# A Practical Linear Time Algorithm for Trivial Automata Model Checking of Higher-Order Recursion Schemes

Naoki Kobayashi

Tohoku University

**Abstract.** The model checking of higher-order recursion schemes has been actively studied and is now becoming a basis of higher-order program verification. We propose a new algorithm for trivial automata model checking of higher-order recursion schemes. To our knowledge, this is the first practical model checking algorithm for recursion schemes that runs in time linear in the size of the higher-order recursion scheme, under the assumption that the size of trivial automata and the largest order and arity of functions are fixed. The previous linear time algorithm was impractical due to a huge constant factor, and the only practical previous algorithm suffers from the hyper-exponential worst-case time complexity, under the same assumption. The new algorithm is remarkably simple, consisting of just two fixed-point computations. We have implemented the algorithm and confirmed that it outperforms Kobayashi's previous algorithm in a certain case.

## 1 Introduction

The model checking of higher-order recursion schemes [9, 19] (higher-order model checking, for short) has been studied extensively, and recently applied to higher-order program verification [12, 10, 17, 16, 20]. A higher-order recursion scheme [9, 19] is a grammar for describing a possibly infinite tree, and the higher-order model checking is concerned about whether the tree described by a given higher-order recursion scheme satisfies a given property (typically expressed by the modal $\mu$-calculus or tree automata). Higher-order model checking can be considered a generalization of finite-state and pushdown model checking. Just as software model checkers for procedural languages like BLAST [4] and SLAM [3] have been constructed based on finite-state and pushdown model checking, one may hope to construct software model checkers for higher-order functional languages based on higher-order model checking. Some evidence for such a possibility has been provided recently [12, 10, 17, 16, 20].

The main obstacle in applying higher-order model checking to program verification is its extremely high worst-case complexity: $n$-EXPTIME completeness [19] (where $n$ is the order of a given higher-order recursion scheme). Kobayashi and Ong [12, 15] showed that, under the assumption that the size of properties and the largest order and arity of functions are fixed, the model checking is actually linear time in the size of the higher-order recursion scheme for the class of properties described by trivial automata [2], and polynomial time for the full modal $\mu$-calculus. Their algorithms were however of only theoretical interest; because of a huge constant factor (which is $n$-fold exponential in the other parameters), they are runnable only for recursion schemes of order 2 at highest.

The only practical algorithm known to date is Kobayashi's hybrid algorithm [10], used in the first higher-order model checker TRecS [11]. According to experiments, the algorithm runs remarkably fast in practice, considering the worst-case complexity of the problem. The worst-case complexity of the hybrid algorithm is, however, actually worse than Kobayashi's naïve algorithm [12]: Under the same assumption that the other parameters are fixed, the worst-case time complexity of the hybrid algorithm [10] is still hyper-exponential in the size of the recursion scheme. In fact, one can easily construct a higher-order recursion scheme for which the hybrid algorithm suffers from an $n$-EXPTIME bottleneck in the size of the recursion scheme. Thus, it remained as a question whether there is a practical algorithm that runs in time polynomial in the size of the higher-order recursion scheme. The question is highly relevant for applications to program verification [12, 17], as the size of a higher-order recursion scheme corresponds to the size of a program.

The present paper proposes the first (arguably) *practical*, *linear time*[1] algorithm for trivial automata model checking of recursion schemes (i.e. the problem of deciding whether the tree generated by a given recursion scheme $\mathcal{G}$ is accepted by a given trivial automaton $\mathcal{B}$). Like Kobayashi and Ong's previous algorithms [12, 10, 15], the new algorithm is based on a reduction of model checking to intersection type inference, but the algorithm has also been inspired by game semantics [19, 1] (though the game semantics is not explicitly used). The resulting algorithm is remarkably simple, consisting of just two fixedpoint computations. We have implemented the new algorithm, and confirmed that it outperforms Kobayashi's hybrid algorithm [10] in a certain case. Another advantage of the new algorithm is that it works for *non-deterministic* trivial automata, unlike the hybrid algorithm (which works only for deterministic trivial automata).

Unfortunately, the current implementation of the new algorithm is significantly slower than the hybrid algorithm [10] (which already incorporates a number of optimizations) in most cases. However, the new algorithm provides a hope that, with further optimizations, one may eventually obtain a model checking algorithm that scales to large programs (because of the fixed-parameter linear time complexity).

The rest of this paper is structured as follows. Section 2 reviews higher-order recursion schemes and previous type-based model checking algorithms for recursion schemes. Section 3 presents a new model checking algorithm, and Section 4 proves the correctness of the algorithm. Section 5 reports preliminary experiments. Section 6 discusses related work and Section 7 concludes.

[1] Under the same assumption as [12, 15] that the other parameters are fixed.

## 2 Preliminaries

We write $\mathit{dom}(f)$ for the domain of a map $f$. We write $\widetilde{x}$ for a sequence $x_1, \ldots, x_k$, and write $[t_1/x_1, \ldots, t_k/x_k]u$ for the term obtained from $u$ by replacing $x_1, \ldots, x_k$ in $u$ with $t_1, \ldots, t_k$. A $\Sigma$-labeled tree (where $\Sigma$ is a set of symbols), written $T$, is a map from $[m]^*$ (where $m$ is a positive integer and $[m] = \{1, \ldots, m\}$) to $\Sigma$ such that (i) $\epsilon \in \mathit{dom}(T)$, (ii) $xi \in \mathit{dom}(T)$ implies $\{x, x1, \ldots, x(i-1)\} \subseteq \mathit{dom}(T)$ for any $x \in [m]^*$ and $i \in [m]$.

*Higher-Order Recursion Schemes.* A higher-order recursion scheme [19] is a tree grammar for generating an infinite tree, where non-terminal symbols can take parameters. To preclude illegal parameters, each non-terminal symbol has a sort. The set of *sorts* is given by: $\kappa ::= \mathtt{o} \mid \kappa_1 \to \kappa_2$. Intuitively, the sort $\mathtt{o}$ describes trees, and the sort $\kappa_1 \to \kappa_2$ describes functions that take an element of sort $\kappa_1$ as input, and return an element of sort $\kappa_2$. The arity and order are defined by:

$$\begin{array}{ll} \mathit{arity}(\mathtt{o}) = 0 & \mathit{arity}(\kappa_1 \to \kappa_2) = 1 + \mathit{arity}(\kappa_2) \\ \mathit{order}(\mathtt{o}) = 0 & \mathit{order}(\kappa_1 \to \kappa_2) = \mathit{max}(1 + \mathit{order}(\kappa_1), \kappa_2) \end{array}$$

Formally, a *higher-order recursion scheme* (recursion scheme, for short) is a quadruple $\mathcal{G} = (\Sigma, \mathcal{N}, \mathcal{R}, S)$, where $\Sigma$ is a set of symbols called *terminals* and $\mathcal{N}$ is a set of symbols called *non-terminals*. Each terminal or non-terminal $\alpha$ has an associated sort, denoted by $\mathit{sort}(\alpha)$. The order of the sort of a terminal must be 0 or 1. We write $\mathit{arity}(\alpha)$ for $\mathit{arity}(\mathit{sort}(\alpha))$ and call it the arity of $\alpha$. (Thus, $\Sigma$ is a ranked alphabet.) $\mathcal{R}$, called *rewriting rules*, is a finite map from $\mathcal{N}$ to $\lambda$-terms of the form $\lambda\widetilde{x}.t$ where $t$ is an applicative term constructed from the variables $\widetilde{x}$, terminals, and non-terminals. $\mathcal{R}(F)$ must have sort $\mathit{sort}(F)$ (under the standard simple type system). $S$ is a special non-terminal called the *start symbol*. The *order* of a recursion scheme $\mathcal{G}$ is the largest order of the sorts of its non-terminals. We use lower letters for terminals, and upper letters for non-terminals.

Given a recursion scheme $\mathcal{G}$, the rewriting relation $\longrightarrow_{\mathcal{G}}$ is the least relation that satisfies: (i) $F\,u_1 \ldots, u_k \longrightarrow_{\mathcal{G}} [u_1/x_1, \ldots, u_k/x_k]t$ if $\mathcal{R}(F) = \lambda x_1. \cdots \lambda x_k.t$, and (ii) $a\,t_1 \cdots t_n \longrightarrow_{\mathcal{G}} a\,t_1 \cdots t_{i-1}\,t'_i\,t_{i+1} \cdots t_n$ if $t_i \longrightarrow_{\mathcal{G}} t'_i$.[2] The *value tree* of $\mathcal{G}$, written $[\![\mathcal{G}]\!]$, is the (possibly infinite) $(\Sigma \cup \{\bot\})$-labelled tree generated by a fair, maximal reduction sequence from $S$. More precisely, define $t^{\bot}$ by $(a\,t_1 \cdots t_k)^{\bot} = a\,t_1{}^{\bot} \cdots t_k{}^{\bot}$, and $(F\,t_1 \cdots t_k)^{\bot} = \bot$. $[\![\mathcal{G}]\!]$ is $\bigsqcup\{t^{\bot} \mid S \longrightarrow^*_{\mathcal{G}} t\}$, where $\bigsqcup$ is the least upper bound with respect to the least compatible (i.e. closed under contexts) relation $\sqsubseteq$ on trees that satisfies $\bot \sqsubseteq T$ for every tree $T$.

*Example 1.* Consider the recursion scheme $\mathcal{G}_0 = (\{\mathtt{a}, \mathtt{b}, \mathtt{c}\}, \{S, F\}, \mathcal{R}, S)$, where $\mathcal{R}$ and the sorts of symbols are given by:

$$\begin{array}{l} \mathcal{R} = \{S \mapsto F\,\mathtt{b}\,\mathtt{c}, \quad F \mapsto \lambda f.\lambda x.(\mathtt{a}\,(f\,x)\,(F\,f\,(f\,x)))\} \\ \mathtt{a} : \mathtt{o} \to \mathtt{o} \to \mathtt{o},\ \mathtt{b} : \mathtt{o} \to \mathtt{o},\ \mathtt{c} : \mathtt{o},\ S : \mathtt{o},\ F : (\mathtt{o} \to \mathtt{o}) \to \mathtt{o} \to \mathtt{o} \end{array}$$

[2] Note that we allow only reductions of outermost redexes.

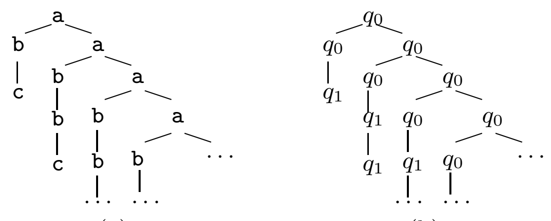

**Fig. 1.** The tree generated by the recursion scheme $\mathcal{G}_0$ and a run tree of it

$S$ is rewritten as follows, and the tree in Figure 1(a) is generated.

$$S \longrightarrow F\,\mathtt{b}\,\mathtt{c} \to \mathtt{a}\,(\mathtt{b}\,\mathtt{c})\,(F\,\mathtt{b}\,(\mathtt{b}\,\mathtt{c})) \to \mathtt{a}\,(\mathtt{b}\,\mathtt{c})\,(\mathtt{a}\,(\mathtt{b}(\mathtt{b}\,\mathtt{c}))\,(F\;\mathtt{b}\;(\mathtt{b}\,(\mathtt{b}\,\mathtt{c})))) \to \cdots$$

*Trivial Automata Model Checking.* The aim of model-checking a higher-order recursion scheme is to check whether the tree generated by the recursion scheme satisfies a certain regular property. In the present paper, we consider the properties described by *trivial automata* [2], which are sufficient for program verification problems considered in [12, 17].

A *trivial automaton* $\mathcal{B}$ is a quadruple $(\Sigma, Q, \Delta, q_0)$, where $\Sigma$ is a set of input symbols, $Q$ is a finite set of states, $\Delta \subseteq Q \times \Sigma \times Q^*$ is a transition function, and $q_0$ is the initial state. A $\Sigma$-labeled tree $T$ is *accepted* by $\mathcal{B}$ if there is a $Q$-labeled tree $R$ (called a *run tree*) such that: (i) $\mathit{dom}(T) = \mathit{dom}(R)$; (ii) $R(\epsilon) = q_0$; and (iii) for every $x \in \mathit{dom}(R)$, $(R(x), T(x), R(x1) \cdots R(xm)) \in \Delta$ where $m = \mathit{arity}(T(x))$. For a trivial automaton $\mathcal{B} = (\Sigma, Q, \Delta, q_0)$ (with $\perp \notin \Sigma$), we write $\mathcal{B}^{\perp}$ for the trivial automaton $(\Sigma \cup \{\perp\}, Q, \Delta \cup \{(q, \perp, \epsilon) \mid q \in Q\}, q_0)$.

The *trivial automata model checking* is the problem of deciding whether $[\![\mathcal{G}]\!]$ is accepted by $\mathcal{B}^{\perp}$, given a recursion scheme $\mathcal{G}$ and a trivial automaton $\mathcal{B}$.[3]

*Example 2.* Consider the trivial automaton $\mathcal{B}_0 = (\{\mathtt{a}, \mathtt{b}, \mathtt{c}\}, \{q_0, q_1\}, \Delta, q_0)$, where $\Delta = \{(q_0, \mathtt{a}, q_0q_0), (q_0, \mathtt{b}, q_1), (q_1, \mathtt{b}, q_1), (q_0, \mathtt{c}, \epsilon), (q_1, \mathtt{c}, \epsilon)\}$. $\mathcal{B}_0$ accepts a $\{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$-labeled ranked tree just if $\mathtt{a}$ does not occur below $\mathtt{b}$. The tree generated by $\mathcal{G}_0$ of Example 1 is accepted by $\mathcal{B}_0$. The run tree is shown in Figure 1(b).

*Type Systems Equivalent to Trivial Automata Model Checking* One can construct an intersection type system (parameterized by a trivial automaton $\mathcal{B} = (\Sigma, Q, \Delta, q_0)$) that is equivalent to trivial automata model checking, in the sense that a recursion scheme $\mathcal{G}$ is well typed if, and only if, $[\![\mathcal{G}]\!]$ is accepted by $\mathcal{B}^{\perp}$ [12]. Define the set of types by:

$$\theta \text{ (atomic types) } ::= q \mid \tau \to \theta \qquad \tau \text{ (intersections) } ::= \bigwedge\{\theta_1, \ldots, \theta_m\}$$

[3] In the literature [19, 15], it is often assumed that the value tree of $\mathcal{G}$ does not contain $\perp$. Under that assumption, the acceptance by $\mathcal{B}^{\perp}$ and $\mathcal{B}$ are equivalent.

Here, $q$ ranges over the states of $\mathcal{B}$. We often write $\theta_1 \wedge \cdots \wedge \theta_m$ or $\bigwedge_{i\in\{1,\ldots,m\}} \theta_i$ for $\bigwedge\{\theta_1,\ldots,\theta_m\}$. We also write $\top$ for $\bigwedge\emptyset$, and $\theta$ for $\bigwedge\{\theta\}$. $\bigwedge$ binds tighter than $\to$. Intuitively, $q$ is a refinement of sort $\mathsf{o}$, describing trees accepted by $\mathcal{B}$ with the initial state replaced by $q$. $\theta_1 \wedge \cdots \wedge \theta_m \to \theta$ describes functions that take an element that has types $\theta_1,\ldots,\theta_m$ as input, and return an element of type $\theta$. For example, $q_0 \wedge q_1 \to q_0$ describes a function that takes a tree that can be accepted from both $q_0$ and $q_1$, and returns a tree accepted from $q_0$. We define the refinement relation $\theta :: \kappa$ inductively by: (i) $q :: \mathsf{o}$ for every $q \in Q$ and (ii) $(\bigwedge_{i\in S} \theta_i \to \theta) :: (\kappa_1 \to \kappa_2)$ if $\forall i \in S.\theta_i :: \kappa_1$ and $\theta :: \kappa_2$.

The typing rules for terms and rewriting rules are given as follows.

$$\frac{(q,a,q_1\cdots q_k)\in\Delta}{\Gamma\vdash^{\mathcal{B}} a : q_1 \to \cdots q_k \to q} \qquad \frac{x:\theta\in\Gamma}{\Gamma\vdash^{\mathcal{B}} x:\theta} \qquad \frac{\forall i\in S.(\Gamma\vdash^{\mathcal{B}} t:\theta_i)}{\Gamma\vdash^{\mathcal{B}} t : \bigwedge_{i\in S}\theta_i}$$

$$\frac{\Gamma\vdash^{\mathcal{B}} t_1 : \tau\to\theta \quad \Gamma\vdash^{\mathcal{B}} t_2 : \tau}{\Gamma\vdash^{\mathcal{B}} t_1t_2 : \theta} \qquad \frac{\Gamma, x:\theta_1,\ldots,x:\theta_m \vdash^{\mathcal{B}} t:\theta \quad x\notin dom(\Gamma)}{\Gamma\vdash^{\mathcal{B}} \lambda x.t : \theta_1\wedge\cdots\wedge\theta_m\to\theta} \qquad \frac{dom(\Gamma)\subseteq dom(\mathcal{R}) \quad \forall(F:\theta)\in\Gamma.(\Gamma\vdash^{\mathcal{B}} \mathcal{R}(F):\theta)}{\vdash^{\mathcal{B}} \mathcal{R}:\Gamma}$$

Here, $\Gamma$ is a set of bindings of the form $x:\theta$ where non-terminals are also treated as variables, and $\Gamma$ may contain more than one binding for each variable. We write $dom(\Gamma)$ for $\{x \mid x:\theta\in S\}$. A recursion scheme $\mathcal{G} = (\Sigma,\mathcal{N},\mathcal{R},S)$ is well typed under $\Gamma$, written $\vdash^{\mathcal{B}} \mathcal{G} : \Gamma$, if $\vdash^{\mathcal{B}} \mathcal{R} : \Gamma$, $\forall(F:\theta)\in\Gamma.(\theta :: sort(F))$, and $S:q_0\in\Gamma$. We write $\vdash^{\mathcal{B}} \mathcal{G}$ if $\vdash^{\mathcal{B}} \mathcal{G} : \Gamma$ for some $\Gamma$.

The following theorem guarantees the correspondence between model checking and type checking.

**Theorem 1 (Kobayashi [12]).** *$[\![\mathcal{G}]\!]$ is accepted by $\mathcal{B}^{\perp}$ if and only if $\vdash^{\mathcal{B}} \mathcal{G}$.*

*Example 3.* Recall the recursion scheme $\mathcal{G}_0$ in Example 1 and the trivial automaton $\mathcal{B}_0$ in Example 2. $\vdash^{\mathcal{B}_0} \mathcal{G}_0 : \Gamma$ holds for $\Gamma = \{S:q_0, F:(q_1\to q_1)\wedge(q_1\to q_0)\to q_1\to q_0\}$

Theorem 1 above yields a straightforward, fixedpoint-based model checking algorithm. Let $\mathbf{Shrink}_{\mathcal{G},\mathcal{B}}$ be the function on type environments defined by: $\mathbf{Shrink}_{\mathcal{G},\mathcal{B}}(\Gamma) = \{F:\theta\in\Gamma \mid \Gamma\vdash^{\mathcal{B}} \mathcal{R}(F):\theta\}$, and let $\Gamma_{\mathbf{max}}$ be $\{F:\theta \mid F\in\mathcal{N}, \theta :: sort(F)\}$. Then, by the definition, $\vdash^{\mathcal{B}} \mathcal{G} : \Gamma$ if and only if there exists $\Gamma\subseteq\Gamma_{\mathbf{max}}$ such that $\Gamma\subseteq\mathbf{Shrink}_{\mathcal{G},\mathcal{B}}(\Gamma)$ and $S:q_0\in\Gamma$. (Note that $\Gamma\subseteq\mathbf{Shrink}_{\mathcal{G},\mathcal{B}}(\Gamma)$ if and only if $\vdash^{\mathcal{B}} \mathcal{R}:\Gamma$.) Thus, to check whether $\vdash^{\mathcal{B}} \mathcal{G}$ holds, it is sufficient to compute the greatest fixedpoint $\Gamma_{\mathbf{gfp}}$ of $\mathbf{Shrink}_{\mathcal{G},\mathcal{B}}$ and checks whether $S:q_0\in\Gamma_{\mathbf{gfp}}$. This is Kobayashi's naïve algorithm.

```
NAIVE ALGORITHM [12]:
1. $\Gamma$ := $\Gamma_{\mathbf{max}}$;
2. Repeat $\Gamma$ := $\mathbf{Shrink}_{\mathcal{G},\mathcal{B}}(\Gamma)$ until $\Gamma = \mathbf{Shrink}_{\mathcal{G},\mathcal{B}}(\Gamma)$;
3. Output whether $S:q_0\in\Gamma$.
```

Suppose that the size of $\mathcal{B}$ and the largest size of sorts of symbols are fixed. Then, the size of $\Gamma_{\mathbf{max}}$ is linear in the size of $\mathcal{G}$, since for a given $\kappa$, the number of types that satisfy $\theta :: \kappa$ is bounded above by a constant. Thus, using Rehof and Mogensen's optimization [21], the above algorithm is made linear in the size of $\mathcal{G}$. The algorithm does not work in practice, however, as the constant factor is too large. Even at the first iteration of the fixedpoint computation, we need to pick each binding $F:\theta$ from $\Gamma_{\mathbf{max}}$ and check whether $\Gamma_{\mathbf{max}} \vdash^{\mathcal{B}} \mathcal{R}(F) : \theta$ holds. This is impractical, as $\Gamma_{\mathbf{max}}$ is too large; In fact, even when $|Q| = 2$, for a symbol of sort $((\mathtt{o} \to \mathtt{o}) \to \mathtt{o}) \to \mathtt{o}$, the number of corresponding types is $2^{513} \approx 10^{154}$.

Kobayashi's hybrid algorithm [10] starts the greatest fixedpoint computation from a type environment much smaller than $\Gamma_{\mathbf{max}}$. To find an appropriate start-point of the fixedpoint computation, his algorithm reduces the recursion scheme a finite number of times, and infers candidates of the types of each non-terminal, by observing how each non-terminal is used in the reduction sequence. The following is an outline of the hybrid algorithm (see [10] for more details):

```
HYBRID ALGORITHM [10]:
1. Reduce $S$ a finite number of steps;
2. If a property violation is found, output 'no' and halt;
3. $\Gamma$ := type bindings extracted from the reduction sequence;
4. Repeat $\Gamma$ := $\mathbf{Shrink}_{\mathcal{G},\mathcal{B}}(\Gamma)$ until $\Gamma = \mathbf{Shrink}_{\mathcal{G},\mathcal{B}}(\Gamma)$;
5. If $S : q_0 \in \Gamma$ then output 'yes' and halt;
6. Go back to 1 and reduce $S$ further.
```

The algorithm works well in practice [10, 17], but has some limitations: (i) No theoretical guarantee that the algorithm is efficient. In fact, the worst-case running time is hyper-exponential [13]: see Section 5. (ii) The efficiency of the algorithm crucially depends on the selection of terms to be reduced in Step 1. The implementation relies on heuristics for choosing reduced terms, and there is no theoretical justification for it. (iii) It works only for *deterministic* trivial automata (i.e. trivial automata such that $|\Delta \cap \{q\} \times \{a\} \times Q^*| \leq 1$ for every $q \in Q, a \in \Sigma$.) Though it is possible to extend the algorithm to remove the restriction, it is unclear whether the resulting algorithm is efficient in practice.

The limitations above motivated us to look for yet another algorithm, which is efficient *both in practice and in theory* (where an important criterion for the latter is that the time complexity should be linear in the size of the recursion scheme, under the assumption that the other parameters are fixed). That is the subject of this paper, discussed in the following sections.

## 3 The New Model Checking Algorithm

### 3.1 Main Idea

In the previous section, a reader may have wondered why we do not compute the *least* fixedpoint, instead of the *greatest* one. The naïve least fixedpoint computation however does not work. Let us define $\mathcal{F}_{\mathcal{B}}$ by: $\mathcal{F}_{\mathcal{B}}(\Gamma) = \{F : \theta \mid \Gamma \vdash^{\mathcal{B}} \mathcal{R}(F) : \theta\}$. How about computing the least fixedpoint $\Gamma_{\mathbf{lfp}}$ of $\mathcal{F}_{\mathcal{B}}$, and checking whether $S : q_0 \in \Gamma_{\mathbf{lfp}}$? This does not work for two reasons. First of all, $S : q_0 \in \Gamma_{\mathbf{lfp}}$ is not a necessary condition for the well-typedness of $\mathcal{G}$. For example, for $\mathcal{G}_0$ of Example 1, $\Gamma_{\mathbf{lfp}} = \emptyset$. Secondly, for each iteration to compute $\mathcal{F}_{\mathcal{B}}(\emptyset), \mathcal{F}^2_{\mathcal{B}}(\emptyset), \mathcal{F}^3_{\mathcal{B}}(\emptyset), \ldots$, we have to *guess* a type $\theta$ of $F$ and check whether $\Gamma \vdash^{\mathcal{B}} \mathcal{R}(F) : \theta$. The possible types of $F$ are however too many, hence the same problem as the greatest fixedpoint computation.

The discussion above however suggests that a least fixedpoint computation may work if, in $\Gamma \vdash^{\mathcal{B}} \mathcal{R}(F) : \theta$, (i) we relax the condition on $\Gamma$ (that $\Gamma$ must have been obtained from the previous iteration steps), and (ii) we impose a restriction on $\theta$, to disallow $\theta$ to be synthesized "out of thin air". This observation motivates us to modify $\mathcal{F}_{\mathcal{B}}(\Gamma)$ as follows:

$$\mathcal{F}'_{\mathcal{B}}(\Gamma) = \bigcup \{\{F : \theta'\} \cup \Gamma' \mid \Gamma' \vdash^{\mathcal{B}} \mathcal{R}(F) : \theta', \Gamma \preceq_O \Gamma', \theta \preceq_P \theta', (F : \theta) \in \Gamma\}.$$

Here, $\Gamma \preceq_O \Gamma'$ (which will be defined later) indicates that $\Gamma'$ is not identical, but somehow similar to $\Gamma$. The condition "$\theta \preceq_P \theta'$ for some $F : \theta \in \Gamma$" also indicates that $F : \theta'$ is somehow similar to an existing type binding $F : \theta$ of $\Gamma$.

To see how $\preceq_O$ and $\preceq_P$ may be defined, let us consider $\mathcal{G}_0$ and $\mathcal{B}_0$ of Examples 1 and 2. In order for $[\![\mathcal{G}_0]\!]$ to be accepted by ${\mathcal{B}_0}^{\perp}$, $S$ should have type $q_0$. So, let us first put $S : q_0$ into the initial type environment: $\Gamma_0 := \{S : q_0\}$.
Now, in order for the body $F\,\mathtt{b}\,\mathtt{c}$ of $S$ to have type $q_0$, $F$ must have a type of the form $\cdots \to \cdots \to q_0$. So, let us put $F : \top \to \top \to q_0$ (note that $\top \to \top \to q_0$ is a subtype of any type of the form $\tau_1 \to \tau_2 \to q_0$ with respect to the standard subtype relation; see Appendix A): $\Gamma_1 := \{S : q_0, F : \top \to \top \to q_0\}$.

Let us now look at the definition of $F$, to check whether the body of $F$ has type $\top \to \top \to q_0$. The body doesn't, but it has a slightly modified type: $(\top \to q_0) \to \top \to q_0$, so we update the type of $F$: $\Gamma_2 := \{S : q_0, F : (\top \to q_0) \to \top \to q_0\}$.
Going back to the definition of $S$, we know that $F$ is required to have a type like $(q_1 \to q_0) \to \top \to q_0$ (because $\mathtt{b}$ has type $q_1 \to q_0$, not $\top \to q_0$). Thus, we further update the type of $F$: $\Gamma_3 := \{S : q_0, F : (q_1 \to q_0) \to \top \to q_0\}$.

By looking at the definition of $F$ again, we know that $x$ should have type $q_1$ and that $f$ should have $q_1$ as a return type, from the first and second arguments of $\mathtt{a}$ respectively. Thus, we get an updated type environment: $\Gamma_4 := \{S{:}q_0, F{:}(q_1 \to q_0) \wedge (\top \to q_1) \to q_1 \to q_0\}$. By checking the definition of $S$ again, we get:

$$\Gamma_5 := \{S : q_0, F : (q_1 \to q_0) \wedge (q_1 \to q_1) \to q_1 \to q_0\}.$$

Thus, we have obtained enough type information for $\mathcal{G}_0$ (recall Example 3).

In the above example, the type of $F$ has been expanded as follows.

$$\begin{aligned}&\top \preceq_O \top \to \top \to q_0 \preceq_P (\top \to q_0) \to \top \to q_0 \preceq_O (q_1 \to q_0) \to \top \to q_0\\ &\preceq_P (q_1 \to q_0) \wedge (\top \to q_1) \to q_1 \to q_0 \preceq_O (q_1 \to q_0) \wedge (q_1 \to q_1) \to q_1 \to q_0.\end{aligned}$$

Here, the expansions represented by $\preceq_O$ come from constraints on call sites of $F$, and those represented by $\preceq_P$ come from constraints on the definition of $F$. We shall formally define these expansion relations and obtain a fixedpoint-based model checking algorithm in the following sections.

*Remark 1.* A reader familiar with game semantics [1, 19] may find a connection between the type expansion sequence above and a play of a function. For example, the type $(\top \to q_0) \to \top \to q_0$ may be considered an abstraction of a state of a play where the opponent of $F$ has requested a tree of type $q_0$, and the proponent has requested a tree of type $q_0$ in response. The type $(q_1 \to q_0) \to \top \to q_0$ represents the next state, where the opponent has requested a tree of type $q_1$ in response, and the type $(q_1 \to q_0) \wedge (\top \to q_1) \to q_1 \to q_0$ represents the state where, in response to it, the proponent has requested trees of type $q_1$ to the first and second arguments. Thus, the expansion relations $\preceq_O$ and $\preceq_P$ represent opponent's and proponent's moves respectively. Although we do not make this connection formal, this game semantic intuition may help understand how our algorithm described below works. In particular, the intuition helps us understand why necessary type information can be obtained by gradually expanding types as in the example above; for a valid type of a function,[4] there should be a corresponding play of the function, and by following the play, one can obtain a type expansion sequence that leads to the valid type.

## 3.2 Expansion relations

We now formally define the expansion relations $\preceq_O$ and $\preceq_P$ mentioned above. They are inductively defined by the following rules.

$$\frac{}{q \preceq_O q} \qquad \frac{}{q \preceq_P q} \qquad \frac{\tau \preceq_P \tau' \qquad \theta \preceq_O \theta'}{\tau \to \theta \preceq_O \tau' \to \theta'} \qquad \frac{\tau \preceq_O \tau' \qquad \theta \preceq_P \theta'}{\tau \to \theta \preceq_P \tau' \to \theta'}$$

$$\frac{\begin{array}{c}\forall j \in S' \setminus S. \exists q. \theta'_j = \top \to \cdots \to \top \to q \\ \forall j \in S. \theta_j \preceq_O \theta'_j \qquad S \subseteq S'\end{array}}{\bigwedge_{j \in S} \theta_j \preceq_O \bigwedge_{j \in S'} \theta'_j} \qquad \frac{\forall j \in S. \theta_j \preceq_P \theta'_j}{\bigwedge_{j \in S} \theta_j \preceq_P \bigwedge_{j \in S} \theta'_j}$$

Note that the four relations: $\theta \preceq_O \theta'$, $\tau \preceq_O \tau'$, $\theta \preceq_P \theta'$, and $\tau \preceq_P \tau'$ are defined simultaneously. Notice also that $\preceq_P$ and $\preceq_O$ are swapped in the argument position of arrow types; this is analogous to the contravariance of the standard subtyping relation in the argument position of function types. Another way to understand the relations is: $\theta \preceq_O \theta'$ ($\theta \preceq_P \theta'$, resp.) holds if $\theta'$ is obtained from $\theta$ by adding atomic types (of the form $q$) to positive (negative, resp.) positions.

In the last two rules, $S$ can be an empty set. So, we can derive $\top \preceq_O \top \to q_0$, from which $\top \to \top \to q_0 \preceq_P (\top \to q_0) \to \top \to q_0$ follows. The expansion relation $\preceq_O$ is extended to type environments by: $\Gamma \preceq_O \Gamma'$ if and only if $\forall x \in \mathit{dom}(\Gamma) \cup \mathit{dom}(\Gamma').\Gamma(x) \preceq_O \Gamma'(x))$. Here, $\Gamma(x) = \bigwedge\{\theta \mid x : \theta \in \Gamma\}$. $\Gamma \preceq_P \Gamma'$ is defined in a similar manner.

Let $\leq$ be the standard subtyping relation on intersection types (see Appendix A). The following lemmas state the relationship between the expansion and subtyping relations.

[4] Strictly speaking, we should interpret a type as a kind of linear type; for example, the type $q_1 \to q_0$ should be interpreted as a function that takes a tree of type $q_1$ and uses it *at least once* to return a tree of type $q_0$.

**Lemma 1.** *1. $\theta_1 \preceq_O \theta_2$ and $\tau_1 \preceq_O \tau_2$ imply $\theta_1 \geq \theta_2$ and $\tau_1 \geq \tau_2$ respectively.*
*2. $\theta_1 \preceq_P \theta_2$ and $\tau_1 \preceq_P \tau_2$ imply $\theta_1 \leq \theta_2$ and $\tau_1 \leq \tau_2$ respectively.*

*Proof.* This follows by a straightforward mutual induction on the derivations.

Note that the converse does not hold; for example, $\top \to q_0 \leq (q_1 \to q_0) \to q_0$ but $\top \to q_0 \not\preceq_P (q_1 \to q_0) \to q_0$. In the game-semantic view (recall Remark 1), $\theta \preceq_P \theta'$ ($\theta \preceq_O$, respectively) mean that $\theta'$ is obtained from $\theta$ by only adding base types to positions corresponding to proponent's moves (opponent's moves, resp.).

## 3.3 Type generation rules

We now define the relation $\Gamma_1 \triangleright \Gamma_2 \vdash_P^{\mathcal{B}} t : \theta_1 \triangleright \theta_2$, which means that, given a candidate of type judgment $\Gamma_1 \vdash t : \theta_1$ (which may not be valid), a valid judgment $\Gamma_2 \vdash^{\mathcal{B}} t : \theta_2$ is obtained by "adjusting" $\Gamma_1$ and $\theta_1$ in a certain manner. $\Gamma_1 \triangleright \Gamma_2 \vdash_P^{\mathcal{B}} t : \theta_1 \triangleright \theta_2$ corresponds to the condition $(\Gamma_2 \vdash^{\mathcal{B}} t : \theta_2) \wedge (\Gamma_1 \preceq_O \Gamma_2) \wedge (\theta_1 \preceq_P \theta_2)$ used in the informal definition of $\mathcal{F}'_{\mathcal{B}}$ in Section 3.1. In fact, the rules below ensure that $\Gamma_1 \triangleright \Gamma_2 \vdash_P^{\mathcal{B}} t : \theta_1 \triangleright \theta_2$ implies that $(\Gamma_2 \vdash^{\mathcal{B}} t : \theta_2) \wedge (\Gamma_1 \preceq_O \Gamma_2) \wedge (\theta_1 \preceq_P \theta_2)$ holds. Thus, the "adjustment" of $\Gamma_1$ and $\theta_1$ is allowed only in a restricted manner. For example, $(f : q_1 \to q_0) \triangleright (f : q_1 \to q_0, x : q_1) \vdash_P^{\mathcal{B}} f\,x : q_0 \triangleright q_0$ is allowed, but $(f : \top \to q_0) \triangleright (f : q_1 \to q_0, x : q_1) \vdash_P^{\mathcal{B}} f\,x : q_0 \triangleright q_0$ is not. In the latter, the change of the type of function $f$ makes the assumption on the behavior of $f$ that upon receiving a request for tree of type $q_0$, $f$ requests a tree of type $q_1$ as an argument. That assumption is speculative and should be avoided, as its validity can be determined only by looking at the environment, not the term $f\,x$.

**Definition 1.** *$\Gamma_1 \triangleright \Gamma_2 \vdash_P^{\mathcal{B}} t : \theta_1 \triangleright \theta_2$ and $\Gamma_1 \triangleright \Gamma_2 \vdash_P^{\mathcal{B}} t : \tau_1 \triangleright \tau_2$ are the least relations that satisfy the following rules.*

$$\frac{\theta_1 \preceq_P \theta_2}{x : \theta_2 \triangleright x : \theta_2 \vdash_P^{\mathcal{B}} x : \theta_1 \triangleright \theta_2}\ (\textsc{VarP}) \qquad \frac{\tau_1 \preceq_O \theta_2}{x : \tau_1 \triangleright x : \theta_2 \vdash_P^{\mathcal{B}} x : \theta_2 \triangleright \theta_2}\ (\textsc{VarO})$$

$$\frac{(q, a, q_1 \cdots q_n) \in \Delta \qquad \forall i \in \{1, \ldots, n\}.\tau_i \in \{\top, q_i\}}{\emptyset \triangleright \emptyset \vdash_P^{\mathcal{B}} a : (\tau_1 \to \cdots \to \tau_n \to q) \triangleright (q_1 \to \cdots \to q_n \to q)}\ (\textsc{Const})$$

$$\frac{\Gamma_1 \triangleright \Gamma'_1 \vdash_P^{\mathcal{B}} t_1 : (\tau \to \theta) \triangleright (\tau' \to \theta') \qquad \Gamma_2 \triangleright \Gamma'_2 \vdash_P^{\mathcal{B}} t_2 : \tau'' \triangleright \tau'}{\Gamma_1 \cup \Gamma_2 \triangleright \Gamma'_1 \cup \Gamma'_2 \vdash_P^{\mathcal{B}} t_1 t_2 : \theta \triangleright \theta'}\ (\textsc{App})$$

$$\frac{(\Gamma_1, x : \tau_1) \triangleright (\Gamma_2, x : \tau_2) \vdash_P^{\mathcal{B}} t : \theta_1 \triangleright \theta_2}{\Gamma_1 \triangleright \Gamma_2 \vdash_P^{\mathcal{B}} \lambda x.t : (\tau_1 \to \theta_1) \triangleright (\tau_2 \to \theta_2)}\ (\textsc{Abs})$$

$$\frac{\forall i \in S.(\Gamma_i \triangleright \Gamma'_i \vdash_P^{\mathcal{B}} t : \theta_i \triangleright \theta'_i)}{(\bigcup_{i \in S} \Gamma_i) \triangleright (\bigcup_{i \in S} \Gamma'_i) \vdash_P^{\mathcal{B}} t : (\bigwedge_{i \in S} .\theta_i) \triangleright (\bigwedge_{i \in S} .\theta'_i)}\ (\textsc{Int})$$

In the rules above, we write $x : \bigwedge_{i \in S} \theta_i$ for $\{x : \theta_i \mid i \in S\}$. It is implicitly assumed that the sort of each variable is respected, i.e., if $x{:}\theta \in \Gamma$, then $\theta{::}\mathit{sort}(x)$ must hold. The rule VarP is used for adjusting the type of $x$ to the type provided by the environment, while the rule VarO is used for adjusting the environment to the type of $x$. For example, $(x{:}q_1 \to q_2) \triangleright (x : q_1 \to q_2) \vdash_P^{\mathcal{B}} x : (\top \to q_2) \triangleright (q_1 \to q_2)$ is obtained from the former, and $x : \top \triangleright (x : \top \to q_2) \vdash_P^{\mathcal{B}} x : (\top \to q_2) \triangleright (\top \to q_2)$ is obtained from the latter. In the rule App, the left and right premises adjust the types of the function and the argument respectively. (In the game semantic view, the former accounts for a move of the function, and the latter accounts for a move of the argument.)

## 3.4 Model Checking Algorithm

We are now ready to describe the new algorithm. Let $\mathcal{G} = (\Sigma, \mathcal{N}, \mathcal{R}, S)$ be a recursion scheme and $\mathcal{B}$ be a trivial automaton. Define $\mathbf{Expand}_{\mathcal{G},\mathcal{B}}$ by:

$$\mathbf{Expand}_{\mathcal{G},\mathcal{B}}(\Gamma) = \Gamma \cup (\bigcup \{\Gamma' \cup \{F{:}\theta'\} \mid \Gamma_1 \triangleright \Gamma' \vdash_P^{\mathcal{B}} \mathcal{R}(F) : \theta \triangleright \theta', F{:}\theta \in \Gamma, \Gamma_1 \subseteq \Gamma\})$$

Our new algorithm just consists of two fixedpoint computations:

```
NEW ALGORITHM:
1. $\Gamma$ := $\{S : q_0\}$;
2. Repeat $\Gamma$ := $\mathbf{Expand}_{\mathcal{G},\mathcal{B}}(\Gamma)$ until $\Gamma = \mathbf{Expand}_{\mathcal{G},\mathcal{B}}(\Gamma)$;
3. Repeat $\Gamma$ := $\mathbf{Shrink}_{\mathcal{G},\mathcal{B}}(\Gamma)$ until $\Gamma = \mathbf{Shrink}_{\mathcal{G},\mathcal{B}}(\Gamma)$;
4. Output whether $S : q_0 \in \Gamma$.
```

We first expand the set of type candidates by using $\mathbf{Expand}_{\mathcal{G},\mathcal{B}}$, and then shrink it by filtering out invalid types by $\mathbf{Shrink}_{\mathcal{G},\mathcal{B}}$. The only change from the naïve algorithm is that $\Gamma_{\mathbf{max}}$ has been replaced by the least fixedpoint of $\mathbf{Expand}_{\mathcal{G},\mathcal{B}}$.

*Example 4.* Recall $\mathcal{G}_0$ and $\mathcal{B}_0$ of Examples 1 and 2. Let $\Gamma_0 = \{S : q_0\}$. We have:

$$\begin{aligned}
\Gamma_1 &= \mathbf{Expand}_{\mathcal{G}_0,\mathcal{B}_0}(\Gamma_0) = \{S : q_0, F : \top \to \top \to q_0\}\\
\Gamma_2 &= \mathbf{Expand}_{\mathcal{G}_0,\mathcal{B}_0}(\Gamma_1) = \Gamma_1 \cup \{F : (\top \to q_0) \to \top \to q_0\}\\
\Gamma_3 &= \mathbf{Expand}_{\mathcal{G}_0,\mathcal{B}_0}(\Gamma_2) = \Gamma_2 \cup \{F : (q_1 \to q_0) \to \top \to q_0, \ldots\}\\
\Gamma_4 &= \mathbf{Expand}_{\mathcal{G}_0,\mathcal{B}_0}(\Gamma_3) = \Gamma_3 \cup \{F : (q_1 \to q_0) \to q_1 \to q_0, \ldots\}\\
\Gamma_5 &= \mathbf{Expand}_{\mathcal{G}_0,\mathcal{B}_0}(\Gamma_4) = \Gamma_3 \cup \{F : (q_1 \to q_0) \wedge (\top \to q_1) \to q_1 \to q_0, \ldots\}\\
\Gamma_6 &= \mathbf{Expand}_{\mathcal{G}_0,\mathcal{B}_0}(\Gamma_5) = \Gamma_4 \cup \{F : (q_1 \to q_0) \wedge (q_1 \to q_1) \to q_1 \to q_0, \ldots\}\\
\Gamma_7 &= \mathbf{Expand}_{\mathcal{G}_0,\mathcal{B}_0}(\Gamma_6) = \Gamma_6
\end{aligned}$$

In the second step (for computing $\Gamma_2$), the type $(\top \to q_0) \to \top \to q_0$ of $F$ is obtained from the following derivation.

$$\dfrac{\dfrac{\emptyset \triangleright f : \top \to q_0 \vdash_P^{\mathcal{B}} \mathtt{a}\ (f\ x) : (\top \to q_0) \triangleright (q_0 \to q_0) \qquad \Delta_1 \triangleright \Delta_1 \vdash_P^{\mathcal{B}} (F\ f\ (f\ x)) : q_0 \triangleright q_0}{\Delta_1 \triangleright \Delta_2 \vdash_P^{\mathcal{B}} \mathtt{a}\ (f\ x)\ (F\ f\ (f\ x)) : q_0 \triangleright q_0}}{\Delta_1 \triangleright \Delta_1 \vdash_P^{\mathcal{B}} \lambda f.\lambda x.\mathtt{a}\ (f\ x)\ (F\ f\ (f\ x)) : \top \to \top \to q_0 \triangleright (\top \to q_0) \to \top \to q_0}$$

Here, $\Delta_1 = F : \top \to \top \to q_0$, $\Delta_2 = \Delta_1 \cup \{f : \top \to q_0\}$, and $\emptyset \triangleright f : \top \to q_0 \vdash^{\mathcal{B}}_{P}$ $\mathtt{a}\ (f\ x) : (\top \to q_0) \triangleright (q_0 \to q_0)$ is derivable from $\emptyset \triangleright f : (\top \to q_0) \vdash^{\mathcal{B}}_{P} f : \top \to q_0 \triangleright \top \to q_0$ and $\emptyset \triangleright \emptyset \vdash^{\mathcal{B}}_{P} \mathtt{a} : (\top \to \top \to q_0) \triangleright (q_0 \to q_0 \to q_0)$.

By repeatedly applying $\mathbf{Shrink}_{\mathcal{G},\mathcal{B}}$ to $\Gamma_6$, we obtain: $\Gamma = \{S : q_0, F : (q_1 \to q_0) \wedge (q_1 \to q_1) \to q_1 \to q_0, F : (\top \to q_0) \to \top \to q_0, \ldots\}$ as a fixedpoint. Since $S : q_0 \in \Gamma$, we know that $\mathcal{G}_0$ is well-typed, i.e. $[\![\mathcal{G}]\!]$ is accepted by $\mathcal{B}^{\perp}$.

The least fixedpoint $\Gamma_6$ of $\mathbf{Expand}_{\mathcal{G}_0,\mathcal{B}_0}$ contains type bindings like $F : (\top \to q_0) \to \top \to q_0$, which are not required for typing $\mathcal{G}$ (recall Example 3). However, $\mathbf{Expand}_{\mathcal{G}_0,\mathcal{B}_0}$ does not add completely irrelevant type bindings like $F : \top \to \top \to q_1$. Thus, we can expect that the least fixedpoint of $\mathbf{Expand}_{\mathcal{G}_0,\mathcal{B}_0}$ is often much smaller than $\Gamma_{\mathbf{max}}$, which will be confirmed by the experiments in Section 5.

## 4 Correctness of the Algorithm

This section discusses the correctness and complexity of the algorithm.

### 4.1 Termination and Complexity

We first show that the algorithm always terminates.

The termination follows immediately from the following facts: (i) $\Gamma$ increases monotonically in the first loop, (ii) $\Gamma$ decreases monotonically in the second loop, and (iii) $\Gamma$ ranges over a finite set.

**Theorem 2.** *The algorithm always terminates and outputs "yes" or "no".*

From the termination argument, the complexity result also follows.

**Theorem 3.** *Suppose that both (i) the largest size of the sorts of non-terminals and terminals of $\mathcal{G}$ and (ii) the size of automaton $\mathcal{B}$ are fixed. Then, the algorithm terminates in time quadratic in $|\mathcal{G}|$.*

*Proof.* By the assumption, the size of $\Gamma_{\mathbf{max}}$ is $O(|\mathcal{G}|)$. Thus, the two loops terminate in $O(|\mathcal{G}|)$ iterations. At each iteration, $\mathbf{Expand}_{\mathcal{G},\mathcal{B}}(\mathcal{G})$ and $\mathbf{Shrink}_{\mathcal{G},\mathcal{B}}(\mathcal{G})$ can be computed in time $O(|\mathcal{G}|)$,[5] hence the result. $\square$

Actually, we can use Rehof and Mogensen's algorithm [21] to accelerate the above algorithm, and obtain a linear time algorithm. (The idea is just to recompute $\mathbf{Expand}_{\mathcal{G},\mathcal{B}}$ and $\mathbf{Shrink}_{\mathcal{G},\mathcal{B}}$ only for variables whose relevant bindings were updated. As $\Gamma(x)$ is updated only a constant number of times for each variable $x$, and the number of typing bindings that are affected by the update is a constant, the resulting algorithm runs in time linear in $|\mathcal{G}|$.) More precisely, if the number of states of $\mathcal{B}$ is $|Q|$ and the largest arity of functions is $A$, the algorithm runs in time $O(|\mathcal{G}|\mathbf{exp}_n(p(A|Q|)))$, where $p(x)$ is a polynomial of $x$, and $\mathbf{exp}_n(x)$ is defined by: $\mathbf{exp}_0(x) = x$, $\mathbf{exp}_{k+1}(x) = 2^{\mathbf{exp}_k(x)}$.

[5] To guarantee this, we need to normalize the rewrite rules of $\mathcal{G}$ in advance [15], so that the size of body $\mathcal{R}(F)$ of each non-terminal $F$ is bounded by a constant.

### 4.2 Soundness

The soundness of the algorithm follows immediately from that of Kobayashi's type system [12].

**Theorem 4.** *If the algorithm outputs "yes", then $[\![\mathcal{G}]\!]$ is accepted by $\mathcal{B}^{\perp}$.*

*Proof.* By the definition of the algorithm, $\Gamma$ at the last line satisfies $\Gamma \supseteq \mathbf{Shrink}_{\mathcal{G},\mathcal{B}}(\Gamma)$. Thus, the result follows by Theorem 1.

### 4.3 Completeness

A recursion scheme is in *normal form* if each rewrite rule is either of the form (i) $F \mapsto \lambda\widetilde{x}.t$ where $t$ does not contain terminals, or (ii) $F \mapsto \lambda\widetilde{x}.a\,\widetilde{x}$. We show that the algorithm is complete when the given recursion scheme is in normal form.[6] We now prove the completeness, i.e., if $[\![\mathcal{G}]\!]$ is accepted by $\mathcal{B}^{\perp}$, then the algorithm outputs "yes". From the game semantic intuition described in Remark 1, the reason for the completeness is intuitively clear: If a function behaves as described by a type $\theta$ in a reduction sequence of the given recursion scheme, then there should be a sequence of interactions between the function and the environment that conforms to $\theta$. As the sequence of interactions evolves, the function's behavior should gradually evolve from $\top \to \cdots \to \top \to q$ (which represents a state where the environment has just called the function to ask for a tree of type $q$) to $\top \to \cdots \to (\top \to \cdots \to \top \to q') \to \cdots \to \top \to q$ (which represents a state where the function has responded to ask the environment to provide a tree of type $q'$), and eventually to $\theta$. Such evolution of the function's type can be computed by $\mathbf{Expand}_{\mathcal{G},\mathcal{B}}$, and $\theta$ should be eventually generated.

The actual proof is however rather involved. We defer the proof to Appendix C and just state the theorem here.

**Theorem 5.** *Suppose $\mathcal{G}$ is in normal form. If $[\![\mathcal{G}]\!]$ is accepted by $\mathcal{B}^{\perp}$, then the algorithm outputs "yes".*

## 5 Experiments

We have implemented the new model checking algorithm, and tested it for several recursion schemes. The result of the preliminary experiments is shown in Table 1. The experiments were conducted on a machine with Intel(R) Xeon(R) CPU with 3Ghz and 8GB memory. More information about the benchmark is available at http://www.kb.ecei.tohoku.ac.jp/~koba/gtrecs/.

[6] Note that this does not lose generality, as we can always transform a recursion scheme into an equivalent recursion scheme in normal form before applying the algorithm, by introducing the rule $A \mapsto \lambda\widetilde{x}.a\,\widetilde{x}$ for each terminal $a$, and replace all the other occurrences of $a$ with $A$. We conjecture that the algorithm is complete without the normal form assumption.

The column "order" shows the order of the recursion scheme. The columns "hybrid" and "new" show the running times of the hybrid and new algorithms respectively, measured in seconds. The cell marked by "–" in the column "hybrid" shows that the hybrid algorithm has timed out (where the time limit is 10 min.) or run out of stack. The columns "$\Gamma_1$" and "$\Gamma_2$" show the numbers of atomic types in the type environment $\Gamma$ after the first and second loops of the new algorithm.

The table on the lefthand side shows the result for the following recursion scheme $\mathcal{G}_{n,m}$ [13]:

$$\begin{array}{l}\{S \mapsto F_0\ G_{n-1}\ \cdots\ G_2\ G_1\ G_0,\\ F_0 \mapsto \lambda f.\lambda\widetilde{x}.F_1\,(F_1\ f)\ \widetilde{x}, \cdots, F_{m-1} \mapsto \lambda f.\lambda\widetilde{x}.F_m\,(F_m\,f)\ \widetilde{x}, F_m \mapsto \lambda f.\lambda\widetilde{x}.G_n\ f\ \widetilde{x},\\ G_n \mapsto \lambda f.\lambda z.\lambda.\widetilde{x}.f\,(f\,z)\,\widetilde{x}, \cdots, G_2 \mapsto \lambda f.\lambda z.f\,(f\ z), G_1 \mapsto \lambda z.\mathtt{a}\ z, G_0 \mapsto \mathtt{c}\}\end{array}$$

$S$ is reduced to $\mathtt{a}^{\mathbf{exp}_n(m)}(G_0)$ and then to $\mathtt{a}^{\mathbf{exp}_n(m)}(\mathtt{c})$. The verified property is that the number of $\mathtt{a}$ is even. The hybrid algorithm [10] requires $O(\mathbf{exp}_n(m))$ expansions to extract the type information on $G_0$, so that it times out except for the case $n = 3, m = 1$. In contrast, the new algorithm works even for the case $n = 4, m = 10$. For a fixed $n$, the size of type environments (the columns $\Gamma_1$ and $\Gamma_2$) is almost linear in $m$. The running times are not linear in $m$ due to the naïveness of the current implementation, but exponential slowdown with respect to $m$ is not observed. As expected, the sizes of type environments (the columns $\Gamma_1$ and $\Gamma_2$) are much smaller than that of $\Gamma_{\mathbf{max}}$. For $\mathcal{G}_{3,1}$, the size of $\Gamma_{\mathbf{max}}$ is about $3 \times 2^{2057}$, so that the naïve algorithm does not work even for $\mathcal{G}_{3,1}$.

In the table on the righthand side, `Example1` is the recursion scheme given in Example 1, where the trivial automaton is given in Example 2. The recursion schemes `Twofiles` – `Lock2` have been taken from the benchmark set used in [10], obtained by encoding the resource usage verification problems [12]. We have used the refined encoding given in [14] however.[7] Unfortunately, for these recursion schemes, the new algorithm is slower than the hybrid algorithm by several orders of magnitude. Further optimization is required to see whether this is a fundamental limitation of the new algorithm. Finally, `Nondet` is an order-3 recursion scheme that generates a tree representation of an infinite list $[(0,1);(1,2);(2,3);\cdots]$, where each natural number $n$ is represented by the tree $s^n(z)$. A non-deterministic trivial automaton is used for expressing the property that each pair in the list is either a pair of an even number and an odd number, or a pair of an odd number and an even number. Our new algorithm works well, while the hybrid algorithm (which works only for deterministic trivial automata) is not directly applicable.[8]

[7] The encoding in [12] produces order-4 recursion schemes, while that of [14] produces order-3 recursion schemes. An additional optimization is required to handle the encoding of [12]: see Appendix D.

[8] As mentioned in Section 6, Lester et al. [18] recently extended the hybrid algorithm to deal with alternating Büchi automata.

**Table 1.** The result of experiments. Times are in seconds.

| | order | hybrid | new | $\mathit{\Gamma}_1$ | $\mathit{\Gamma}_2$ |
|---|---|---|---|---|---|
| $\mathcal{G}_{3,1}$ | 3 | 0.002 | 0.021 | 61 | 41 |
| $\mathcal{G}_{3,5}$ | 3 | – | 0.135 | 161 | 97 |
| $\mathcal{G}_{3,10}$ | 3 | – | 0.382 | 286 | 167 |
| $\mathcal{G}_{4,1}$ | 4 | – | 0.563 | 302 | 206 |
| $\mathcal{G}_{4,5}$ | 4 | – | 14.856 | 1079 | 703 |
| $\mathcal{G}_{4,10}$ | 4 | – | 43.815 | 2054 | 1328 |

| | order | hybrid | new | $\mathit{\Gamma}_1$ | $\mathit{\Gamma}_2$ |
|---|---|---|---|---|---|
| `Example1` | 2 | 0.002 | 0.002 | 15 | 13 |
| `Twofiles` | 3 | 0.001 | 0.228 | 468 | 187 |
| `FileWrong` | 3 | 0.001 | 0.116 | 398 | 142 |
| `FileOcamlc` | 3 | 0.003 | 1.162 | 1610 | 414 |
| `Lock2` | 3 | 0.013 | 98.785 | 2464 | 1191 |
| `Nondet` | 3 | N.A. | 0.013 | 77 | 63 |

## 6 Related Work

We have already discussed the main related work in Section 1. There are several algorithms for the model checking of higher-order recursion schemes (some of which are presented in the context of showing the decidability). Besides those already mentioned [19, 12, 10, 15], Hague et al. [6] reduce the modal $\mu$-calculus model checking to a parity game over the configuration graph of a collapsible pushdown automaton. Aehlig [2] gives a trivial automata model checking algorithm based on a finite semantics, which runs in a fixed-parameter *non-deterministic* linear time in the size of the recursion scheme. For recursion schemes with the so called *safety* restriction, Knapik et al. [9] give another decision procedure, which reduces a model checking problem for an order-$n$ recursion scheme to that for an order-$(n-1)$ recursion scheme. As mentioned already, however, the only practical previous algorithm (which was ever implemented) is Kobayashi's hybrid algorithm [10], to our knowledge. Its worst-case complexity is hyper-exponential in the size of recursion schemes, unlike our new algorithm. Recently, Lester et al. [18] extended the hybrid algorithm to deal with alternating Büchi automata. As the basic mechanism for collecting type information remains the same, their algorithm also suffers from the same worst-case behavior as Kobayashi's hybrid algorithm.

As mentioned already, though our new algorithm is type-based, it has been inspired from game semantics [1, 19]. In the previous type-based approach [12], the types of a function provide coarse-grained information about the function's behavior, in the sense that the types tell us information about *complete* runs of the function. On the other hand, the game-semantic view provides more fine-grained information, about partial runs of a function. For example, $F : \top \to q_0$ belonging to the least fixedpoint of $\mathbf{Expand}_{\mathcal{G},\mathcal{B}}$ means that $F$ may be called in a context where a tree of type $q_0$ is required, not necessarily that $F$ returns a tree of type $q_0$ for arbitrary arguments. This enabled us to collect type information by a least fixedpoint computation, yielding a realistic linear time algorithm. A price to pay is that the information is too fine-grained, so that, as observed in Section 5, it is actually often slower than the hybrid algorithm though the worst-case complexity of the latter is hyper-exponential.

As explained in Section 2, the model checking of higher-order recursion schemes has been reduced to the type checking problem for an intersection type system. Thus, our algorithm may have some connection to intersection type inference algorithms [22, 7, 5]. The connection is however not so clear. To our knowledge, the existing inference algorithms have a process corresponding to $\beta$-normalization [5], so that even for terms without recursion, the worst-case complexity of intersection type inference is non-elementary in the program size.

## 7 Conclusion

Studies of the model checking of higher-order recursion schemes have started from theoretical interests [8, 9], but it is now becoming the basis of automated verification tools for higher-order functional programs [12, 17, 16, 20]. Thus, it is very important to develop an efficient model checker for higher-order recursion schemes. The new algorithm presented in this paper is the first one that is efficient both in theory (in the sense that it is fixed-parameter linear time) and in practice (in the sense that it is runnable for recursion schemes of order 3 or higher). The practical efficiency is however far from satisfactory (recall Section 5), so that further optimization of the algorithm is necessary. As the structure of the new algorithm is simple, we expect that it is more amenable to various optimization techniques, such as BDD representation of types. A combination of the hybrid and new algorithms also seems useful. It does not seem so difficult to extend the new algorithm to obtain a practical fixed-parameter polynomial time algorithm for the full modal $\mu$-calculus; It is left for future work.

## References

1. S. Abramsky and G. McCusker. Game semantics. In *Computational Logic: Proceedings of the 1997 Marktoberdorf Summer School*, pages 1–56. Springer-Verlag, 1999.
2. K. Aehlig. A finite semantics of simply-typed lambda terms for infinite runs of automata. *Logical Methods in Computer Science*, 3(3), 2007.
3. T. Ball and S. K. Rajamani. The SLAM project: Debugging system software via static analysis. In *Proceedings of ACM SIGPLAN/SIGACT Symposium on Principles of Programming Languages (POPL)*, pages 1–3, 2002.
4. D. Beyer, T. A. Henzinger, R. Jhala, and R. Majumdar. The software model checker Blast. *International Journal on Software Tools for Technology Transfer*, 9(5-6):505–525, 2007.
5. S. Carlier, J. Polakow, J. B. Wells, and A. J. Kfoury. System E: Expansion variables for flexible typing with linear and non-linear types and intersection types. In *Proceedings of ESOP'04*, volume 2986 of *Lecture Notes in Computer Science*, pages 294–309, 2004.
6. M. Hague, A. Murawski, C.-H. L. Ong, and O. Serre. Collapsible pushdown automata and recursion schemes. In *Proceedings of 23rd Annual IEEE Symposium on Logic in Computer Science*, pages 452–461. IEEE Computer Society, 2008.
7. A. J. Kfoury and J. B. Wells. Principality and type inference for intersection types using expansion variables. *Theor. Comput. Sci.*, 311(1-3):1–70, 2004.

8. T. Knapik, D. Niwinski, and P. Urzyczyn. Deciding monadic theories of hyperalgebraic trees. In *TLCA 2001*, volume 2044 of *Lecture Notes in Computer Science*, pages 253–267. Springer-Verlag, 2001.
9. T. Knapik, D. Niwinski, and P. Urzyczyn. Higher-order pushdown trees are easy. In *FoSSaCS 2002*, volume 2303 of *Lecture Notes in Computer Science*, pages 205–222. Springer-Verlag, 2002.
10. N. Kobayashi. Model-checking higher-order functions. In *Proceedings of PPDP 2009*, pages 25–36. ACM Press, 2009. See also [13].
11. N. Kobayashi. TRecS: A type-based model checker for recursion schemes. http://www.kb.ecei.tohoku.ac.jp/ koba/trecs/, 2009.
12. N. Kobayashi. Types and higher-order recursion schemes for verification of higher-order programs. In *Proceedings of ACM SIGPLAN/SIGACT Symposium on Principles of Programming Languages (POPL)*, pages 416–428, 2009. See also [13].
13. N. Kobayashi. Model checking higher-order programs. Available at http://www.kb.ecei.tohoku.ac.jp/˜koba/papers/hmc.pdf. A revised and extended version of [12] and [10], 2010.
14. N. Kobayashi and C.-H. L. Ong. Complexity of model checking recursion schemes for fragments of the modal mu-calculus. In *Proceedings of ICALP 2009*, volume 5556 of *Lecture Notes in Computer Science*, pages 223–234. Springer-Verlag, 2009.
15. N. Kobayashi and C.-H. L. Ong. A type system equivalent to the modal mu-calculus model checking of higher-order recursion schemes. In *Proceedings of LICS 2009*, pages 179–188. IEEE Computer Society Press, 2009.
16. N. Kobayashi, R. Sato, and H. Unno. Predicate abstraction and cegar for higher-order model checking. In *Proceedings of ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)*, 2011.
17. N. Kobayashi, N. Tabuchi, and H. Unno. Higher-order multi-parameter tree transducers and recursion schemes for program verification. In *Proceedings of ACM SIGPLAN/SIGACT Symposium on Principles of Programming Languages (POPL)*, pages 495–508, 2010.
18. M. M. Lester, R. P. Neatherway, C.-H. L. Ong, and S. J. Ramsay. Model checking liveness properties of higher-order functional programs. Unpublished manuscript, 2010.
19. C.-H. L. Ong. On model-checking trees generated by higher-order recursion schemes. In *LICS 2006*, pages 81–90. IEEE Computer Society Press, 2006.
20. C.-H. L. Ong and S. Ramsay. Verifying higher-order programs with pattern-matching algebraic data types. In *Proceedings of ACM SIGPLAN/SIGACT Symposium on Principles of Programming Languages (POPL)*, pages 587–598, 2011.
21. J. Rehof and T. Mogensen. Tractable constraints in finite semilattices. *Science of Computer Programming*, 35(2):191–221, 1999.
22. S. R. D. Rocca and B. Venneri. Principal type schemes for an extended type theory. *Theor. Comput. Sci.*, 28:151–169, 1984.

## Appendix

## A Subtyping

The subtyping relations $\theta \leq \theta'$ and $\tau \leq \tau'$ are inductively defined by the following rules.

$$\frac{}{q \to q}$$

$$\frac{\tau' \leq \tau \qquad \theta \leq \theta'}{\tau \to \theta \leq \tau' \to \theta'}$$

$$\frac{\forall j \in S'.\theta_j \leq \theta'_j \qquad S' \subseteq S}{\bigwedge_{j\in S} \theta_j \leq \bigwedge_{j\in S'} \theta'_j}$$

## B Algorithmic Interpretation of Type Expansion Rules

When computing $\mathbf{Expand}_{\mathcal{G},\mathcal{B}}(\Gamma)$ defined in Section 3.4, we need to find $\Gamma'$, $\theta'$, and $\Gamma_1$ such that $\Gamma_1 \rhd \Gamma' \vdash^{\mathcal{B}}_{P} t : \theta \rhd \theta'$ and $\Gamma_1 \subseteq \Gamma$, for given $\Gamma$, $t$, and $\theta$ based on the rules for the type expansion relation $\Gamma \rhd \Gamma' \vdash^{\mathcal{B}}_{P} t : \theta \rhd \theta'$. Figure 2 shows such an algorithm $\mathbf{ExpSub}(\Gamma, t, \theta)$, which returns a set of tuples $(\Gamma', \theta', \Gamma_1)$ mentioned above.

## C Proof of Completeness (Theorem 5)

### C.1 Proof of Theorem 5

The following is the key lemma, which states that by repeated applications of the transformation represented by the expansition relation $\vdash^{\mathcal{B}}_{P}$, one can eventually obtain a valid type judgment $\Gamma \vdash^{\mathcal{B}} t : q$ such that $\Gamma$ contains only type bindings generated by $\mathbf{Expand}_{\mathcal{G},\mathcal{B}}$.

**Lemma 2.** *Let $t$ be a term consisting of only non-terminals, $T$ be a finite $\Sigma$-labeled tree, and suppose $t \longrightarrow^*_{\mathcal{G}} T$. If $T$ is accepted by $\mathcal{B}$ from $q$, then there exist $\Gamma_1(=\emptyset), \Gamma_2, \dots, \Gamma_{2m}, \Theta$, and $\ell$ such that:*

*(i) $\Gamma_{2i+1} \rhd \Gamma_{2i+2} \vdash^{\mathcal{B}}_{P} t : q \rhd q$ (for each $i \in \{0, \dots, m-1\}$);*

*(ii) $\Gamma_{2i+1} \subseteq \mathbf{Expand}^{\ell}_{\mathcal{G},\mathcal{B}}(\bigcup_{j\leq 2i} \Gamma_j)$ for each $i \in \{1, \dots, m-1\}$; and*

*(iii) $\vdash^{\mathcal{B}} \mathcal{R} : \Theta \wedge \Gamma_{2m} \subseteq \Theta \subseteq \mathbf{Expand}^{\ell}_{\mathcal{G},\mathcal{B}}(\bigcup_{j\leq 2m} \Gamma_j)$.*

*Proof.* See Appendix C.2.

```
$\mathbf{ExpSub}(\Gamma, x\,t_1 \cdots t_k, \theta) =$ (* $k$ may be 0 *)
    let $S = \{\theta_1 \mid \theta_1 = \tau_1 \to \cdots \to \tau_k \to \theta', x : \theta_1 \in \Gamma, \theta \preceq_P \theta'\}$
    in $\{(\Theta, \theta', \Gamma') \mid$
        $\tau_1 \to \cdots \to \tau_k \to \theta' \in S,$
        $S_i = \mathbf{ExpSub}(\Gamma, t_i, \tau_i)($ for each $i \in \{1, \ldots, k\}),$
        $(\Theta_i, \tau_i', \Gamma_i) \in S_i,$
        $\Theta = \{x : \tau_1 \to \cdots \to \tau_k \to \theta'\} \cup \Theta_1 \cup \cdots \cup \Theta_k,$
        $\Gamma' = \{x : \tau_1' \to \cdots \to \tau_k' \to \theta'\} \cup \Gamma_1 \cup \cdots \cup \Gamma_k\}$
$\mathbf{ExpSub}(\Gamma, a\,t_1 \cdots t_k, \theta) =$
    let $S = \{\theta_1 \mid \theta_1 = q_1 \to \cdots \to q_n \to q, (q, a, q_1 \cdots q_n) \in \Delta,$
                $\theta \preceq_P q_{k+1} \to \cdots \to q_n \to q\}$
    in $\{(\Theta, \theta', \Gamma') \mid$
        $\tau_1 \to \cdots \to \tau_k \to \theta' \in S,$
        $S_i = \mathbf{ExpSub}(\Gamma, t_i, \tau_i)($ for each $i \in \{1, \ldots, k\}),$
        $(\Theta_i, \tau_i', \Gamma_i) \in S_i,$
        $\Theta = \Theta_1 \cup \cdots \cup \Theta_k,$
        $\Gamma' = \Gamma_1 \cup \cdots \cup \Gamma_k\}$
$\mathbf{ExpSub}(\Gamma, \lambda x.t, \tau \to \theta) =$
    let $S = \mathbf{ExpSub}(\Gamma \cup \{x : \tau\}, t, \theta)$
    in $\{(\Theta, \tau' \to \theta', \Gamma') \mid$
        $(\Theta \cup \{x : \tau\}, \theta', \Gamma' \cup \{x : \tau'\}) \in S, x \notin dom(\Gamma')\}$
$\mathbf{ExpSub}(\Gamma, t, \bigwedge_{i \in I} \theta_i) =$
    let $S_i = \mathbf{ExpSub}(\Gamma, t, \theta_i)$ (for each $i \in I$)
    in $\{(\bigcup_{i \in I} \Theta_i, \bigwedge_{i \in I} \theta_i', \bigcup_{i \in I} \Gamma_i) \mid (\Theta_i, \theta_i', \Gamma_i) \in S_i$ for each $i \in I\}$
```

**Fig. 2.** Algorithm **ExpSub**

Lemma 2 applies only to recursion schemes that generate finite trees. To deal with a general recursion scheme $\mathcal{G} = (\Sigma, \{F_1, \ldots, F_m\}, \mathcal{R}, S)$, we construct an approximation $\mathcal{G}^{(\ell)} = (\Sigma \cup \{\bot\}, \mathcal{N}^{(\ell)}, \mathcal{R}^{(\ell)}, S^{(0)})$ where $\mathcal{N}^{(\ell)}$ and $\mathcal{R}^{(\ell)}$ are:

$$\begin{aligned}
\mathcal{N}^{(\ell)} &= \{F_i^{(k)} | 1 \le i \le m, 0 \le k \le \ell\} \\
\mathcal{R}^{(\ell)} &= \{F_i^{(k)} \mapsto [F_1^{(k+1)}/F_1, \ldots, F_m^{(k+1)}/F_m] t_i \mid 1 \le i \le m, 0 \le k < \ell\} \\
&\quad \cup \{F_i^{(\ell)} \mapsto \lambda \widetilde{x}.\bot \mid 1 \le i \le m\}
\end{aligned}$$

Here, $\bot$ is the special terminal of arity 0, used in the definition of $[\![\mathcal{G}]\!]$.

Lemma 2 guarantees that if $\mathcal{G}^{(\ell)}$ is well-typed, then (a superset of) a type environment for $\mathcal{G}^{(\ell)}$ can be obtained by repeatedly applying $\mathbf{Expand}_{\mathcal{G},\mathcal{B}}$ to $\{S^{(0)} : q_0\}$. Let $\Gamma$ be a type environment such that $dom(\Gamma) \subseteq \mathcal{N}^{(\ell)}$ and let $\mathbf{Collapse}(\Gamma)$ be $\{F : \theta \mid F^{(k)} : \theta \in \Gamma\}$. The following lemma establishes a relationship between the typings of $\mathcal{G}^{(\ell)}$ and $\mathcal{G}$.

**Lemma 3 ([13], Section 5.3.3).** *Suppose* $\vdash^{\mathcal{B}^\bot} \mathcal{G}^{(\ell)} : \Gamma'$. *If* $\ell$ *is sufficiently large, then* $\vdash^{\mathcal{B}} \mathcal{G} : \Gamma$ *for some* $\Gamma \subseteq \mathbf{Collapse}(\Gamma')$.

Now we are ready to prove the completeness.

$$\frac{\theta_1 \preceq_P \theta_2}{x : \theta_1 \triangleright x : \theta_2 \vdash^{\mathcal{B}}_{O} x : \theta_1 \triangleright \theta_1} \qquad \text{(O-VarP)}$$

$$\frac{\theta_1 \preceq_O \theta_2}{x : \theta_1 \triangleright x : \theta_1 \vdash^{\mathcal{B}}_{O} x : \theta_1 \triangleright \theta_2} \qquad \text{(O-VarO)}$$

$$\frac{\Gamma_1 \triangleright \Gamma_1' \vdash^{\mathcal{B}}_{O} t_1 : (\tau \to \theta) \triangleright (\tau' \to \theta') \qquad \Gamma_2 \triangleright \Gamma_2' \vdash^{\mathcal{B}}_{O} t_2 : \tau \triangleright \tau''}{(\Gamma_1 \cup \Gamma_2) \triangleright (\Gamma_1' \cup \Gamma_2') \vdash^{\mathcal{B}}_{O} t_1 t_2 : \theta \triangleright \theta'} \qquad \text{(O-App)}$$

$$\frac{(\Gamma_1, x : \tau_1) \triangleright (\Gamma_2, x : \tau_2) \vdash^{\mathcal{B}}_{O} t : \theta_1 \triangleright \theta_2}{\Gamma_1 \triangleright \Gamma_2 \vdash^{\mathcal{B}}_{O} \lambda x.t : (\tau_1 \to \theta_1) \triangleright (\tau_2 \to \theta_2)} \qquad \text{(O-Abs)}$$

$$\frac{\emptyset \triangleright \emptyset \vdash^{\mathcal{B}}_{O} t_1 : \top \triangleright (\underbrace{\top \to \cdots \to \top}_{n+1} \to q)}{\emptyset \triangleright \emptyset \vdash^{\mathcal{B}}_{O} t_1 t_2 : \top \triangleright (\underbrace{\top \to \cdots \to \top}_{n} \to q)} \qquad \text{(O-TopFun)}$$

$$\frac{}{\emptyset \triangleright \emptyset \vdash^{\mathcal{B}}_{O} x : \top \triangleright (\top \to \cdots \to \top \to q)} \qquad \text{(O-TopVar)}$$

$$\frac{\forall i \in S.(\Gamma_i \triangleright \Gamma_i' \vdash^{\mathcal{B}}_{O} t : \theta_i \triangleright \theta_i') \qquad \forall i \in S' \setminus S.(\emptyset \triangleright \emptyset \vdash^{\mathcal{B}}_{O} t : \top \triangleright \theta_i') \qquad S \subseteq S'}{(\bigcup_{i \in S} \Gamma_i) \triangleright (\bigcup_{i \in S} \Gamma_i') \vdash^{\mathcal{B}}_{O} t : (\bigwedge_{i \in S} \theta_i) \triangleright (\bigwedge_{i \in S'} \theta_i')} \qquad \text{(O-INT)}$$

**Fig. 3.** The relation $\Gamma \triangleright \Gamma' \vdash^{\mathcal{B}}_{O} t : \theta \triangleright \theta'$

*Proof of Theorem 5* Suppose that $[\![\mathcal{G}]\!]$ is accepted by $\mathcal{B}^{\perp}$. Let $\ell$ be a sufficiently large number that satisfies Lemma 3. By the construction of $\mathcal{G}^{(\ell)}$, $[\![\mathcal{G}^{(\ell)}]\!]$ is also accepted by $\mathcal{B}^{\perp}$. By Lemma 2, we have $\Gamma'$ such that $\vdash^{\mathcal{B}^{\perp}} \mathcal{G}^{(\ell)} : \Gamma'$ and $\Gamma' \subseteq \mathbf{Expand}^{m}_{\mathcal{G}^{(\ell)},\mathcal{B}}(\{S^{(0)} : q_0\})$ for some $m$. By Lemma 3, we have $\Gamma''$ such that $\vdash^{\mathcal{B}} \mathcal{G} : \Gamma''$ and $\Gamma'' \subseteq \mathbf{Collapse}(\Gamma') \subseteq \mathbf{Collapse}(\mathbf{Expand}^{m}_{\mathcal{G}^{(\ell)},\mathcal{B}}(\{S^{(0)} : q_0\})) \subseteq \mathbf{Expand}^{m}_{\mathcal{G},\mathcal{B}}(\{S : q_0\})$. Let $\Gamma_r$ be the final value of $\Gamma$ in the new algorithm. By the standard fixedpoint theorem, $\Gamma_r \supseteq \Gamma'' \ni S : q_0$, so that the algorithm must output “yes”. □

## C.2 Proof of Lemma 2

To prove Lemma 2, we need to strengthen the statement by using another relation $\Gamma \triangleright \Gamma' \vdash^{\mathcal{B}}_{O} t : \theta \triangleright \theta'$, which describes an expansion of types by the environment' move. It is defined by the rules in Figure 3. For example, using O-VarP, we can derive $(F : \top \to q_0) \triangleright (F : q_1 \to q_0) \vdash^{\mathcal{B}}_{O} F : (\top \to q_0) \triangleright (\top \to q_0)$, which represents the move of an environment (or, the function $F$) that requests an argument of type $q_1$ in response to the request for an output of type $q_0$. There is no rule for constants; this is because the term $t$ in Lemma 7 does not contain terminal symbols.

The relations $\Gamma \triangleright \Gamma' \vdash^{\mathcal{B}}_{P} t : \theta \triangleright \theta'$ and $\Gamma \triangleright \Gamma' \vdash^{\mathcal{B}}_{O} t : \theta \triangleright \theta'$ satisfy the following properties, which follow by straightforward induction on the derivations of $\Gamma \triangleright \Gamma' \vdash^{\mathcal{B}}_{P} t : \theta \triangleright \theta'$ and $\Gamma \triangleright \Gamma' \vdash^{\mathcal{B}}_{O} t : \theta \triangleright \theta'$.

**Lemma 4.** *If $\Gamma \triangleright \Gamma' \vdash^{\mathcal{B}}_{P} t : \theta \triangleright \theta'$, then $\Gamma' \vdash^{\mathcal{B}} t : \theta'$, $\Gamma \preceq_O \Gamma'$, and $\theta \preceq_P \theta'$.*

**Lemma 5.** *If $\Gamma \triangleright \Gamma' \vdash^{\mathcal{B}}_{O} t : \theta \triangleright \theta'$, then $\Gamma \vdash^{\mathcal{B}} t : \theta$, $\Gamma \preceq_P \Gamma'$, and $\theta \preceq_O \theta'$.*

The rules for $\Gamma \triangleright \Gamma' \vdash^{\mathcal{B}}_{P} t : \theta \triangleright \theta'$ and $\Gamma \triangleright \Gamma' \vdash^{\mathcal{B}}_{O} t : \theta \triangleright \theta'$ can be interpreted as those for transforming derivations for $\Gamma \vdash t : \theta$. (In fact, without that purpose, we could define $\Gamma \triangleright \Gamma' \vdash^{\mathcal{B}}_{O} t : \theta \triangleright \theta'$ as $\Gamma \vdash^{\mathcal{B}} t : \theta \wedge \Gamma \preceq_P \Gamma' \wedge \theta \preceq_O \theta'$.) We define the consistency and stability of derivations of $\Gamma \triangleright \Gamma' \vdash^{\mathcal{B}}_{P} t : \theta \triangleright \theta'$ and $\Gamma \triangleright \Gamma' \vdash^{\mathcal{B}}_{O} t : \theta \triangleright \theta'$ as follows.

**Definition 2.** *A derivation $\pi_P$ of $\Gamma' \triangleright \Gamma'' \vdash^{\mathcal{B}}_{P} t : \theta' \triangleright \theta''$ is* consistent *with a derivation $\pi_O$ of $\Gamma \triangleright \Gamma' \vdash^{\mathcal{B}}_{O} t : \theta \triangleright \theta'$ if the lefthand side of $\pi_P$ matches that of $\pi_O$, and for each subterm $u$ of $t$, the corresponding types $\theta_1$, $\theta_2$, and $\theta_3$ assigned in $\pi_O$ and $\pi_P$ satisfies $\theta_1 \preceq_O \theta_2 = \theta_3$ or $\theta_1 = \theta_2 \preceq_P \theta_3$. Similarly, a derivation $\pi_O$ of $\Gamma' \triangleright \Gamma'' \vdash^{\mathcal{B}}_{O} t : \theta' \triangleright \theta''$ is consistent with a derivation $\pi_P$ of $\Gamma \triangleright \Gamma' \vdash^{\mathcal{B}}_{P} t : \theta \triangleright \theta'$ if the lefthand side of $\pi_O$ matches that of $\pi_P$.*

*A derivation for $\Gamma \triangleright \Gamma' \vdash^{\mathcal{B}}_{P} t : \theta \triangleright \theta'$ is* stable *if in every node $\Gamma_1 \triangleright \Gamma'_1 \vdash^{\mathcal{B}}_{P} u : \theta_1 \triangleright \theta'_1$ of the derivation tree, $\Gamma_1 = \Gamma'_1$ and $\theta_1 = \theta'_1$.*

*Remark 2.* More formally, consistent derivations should be inductively defined. For example, the derivation $\dfrac{\dfrac{\pi_1}{\Gamma_{1,1} \triangleright \Gamma_{1,2} \vdash^{\mathcal{B}}_{P} t_1 : (\tau_1 \to \theta_1) \triangleright (\tau_2 \to \theta_2)} \quad \dfrac{\pi'_1}{\Gamma_{2,1} \triangleright \Gamma_{2,2} \vdash^{\mathcal{B}}_{P} t_2 : \tau'_1 \triangleright \tau_2}}{\Gamma_{1,1} \cup \Gamma_{2,1} \triangleright \Gamma_{1,2} \cup \Gamma_{2,2} \vdash^{\mathcal{B}}_{P} t_1 t_2 : \theta_1 \triangleright \theta_2}$ is consistent with $\dfrac{\dfrac{\pi_0}{\Gamma_{1,0} \triangleright \Gamma_{1,1} \vdash^{\mathcal{B}}_{O} t_1 : (\tau_0 \to \theta_0) \triangleright (\tau_1 \to \theta_1)} \quad \dfrac{\pi'_0}{\Gamma_{2,0} \triangleright \Gamma_{2,1} \vdash^{\mathcal{B}}_{O} t_2 : \tau_0 \triangleright \tau'_1}}{(\Gamma_{1,0} \cup \Gamma_{2,0}) \triangleright (\Gamma_{1,1} \cup \Gamma_{2,1}) \vdash^{\mathcal{B}}_{O} t_1 t_2 : \theta_0 \triangleright \theta_1}$ if $\dfrac{\pi_1}{\Gamma_{1,1} \triangleright \Gamma_{1,2} \vdash^{\mathcal{B}}_{P} t_1 : (\tau_1 \to \theta_1) \triangleright (\tau_2 \to \theta_2)}$ is consistent with $\dfrac{\pi_0}{\Gamma_{1,0} \triangleright \Gamma_{1,1} \vdash^{\mathcal{B}}_{O} t_1 : (\tau_0 \to \theta_0) \triangleright (\tau_1 \to \theta_1)}$, and $\dfrac{\pi'_1}{\Gamma_{2,1} \triangleright \Gamma_{2,2} \vdash^{\mathcal{B}}_{P} t_2 : \tau'_1 \triangleright \tau_2}$ is consistent with $\dfrac{\pi'_0}{\Gamma_{2,0} \triangleright \Gamma_{2,1} \vdash^{\mathcal{B}}_{O} t_2 : \tau_0 \triangleright \tau'_1}$.

**Lemma 6 (inverse substitution).** *If $\Gamma \triangleright \Gamma' \vdash^{\mathcal{B}}_{P} [t/x]u : \theta \triangleright \theta'$, then we have:*

$$\begin{array}{l} \Gamma = \Gamma_1 \cup \Gamma_2 \\ \Gamma' = \Gamma'_1 \cup \Gamma'_2 \\ (\Gamma_1, x : \tau') \triangleright (\Gamma'_1, x : \tau') \vdash^{\mathcal{B}}_{P} u : \theta \triangleright \theta' \\ \Gamma_2 \triangleright \Gamma'_2 \vdash^{\mathcal{B}}_{P} t : \tau \triangleright \tau' \end{array}$$

*for some $\Gamma_1, \Gamma'_1, \tau, \tau'$.*

*Similarly, if $\Gamma \triangleright \Gamma' \vdash^{\mathcal{B}}_{O} [t/x]u : \theta \triangleright \theta'$, then we have:*

$$\begin{array}{l} \Gamma = \Gamma_1 \cup \Gamma_2 \\ \Gamma' = \Gamma'_1 \cup \Gamma'_2 \\ (\Gamma_1, x : \tau) \triangleright (\Gamma'_1, x : \tau) \vdash^{\mathcal{B}}_{O} u : \theta \triangleright \theta' \\ \Gamma_2 \triangleright \Gamma'_2 \vdash^{\mathcal{B}}_{O} t : \tau \triangleright \tau' \end{array}$$

*for some $\Gamma_1, \Gamma'_1, \tau, \tau'$.*

*Proof.* $(\Gamma_1, x : \tau') \triangleright (\Gamma'_1, x : \tau') \vdash^{\mathcal{B}}_{P} u : \theta \triangleright \theta'$ can be obtained by replacing each sub-derivation of the form $\Delta_i \triangleright \Delta'_i \vdash^{\mathcal{B}}_{P} t : \theta_i \triangleright \theta'_i$ with $x : \theta'_i \triangleright x : \theta'_i \vdash^{\mathcal{B}}_{P} x : \theta_i \triangleright \theta'_i$. $\tau$ and $\tau'_i$ are the intersections of all $\theta_i$ and $\theta'_i$ respectively. $\Gamma_2$ and $\Gamma'_2$ are the unions of all $\Delta_i$ and $\Delta'_i$ respectively.

Similarly, $(\Gamma_1, x : \tau) \triangleright (\Gamma'_1, x : \tau) \vdash^{\mathcal{B}}_{O} u : \theta \triangleright \theta'$ can be obtained by replacing each sub-derivation of the form $\Delta_i \triangleright \Delta'_i \vdash^{\mathcal{B}}_{O} t : \theta_i \triangleright \theta'_i$ with $x : \theta_i \triangleright x : \theta_i \vdash^{\mathcal{B}}_{O} x : \theta_i \triangleright \theta'_i$. □

We are now ready to prove the following strengthened version of Lemma 2.

**Lemma 7.** *Suppose $\mathcal{G}$ is in normal form. Let $t$ be a term consisting of only non-terminals, $T$ be a finite $\Sigma$-labeled tree, and suppose $t \longrightarrow^*_{\mathcal{G}} T$. If $T$ is accepted by $\mathcal{B}$ from $q$, then there exist $\Gamma_0(=\emptyset), \Gamma_1, \Gamma_2, \ldots, \Gamma_{2m}, \Theta, \ell$ such that:*

*(i)* $\Gamma_0 \triangleright \Gamma_1 \vdash^{\mathcal{B}}_O t : \top \triangleright q$;

*(ii)* $\Gamma_{2i+1} \triangleright \Gamma_{2i+2} \vdash^{\mathcal{B}}_P t : q \triangleright q$ *(for each* $i \in \{0, \ldots, m-1\}$*)*;

*(iii)* $\Gamma_{2i} \triangleright \Gamma_{2i+1} \vdash^{\mathcal{B}}_O t : q \triangleright q$ *(for each* $i \in \{1, \ldots, m-1\}$*)*;

*(iv)* $\Gamma_{2i+1} \subseteq \mathbf{Expand}^{\ell}_{\mathcal{G},\mathcal{B}}(\bigcup_{j \le 2i} \Gamma_j)$ *for every* $i \in \{0, \ldots, m-1\}$;

*(v) There exists* $\Theta$ *such that* $\vdash^{\mathcal{B}} \mathcal{R} : \Theta \wedge \Gamma_{2m} \subseteq \Theta \subseteq \mathbf{Expand}^{\ell}_{\mathcal{G},\mathcal{B}}(\bigcup_{j \le 2m} \Gamma_j)$;

*(vi)* $\Gamma_{i+1} \triangleright \Gamma_{i+2} \vdash_{\psi} t : q \triangleright q$ *is consistent with* $\Gamma_i \triangleright \Gamma_{i+1} \vdash_{\overline{\psi}} t : q \triangleright q$ *where* $\psi \in \{P, O\}$ *and* $\overline{P} = O$, $\overline{O} = P$*; and*

*(vii) The derivation for* $\Gamma_{2m-1} \triangleright \Gamma_{2m} \vdash^{\mathcal{B}}_P t : q \triangleright q$ *is stable.*

*Proof.* We fix $\ell$ to be $|\Gamma_{\mathbf{max}}| + 1$, so that $\mathbf{Expand}^{\ell}_{\mathcal{G},\mathcal{B}}(\Gamma) = \mathbf{Expand}^{\ell+1}_{\mathcal{G},\mathcal{B}}(\Gamma)$ for any $\Gamma$.

The proof proceeds by induction on the length of the sequence $t \longrightarrow^*_{\mathcal{G}} T$. (Note that the length of the sequence $t \longrightarrow^*_{\mathcal{G}} T$ is greater than 0, as $t$ does not contain terminals.)

Suppose $t = F\,\widetilde{s} \longrightarrow_{\mathcal{G}} [\widetilde{s}/\widetilde{x}]u \longrightarrow^*_{\mathcal{G}} T$, where $\mathcal{R}(F) = \lambda\widetilde{x}.u$. By the normal form assumption, either $u$ is of the form $a\,x_1 \cdots x_k$ (where $k$ may be 0), or $u$ does not contain terminals.

If $u$ is of the form $a\,x_1 \cdots x_k$, then $[\widetilde{s}/\widetilde{x}]u$ and $T$ must be of the form $a\ s_1 \cdots s_k$ and $a\ T_1 \cdots T_k$, with $s_j \longrightarrow^*_{\mathcal{G}} T_j (j \in \{1, \ldots, k\})$. As $T$ is accepted by $\mathcal{B}$ from $q$, there exists $q_1, \ldots, q_k$ such that $(q, a, q_1 \cdots q_k) \in \Delta_{\mathcal{B}}$ and $T_1, \ldots, T_k$ are accepted from $q_1, \ldots, q_k$. By the induction hypothesis (note that $\widetilde{s}$ consist of only non-terminals), we have:

$$\begin{array}{l}
\Gamma_{2i,j} \triangleright \Gamma_{2i+1,j} \vdash^{\mathcal{B}}_O u_j : q^?_j \triangleright q_j \ \ (q^?_j = \top \text{ if } i = 0 \text{ and } q_j \text{ otherwise}) \\
\Gamma_{2i+1,j} \triangleright \Gamma_{2i+2,j} \vdash^{\mathcal{B}}_P u_j : q_j \triangleright q_j \\
\Gamma_{2i+1,j} \subseteq \mathbf{Expand}^{\ell}_{\mathcal{G},\mathcal{B}}(\bigcup_{i' \le 2i} \Gamma_{i',j}) \\
\vdash^{\mathcal{B}} \mathcal{R} : \Theta_j \wedge \Gamma_{2m} \subseteq \Theta_j \subseteq \mathbf{Expand}^{\ell}_{\mathcal{G},\mathcal{B}}(\bigcup_{j \le 2m} \Gamma_j) \\
\Gamma_{2m_j-1,j} = \Gamma_{2m_j,j}
\end{array}$$

for every $j \in \{1, \ldots, k\}$. Let $m$ be $max(m_1, \ldots, m_k) + 1$ and let $\Gamma_{i,j}$ be $\Gamma_{2m_j,j}$ for every $2m_j < i \le 2m - 2$. Then, the required conditions hold for:

$$\begin{array}{l}
\Gamma_0 = \Gamma_1 = \emptyset \\
\Gamma_2 = F : \top \to \cdots \to \top \to q \\
\Gamma_{i+2} = \bigcup_{j \in \{1,\ldots,k\}} \Gamma_{i,j}, F : q_1 \to \cdots \to q_k \to q \ \ (i \in \{1, \ldots, 2m-2\}) \\
\Theta = \{F : q_1 \to \cdots \to q_k \to q\} \cup (\bigcup_{j \in \{1,\ldots,k\}} \Theta_j)
\end{array}$$

Note that the initial sequence of derivations is obtained by:

$$\frac{\emptyset \triangleright \emptyset \vdash^{\mathcal{B}}_O F : \top \triangleright (\top \to \cdots \to \top \to q)}{\Gamma_0 \triangleright \Gamma_1 \vdash^{\mathcal{B}}_O F\,s_1 \cdots s_k : \top \triangleright q}$$

$$\frac{\emptyset \triangleright (F : \top \to \cdots \to \top \to q) \vdash_P^{\mathcal{B}} F : (\top \to \cdots \to \top \to q) \triangleright (\top \to \cdots \to \top \to q)}{\Gamma_1 \triangleright \Gamma_2 \vdash_P^{\mathcal{B}} F\, s_1 \, \cdots \, s_k : q \triangleright q}$$

$$\frac{\begin{array}{c}(F : \top \to \cdots \to \top \to q) \triangleright (F : q_1 \to \cdots \to q_k \to q) \vdash_O^{\mathcal{B}} \\ \quad F : (\top \to \cdots \to \top \to q) \triangleright (\top \to \cdots \to \top \to q) \\ \Gamma_{0,j} \triangleright \Gamma_{1,j} \vdash_O^{\mathcal{B}} s_j : \top \triangleright q_j \text{ for each } j \in \{1, \ldots, k\}\end{array}}{\Gamma_2 \triangleright \Gamma_3 \vdash_O^{\mathcal{B}} F\, s_1 \, \cdots \, s_k : q \triangleright q}$$

$$\frac{\begin{array}{c}(F : q_1 \to \cdots \to q_k \to q) \triangleright (F : q_1 \to \cdots \to q_k \to q) \vdash_P^{\mathcal{B}} \\ \quad F : (\top \to \cdots \to \top \to q) \triangleright (q_1 \to \cdots \to q_k \to q) \\ \Gamma_{1,j} \triangleright \Gamma_{2,j} \vdash_P^{\mathcal{B}} s_j : q_j \triangleright q_j \text{ for each } j \in \{1, \ldots, k\}\end{array}}{\Gamma_3 \triangleright \Gamma_4 \vdash_P^{\mathcal{B}} F\, s_1 \, \cdots \, s_k : q \triangleright q}$$

If $u$ consists only of non-terminal symbols, then by the induction hypothesis, we have:

$$\begin{array}{l}\Gamma'_{2i} \triangleright \Gamma'_{2i+1} \vdash_O^{\mathcal{B}} [s_1/x_1, \ldots, s_k/x_k]u : q^? \triangleright q \\ \Gamma'_{2i+1} \triangleright \Gamma'_{2i+2} \vdash_P^{\mathcal{B}} [s_1/x_1, \ldots, s_k/x_k]u : q \triangleright q \\ \Gamma'_{2i+1} \subseteq \mathbf{Expand}^{\ell}_{\mathcal{G},\mathcal{B}}(\bigcup_{j \le 2i} \Gamma'_j) \\ \vdash^{\mathcal{B}} \mathcal{R} : \Theta' \\ \Gamma'_{2m'} \subseteq \Theta' \subseteq \mathbf{Expand}^{\ell}_{\mathcal{G},\mathcal{B}}(\bigcup_{j \le 2m'} \Gamma'_j) \\ \Gamma'_{2m'} = \Gamma'_{2m'-1}\end{array}$$

By Lemma 6, we have

$$\begin{array}{l}(\Gamma'_{2i,1}, \widetilde{x} : \widetilde{\tau}_{2i}) \triangleright (\Gamma'_{2i+1,1}, \widetilde{x} : \widetilde{\tau}_{2i}) \vdash_O^{\mathcal{B}} u : q^? \triangleright q \\ \Gamma'_{2i,2} \triangleright \Gamma'_{2i+1,2} \vdash_O^{\mathcal{B}} \widetilde{s} : \widetilde{\tau}_{2i} \triangleright \widetilde{\tau}_{2i+1} \\ (\Gamma'_{2i+1,1}, \widetilde{x} : \widetilde{\tau}_{2i+2}) \triangleright (\Gamma'_{2i+2,1}, \widetilde{x} : \widetilde{\tau}_{2i+2}) \vdash_P^{\mathcal{B}} u : q \triangleright q \\ \Gamma'_{2i+1,2} \triangleright \Gamma'_{2i+2,2} \vdash_P^{\mathcal{B}} \widetilde{s} : \widetilde{\tau}_{2i+1} \triangleright \widetilde{\tau}_{2i+2} \\ \Gamma'_i = \Gamma'_{i,1} \cup \Gamma'_{i,2}\end{array}$$

Here, $\Gamma_1 \triangleright \Gamma_2 \vdash_P^{\mathcal{B}} \widetilde{s} : \widetilde{\tau} \triangleright \widetilde{\tau}'$ means that there exist $\Gamma_{1,i}$ and $\Gamma_{2,i}$ $(i \in \{1, \ldots, k\})$ such that $\Gamma_{1,i} \triangleright \Gamma_{2,i} \vdash_P^{\mathcal{B}} s_i : \tau_i \triangleright \tau'_i$ for each $i \in \{1, \ldots, k\}$ with $\Gamma_1 = \Gamma_{1,1} \cup \cdots \cup \Gamma_{1,k}$ and $\Gamma_2 = \Gamma_{2,1} \cup \cdots \cup \Gamma_{2,k}$. Similarly for $\Gamma_1 \triangleright \Gamma_2 \vdash_O^{\mathcal{B}} \widetilde{s} : \widetilde{\tau} \triangleright \widetilde{\tau}'$.

Furthermore, by the construction of such derivations in the proof of Lemma 6, the derivations for $\Gamma'_{2i,2} \triangleright \Gamma'_{2i+1,2} \vdash_O^{\mathcal{B}} \widetilde{s} : \widetilde{\tau}_{2i} \triangleright \widetilde{\tau}_{2i+1}$ and $\Gamma'_{2i+1,2} \triangleright \Gamma'_{2i+2,2} \vdash_P^{\mathcal{B}} \widetilde{s} : \widetilde{\tau}_{2i+1} \triangleright \widetilde{\tau}_{2i+2}$ are consistent.

Now, by the consistency condition, $\widetilde{\tau}_{2i}, \widetilde{\tau}_{2i+1}, \widetilde{\tau}_{2i+2}$ can be divided into two parts:

$$\begin{array}{l}\widetilde{\tau}_{2i} = \widetilde{\tau}_{2i,1} \wedge \widetilde{\tau}_{2i,2} \\ \widetilde{\tau}_{2i+1} = \widetilde{\tau}_{2i+1,1} \wedge \widetilde{\tau}_{2i+1,2} \\ \widetilde{\tau}_{2i+2} = \widetilde{\tau}_{2i+2,1} \wedge \widetilde{\tau}_{2i+2,2} \\ \widetilde{\tau}_{2i,1} \preceq_O \widetilde{\tau}_{2i+1,1} = \widetilde{\tau}_{2i+2,1} \\ \widetilde{\tau}_{2i,2} = \widetilde{\tau}_{2i+1,2} \preceq_P \widetilde{\tau}_{2i+2,2}\end{array}$$

Thus, from $(\Gamma'_{2i+1,1}, \widetilde{x} : \widetilde{\tau}_{2i+2}) \triangleright (\Gamma'_{2i+2,1}, \widetilde{x} : \widetilde{\tau}_{2i+2}) \vdash_P^{\mathcal{B}} u : q \triangleright q$, we obtain

$$(\Gamma'_{2i+1,1}, \widetilde{x} : \widetilde{\tau}_{2i,1} \wedge \widetilde{\tau}_{2i+2,2}) \triangleright (\Gamma'_{2i+2,1}, \widetilde{x} : \widetilde{\tau}_{2i+2}) \vdash_P^{\mathcal{B}} u : q \triangleright q.$$

Therefore, we have:

$$\Gamma'_{2i+1,1} \triangleright \Gamma'_{2i+2,1} \vdash^{\mathcal{B}}_{P} \lambda\widetilde{x}.u : (\widetilde{\tau}_{2i,1} \wedge \widetilde{\tau}_{2i+2,2} \to q) \triangleright (\widetilde{\tau}_{2i+2} \to q) \tag{1}$$

The derivations for $\widetilde{s}$ can also be split as follows.

$$\begin{aligned}
\Gamma'_{2i,2,1} \triangleright \Gamma'_{2i+1,2,1} &\vdash^{\mathcal{B}}_{O} \widetilde{s} : \widetilde{\tau}_{2i,1} \triangleright \widetilde{\tau}_{2i+1,1} && (2)\\
\Gamma'_{2i,2,2} \triangleright \Gamma'_{2i+1,2,2} &\vdash^{\mathcal{B}}_{O} \widetilde{s} : \widetilde{\tau}_{2i,2} \triangleright \widetilde{\tau}_{2i+1,2} && (3)\\
\Gamma'_{2i+1,2,1} \triangleright \Gamma'_{2i+2,2,1} &\vdash^{\mathcal{B}}_{P} \widetilde{s} : \widetilde{\tau}_{2i+1,1} \triangleright \widetilde{\tau}_{2i+2,1} && (4)\\
\Gamma'_{2i+1,2,2} \triangleright \Gamma'_{2i+2,2,2} &\vdash^{\mathcal{B}}_{P} \widetilde{s} : \widetilde{\tau}_{2i+1,2} \triangleright \widetilde{\tau}_{2i+2,2} && (5)\\
\Gamma'_{2i,2} &= \Gamma'_{2i,2,1} \cup \Gamma'_{2i,2,2} && (6)\\
\Gamma'_{2i+1,2} &= \Gamma'_{2i+1,2,1} \cup \Gamma'_{2i+1,2,2} && (7)
\end{aligned}$$

From these, we obtain:

*(i)* $(\Gamma'_{2i,2,1} \cup \Gamma'_{2i,2,2}, F : \tau_{F,2i}) \triangleright (\Gamma'_{2i,2,1} \cup \Gamma'_{2i+1,2,2}, F : \tau_{F,2i}) \vdash^{\mathcal{B}}_{O} F\,\widetilde{s} : q^{?} \triangleright q$
*(ii)* $(\Gamma'_{2i,2,1} \cup \Gamma'_{2i+1,2,2}, F : \tau_{F,2i})$
$\quad \triangleright(\Gamma'_{2i,2,1} \cup \Gamma'_{2i+2,2,2}, F : \widetilde{\tau}_{2i,1} \wedge \widetilde{\tau}_{2i+2,2} \to q) \vdash^{\mathcal{B}}_{P} F\,\widetilde{s} : q \triangleright q$
*(iii)* $(\Gamma'_{2i,2,1} \cup \Gamma'_{2i+2,2,2}, F : \widetilde{\tau}_{2i,1} \wedge \widetilde{\tau}_{2i+2,2} \to q)$
$\quad \triangleright(\Gamma'_{2i+1,2,1} \cup \Gamma'_{2i+2,2,2}, F : \widetilde{\tau}_{2i+2} \to q) \vdash^{\mathcal{B}}_{O} F\,\widetilde{s} : q \triangleright q$
*(iv)* $(\Gamma'_{2i+1,2,1} \cup \Gamma'_{2i+2,2,2}, F : \widetilde{\tau}_{2i+2} \to q) \triangleright (\Gamma'_{2i+2,2,2}, F : \widetilde{\tau}_{2i+2} \to q) \vdash^{\mathcal{B}}_{P} F\,\widetilde{s} : q \triangleright q$

Here, $\tau_{F,2i}$ is $\top$ if $i = 0$, and $\widetilde{\tau}_{2i} \to q$ otherwise. (i) is obtained from (3) (and $\Gamma'_{2i,2,1} \triangleright \Gamma'_{2i,2,1} \vdash^{\mathcal{B}}_{O} \widetilde{s} : \widetilde{\tau}_{2i,1} \triangleright \widetilde{\tau}_{2i,1}$). (ii) is obtained from (5) and $(F : \tau_{F,2i}) \triangleright (F : \widetilde{\tau}_{2i,1} \wedge \widetilde{\tau}_{2i+2,2} \to q) \vdash^{\mathcal{B}}_{P} F : (\widetilde{\tau}_{2i,1} \wedge \widetilde{\tau}_{2i+2,2} \to q) \triangleright (\widetilde{\tau}_{2i,1} \wedge \widetilde{\tau}_{2i+2,2} \to q)$. (iii) and (iv) follow from (2) and (4) respectively.

Now, let $m = 2m'$ and define $\Gamma_0, \dots, \Gamma_{2m}, \Theta$ by:

$$\begin{aligned}
&\Gamma_{4i} = \Gamma'_{2i,2}, F : \tau_{F,2i}\\
&\Gamma_{4i+1} = \Gamma'_{2i,2,1} \cup \Gamma'_{2i+1,2,2}, F : \tau_{F,2i}\\
&\Gamma_{4i+2} = \Gamma'_{2i,2,1} \cup \Gamma'_{2i+2,2,2}, F : \widetilde{\tau}_{2i,1} \wedge \widetilde{\tau}_{2i+2,2} \to q\\
&\Gamma_{4i+3} = \Gamma'_{2i+1,2,1} \cup \Gamma'_{2i+2,2,2}, F : \widetilde{\tau}_{2i+2} \to q\\
&\Theta = \Theta' \cup \{F : \widetilde{\tau}_{2m} \to q\}
\end{aligned}$$

By the above construction, we have consistent derivations for:

$$\begin{aligned}
&\Gamma_0 \triangleright \Gamma_1 \vdash^{\mathcal{B}}_{O} t : \top \triangleright q\\
&\Gamma_{2i} \triangleright \Gamma_{2i+1} \vdash^{\mathcal{B}}_{O} t : q \triangleright q \text{ (for each } i \in \{1, \dots, m-1\})\\
&\Gamma_{2i+1} \triangleright \Gamma_{2i+2} \vdash^{\mathcal{B}}_{P} t : q \triangleright q \text{ (for each } i \in \{0, \dots, m-1\})
\end{aligned}$$

Furthermore, $\Gamma_{2m-1} \triangleright_O \Gamma_{2m} \vdash^{\mathcal{B}}_{P} t : q \triangleright_P q$ is stable.

By the definition of $\Theta$ and $\Gamma_{2m}$, and the induction hypothesis $\Gamma'_{2m'} \subseteq \Theta'$, we have $\Gamma_{2m} = \Gamma_{4m'} = \Gamma'_{2m',2} \cup \{F : \widetilde{\tau}_{2m'} \to q\} \subseteq \Theta$. From (1), we obtain

$$\Gamma'_{2m'} \vdash^{\mathcal{B}}_{P} \lambda\widetilde{x}.u : \widetilde{\tau}_{2m'} \to q,$$

so that we have $\vdash \mathcal{R} : \Theta$.

It remains to check $\Gamma_{2i+1} \subseteq \mathbf{Expand}^{\ell}_{\mathcal{G},\mathcal{B}}(\bigcup_{j\le 2i}\Gamma_j)$ and $\Theta \subseteq \mathbf{Expand}^{\ell}_{\mathcal{G},\mathcal{B}}(\bigcup_{j\le 2m}\Gamma_j)$. We show that

$$\begin{array}{l}\Gamma'_{2i+1} \subseteq \mathbf{Expand}^{\ell}_{\mathcal{G},\mathcal{B}}(\bigcup_{j\le 4i}\Gamma_j)\\ \Gamma'_{2i+2,1} \cup \{F:\widetilde{\tau}_{2i+2} \to q\} \subseteq \mathbf{Expand}^{\ell}_{\mathcal{G},\mathcal{B}}(\bigcup_{j\le 4i+2}\Gamma_j)\end{array}$$

holds for every $i \in \{0,\dots,m'-1\}$ by induction on $i$. For the base case (i.e. when $i=0$),

$$\Gamma'_{2i+1} \subseteq \mathbf{Expand}^{\ell}_{\mathcal{G},\mathcal{B}}(\bigcup_{j\le 4i}\Gamma_j) = \mathbf{Expand}^{\ell}_{\mathcal{G},\mathcal{B}}(\emptyset)$$

follows immediately from the hypothesis $\Gamma'_{2i+1} \subseteq \mathbf{Expand}_{\mathcal{G},\mathcal{B}}(\bigcup_{j\le 2i}\Gamma'_j)$ of the outermost induction. $\Gamma'_{2i+2,1} \cup \{F:\widetilde{\tau}_{2i+2} \to q\} \subseteq \mathbf{Expand}^{\ell}_{\mathcal{G},\mathcal{B}}(\bigcup_{j\le 4i+2}\Gamma_j)$ (for $i=0$) then follows from (1).

For the induction step (i.e. when $i>0$), by the induction hypothesis, we have:

$$\Gamma'_{2i,1} \subseteq \mathbf{Expand}^{\ell}_{\mathcal{G},\mathcal{B}}(\bigcup_{j\le 4i-2}\Gamma_j)$$

and

$$\Gamma'_{2i-1} \subseteq \mathbf{Expand}^{\ell}_{\mathcal{G},\mathcal{B}}(\bigcup_{j\le 4i-4}\Gamma_j).$$

Thus, by using the outer induction hypothesis, we get

$$\Gamma'_{2i+1} \subseteq \mathbf{Expand}^{\ell}_{\mathcal{G},\mathcal{B}}(\bigcup_{j\le 2i}\Gamma'_j) \subseteq \mathbf{Expand}^{\ell}_{\mathcal{G},\mathcal{B}}(\bigcup_{j\le 4i}\Gamma_j).$$

By using (1), we obtain $\Gamma'_{2i+2,1} \cup \{F:\widetilde{\tau}_{2i+2} \to q\} \subseteq \mathbf{Expand}^{\ell}_{\mathcal{G},\mathcal{B}}(\bigcup_{j\le 4i+2}\Gamma_j)$ as required.

Thus, we have

$$\begin{array}{l}\Gamma_{4i+1} \subseteq \mathbf{Expand}^{\ell}_{\mathcal{G},\mathcal{B}}(\bigcup_{j\le 4i}\Gamma_j)\\ \Gamma_{4i+3} \subseteq \mathbf{Expand}^{\ell}_{\mathcal{G},\mathcal{B}}(\bigcup_{j\le 4i+2}\Gamma_j),\end{array}$$

i.e., $\Gamma_{2i+1} \subseteq \mathbf{Expand}^{\ell}_{\mathcal{G},\mathcal{B}}(\bigcup_{j\le 2i}\Gamma_j)$ for $0 \le i \le m-1$.

Finally, the above condition also implies $\bigcup_{j\le 2m'}\Gamma'_j \subseteq \mathbf{Expand}^{\ell}_{\mathcal{G},\mathcal{B}}(\bigcup_{j\le 2m}\Gamma_j)$, so that we have:

$$\Theta = \Theta' \cup \{F:\widetilde{\tau}_{2m} \to q\} \subseteq \mathbf{Expand}^{\ell}_{\mathcal{G},\mathcal{B}}(\bigcup_{j\le 2m'}\Gamma'_j) \cup \{F:\widetilde{\tau}_{2m} \to q\} \subseteq \mathbf{Expand}^{\ell}_{\mathcal{G},\mathcal{B}}(\bigcup_{j\le 2m}\Gamma_j)$$

as required. □

## D Optimization

As mentioned in the footnote of Section 5, the implementation of the new algorithm times out for the order-4 encodings of resource usage verification problems [12]. That is because the least-fixedpoint of $\mathbf{Expand}_{\mathcal{G},\mathcal{B}}$ is still too large for those order-4 recursion schemes.

One way to overcome that problem is to interleave the least and greatest fixedpoint computations (the 2nd and 3rd lines of the new algorithm). Let us consider a finite increasing sequence of type environments: $\Theta_1 \subseteq \Theta_2 \subseteq \cdots \subseteq \Theta_m (= \Gamma_{\mathbf{max}})$, and define $\mathbf{Expand}^{(j)}_{\mathcal{G},\mathcal{B}}$ by: $\mathbf{Expand}^{(j)}_{\mathcal{G},\mathcal{B}}(\Gamma) = \mathbf{Expand}_{\mathcal{G},\mathcal{B}}(\Gamma) \cap \Theta_j$. We modify the new algorithm as follows.

```
OPTIMIZED ALGORITHM:
1. $\Gamma_1$ := $\{S : q_0\}$; $j$ := 1;
2. Repeat $\Gamma_1$ := $\mathbf{Expand}^{(j)}_{\mathcal{G},\mathcal{B}}(\Gamma_1)$ until $\Gamma_1 = \mathbf{Expand}^{(j)}_{\mathcal{G},\mathcal{B}}(\Gamma_1)$;
3. $\Gamma_2$ := $\Gamma_1$;
4. Repeat $\Gamma_2$ := $\mathbf{Shrink}_{\mathcal{G},\mathcal{B}}(\Gamma_2)$ until $\Gamma = \mathbf{Shrink}_{\mathcal{G},\mathcal{B}}(\Gamma_2)$;
4. If $S : q_0 \in \Gamma_2$ then output ``yes''
   else if $j = m$ then output ``no'' else ($j$ := $j+1$; goto 2).
```

Since a fixedpoint of $\mathbf{Shrink}_{\mathcal{G},\mathcal{B}}$ is computed before the least fixedpoint of $\mathbf{Expand}_{\mathcal{G},\mathcal{B}}$ is computed, the optimized algorithm often terminates faster if $[\![\mathcal{G}]\!]$ is accepted by $\mathcal{B}$. (On the other hand, the refutation of the property does not get faster; we need to run a similar algorithm for the negation of the automaton.)

We have implemented the optimized algorithm, where $\Theta_i$ is defined by:

$$\Theta_i = \{F : \theta \mid \theta :: \mathit{sort}(F), \mathtt{width}(\theta) \leq i\}.$$

Here, $\mathtt{width}(\theta)$ is the largest size of intersections, defined by:

$$\mathtt{width}(q) = 1 \qquad \mathtt{width}(\theta_1 \wedge \cdots \wedge \theta_k \to \theta) = \mathit{max}(k, \mathtt{width}(\theta_1), \ldots, \mathtt{width}(\theta_k), \mathtt{width}(\theta))$$

Table 2 shows the running times for the optimized algorithm. `Twofiles-4`, `FileOcamlc-4`, and `Lock2-4` are order-4 recursion schemes obtained by using the original encoding of resource usage verification problems [12]. We can observe good speed-ups for those order-4 recursion schemes (for which the unoptimized version times out) and for `Lock2`.

| | order | opt (sec.) |
|---|---|---|
| $\mathcal{G}_{4,10}$ | 4 | 13.580 |
| `Example1` | 2 | 0.001 |
| `Twofiles` | 3 | 0.145 |
| `FileOcamlc` | 3 | 0.486 |
| `Lock2` | 3 | 4.794 |
| `Nondet` | 3 | 0.001 |
| `Twofiles-4` | 4 | 7.142 |
| `FileOcamlc-4` | 4 | 1.737 |
| `Lock2-4` | 4 | 22.142 |

**Table 2.** Benchmark results for an optimized version