ⟨g | g = ⌊g1⌋, g1 = {{e.1.1.1+f.1.1.1}}⟩⟩}⟩⟩

$\rightarrow^*_{7,2,2,3}$

⟨a1.3.1 | a = ⌊a1⌋, a1 = {⌊m1⌋; ⌊m2⌋; {b1.2.1.1.2.1.1.1.1}}
  m1 = ⌊b1⌋, b = ⌊b1⌋, b1 = {⌊3⌋; ⟨⌊m1'⌋ |
    m1' = ⟨c | c = ⌊c1⌋, c1 = {{b.1.1.1+2}; ⟨⌊m1"⌋ |
      m1" = ⟨d | d = ⌊d1⌋, d1 = {{a.1.2.1.1.2.1.1.2.1.1.1.1+c.1.1.1}}⟩⟩}⟩⟩}
  m2 = ⌊e1⌋, e = ⌊e1⌋, e1 = ⌊⌊3⌋; ⌊m2'⌋⌋, m2' = ⟨f | f = ⌊f1⌋, f1 = {{a.1.1.1.1.2.1.1.1.1}; ⟨⌊m2"⌋ |
    m2" = ⟨g | g = ⌊g1⌋, g1 = {{e.1.1.1+f.1.1.1}}⟩⟩}⟩⟩

$\rightarrow_{9}$

⟨a1.3.1 | a = ⌊a1⌋, a1 = {⌊m1⌋; ⌊m2⌋; {b1.2.1.1.2.1.1.1.1}}
  m1 = ⌊b1⌋, b = ⌊b1⌋, b1 = ⟨{⌊3⌋; ⌊m1'⌋} |
    m1' = ⟨c | c = ⌊c1⌋, c1 = {{b.1.1.1+2}; ⟨⌊m1"⌋ |
      m1" = ⟨d | d = ⌊d1⌋, d1 = {{a.1.2.1.1.2.1.1.2.1.1.1.1+c.1.1.1}}⟩⟩}⟩⟩
  m2 = ⌊e1⌋, e = ⌊e1⌋, e1 = ⌊⌊3⌋; ⌊m2'⌋⌋, m2' = ⟨f | f = ⌊f1⌋, f1 = {{a.1.1.1.1.2.1.1.1.1}; ⟨⌊m2"⌋ |
    m2" = ⟨g | g = ⌊g1⌋, g1 = {{e.1.1.1+f.1.1.1}}⟩⟩}⟩⟩

$\rightarrow^*_{3,11}$

⟨a1.3.1 | a = ⌊a1⌋, a1 = {⌊m1⌋; ⌊m2⌋; {b1.2.1.1.2.1.1.1.1}}
  m1 = ⌊b1⌋, b = ⌊b1⌋, b1 = ⌊⌊3⌋; ⌊m1'⌋⌋, m1' = ⟨c | c = ⌊c1⌋, c1 = {{b.1.1.1+2}; ⟨⌊m1"⌋ |
    m1" = ⟨d | d = ⌊d1⌋, d1 = {{a.1.2.1.1.2.1.1.2.1.1.1.1+c.1.1.1}}⟩⟩}⟩
  m2 = ⌊e1⌋, e = ⌊e1⌋, e1 = ⌊⌊3⌋; ⌊m2'⌋⌋, m2' = ⟨f | f = ⌊f1⌋, f1 = {{a.1.1.1.1.2.1.1.1.1}; ⟨⌊m2"⌋ |
    m2" = ⟨g | g = ⌊g1⌋, g1 = {{e.1.1.1+f.1.1.1}}⟩⟩}⟩⟩

$\rightarrow^*_{7,2,2}$

⟨a1.3.1 | a = ⌊a1⌋, a1 = {⌊m1⌋; ⌊m2⌋; {m1'.1.2.1.1.1.1}}
  m1 = ⌊b1⌋, b = ⌊b1⌋, b1 = ⌊⌊3⌋; ⌊m1'⌋⌋, m1' = ⟨c | c = ⌊c1⌋, c1 = {{b.1.1.1+2}; ⟨⌊m1"⌋ |
    m1" = ⟨d | d = ⌊d1⌋, d1 = {{a.1.2.1.1.2.1.1.2.1.1.1.1+c.1.1.1}}⟩⟩}⟩
  m2 = ⌊e1⌋, e = ⌊e1⌋, e1 = ⌊⌊3⌋; ⌊m2'⌋⌋, m2' = ⟨f | f = ⌊f1⌋, f1 = {{a.1.1.1.1.2.1.1.1.1}; ⟨⌊m2"⌋ |
    m2" = ⟨g | g = ⌊g1⌋, g1 = {{e.1.1.1+f.1.1.1}}⟩⟩}⟩⟩

$\rightarrow^*_{6,11,7,2}$

⟨a1.3.1 | a = ⌊a1⌋, a1 = {⌊m1⌋; ⌊m2⌋; {c1.2.1.1.1.1}}
  m1 = ⌊b1⌋, b = ⌊b1⌋, b1 = ⌊⌊3⌋; ⌊m1'⌋⌋, m1' = ⌊c1⌋, c = ⌊c1⌋, c1 = {{b.1.1.1+2}; ⟨⌊m1"⌋ |
    m1" = ⟨d | d = ⌊d1⌋, d1 = {{a.1.2.1.1.2.1.1.2.1.1.1.1+c.1.1.1}}⟩⟩}
  m2 = ⌊e1⌋, e = ⌊e1⌋, e1 = ⌊⌊3⌋; ⌊m2'⌋⌋, m2' = ⟨f | f = ⌊f1⌋, f1 = {{a.1.1.1.1.2.1.1.1.1}; ⟨⌊m2"⌋ |
    m2" = ⟨g | g = ⌊g1⌋, g1 = {{e.1.1.1+f.1.1.1}}⟩⟩}⟩⟩

$\rightarrow^*_{7,2,7,2,2,3}$

⟨a1.3.1 | a = ⌊a1⌋, a1 = {⌊m1⌋; ⌊m2⌋; {c1.2.1.1.1.1}}
  m1 = ⌊b1⌋, b = ⌊b1⌋, b1 = ⌊⌊3⌋; ⌊m1'⌋⌋, m1' = ⌊c1⌋, c = ⌊c1⌋, c1 = {⌊5⌋; ⟨⌊m1"⌋ |
    m1" = ⟨d | d = ⌊d1⌋, d1 = {{a.1.2.1.1.2.1.1.2.1.1.1.1+c.1.1.1}}⟩⟩}
  m2 = ⌊e1⌋, e = ⌊e1⌋, e1 = ⌊⌊3⌋; ⌊m2'⌋⌋, m2' = ⟨f | f = ⌊f1⌋, f1 = {{a.1.1.1.1.2.1.1.1.1}; ⟨⌊m2"⌋ |
    m2" = ⟨g | g = ⌊g1⌋, g1 = {{e.1.1.1+f.1.1.1}}⟩⟩}⟩⟩

$\rightarrow^*_{9,3}$

⟨a1.3.1 | a = ⌊a1⌋, a1 = {⌊m1⌋; ⌊m2⌋; {c1.2.1.1.1.1}}
  m1 = ⌊b1⌋, b = ⌊b1⌋, b1 = ⌊⌊3⌋; ⌊m1'⌋⌋, m1' = ⌊c1⌋, c = ⌊c1⌋, c1 = ⟨⌊⌊5⌋; ⌊m1"⌋⌋ |
    m1" = ⟨d | d = ⌊d1⌋, d1 = {{a.1.2.1.1.2.1.1.2.1.1.1.1+c.1.1.1}}⟩⟩
  m2 = ⌊e1⌋, e = ⌊e1⌋, e1 = ⌊⌊3⌋; ⌊m2'⌋⌋, m2' = ⟨f | f = ⌊f1⌋, f1 = {{a.1.1.1.1.2.1.1.1.1}; ⟨⌊m2"⌋ |
    m2" = ⟨g | g = ⌊g1⌋, g1 = {{e.1.1.1+f.1.1.1}}⟩⟩}⟩⟩

$\rightarrow^*_{11,7,2,2}$

⟨a1.3.1 | a = ⌊a1⌋, a1 = {⌊m1⌋; ⌊m2⌋; {m1".1.1.1}}
  m1 = ⌊b1⌋, b = ⌊b1⌋, b1 = ⌊⌊3⌋; ⌊m1'⌋⌋, m1' = ⌊c1⌋, c = ⌊c1⌋, c1 = ⌊⌊5⌋; ⌊m1"⌋⌋,
  m1" = ⟨d | d = ⌊d1⌋, d1 = {{a.1.2.1.1.2.1.1.2.1.1.1.1+c.1.1.1}}⟩
  m2 = ⌊e1⌋, e = ⌊e1⌋, e1 = ⌊⌊3⌋; ⌊m2'⌋⌋, m2' = ⟨f | f = ⌊f1⌋, f1 = {{a.1.1.1.1.2.1.1.1.1}; ⟨⌊m2"⌋ |
    m2" = ⟨g | g = ⌊g1⌋, g1 = {{e.1.1.1+f.1.1.1}}⟩⟩}⟩⟩
$\rightarrow^{*}_{6,11,7,2}$
⟨a1.3.1 | a = ⌊a1⌋, a1 = {⌊m1⌋; ⌊m2⌋; {d1.1.1}}
  m1 = ⌊b1⌋, b = ⌊b1⌋, b1 = ⌊⌊3⌋; ⌊m1'⌋⌋, m1' = ⌊c1⌋, c = ⌊c1⌋, c1 = ⌊⌊5⌋; ⌊m1"⌋⌋,
  m1" = ⌊d1⌋, d1 = {{a.1.2.1.1.2.1.1.2.1.1.1.1+c.1.1.1}}
  m2 = ⌊e1⌋, e = ⌊e1⌋, e1 = ⌊⌊3⌋; ⌊m2'⌋⌋, m2' = ⟨f | f = ⌊f1⌋, f1 = {{a.1.1.1.1.2.1.1.1.1}; ⟨⌊m2"⌋ |
    m2" = ⟨g | g = ⌊g1⌋, g1 = {{e.1.1.1+f.1.1.1}}⟩⟩}⟩⟩
$\rightarrow^{*}_{7,2,7,2,2}$
⟨a1.3.1 | a = ⌊a1⌋, a1 = {⌊m1⌋; ⌊m2⌋; {d1.1.1}}
  m1 = ⌊b1⌋, b = ⌊b1⌋, b1 = ⌊⌊3⌋; ⌊m1'⌋⌋, m1' = ⌊c1⌋, c = ⌊c1⌋, c1 = ⌊⌊5⌋; ⌊m1"⌋⌋,
  m1" = ⌊d1⌋, d1 = {{m2.1.2.1.1.2.1.1.1.1+c.1.1.1}}
  m2 = ⌊e1⌋, e = ⌊e1⌋, e1 = ⌊⌊3⌋; ⌊m2'⌋⌋, m2' = ⟨f | f = ⌊f1⌋, f1 = {{a.1.1.1.1.2.1.1.1.1}; ⟨⌊m2"⌋ |
    m2" = ⟨g | g = ⌊g1⌋, g1 = {{e.1.1.1+f.1.1.1}}⟩⟩}⟩⟩
$\rightarrow^{*}_{7,2,7,2,2}$
⟨a1.3.1 | a = ⌊a1⌋, a1 = {⌊m1⌋; ⌊m2⌋; {d1.1.1}}
  m1 = ⌊b1⌋, b = ⌊b1⌋, b1 = ⌊⌊3⌋; ⌊m1'⌋⌋, m1' = ⌊c1⌋, c = ⌊c1⌋, c1 = ⌊⌊5⌋; ⌊m1"⌋⌋,
  m1" = ⌊d1⌋, d1 = {{m2'.1.2.1.1.1.1+c.1.1.1}}
  m2 = ⌊e1⌋, e = ⌊e1⌋, e1 = ⌊⌊3⌋; ⌊m2'⌋⌋, m2' = ⟨f | f = ⌊f1⌋, f1 = {{a.1.1.1.1.2.1.1.1.1}; ⟨⌊m2"⌋ |
    m2" = ⟨g | g = ⌊g1⌋, g1 = {{e.1.1.1+f.1.1.1}}⟩⟩}⟩⟩
$\rightarrow^{*}_{6,11,7,2}$
⟨a1.3.1 | a = ⌊a1⌋, a1 = {⌊m1⌋; ⌊m2⌋; {d1.1.1}}
  m1 = ⌊b1⌋, b = ⌊b1⌋, b1 = ⌊⌊3⌋; ⌊m1'⌋⌋, m1' = ⌊c1⌋, c = ⌊c1⌋, c1 = ⌊⌊5⌋; ⌊m1"⌋⌋,
  m1" = ⌊d1⌋, d1 = {{f1.2.1.1.1.1+c.1.1.1}}
  m2 = ⌊e1⌋, e = ⌊e1⌋, e1 = ⌊⌊3⌋; ⌊m2'⌋⌋, m2' = ⌊f1⌋, f = ⌊f1⌋, f1 = {{a.1.1.1.1.2.1.1.1.1}; ⟨⌊m2"⌋ |
    m2" = ⟨g | g = ⌊g1⌋, g1 = {{e.1.1.1+f.1.1.1}}⟩⟩⟩
$\rightarrow^{*}_{7,2,8,2,2}$
⟨a1.3.1 | a = ⌊a1⌋, a1 = {⌊m1⌋; ⌊m2⌋; {d1.1.1}}
  m1 = ⌊b1⌋, b = ⌊b1⌋, b1 = ⌊⌊3⌋; ⌊m1'⌋⌋, m1' = ⌊c1⌋, c = ⌊c1⌋, c1 = ⌊⌊5⌋; ⌊m1"⌋⌋,
  m1" = ⌊d1⌋, d1 = {{f1.2.1.1.1.1+c.1.1.1}}
  m2 = ⌊e1⌋, e = ⌊e1⌋, e1 = ⌊⌊3⌋; ⌊m2'⌋⌋, m2' = ⌊f1⌋, f = ⌊f1⌋, f1 = {{m1.1.2.1.1.1.1}; ⟨⌊m2"⌋ |
    m2" = ⟨g | g = ⌊g1⌋, g1 = {{e.1.1.1+f.1.1.1}}⟩⟩⟩
$\rightarrow^{*}_{7,2,7,2,2}$
⟨a1.3.1 | a = ⌊a1⌋, a1 = {⌊m1⌋; ⌊m2⌋; {d1.1.1}}
  m1 = ⌊b1⌋, b = ⌊b1⌋, b1 = ⌊⌊3⌋; ⌊m1'⌋⌋, m1' = ⌊c1⌋, c = ⌊c1⌋, c1 = ⌊⌊5⌋; ⌊m1"⌋⌋,
  m1" = ⌊d1⌋, d1 = {{f1.2.1.1.1.1+c.1.1.1}}
  m2 = ⌊e1⌋, e = ⌊e1⌋, e1 = ⌊⌊3⌋; ⌊m2'⌋⌋, m2' = ⌊f1⌋, f = ⌊f1⌋, f1 = {{m1'.1.1.1}; ⟨⌊m2"⌋ |
    m2" = ⟨g | g = ⌊g1⌋, g1 = {{e.1.1.1+f.1.1.1}}⟩⟩⟩
$\rightarrow^{*}_{7,2,7,2,2,3}$
⟨a1.3.1 | a = ⌊a1⌋, a1 = {⌊m1⌋; ⌊m2⌋; {d1.1.1}}
  m1 = ⌊b1⌋, b = ⌊b1⌋, b1 = ⌊⌊3⌋; ⌊m1'⌋⌋, m1' = ⌊c1⌋, c = ⌊c1⌋, c1 = ⌊⌊5⌋; ⌊m1"⌋⌋,
  m1" = ⌊d1⌋, d1 = {{f1.2.1.1.1.1+c.1.1.1}}
  m2 = ⌊e1⌋, e = ⌊e1⌋, e1 = ⌊⌊3⌋; ⌊m2'⌋⌋, m2' = ⌊f1⌋, f = ⌊f1⌋, f1 = {⌊5⌋; ⟨⌊m2"⌋ |
    m2" = ⟨g | g = ⌊g1⌋, g1 = {{e.1.1.1+f.1.1.1}}⟩⟩}⟩
$\rightarrow^{*}_{9,3,11}$
⟨a1.3.1 | a = ⌊a1⌋, a1 = {⌊m1⌋; ⌊m2⌋; {d1.1.1}}
  m1 = ⌊b1⌋, b = ⌊b1⌋, b1 = ⌊⌊3⌋; ⌊m1'⌋⌋, m1' = ⌊c1⌋, c = ⌊c1⌋, c1 = ⌊⌊5⌋; ⌊m1"⌋⌋,
  m1" = ⌊d1⌋, d1 = {{f1.2.1.1.1.1+c.1.1.1}}
  m2 = ⌊e1⌋, e = ⌊e1⌋, e1 = ⌊⌊3⌋; ⌊m2'⌋⌋, m2' = ⌊f1⌋, f = ⌊f1⌋, f1 = ⌊⌊5⌋; ⌊m2"⌋⌋,
  m2" = ⟨g | g = ⌊g1⌋, g1 = {{e.1.1.1+f.1.1.1}}⟩⟩
$\rightarrow^{*}_{7,2,2}$

⟨a1.3.1 | a = ⌊a1⌋, a1 = {⌊m1⌋; ⌊m2⌋; {d1.1.1}}
m1 = ⌊b1⌋, b = ⌊b1⌋, b1 = ⌊⌊3⌋; ⌊m1'⌋⌋, m1' = ⌊c1⌋, c = ⌊c1⌋, c1 = ⌊⌊5⌋; ⌊m1"⌋⌋,
m1" = ⌊d1⌋, d1 = {{m2".1.1.1+c.1.1.1}}
m2 = ⌊e1⌋, e = ⌊e1⌋, e1 = ⌊⌊3⌋; ⌊m2'⌋⌋, m2' = ⌊f1⌋, f = ⌊f1⌋, f1 = ⌊⌊5⌋; ⌊m2"⌋⌋,
m2" = ⟨g | g = ⌊g1⌋, g1 = {{e.1.1.1+f.1.1.1}}⟩⟩
$\longrightarrow^*_{6,11,7,2}$
⟨a1.3.1 | a = ⌊a1⌋, a1 = {⌊m1⌋; ⌊m2⌋; {d1.1.1}}
m1 = ⌊b1⌋, b = ⌊b1⌋, b1 = ⌊⌊3⌋; ⌊m1'⌋⌋, m1' = ⌊c1⌋, c = ⌊c1⌋, c1 = ⌊⌊5⌋; ⌊m1"⌋⌋,
m1" = ⌊d1⌋, d1 = {{g1.1.1+c.1.1.1}}
m2 = ⌊e1⌋, e = ⌊e1⌋, e1 = ⌊⌊3⌋; ⌊m2'⌋⌋, m2' = ⌊f1⌋, f = ⌊f1⌋, f1 = ⌊⌊5⌋; ⌊m2"⌋⌋,
m2" = ⌊g1⌋, g = ⌊g1⌋, g1 = {{e.1.1.1+f.1.1.1}}⟩
$\longrightarrow^*_{7,2,2,7,2,2,3,3}$
⟨a1.3.1 | a = ⌊a1⌋, a1 = {⌊m1⌋; ⌊m2⌋; {d1.1.1}}
m1 = ⌊b1⌋, b = ⌊b1⌋, b1 = ⌊⌊3⌋; ⌊m1'⌋⌋, m1' = ⌊c1⌋, c = ⌊c1⌋, c1 = ⌊⌊5⌋; ⌊m1"⌋⌋,
m1" = ⌊d1⌋, d1 = {{g1.1.1+c.1.1.1}}
m2 = ⌊e1⌋, e = ⌊e1⌋, e1 = ⌊⌊3⌋; ⌊m2'⌋⌋, m2' = ⌊f1⌋, f = ⌊f1⌋, f1 = ⌊⌊5⌋; ⌊m2"⌋⌋,
m2" = ⌊g1⌋, g = ⌊g1⌋, g1 = ⌊⌊8⌋⌋⟩
$\longrightarrow^*_{7,2,2}$
⟨a1.3.1 | a = ⌊a1⌋, a1 = {⌊m1⌋; ⌊m2⌋; {d1.1.1}}
m1 = ⌊b1⌋, b = ⌊b1⌋, b1 = ⌊⌊3⌋; ⌊m1'⌋⌋, m1' = ⌊c1⌋, c = ⌊c1⌋, c1 = ⌊⌊5⌋; ⌊m1"⌋⌋,
m1" = ⌊d1⌋, d1 = {{8+c.1.1.1}}
m2 = ⌊e1⌋, e = ⌊e1⌋, e1 = ⌊⌊3⌋; ⌊m2'⌋⌋, m2' = ⌊f1⌋, f = ⌊f1⌋, f1 = ⌊⌊5⌋; ⌊m2"⌋⌋,
m2" = ⌊g1⌋, g = ⌊g1⌋, g1 = ⌊⌊8⌋⌋⟩
$\longrightarrow^*_{7,2,7,2,2,3,3}$
⟨a1.3.1 | a = ⌊a1⌋, a1 = {⌊m1⌋; ⌊m2⌋; {d1.1.1}}
m1 = ⌊b1⌋, b = ⌊b1⌋, b1 = ⌊⌊3⌋; ⌊m1'⌋⌋, m1' = ⌊c1⌋, c = ⌊c1⌋, c1 = ⌊⌊5⌋; ⌊m1"⌋⌋,
m1" = ⌊d1⌋, d1 = ⌊⌊13⌋⌋
m2 = ⌊e1⌋, e = ⌊e1⌋, e1 = ⌊⌊3⌋; ⌊m2'⌋⌋, m2' = ⌊f1⌋, f = ⌊f1⌋, f1 = ⌊⌊5⌋; ⌊m2"⌋⌋,
m2" = ⌊g1⌋, g = ⌊g1⌋, g1 = ⌊⌊8⌋⌋⟩
$\longrightarrow^*_{7,2,2,3,3}$
⟨a1.3.1 | a = ⌊a1⌋, a1 = ⌊⌊m1⌋; ⌊m2⌋; ⌊13⌋⌋
m1 = ⌊b1⌋, b = ⌊b1⌋, b1 = ⌊⌊3⌋; ⌊m1'⌋⌋, m1' = ⌊c1⌋, c = ⌊c1⌋, c1 = ⌊⌊5⌋; ⌊m1"⌋⌋,
m1" = ⌊d1⌋, d1 = ⌊⌊13⌋⌋
m2 = ⌊e1⌋, e = ⌊e1⌋, e1 = ⌊⌊3⌋; ⌊m2'⌋⌋, m2' = ⌊f1⌋, f = ⌊f1⌋, f1 = ⌊⌊5⌋; ⌊m2"⌋⌋,
m2" = ⌊g1⌋, g = ⌊g1⌋, g1 = ⌊⌊8⌋⌋⟩
$\longrightarrow^*_{6,2,2}$
⟨13 | a = ⌊a1⌋, a1 = ⌊⌊m1⌋; ⌊m2⌋; ⌊13⌋⌋
m1 = ⌊b1⌋, b = ⌊b1⌋, b1 = ⌊⌊3⌋; ⌊m1'⌋⌋, m1' = ⌊c1⌋, c = ⌊c1⌋, c1 = ⌊⌊5⌋; ⌊m1"⌋⌋,
m1" = ⌊d1⌋, d1 = ⌊⌊13⌋⌋
m2 = ⌊e1⌋, e = ⌊e1⌋, e1 = ⌊⌊3⌋; ⌊m2'⌋⌋, m2' = ⌊f1⌋, f = ⌊f1⌋, f1 = ⌊⌊5⌋; ⌊m2"⌋⌋,
m2" = ⌊g1⌋, g = ⌊g1⌋, g1 = ⌊⌊8⌋⌋⟩

7.9 The example in Motivation (page 8)

The original program written in the syntax of *Osan* is:

```
{(X)
 M1 = { (X1)
   m1 = print 1;
   M2 = { m1 = print 2; m2 = X1.M3.m1; m3 = print 3 };
   m2 = print 4;
   M3 = { m1 = print 5 }
   m3 = print 6 }
 main = X.M1.M2.m3}
```

Execution of the above program succeeds and print “1 4 6 2 5 3” in this order. Below are the reductions.

$\langle$a.1.2.1 | a = $\lfloor$a1$\rfloor$, a1 = {
  $\langle\lfloor$m1$\rfloor$ | m1 = $\langle$b | b = $\lfloor$b1$\rfloor$, b1 = {{print 1}; $\langle\lfloor$m2$\rfloor$ |
    m2 = $\langle$c | c = $\lfloor$c1$\rfloor$, c1 = {{print 2}; {b.1.4.1.1.1.1}; {print 3}}$\rangle$; {print 4}; $\langle\lfloor$m3$\rfloor$ |
      m3 = $\langle$d | d = $\lfloor$d1$\rfloor$, d1 = {{print 5}}$\rangle$; {print 6}}$\rangle$;
  {a.1.1.1.1.2.1.1.3.1}}$\rangle$
$\rightarrow^{*}_{6,2}$
$\langle$a1.2.1 | a = $\lfloor$a1$\rfloor$, a1 = {
  $\langle\lfloor$m1$\rfloor$ | m1 = $\langle$b | b = $\lfloor$b1$\rfloor$, b1 = {{print 1}; $\langle\lfloor$m2$\rfloor$ |
    m2 = $\langle$c | c = $\lfloor$c1$\rfloor$, c1 = {{print 2}; {b.1.4.1.1.1.1}; {print 3}}$\rangle$; {print 4}; $\langle\lfloor$m3$\rfloor$ |
      m3 = $\langle$d | d = $\lfloor$d1$\rfloor$, d1 = {{print 5}}$\rangle$; {print 6}}$\rangle$;
  {a.1.1.1.1.2.1.1.3.1}}$\rangle$
$\rightarrow_{9}$
$\langle$a1.2.1 | a = $\lfloor$a1$\rfloor$, a1 = $\langle${$\lfloor$m1$\rfloor$; {a.1.1.1.1.2.1.1.3.1}} |
  m1 = $\langle$b | b = $\lfloor$b1$\rfloor$, b1 = {{print 1}; $\langle\lfloor$m2$\rfloor$ |
    m2 = $\langle$c | c = $\lfloor$c1$\rfloor$, c1 = {{print 2}; {b.1.4.1.1.1.1}; {print 3}}$\rangle$; {print 4}; $\langle\lfloor$m3$\rfloor$ |
      m3 = $\langle$d | d = $\lfloor$d1$\rfloor$, d1 = {{print 5}}$\rangle$; {print 6}}$\rangle\rangle$
$\rightarrow^{*}_{7,2}$
$\langle$a1.2.1 | a = $\lfloor$a1$\rfloor$, a1 = $\langle${$\lfloor$m1$\rfloor$; {a1.1.1.1.2.1.1.3.1}} |
  m1 = $\langle$b | b = $\lfloor$b1$\rfloor$, b1 = {{print 1}; $\langle\lfloor$m2$\rfloor$ |
    m2 = $\langle$c | c = $\lfloor$c1$\rfloor$, c1 = {{print 2}; {b.1.4.1.1.1.1}; {print 3}}$\rangle$; {print 4}; $\langle\lfloor$m3$\rfloor$ |
      m3 = $\langle$d | d = $\lfloor$d1$\rfloor$, d1 = {{print 5}}$\rangle$; {print 6}}$\rangle\rangle$
$\rightarrow_{12}$
$\langle$a1.2.1 | a = $\lfloor$a1$\rfloor$, a1 = {$\lfloor$m1$\rfloor$; {a1.1.1.1.2.1.1.3.1}},
  m1 = $\langle$b | b = $\lfloor$b1$\rfloor$, b1 = {{print 1}; $\langle\lfloor$m2$\rfloor$ |
    m2 = $\langle$c | c = $\lfloor$c1$\rfloor$, c1 = {{print 2}; {b.1.4.1.1.1.1}; {print 3}}$\rangle$; {print 4}; $\langle\lfloor$m3$\rfloor$ |
      m3 = $\langle$d | d = $\lfloor$d1$\rfloor$, d1 = {{print 5}}$\rangle$; {print 6}}$\rangle$
$\rightarrow^{*}_{8,2,2}$
$\langle$a1.2.1 | a = $\lfloor$a1$\rfloor$, a1 = {$\lfloor$m1$\rfloor$; {m1.1.2.1.1.3.1}},
  m1 = $\langle$b | b = $\lfloor$b1$\rfloor$, b1 = {{print 1}; $\langle\lfloor$m2$\rfloor$ |
    m2 = $\langle$c | c = $\lfloor$c1$\rfloor$, c1 = {{print 2}; {b.1.4.1.1.1.1}; {print 3}}$\rangle$; {print 4}; $\langle\lfloor$m3$\rfloor$ |
      m3 = $\langle$d | d = $\lfloor$d1$\rfloor$, d1 = {{print 5}}$\rangle$; {print 6}}$\rangle$
$\rightarrow^{*}_{6,11}$
$\langle$a1.2.1 | a = $\lfloor$a1$\rfloor$, a1 = {$\lfloor$m1$\rfloor$; {m1.1.2.1.1.3.1}},
  m1 = $\lfloor$b1$\rfloor$, b = $\lfloor$b1$\rfloor$, b1 = {{print 1}; $\langle\lfloor$m2$\rfloor$ |
    m2 = $\langle$c | c = $\lfloor$c1$\rfloor$, c1 = {{print 2}; {b.1.4.1.1.1.1}; {print 3}}$\rangle$; {print 4}; $\langle\lfloor$m3$\rfloor$ |
      m3 = $\langle$d | d = $\lfloor$d1$\rfloor$, d1 = {{print 5}}$\rangle$; {print 6}}$\rangle$
$\rightarrow^{*}_{7,2}$
$\langle$a1.2.1 | a = $\lfloor$a1$\rfloor$, a1 = {$\lfloor$m1$\rfloor$; {b1.2.1.1.3.1}},
  m1 = $\lfloor$b1$\rfloor$, b = $\lfloor$b1$\rfloor$, b1 = {{print 1}; $\langle\lfloor$m2$\rfloor$ |
    m2 = $\langle$c | c = $\lfloor$c1$\rfloor$, c1 = {{print 2}; {b.1.4.1.1.1.1}; {print 3}}$\rangle$; {print 4}; $\langle\lfloor$m3$\rfloor$ |
      m3 = $\langle$d | d = $\lfloor$d1$\rfloor$, d1 = {{print 5}}$\rangle$; {print 6}}$\rangle$

1 is printed.

$\rightarrow^{*}_{3}$
$\langle$a1.2.1 | a = $\lfloor$a1$\rfloor$, a1 = {$\lfloor$m1$\rfloor$; {b1.2.1.1.3.1}},
  m1 = $\lfloor$b1$\rfloor$, b = $\lfloor$b1$\rfloor$, b1 = {$\lfloor$1$\rfloor$; $\langle\lfloor$m2$\rfloor$ |
    m2 = $\langle$c | c = $\lfloor$c1$\rfloor$, c1 = {{print 2}; {b.1.4.1.1.1.1}; {print 3}}$\rangle$; {print 4}; $\langle\lfloor$m3$\rfloor$ |
      m3 = $\langle$d | d = $\lfloor$d1$\rfloor$, d1 = {{print 5}}$\rangle$; {print 6}}$\rangle$
$\rightarrow_{9}$
$\langle$a1.2.1 | a = $\lfloor$a1$\rfloor$, a1 = {$\lfloor$m1$\rfloor$; {b1.2.1.1.3.1}},
  m1 = $\lfloor$b1$\rfloor$, b = $\lfloor$b1$\rfloor$, b1 = $\langle${$\lfloor$1$\rfloor$; $\lfloor$m2$\rfloor$; {print 4}; $\langle\lfloor$m3$\rfloor$ |
    m3 = $\langle$d | d = $\lfloor$d1$\rfloor$, d1 = {{print 5}}$\rangle$; {print 6}} |
      m2 = $\langle$c | c = $\lfloor$c1$\rfloor$, c1 = {{print 2}; {b.1.4.1.1.1.1}; {print 3}}$\rangle\rangle$

4 is printed.

$\rightarrow^*_3$
⟨a1.2.1 | a = ⌊a1⌋, a1 = {⌊m1⌋; {b1.2.1.1.3.1}},
m1 = ⌊b1⌋, b = ⌊b1⌋, b1 = ⟨{⌊1⌋; ⌊m2⌋; ⌊4⌋; ⟨⌊m3⌋ |
m3 = ⟨d | d = ⌊d1⌋, d1 = {{print 5}}⟩⟩; {print 6}} |
m2 = ⟨c | c = ⌊c1⌋, c1 = {{print 2}; {b.1.4.1.1.1.1}; {print 3}}⟩⟩⟩
$\rightarrow_9$
⟨a1.2.1 | a = ⌊a1⌋, a1 = {⌊m1⌋; {b1.2.1.1.3.1}},
m1 = ⌊b1⌋, b = ⌊b1⌋, b1 = ⟨⟨{⌊1⌋; ⌊m2⌋; ⌊4⌋; ⌊m3⌋; {print 6}} |
m2 = ⟨c | c = ⌊c1⌋, c1 = {{print 2}; {b.1.4.1.1.1.1}; {print 3}}⟩⟩ | m3 = ⟨d | d = ⌊d1⌋, d1 = {{print 5}}⟩⟩⟩
6 is printed.
$\rightarrow^*_{3,3,11,11}$
⟨a1.2.1 | a = ⌊a1⌋, a1 = {⌊m1⌋; {b1.2.1.1.3.1}},
m1 = ⌊b1⌋, b = ⌊b1⌋, b1 = ⌊⌊1⌋; ⌊m2⌋; ⌊4⌋; ⌊m3⌋; ⌊6⌋⌋,
m2 = ⟨c | c = ⌊c1⌋, c1 = {{print 2}; {b.1.4.1.1.1.1}; {print 3}}⟩, m3 = ⟨d | d = ⌊d1⌋, d1 = {{print 5}}⟩⟩
$\rightarrow^*_{7,2,2}$
⟨a1.2.1 | a = ⌊a1⌋, a1 = {⌊m1⌋; {m2.1.3.1}},
m1 = ⌊b1⌋, b = ⌊b1⌋, b1 = ⌊⌊1⌋; ⌊m2⌋; ⌊4⌋; ⌊m3⌋; ⌊6⌋⌋,
m2 = ⟨c | c = ⌊c1⌋, c1 = {{print 2}; {b.1.4.1.1.1.1}; {print 3}}⟩, m3 = ⟨d | d = ⌊d1⌋, d1 = {{print 5}}⟩⟩
$\rightarrow^*_{6,11,7,2}$
⟨a1.2.1 | a = ⌊a1⌋, a1 = {⌊m1⌋; {c1.3.1}},
m1 = ⌊b1⌋, b = ⌊b1⌋, b1 = ⌊⌊1⌋; ⌊m2⌋; ⌊4⌋; ⌊m3⌋; ⌊6⌋⌋,
m2 = ⌊c1⌋, c = ⌊c1⌋, c1 = {{print 2}; {b.1.4.1.1.1.1}; {print 3}}, m3 = ⟨d | d = ⌊d1⌋, d1 = {{print 5}}⟩⟩
2 is printed.
$\rightarrow^*_3$
⟨a1.2.1 | a = ⌊a1⌋, a1 = {⌊m1⌋; {c1.3.1}},
m1 = ⌊b1⌋, b = ⌊b1⌋, b1 = ⌊⌊1⌋; ⌊m2⌋; ⌊4⌋; ⌊m3⌋; ⌊6⌋⌋,
m2 = ⌊c1⌋, c = ⌊c1⌋, c1 = {⌊2⌋; {b.1.4.1.1.1.1}; {print 3}}, m3 = ⟨d | d = ⌊d1⌋, d1 = {{print 5}}⟩⟩
$\rightarrow^*_{7,2,7,2,2}$
⟨a1.2.1 | a = ⌊a1⌋, a1 = {⌊m1⌋; {c1.3.1}},
m1 = ⌊b1⌋, b = ⌊b1⌋, b1 = ⌊⌊1⌋; ⌊m2⌋; ⌊4⌋; ⌊m3⌋; ⌊6⌋⌋,
m2 = ⌊c1⌋, c = ⌊c1⌋, c1 = {⌊2⌋; {m3.1.1.1}; {print 3}}, m3 = ⟨d | d = ⌊d1⌋, d1 = {{print 5}}⟩⟩
$\rightarrow^*_{6,11}$
⟨a1.2.1 | a = ⌊a1⌋, a1 = {⌊m1⌋; {c1.3.1}},
m1 = ⌊b1⌋, b = ⌊b1⌋, b1 = ⌊⌊1⌋; ⌊m2⌋; ⌊4⌋; ⌊m3⌋; ⌊6⌋⌋,
m2 = ⌊c1⌋, c = ⌊c1⌋, c1 = {⌊2⌋; {m3.1.1.1}; {print 3}}, m3 = ⌊d1⌋, d = ⌊d1⌋, d1 = {{print 5}}⟩
$\rightarrow^*_{7,2}$
⟨a1.2.1 | a = ⌊a1⌋, a1 = {⌊m1⌋; {c1.3.1}},
m1 = ⌊b1⌋, b = ⌊b1⌋, b1 = ⌊⌊1⌋; ⌊m2⌋; ⌊4⌋; ⌊m3⌋; ⌊6⌋⌋,
m2 = ⌊c1⌋, c = ⌊c1⌋, c1 = {⌊2⌋; {d1.1.1}; {print 3}}, m3 = ⌊d1⌋, d = ⌊d1⌋, d1 = {{print 5}}⟩
5 is printed.
$\rightarrow^*_{3,3,7,2,2,3}$
⟨a1.2.1 | a = ⌊a1⌋, a1 = {⌊m1⌋; {c1.3.1}},
m1 = ⌊b1⌋, b = ⌊b1⌋, b1 = ⌊⌊1⌋; ⌊m2⌋; ⌊4⌋; ⌊m3⌋; ⌊6⌋⌋,
m2 = ⌊c1⌋, c = ⌊c1⌋, c1 = {⌊2⌋; ⌊5⌋; {print 3}}, m3 = ⌊d1⌋, d = ⌊d1⌋, d1 = ⌊⌊5⌋⌋⟩
3 is printed.
$\rightarrow^*_{3,3}$
⟨a1.2.1 | a = ⌊a1⌋, a1 = {⌊m1⌋; {c1.3.1}},
m1 = ⌊b1⌋, b = ⌊b1⌋, b1 = ⌊⌊1⌋; ⌊m2⌋; ⌊4⌋; ⌊m3⌋; ⌊6⌋⌋,
m2 = ⌊c1⌋, c = ⌊c1⌋, c1 = ⌊⌊2⌋; ⌊5⌋; ⌊3⌋⌋, m3 = ⌊d1⌋, d = ⌊d1⌋, d1 = ⌊⌊5⌋⌋⟩
$\rightarrow^*_{7,2,2,3,3}$
⟨a1.2.1 | a = ⌊a1⌋, a1 = ⌊⌊m1⌋; ⌊3⌋⌋,
m1 = ⌊b1⌋, b = ⌊b1⌋, b1 = ⌊⌊1⌋; ⌊m2⌋; ⌊4⌋; ⌊m3⌋; ⌊6⌋⌋,
m2 = ⌊c1⌋, c = ⌊c1⌋, c1 = ⌊⌊2⌋; ⌊5⌋; ⌊3⌋⌋, m3 = ⌊d1⌋, d = ⌊d1⌋, d1 = ⌊⌊5⌋⌋⟩

$\longrightarrow^{*}_{6,2,2}$
⟨3 | a = ⌊a1⌋, a1 = ⌊⌊m1⌋; ⌊3⌋⌋,
m1 = ⌊b1⌋, b = ⌊b1⌋, b1 = ⌊⌊1⌋; ⌊m2⌋; ⌊4⌋; ⌊m3⌋; ⌊6⌋⌋,
m2 = ⌊c1⌋, c = ⌊c1⌋, c1 = ⌊⌊2⌋; ⌊5⌋; ⌊3⌋⌋, m3 = ⌊d1⌋, d = ⌊d1⌋, d1 = ⌊⌊5⌋⌋⟩

7.10 The example for abbreviations (page 12)

The original program written in the syntax of *Osan* is:

{(X) M1 = X.M2; M2 = { m1 = 0; m2 = X.M1.m1 + 1 }; main = X.M1.m2 }

Execution of the above program succeeds. Below are the reductions.

⟨ a.1.3.1 | a = ⌊a1⌋,
a1 = {⟨⌊m1⌋ | m1 = a.1.2.1⟩; ⟨⌊m2⌋ | m2 = ⟨b | b = ⌊b1⌋, b1 = {{0}; {a.1.1.1.1.1.1+1}}⟩⟩; {a.1.1.1.1.2.1}}⟩
$\longrightarrow^{*}_{6,2}$
⟨ a1.3.1 | a = ⌊a1⌋,
a1 = {⟨⌊m1⌋ | m1 = a.1.2.1⟩; ⟨⌊m2⌋ | m2 = ⟨b | b = ⌊b1⌋, b1 = {{0}; {a.1.1.1.1.1.1+1}}⟩⟩; {a.1.1.1.1.2.1}}⟩
$\longrightarrow_{9}$
⟨ a1.3.1 | a = ⌊a1⌋,
a1 = ⟨{⌊m1⌋; ⟨⌊m2⌋ | m2 = ⟨b | b = ⌊b1⌋, b1 = {{0}; {a.1.1.1.1.1.1+1}}⟩⟩; {a.1.1.1.1.2.1}} | m1 = a.1.2.1⟩⟩
$\longrightarrow_{9}$
⟨ a1.3.1 | a = ⌊a1⌋,
a1 = ⟨⟨{⌊m1⌋; ⌊m2⌋; {a.1.1.1.1.2.1}} | m2 = ⟨b | b = ⌊b1⌋, b1 = {{0}; {a.1.1.1.1.1.1+1}}⟩⟩ | m1 = a.1.2.1⟩⟩
$\longrightarrow^{*}_{7,2}$
⟨ a1.3.1 | a = ⌊a1⌋,
a1 = ⟨⟨{⌊m1⌋; ⌊m2⌋; {a1.1.1.1.2.1}} | m2 = ⟨b | b = ⌊b1⌋, b1 = {{0}; {a.1.1.1.1.1.1+1}}⟩⟩ | m1 = a.1.2.1⟩⟩
$\longrightarrow_{12}$
⟨ a1.3.1 | a = ⌊a1⌋,
a1 = ⟨{⌊m1⌋; ⌊m2⌋; {a1.1.1.1.2.1}} | m2 = ⟨b | b = ⌊b1⌋, b1 = {{0}; {a.1.1.1.1.1.1+1}}⟩⟩, m1 = a.1.2.1⟩
$\longrightarrow_{12}$
⟨ a1.3.1 | a = ⌊a1⌋,
a1 = {⌊m1⌋; ⌊m2⌋; {a1.1.1.1.2.1}}, m2 = ⟨b | b = ⌊b1⌋, b1 = {{0}; {a.1.1.1.1.1.1+1}}⟩, m1 = a.1.2.1⟩
$\longrightarrow_{8}$
⟨ a1.3.1 | a = ⌊a1⌋,
a1 = {⌊m1⌋; ⌊m2⌋; {⌊⌊m1⌋;⌊m2⌋⌋.1.1.1.2.1}}, m2 = ⟨b | b = ⌊b1⌋, b1 = {{0}; {a.1.1.1.1.1.1+1}}⟩, m1 = a.1.2.1⟩
$\longrightarrow^{*}_{2,2}$
⟨ a1.3.1 | a = ⌊a1⌋,
a1 = {⌊m1⌋; ⌊m2⌋; {m1.1.2.1}}, m2 = ⟨b | b = ⌊b1⌋, b1 = {{0}; {a.1.1.1.1.1.1+1}}⟩, m1 = a.1.2.1⟩
$\longrightarrow^{*}_{7,2,8,2,2}$
⟨ a1.3.1 | a = ⌊a1⌋,
a1 = {⌊m1⌋; ⌊m2⌋; {m1.1.2.1}}, m2 = ⟨b | b = ⌊b1⌋, b1 = {{0}; {a.1.1.1.1.1.1+1}}⟩, m1 = m2⟩
$\longrightarrow_{6}$
⟨ a1.3.1 | a = ⌊a1⌋,
a1 = {⌊m1⌋; ⌊m2⌋; {m1.1.2.1}}, m2 = ⟨⌊b1⌋ | b = ⌊b1⌋, b1 = {{0}; {a.1.1.1.1.1.1+1}}⟩, m1 = m2⟩
$\longrightarrow_{11}$
⟨ a1.3.1 | a = ⌊a1⌋,
a1 = {⌊m1⌋; ⌊m2⌋; {m1.1.2.1}}, m2 = ⌊b1⌋, b = ⌊b1⌋, b1 = {{0}; {a.1.1.1.1.1.1+1}}, m1 = m2⟩
$\longrightarrow_{7}$
⟨ a1.3.1 | a = ⌊a1⌋,
a1 = {⌊m1⌋; ⌊m2⌋; {m1.1.2.1}}, m2 = ⌊b1⌋, b = ⌊b1⌋, b1 = {{0}; {a.1.1.1.1.1.1+1}}, m1 = ⌊b1⌋⟩
$\longrightarrow^{*}_{7,2}$
⟨ a1.3.1 | a = ⌊a1⌋,
a1 = {⌊m1⌋; ⌊m2⌋; {b1.2.1}}, m2 = ⌊b1⌋, b = ⌊b1⌋, b1 = {{0}; {a.1.1.1.1.1.1+1}}, m1 = ⌊b1⌋⟩
$\longrightarrow_{3}$
⟨ a1.3.1 | a = ⌊a1⌋,
a1 = {⌊m1⌋; ⌊m2⌋; {b1.2.1}}, m2 = ⌊b1⌋, b = ⌊b1⌋, b1 = {⌊0⌋; {a.1.1.1.1.1.1+1}}, m1 = ⌊b1⌋⟩

$\longrightarrow^{*}_{7,2,8,2,2}$
$\langle$ a1.3.1 | a = $\lfloor$a1$\rfloor$,
  a1 = {$\lfloor$m1$\rfloor$; $\lfloor$m2$\rfloor$; {b1.2.1}}, m2 = $\lfloor$b1$\rfloor$, b = $\lfloor$b1$\rfloor$, b1 = {$\lfloor$0$\rfloor$; {m1.1.1.1+1}}, m1 = $\lfloor$b1$\rfloor$$\rangle$
$\longrightarrow^{*}_{7,2}$
$\langle$ a1.3.1 | a = $\lfloor$a1$\rfloor$,
  a1 = {$\lfloor$m1$\rfloor$; $\lfloor$m2$\rfloor$; {b1.2.1}}, m2 = $\lfloor$b1$\rfloor$, b = $\lfloor$b1$\rfloor$, b1 = {$\lfloor$0$\rfloor$; {b1.1.1+1}}, m1 = $\lfloor$b1$\rfloor$$\rangle$
$\longrightarrow_{8}$
$\langle$ a1.3.1 | a = $\lfloor$a1$\rfloor$,
  a1 = {$\lfloor$m1$\rfloor$; $\lfloor$m2$\rfloor$; {b1.2.1}}, m2 = $\lfloor$b1$\rfloor$, b = $\lfloor$b1$\rfloor$, b1 = {$\lfloor$0$\rfloor$; {$\lfloor$0$\rfloor$.1+1}}, m1 = $\lfloor$b1$\rfloor$$\rangle$
$\longrightarrow^{*}_{2}$
$\langle$ a1.3.1 | a = $\lfloor$a1$\rfloor$,
  a1 = {$\lfloor$m1$\rfloor$; $\lfloor$m2$\rfloor$; {b1.2.1}}, m2 = $\lfloor$b1$\rfloor$, b = $\lfloor$b1$\rfloor$, b1 = {$\lfloor$0$\rfloor$; {1}}, m1 = $\lfloor$b1$\rfloor$$\rangle$
$\longrightarrow_{3}$
$\langle$ a1.3.1 | a = $\lfloor$a1$\rfloor$,
  a1 = {$\lfloor$m1$\rfloor$; $\lfloor$m2$\rfloor$; {b1.2.1}}, m2 = $\lfloor$b1$\rfloor$, b = $\lfloor$b1$\rfloor$, b1 = {$\lfloor$0$\rfloor$; $\lfloor$1$\rfloor$}, m1 = $\lfloor$b1$\rfloor$$\rangle$
$\longrightarrow_{3}$
$\langle$ a1.3.1 | a = $\lfloor$a1$\rfloor$,
  a1 = {$\lfloor$m1$\rfloor$; $\lfloor$m2$\rfloor$; {b1.2.1}}, m2 = $\lfloor$b1$\rfloor$, b = $\lfloor$b1$\rfloor$, b1 = $\lfloor\lfloor$0$\rfloor$; $\lfloor$1$\rfloor\rfloor$, m1 = $\lfloor$b1$\rfloor$$\rangle$
$\longrightarrow^{*}_{7,2,2}$
$\langle$ a1.3.1 | a = $\lfloor$a1$\rfloor$,
  a1 = {$\lfloor$m1$\rfloor$; $\lfloor$m2$\rfloor$; {1}}, m2 = $\lfloor$b1$\rfloor$, b = $\lfloor$b1$\rfloor$, b1 = $\lfloor\lfloor$0$\rfloor$; $\lfloor$1$\rfloor\rfloor$, m1 = $\lfloor$b1$\rfloor$$\rangle$
$\longrightarrow_{3}$
$\langle$ a1.3.1 | a = $\lfloor$a1$\rfloor$,
  a1 = {$\lfloor$m1$\rfloor$; $\lfloor$m2$\rfloor$; $\lfloor$1$\rfloor$}, m2 = $\lfloor$b1$\rfloor$, b = $\lfloor$b1$\rfloor$, b1 = $\lfloor\lfloor$0$\rfloor$; $\lfloor$1$\rfloor\rfloor$, m1 = $\lfloor$b1$\rfloor$$\rangle$
$\longrightarrow_{3}$
$\langle$ a1.3.1 | a = $\lfloor$a1$\rfloor$,
  a1 = $\lfloor\lfloor$m1$\rfloor$; $\lfloor$m2$\rfloor$; $\lfloor$1$\rfloor\rfloor$, m2 = $\lfloor$b1$\rfloor$, b = $\lfloor$b1$\rfloor$, b1 = $\lfloor\lfloor$0$\rfloor$; $\lfloor$1$\rfloor\rfloor$, m1 = $\lfloor$b1$\rfloor$$\rangle$
$\longrightarrow^{*}_{6,2,2}$
$\langle$ 1 | a = $\lfloor$a1$\rfloor$,
  a1 = $\lfloor\lfloor$m1$\rfloor$; $\lfloor$m2$\rfloor$; $\lfloor$1$\rfloor\rfloor$, m2 = $\lfloor$b1$\rfloor$, b = $\lfloor$b1$\rfloor$, b1 = $\lfloor\lfloor$0$\rfloor$; $\lfloor$1$\rfloor\rfloor$, m1 = $\lfloor$b1$\rfloor$$\rangle$