Higher-Order Model Checking by Abstraction of Type Derivation Rewriting

Takeshi Tsukada[1,2] and Naoki Kobayashi[3]

[1] Graduate School of Information Science, Tohoku University
[2] JSPS Research Fellow
[3] University of Tokyo

Abstract. Model checking of recursion schemes, known as higher-order model checking, has actively been studied recently and is becoming a basis for verification of higher-order programs. Some practical model checking algorithms have been developed based on the reduction from model checking to intersection type inference, but they are not completely satisfactory. We propose a novel approach to developing model checking algorithms, based on abstract interpretation of a rewriting system on (incomplete) derivations in a "rigid" intersection type system. The derivation rewriting can be regarded as a simple type inference process that is sound and complete in a certain sense. However it is not practical since the state space (i.e. the set of all incomplete derivations) is too large. We construct an abstraction of the rewriting system, and develop an algorithm based on the abstract model. We have implemented the algorithm and confirmed that it outperforms previous model checkers in certain cases.

1 Introduction

The model-checking of higher-order recursion schemes (called higher-order model checking below) [6, 15], has been actively studied recently. A higher-order recursion scheme (a recursion scheme, for short) is a grammar generating a possibly infinite tree, and higher-order model checking is the problem to decide whether the tree generated by a given recursion scheme satisfies a given property (expressed by a formula of the modal μ-calculus or a tree automaton). Since recursion schemes can be regarded as simply-typed lambda terms with recursion and tree constructors, they are natural models of higher-order programs. Higher-order model checking has been applied to verification of higher-order programs [8, 11, 12, 16].

A significant challenge in applying higher-order model checking to program verification is to develop an efficient model-checking algorithm that works in practice. Despite the huge worst-case complexity of higher-order model checking (k-EXPTIME complete for order-k recursion schemes [15]), several practical algorithms [7, 9, 13] have been developed, which work reasonably well for typical inputs. Though TRecS [7], the first implementation of a higher-order model checker, has been successfully used as the back-end of verification tools

for higher-order programs [11, 12], it has shown scalability only up to hundreds of lines of recursion schemes. Since the worst-case complexity of the underlying algorithm is hyper-exponential in the program size, it is unlikely that TRecS scales to handle millions of lines of recursion schemes. To remedy the problem, Kobayashi [9] proposed another algorithm (called GTRecS) that works in time linear in the size of recursion schemes under certain assumptions. The algorithm is, however, slower than TRecS in practice, except for some pathological cases. Neatherway et al. [13] have recently proposed yet another algorithm, but like TRecS, its worst-case behaviour is hyper-exponential in the size of recursion schemes.

The goal of the present paper is to develop a higher-order model checking algorithm that is linear time in the size of recursion schemes but runs much faster than GTRecS [9]. For that purpose, we introduce a new approach based on the notion of *type derivation rewriting*. Actually, all the previous algorithms mentioned above can be viewed as intersection type inference algorithms for higher-order recursive programs, and ours can be considered a new approach to intersection type inference, where a type derivation tree is incrementally constructed by rewriting an incomplete derivation tree step by step. This rewriting process is infinite in general, but by applying an abstraction, we can obtain an *over-approximation* of the set of intersection types required for typing recursion schemes. We can then apply a fixed-point algorithm [9] to obtain valid intersection types. Advantages of the approach are: (i) the algorithm runs in time linear in the size of recursion schemes as in [9], (ii) the incremental construction of derivation trees avoids duplicated computation in the process of intersection type inference (which was one of the main sources of inefficiency of the previous algorithm [9]), and (iii) we can obtain a *family* of model checking algorithms with varying degrees of overapproximation and convergence speed, by modifying the abstraction function. We have implemented a new model checker GTRecS2 based on the proposed approach, and confirmed by experiments that it outperforms a previous linear-time algorithm GTRecS [9], and also outperforms other higher-order model checkers [7, 13] in certain cases.

2 Review of Higher-Order Model Checking

In this section, we review higher-order model checking and its connection to intersection types [8].

We first define simple types with a unique base type o, called *sorts* in the paper in order to avoid confusion with intersection types introduced later. The set of sorts, ranged over by meta-variables A and B, are defined by: $A, B ::= o \mid A \to B$. The *order* and *arity* of sort A is defined by:

$$\begin{array}{ll} \mathrm{order}(o) = 0 & \mathrm{order}(A \to B) = \max\{1 + \mathrm{order}(A), \mathrm{order}(B)\} \\ \mathrm{arity}(o) = 0 & \mathrm{arity}(A \to B) = 1 + \mathrm{arity}(B). \end{array}$$

Assume a countably infinite set Var^A of variables for each sort A. If $x \in \mathrm{Var}^A$, we write x^A and $x :: A$. The set Term^A of *applicative terms of sort* A is inductively

defined by: (1) $x^A \in \mathrm{Term}^A$, (2) if $M \in \mathrm{Term}^{A\to B}$ and $N \in \mathrm{Term}^A$, then $M\ N \in \mathrm{Term}^B$. We write M^A and $M :: A$ if $M \in \mathrm{Term}^A$. For a term M, the set $\mathrm{fv}(M)$ of free variables of M is defined as usual. The order and arity of a term are those of its sort.

Let Σ be a finite set of variables of order 0 or 1, called *terminals*. Although terminals are defined as variables, they are treated as uninterpreted symbols or constants in reduction semantics. Let $\bot^o \notin \Sigma$ be a distinguished variable, and $\Sigma^\bot$ be $\Sigma \cup \{\bot\}$. Let m be the maximal arity of terminals. We write $\mathrm{dom}(f)$ for the domain of f. A *$\Sigma^\bot$-labelled tree* is a partial function $T : \{1, 2, \ldots, m\}^* \rightharpoonup \Sigma^\bot$ (where X^* is the set of all finite sequences of elements of X) such that (1) $\epsilon \in \mathrm{dom}(T)$, (2) if $p \in \mathrm{dom}(T)$ and $T(p) = a$ and the arity of a is n, then $p\,i \in \mathrm{dom}(T)$ if and only if $1 \le i \le n$. For $\Sigma^\bot$-labelled trees T and T', we write $T \sqsubseteq T'$ if and only if $\mathrm{dom}(T) \subseteq \mathrm{dom}(T')$ and for all $p \in \mathrm{dom}(T)$, $T(p) = \bot$ or $T(p) = T'(p)$.

A *higher-order recursion scheme* (*recursion scheme* for short) is a quadruple $\mathcal{G} = (\Sigma, \mathcal{N}, \mathcal{R}, S)$, where Σ is a set of terminals, $\mathcal{N}$ is a finite set of variables called *non-terminals*, $\mathcal{R}$ is a set of *rewriting rules*, and S is a distinguished non-terminal of sort o called the *start symbol*. We assume Σ and $\mathcal{N}$ are disjoint. Each rule in $\mathcal{R}$ is of the form $F\,x_1\,\ldots\,x_n \to M$, where $F :: A_1 \to \cdots \to A_n \to o$, $x_i :: A_i$ $(1 \le i \le n)$ and M is an applicative term of sort o that satisfies $\mathrm{fv}(M) \subseteq \Sigma \cup \mathcal{N} \cup \{x_1, \ldots, x_n\}$. We write $\mathcal{R}(F) = \lambda\widetilde{x}.M$ if $F\ x_1\ \ldots\ x_n \to M \in \mathcal{R}$, where $\lambda\widetilde{x}$ is an abbreviation for $\lambda x_1 \ldots x_n$. We use a and b for terminals and F and G for non-terminals. Given a recursion scheme $\mathcal{G}$, the rewriting relation $\longrightarrow_\mathcal{G}$ is the least relation that satisfies: (1) $F\ N_1\ \ldots\ N_k \longrightarrow_\mathcal{G} M[N_1/x_1, \ldots, N_k/x_k]$ if $\mathcal{R}(F) = \lambda x_1 \ldots x_k.M$ (here $M[N_1/x_1, \ldots, N_k/x_k]$ means the simultaneous substitution of N_i for x_i in M), and (2) $M \longrightarrow_\mathcal{G} M'$ implies $N\ M \longrightarrow_\mathcal{G} N\ M'$ and $M\ N \longrightarrow_\mathcal{G} M'\ N$ for all N. For a term M of sort o that satisfies $\mathrm{fv}(M) \subseteq \Sigma \cup \mathcal{N}$, we write $M^\bot$ for the $\Sigma^\bot$-labelled tree defined by: (1) $(a\,M_1 \ldots M_n)^\bot = a\,M_1^\bot \ldots M_n^\bot$ if $a \in \Sigma$, and (2) $(F\,M_1 \ldots M_n)^\bot = \bot$ if $F \in \mathcal{N}$. The *value tree* of $\mathcal{G}$, written $[\![\mathcal{G}]\!]$, is a (possibly infinite) $\Sigma^\bot$-labelled tree defined as the least upper bound of $\{M^\bot \mid S \longrightarrow_\mathcal{G}^* M\}$ with respect to $\sqsubseteq$, where $\longrightarrow_\mathcal{G}^*$ is the reflexive and transitive closure of $\longrightarrow_\mathcal{G}$.

A *trivial automaton* $\mathcal{A}$ is a quadruple (Σ, Q, δ, q_0), where Σ is the set of terminals, Q is a finite set of states, $\delta \subseteq Q \times \Sigma \times Q^*$ is a transition relation, and $q_0 \in Q$ is the initial state. A Σ-labelled tree T is *accepted* by $\mathcal{A}$ if there exists a (total) function $\varrho : \mathrm{dom}(T) \to Q$ (called a *run tree*) such that (1) $\varrho(\epsilon) = q_0$ and (2) for every $p \in \mathrm{dom}(T)$, $(\varrho(p), T(p), \varrho(p\,1) \ldots \varrho(p\,n)) \in \delta$ where n is the arity of $T(p)$. For a trivial automaton $\mathcal{A} = (\Sigma, Q, \delta, q_0)$, we write $\mathcal{A}^\bot$ for $(\Sigma^\bot, Q, \delta \cup \{(q, \bot, \epsilon) \mid q \in Q\}, q_0)$.

The *model checking of recursion schemes against trivial automata* is the problem to decide, given a recursion scheme $\mathcal{G}$ and a trivial automaton $\mathcal{A}$, whether $[\![\mathcal{G}]\!]$ is accepted by $\mathcal{A}^\bot$. In the sequel, the problem is simply called *higher-order model checking*. In the following, we assume that $[\![\mathcal{G}]\!]$ does not contain $\bot$. This can be assumed without loss of generality [8].

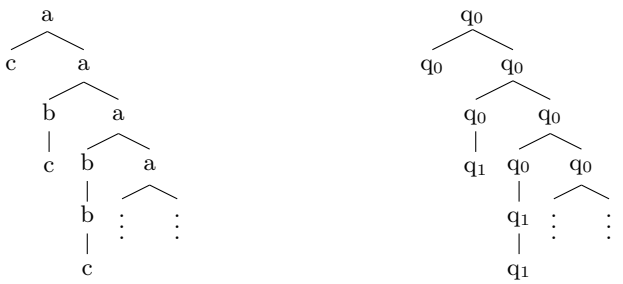

Fig. 1. The value tree $[\![\mathcal{G}_0]\!]$ and a run tree

Example 1. Let $\mathcal{G}_0$ be the recursion scheme $(\Sigma, \{S^o, F^{o\to o}\}, \mathcal{R}, S)$ where $\Sigma = \{a^{o\to o\to o}, b^{o\to o}, c^o\}$ and $\mathcal{R}$ is given by $\mathcal{R} = \{S \to F\,c,\ F\,x \to a\,x\,(F\,(b\,x))\}$. Then

$$S \longrightarrow_{\mathcal{G}_0} F\,c \longrightarrow_{\mathcal{G}_0} a\,c\,(F\,(b\,c)) \longrightarrow_{\mathcal{G}_0} a\,c\,(a\,(b\,c)\,(F\,(b\,(b\,c))) \longrightarrow_{\mathcal{G}_0} \cdots$$

and the value tree $[\![\mathcal{G}_0]\!]$ is shown in Fig. 1. Consider the trivial automaton $\mathcal{A}_0 = (\Sigma, \{q_0, q_1\}, \delta, q_0)$, where $\delta = \{(q_0, a, q_0q_0), (q_0, b, q_1), (q_1, b, q_1), (q_0, c, \epsilon), (q_1, c, \epsilon)\}$. The automaton accepts a tree in which for every node labelled by a, its ancestor is not labelled by b. Thus $\mathcal{A}_0$ accepts $[\![\mathcal{G}_0]\!]$. A run tree is shown in Fig. 1. □

Higher-order model checking can be reduced to a type checking problem of an intersection type system [8]. Let $\mathcal{G} = (\Sigma, \mathcal{N}, \mathcal{R}, S)$ be a recursion scheme and $\mathcal{A} = (\Sigma, Q, \delta, q_0)$ be a trivial automaton, fixed in the following. Let q be a metavariable ranging over Q. The set of *intersection types* are defined by:

$$\begin{array}{lll} \textit{Types} & \phi, \psi & ::= \ q \mid \overline{\phi} \to \psi \\ \textit{Intersections} & \overline{\phi}, \overline{\psi} & ::= \ \omega \mid \phi \mid \overline{\phi} \wedge \overline{\psi} \end{array}$$

The type operator $\wedge$ is called an *intersection operator*. Intuitively $M : \phi \wedge \psi$ means that M has types ϕ and ψ, i.e. $M : \phi$ and $M : \psi$. The intersection operator $\wedge$ is associative, commutative and idempotent, i.e. for any intersections $\overline{\phi}_1$, $\overline{\phi}_2$ and $\overline{\phi}_3$, we have $\overline{\phi}_1 \wedge \overline{\phi}_1 = \overline{\phi}_1$, $\overline{\phi}_1 \wedge \overline{\phi}_2 = \overline{\phi}_2 \wedge \overline{\phi}_1$ and $(\overline{\phi}_1 \wedge \overline{\phi}_2) \wedge \overline{\phi}_3 = \overline{\phi}_1 \wedge (\overline{\phi}_3 \wedge \overline{\phi}_3)$, and ω is the unit element of $\wedge$, i.e. $\omega \wedge \overline{\phi} = \overline{\phi} \wedge \omega = \overline{\phi}$ for any intersection $\overline{\phi}$. By these laws, one can write intersections as $\bigwedge\{\phi_1, \phi_2, \ldots, \phi_n\}$ (here $n \geq 0$) without any confusion. The *refinement relation* $\phi :: A$, which means the structure of type ϕ follows that of sort A, is defined by: (1) $q :: o$, and (2) $\bigwedge\{\phi_1, \ldots, \phi_n\} \to \psi \ :: \ A \to B$ if $\forall i \in \{1, \ldots, n\}$. $\phi_i :: A$ and $\psi :: B$.

A *type environment* is a finite set of bindings of the form $x^A : \bigwedge\{\phi_1, \ldots, \phi_n\}$ such that $\phi_i :: A$ for all $i \leq n$. We write $\mathrm{dom}(\Gamma) = \{x \mid x : \overline{\phi} \in \Gamma\}$. We define $\Gamma(x) = \bigwedge\{\phi_1, \ldots, \phi_n\}$ if $x : \bigwedge\{\phi_1, \ldots, \phi_n\} \in \Gamma$ and $\Gamma(x) = \omega$ otherwise. For type environments Γ and Δ, we define $\Gamma \wedge \Delta = \{x : \Gamma(x) \wedge \Delta(x) \mid x \in \mathrm{dom}(\Gamma) \cup \mathrm{dom}(\Delta)\}$. If $x : \bigwedge\{\phi_1, \ldots, \phi_n\} \in \Gamma$ and $1 \leq i \leq n$, we write $x : \phi_i \in \Gamma$. We often write $\{x : \phi_i \mid 1 \leq i \leq n\}$ for $\{x : \bigwedge\{\phi_1, \ldots, \phi_n\}\ \}$. Typing rules are listed below.

$$\frac{x : \phi \in \Gamma}{\Gamma \vdash x : \phi} \qquad \frac{\Gamma \vdash M : \overline{\phi} \to \psi \qquad \Delta \vdash N : \overline{\phi}}{\Gamma \wedge \Delta \vdash M\ N : \psi} \qquad \frac{\Delta_i \vdash M : \phi_i\ (\forall i \in \{1, \ldots, n\})}{\Delta_1 \wedge \cdots \wedge \Delta_n \vdash M : \bigwedge\{\phi_1, \ldots, \phi_n\}}$$

We write $\Gamma \vdash \mathcal{R}(F) : \overline{\phi}_1 \to \cdots \to \overline{\phi}_n \to q$ if $\mathcal{R}(F) = \lambda x_1 \ldots x_n.M$ and $\Gamma \cap \{x_1, \ldots, x_n\} = \emptyset$ and $\Gamma \wedge \{x_i : \overline{\phi}_i \mid 1 \leq i \leq n\} \vdash M : q$. For a type environment Γ such that $\mathrm{dom}(\Gamma) \subseteq \mathcal{N}$ and a type environment Θ such that $\mathrm{dom}(\Theta) \cap \mathcal{N} = \emptyset$, we write $\Theta \vdash (M, \mathcal{G}) : (\phi, \Gamma)$ if (1) $\Gamma \wedge \Theta \vdash M : \phi$ and (2) $\Gamma \wedge \Theta \vdash \mathcal{R}(F) : \psi$ for all $F : \psi \in \Gamma$. Here Γ describes types of non-terminals that are defined by mutual recursion and condition (2) requires that the recursion is well-typed.

A type binding $a : q_1 \to \cdots \to q_n \to q$ for a terminal *respects the transition relation* δ, written $\delta \vDash a : q_1 \to \cdots \to q_n \to q$, if $(q, a, q_1 \ldots q_n) \in \delta$. Given a type environment Θ such that $\mathrm{dom}(\Theta) \subseteq \Sigma$, we write $\delta \vDash \Theta$ if $\forall a : \phi \in \Theta.\ \delta \vDash a : \phi$.

The model checking problem is reduced to a type checking problem.

Theorem 1 (Kobayashi [8]). *Let $\mathcal{G} = (\Sigma, \mathcal{N}, \mathcal{R}, S)$ be a recursion scheme and $\mathcal{A} = (\Sigma, Q, \delta, q_0)$ be a trivial automaton. Then $[\![\mathcal{G}]\!]$ is accepted by $\mathcal{A}$ if and only if there exist Θ and Γ such that (1) $\Theta \vdash (S, \mathcal{G}) : (q_0, \Gamma)$ and (2) $\delta \vDash \Theta$.* □

Example 2. Consider $\mathcal{G}_0$ and $\mathcal{A}_0$ defined in Example 1. Let $\Theta = \{a : q_0 \to q_0 \to q_0, b : (q_1 \to q_0) \wedge (q_1 \to q_1), c : q_0 \wedge q_1\}$ and $\Gamma = \{S : q_0, F : (q_1 \wedge q_0) \to q_0\}$. Then $\Theta \vdash (S, \mathcal{G}) : (q_0, \Gamma)$ and $\delta \vDash \Theta$. □

By Theorem 1, the model checking problem is reduced to the problem to find a pair (Θ, Γ) of type environments for terminals and non-terminals. Without loss of generality, we can assume Θ to be $\Theta_\delta = \{a : q_1 \to \cdots \to q_n \to q \mid (q, a, q_1 \ldots q_n) \in \delta\}$. Thus the problem is now to find an appropriate type environment Γ for non-terminals. We call Γ in Theorem 1 a *certificate*.

For the purpose of developing a type checking (= model checking) algorithm, we further rephrase the above condition. Let us define the function $\mathbf{Shrink}_{\mathcal{G},\mathcal{A}}$ on type environments by:

$$\mathbf{Shrink}_{\mathcal{G},\mathcal{A}}(\Gamma) = \{F : \phi \in \Gamma \mid \exists \Theta.\ \Theta, \Gamma \vdash \mathcal{R}(F) : \phi \text{ and } \delta \vDash \Theta\}.$$

Then, the condition in the theorem above can be rephrased as the existence of a post-fixedpoint Γ of $\mathbf{Shrink}_{\mathcal{G},\mathcal{A}}$ such that $S : q_0 \in \Gamma$. Let $\mathbf{gfp}(\mathbf{Shrink}_{\mathcal{G},\mathcal{A}}, \Gamma)$ is the greatest fixedpoint of $\mathbf{Shrink}_{\mathcal{G},\mathcal{A}}$ smaller than Γ. Then the condition is further reduced to the existence of a type environment Γ such that $S : q_0 \in \mathbf{gfp}(\mathbf{Shrink}_{\mathcal{G},\mathcal{A}}, \Gamma)$. Here, $\mathbf{gfp}(\mathbf{Shrink}_{\mathcal{G},\mathcal{A}}, \Gamma)$ is the certificate mentioned above, and Γ is an over-approximation of it.

Since $\mathbf{gfp}(\mathbf{Shrink}_{\mathcal{G},\mathcal{A}}, \Gamma)$ can be easily computed if Γ is sufficiently small, all the previous algorithms [7, 9, 13] try to find such an over-approximation Γ by using various methods. TRecS [7] obtains it by partially reducing the start symbol S and observing how non-terminal symbols are used. GTRecS [9] obtains an over-approximation by gradually refining the types of non-terminals using some game-semantic intuitions. Neatherway et al. [13] obtains the exact certificate (without an approximation) by using the notion of traversals [15].

Our approach introduced in the rest of the paper can be considered yet another method to compute an over-approximation of the certificate, based on the notion of type derivation rewriting. As a preparation, we rephrase the condition $\Theta \vdash (S, \mathcal{G}) : (q_0, \Gamma)$ for the certificate. We add the following rule for

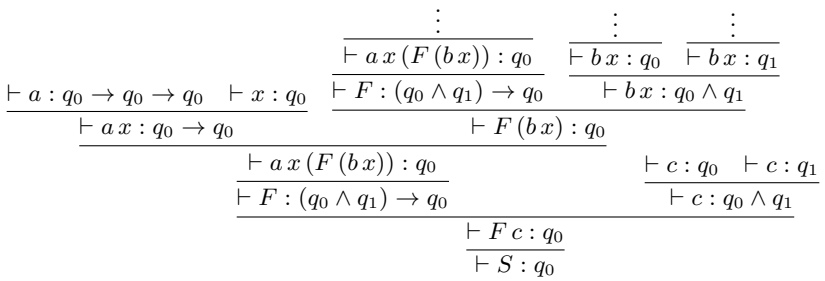

Fig. 2. An infinite derivation (here type environments are omitted)

non-terminals:

$$\frac{\Gamma \wedge \{x_i : \overline{\phi}_i \mid 1 \leq i \leq n\} \vdash M : q}{\Gamma \vdash F : \overline{\phi}_1 \to \cdots \to \overline{\phi}_n \to q} \left[\begin{array}{l}\mathcal{R}(F) = \lambda x_1 \ldots x_n.M \\ \mathrm{dom}(\Gamma) \cap \{x_1, \ldots, x_n\} = \emptyset\end{array}\right]$$

The rule expands the non-terminal and checks if the body has a required type. Then, $\Theta \vdash (S, \mathcal{G}) : (q_0, \Gamma)$ is equivalent to the existence of a possibly *infinite* type derivation whose conclusion is $\Theta_\delta \vdash S : q_0$. Figure 2 shows an example of such an infinite derivation tree. The certificate Γ is then the set of type bindings for non-terminals occurring in the derivation tree. Section 3 gives an algorithm for incrementally constructing such an infinite derivation tree (albeit for a different intersection type system) by type derivation rewriting, and Section 4 gives an abstraction of the derivation rewriting to accelerate the construction and ensure termination. The type environment obtained by the abstraction is an over-approximation of the types of non-terminals, but as explained above, we can apply the fixed-point computation to obtain the actual certificate.

3 Concrete Model: Derivation Rewriting

Given a recursion scheme $\mathcal{G}$ and a trivial tree automaton $\mathcal{A}$, we construct a rewriting system on (incomplete) derivations in an intersection type system. The possibly infinite type derivation mentioned at the end of Section 2 is obtained as the result of infinite rewriting.

3.1 Overview of Derivation Rewriting

We use an example to present the idea of derivation rewriting that will be formalised in the following subsections. Let us consider a term $F\,a$ with $\mathcal{R}(F) = \lambda g.g\,c$ and a trivial automaton $\mathcal{A}$ whose transition rule is $\delta = \{(q_0, a, q_1),\ (q_1, c, \epsilon)\}$, and check if the value tree of $F\,a$ is accepted by $\mathcal{A}$, i.e. if the term has type q_0. Figure 3 illustrates a part of the type inference process based on derivation rewriting.

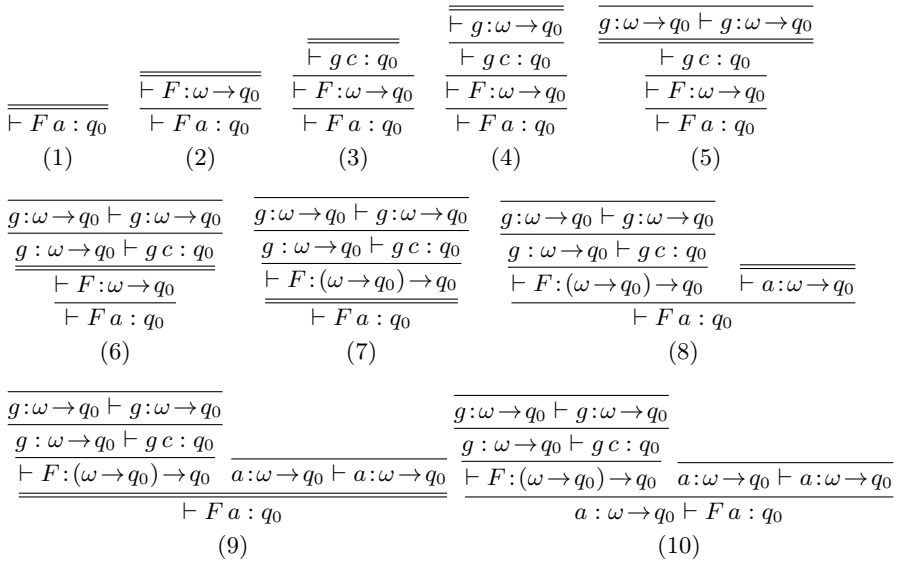

Fig. 3. Overview of derivation rewriting (here $\mathcal{R}(F) = \lambda g.g\,c$)

The type inference process starts from derivation (1) in Fig. 3. Unlike usual type systems, we allow derivations to have a certain kind of type mismatch. For example, derivation (1) concludes $\vdash F\,a : q_0$ without any premises, which is incorrect in the usual sense. We use a double line to indicate the position of a type mismatch. Derivation rewriting locally resolves the type mismatch.

In order to resolve the type mismatch in derivation (1), we need to find appropriate types for F and a. By the application rule, F must have type $\overline{\phi} \to q_0$ and a must have type $\overline{\phi}$ for some $\overline{\phi}$. Since we do not have any constraints on the argument of F at the moment, we choose $\overline{\phi}$ to be the empty intersection ω. In this way, derivation (1) is rewritten to derivation (2) (here we omit $\vdash a : \omega$ as it trivially holds). Now the type mismatch is found at the rule deriving $\vdash F : \omega \to q_0$. To resolve it, one expands the definition of F and checks if the body has the required type, resulting in derivation (3). The type mismatch in derivation (3) can be resolved in the same way as the one in derivation (1), and we obtain derivation (4). In this way, we went up the derivation and reached variable g.

In order for variable g to have type $\omega \to q_0$, the type environment must contain the assumption $g : \omega \to q_0$ (derivation (5)). We now propagate this update of the type environment downwards (derivation (6)). Because variable g is the (formal) argument of F, this update is propagated to the argument type of F (derivation (7)).

Now the type mismatch is at the application of F to a. F requires its argument to have type $\omega \to q_0$, while a has currently type ω (recall that $\vdash a : \omega$ is omitted in derivations (2)–(7)). So we update the type for a (derivation (8)) and in the same way as above, we obtain derivation (10) that has no type mismatch.

Unfortunately derivation (10) in Fig 3 is not satisfactory since the type environment does not respect the transition rule of $\mathcal{A}$. In order to make the type environment respect the transition rule, we need to replace the typing binding $a : \omega \to q_0$ with $a : q_1 \to q_0$ since $(q_0, a, q_1) \in \delta$. This update shall also be propagated to an appropriate position. After several steps of rewriting, one finally reaches a derivation of $a : q_1 \to q_0, c : q_1 \vdash F\, a : q_0$ that respects the transition rule and has no type mismatch.

3.2 Rigid Intersection Type System

The derivation rewriting above is actually formalised for another intersection type system called a "rigid" intersection type system, instead of the type system in Section 2 (see Remark 1 below). An intersection operator is *rigid* [14], if it is not associative, commutative nor idempotent. The intersection operator introduced in this section is rigid in this sense. Several rigid intersection type systems have been studied [5, 14, 3]. The rigid intersection type system in this section is essentially equivalent to the one of Di Gianantonio et al. [3], which has a formal connection to game semantics.

The new type system is also very similar to that in Section 2. In fact the only difference is whether or not the intersection operator is associative, commutative and idempotent. The intersection operator in Section 2 is referred to as a *flexible* intersection operator, in order to distinguish it from the rigid one.

Remark 1. Advantages of developing derivation rewriting for the rigid intersection type system can be listed as follows. First, it has some nice properties such as progress and determinacy of rewriting (Theorem 2). Secondly, we can use the connection to game semantics [4, 1] established by Di Gianantonio et al. [3] to prove properties such as Theorem 3. Thirdly, a derivation rewriting system for the *flexible* intersection type system can be obtained by abstraction of that for the rigid intersection type system, but we cannot do the converse. □

Let Q be a finite set. We use p and q as metavariables ranging over Q. The set of *rigid intersection types* over Q are defined by the following grammar.

$$\begin{array}{lll} \textit{Types} & \tau, \sigma & ::= \; q \mid \alpha \to \tau \\ \textit{Rigid Intersections} & \alpha, \beta & ::= \; \emptyset \mid \tau \mid \alpha \mathbin{⩓} \beta \end{array}$$

Here $\emptyset$ is a constant meaning the empty intersection. Rigid intersections are simply called intersections when it is not confusing. For a sequence of intersections $\widetilde{\alpha} = \alpha_1, \alpha_2, \ldots, \alpha_n$ and a type τ, we write $\widetilde{\alpha} \to \tau$ for $\alpha_1 \to \alpha_2 \to \cdots \to \alpha_n \to \tau$.

As we emphasised above, the intersection connective $⩓$ is not idempotent, commutative nor associative, i.e. in general $\alpha ⩓ \alpha \neq \alpha$, $\alpha ⩓ \beta \neq \beta ⩓ \alpha$ and $(\alpha_1 ⩓ \alpha_2) ⩓ \alpha_3 \neq \alpha_1 ⩓ (\alpha_2 ⩓ \alpha_3)$. The only exception is idempotency of $\emptyset$, and thus $\emptyset = \emptyset ⩓ \emptyset = (\emptyset ⩓ \emptyset) ⩓ \emptyset = \emptyset ⩓ (\emptyset ⩓ \emptyset)$. It is worth noting that $\emptyset$ is *not* a unit element of $⩓$, i.e. $\emptyset ⩓ \alpha \neq \alpha$.

An in the type system in Section 2, we consider only a type that follows the structure of a sort. For a sort A, we write $\tau :: A$ (resp. $\alpha :: A$) when type τ

$$\dfrac{\dfrac{\dagger}{a : q \to q \vdash a : q \to q}\ (1) \qquad \dfrac{\dfrac{\dagger}{a : q \to q \vdash a : q \to q}\ (2) \quad \dfrac{\dagger}{b : q \vdash b : q}}{a : (q \to q) \pitchfork \emptyset,\ b : \emptyset \pitchfork q \vdash a\, b : q}}{a : (q \to q)^3 \pitchfork ((q \to q)^4 \pitchfork \emptyset),\ b : \emptyset \pitchfork (\emptyset \pitchfork q) \vdash a\,(a\, b) : q}$$

Fig. 4. An example of a type derivation in the rigid intersection type system

(resp. intersection α) follows the structure of A. Formally $\tau :: A$ and $\alpha :: A$ are inductively defined by (1) $\emptyset :: A$ for any sort A, (2) $q :: o$, (3) $\alpha :: A$ and $\tau :: B$ implies $\alpha \to \tau :: A \to B$, and (4) $\alpha :: A$ and $\beta :: A$ implies $\alpha \pitchfork \beta :: A$.

A *type environment* is a finite set of bindings $\{x_1 : \alpha_1, x_2 : \alpha_2, \ldots, x_n : \alpha_n\}$ for pairwise distinct variables $x_1, \ldots, x_n$. For a sequence of variables $\widetilde{x} = x_1, x_2, \ldots, x_n$ and intersections $\widetilde{\alpha} = \alpha_1, \ldots, \alpha_n$, we write $\widetilde{x} : \widetilde{\alpha}$ for the type environment $\{x_i : \alpha_i \mid 1 \leq i \leq n\}$. A type environment Γ is *well-sorted* if $\alpha :: A$ holds for every binding $x^A : \alpha \in \Gamma$. By abuse of notation, we write $\emptyset$ for the empty type environment $\{\ \}$. The domain $\mathrm{dom}(\Gamma)$ of a type environment Γ is defined by $\mathrm{dom}(\Gamma) = \{x \mid \exists \alpha.\ x : \alpha \in \Gamma\}$. We write $\Gamma(x) = \alpha$ if $x : \alpha \in \Gamma$ and $\Gamma(x) = \emptyset$ if $x \notin \mathrm{dom}(\Gamma)$. Given type environments Γ and Δ, their intersection is defined by: $\Gamma \pitchfork \Delta = \{x : \Gamma(x) \pitchfork \Delta(x) \mid x \in \mathrm{dom}(\Gamma) \cup \mathrm{dom}(\Delta)\}$. If the domains of Γ and Δ are disjoint, i.e. $\mathrm{dom}(\Gamma) \cap \mathrm{dom}(\Delta) = \emptyset$, we simply write Γ, Δ for their intersection $\Gamma \pitchfork \Delta$. The typing rules are listed below:

$$\frac{\Gamma, \widetilde{x} : \widetilde{\alpha} \vdash M : q}{\Gamma \vdash F : \widetilde{\alpha} \to q}\ [\mathcal{R}(F) = \lambda \widetilde{x}.M^o] \qquad \frac{\dagger}{x : \tau \vdash x : \tau} \qquad \frac{\Diamond}{\emptyset \vdash M : \emptyset}$$

$$\frac{\Gamma \vdash M : \alpha \to \tau \qquad \Delta \vdash N : \alpha}{\Gamma \pitchfork \Delta \vdash M\ N : \tau} \qquad \frac{\Gamma \vdash M : \tau \qquad \Delta \vdash M : \sigma}{\Gamma \pitchfork \Delta \vdash M : \tau \pitchfork \sigma}\ [\tau \pitchfork \sigma \neq \emptyset]$$

where well-sortedness is required in the rule for variables, i.e. there exists sort A such that $x :: A$ and $\tau :: A$. Symbols $\dagger$ and $\Diamond$ are used to record which rule is used there. The side condition $\tau \pitchfork \sigma \neq \emptyset$ is for the technical convenience and it is easy to see that this condition is harmless.

The typing rules above are mostly standard, except for the rule for non-terminals, which expands the non-terminals and checks if the body has a required type, similar to the rule discussed at the end of Section 2. Note that we shall only consider *finite* derivations.

Figure 4 presents an example of a type derivation in the rigid intersection type system. Thanks to the rigidness of the intersection operator, we can distinguish between different occurrences of the same type. This is the reason why we employ the rigid intersection operator. For example, let us consider the derivation in Fig. 4. One can say that the occurrence of $q \to q$ labelled by 3 in the conclusion is used at the rule labelled by (1), and 4 in the conclusion is used at (2). The notion of occurrences plays an important role in the definition of derivation rewriting.

$$(2)\ \frac{}{\vdash F : \omega \to \check{q}_0} \qquad (3)\ \frac{}{\vdash g\, c : \check{q}_0} \qquad (4)\ \frac{}{\vdash g : \omega \to \check{q}_0} \qquad (5)\ \frac{g : \omega \to \check{q}_0 \vdash g : \omega \to q_0 \quad \vdash c : \omega}{\vdash g\, c : q_0}$$

$$(6)\ \frac{g : \omega \to \check{q}_0 \vdash g\, c : q_0}{\vdash F : \omega \to q_0} \qquad (7)\ \frac{\vdash F : (\omega \to \check{q}_0) \to q_0 \quad \vdash a : \omega}{\vdash F\, a : q_0} \qquad (8)\ \frac{}{\vdash a : \omega \to \check{q}_0}$$

$$(9)\ \frac{\vdash F : (\omega \to q_0) \to q_0 \quad a : \omega \to \check{q}_0 \vdash a : \omega \to q_0}{\vdash F\, a : q_0}$$

Fig. 5. Type mismatches seen in Fig. 3

3.3 Incomplete Derivations

We define *incomplete derivations*, which are used to represent intermediate steps of the type inference, like derivations (1) to (9) in Fig. 3. Roughly speaking, incomplete derivations are those with *exactly one* type mismatch, which is marked with a double line in Fig. 3. One may notice that type mismatches appearing in derivations in Fig. 3 are in certain special forms. We formalise them as *incomplete typing rules* and then define incomplete derivations as those with exactly one use of an incomplete typing rule.

We clarify what kind of type mismatch should be allowed to occur, by examining derivations in Fig. 3. Figure 5 shows type mismatches (i.e. rules marked with a double line) in Fig. 3. One may feel that rules in Fig. 5 closely resemble usual typing rules. In fact, removing an occurrence of a base type marked with check ($\check{}$) yields a usual typing rule. For example, removing the marked occurrences[4] from (2), (5) and (7) in Fig. 5 results in the following typing rules:

$$(2')\ \frac{}{\vdash F : \omega} \qquad (5')\ \frac{g : \omega \vdash g : \omega \to q_0 \quad \vdash c : \omega}{\vdash g\, c : q_0} \qquad (7')\ \frac{\vdash F : \omega \to q_0 \quad \vdash a : \omega}{\vdash F\, a : q_0}$$

So the difference between a type mismatch in Fig. 3 and a usual typing rule is just one occurrence of a base type. We then focus on the position of the difference. It is easy to see that the difference is at a covariant position in the conclusion (as in (2), (3), (4) and (8)), or at a contravariant position in a premise (as in (5), (6), (7) and (9)), where the position of a type environment in a judgement is regarded as contravariant, e.g. in the judgement $x : p_1 \to p_2 \vdash M : q_1 \to q_2$, positions of p_1 and q_2 are covariant and those of p_2 and q_1 are contravariant.

The above observation leads to the following definitions.

For types τ and σ, we write $\tau \lhd \sigma$ if removal of an occurrence of a base type in σ yields τ. If the occurrence is at a covariant position (resp. a contravariant position), we write $\tau \lhd_O \sigma$ (resp. $\tau \lhd_P \sigma$) and say that σ is an *O-extension* (resp. a *P-extension*) of τ. The relations $\lhd_O$ and $\lhd_P$ are called *extension relations*. Formally $\lhd_O$ and $\lhd_P$ on types (and on intersections) are inductively defined by

[4] Here we define the result of removal of $\check{q}_0$ from $\omega \to \check{q}_0$ is not $\omega \to \omega$ but ω, since $\omega \to \omega$ is syntactically invalid.

$$\frac{}{\emptyset^{A_1 \to \cdots \to A_n \to o} \lhd_O \emptyset^{A_1} \to \cdots \to \emptyset^{A_n} \to q}$$

$$\frac{\alpha \lhd_X \alpha'}{\alpha \Cap \beta \lhd_X \alpha' \Cap \beta} \qquad \frac{\beta \lhd_X \beta'}{\alpha \Cap \beta \lhd_X \alpha \Cap \beta'} \qquad \frac{\beta \lhd_X \beta'}{\alpha \to \beta \lhd_X \alpha \to \beta'} \qquad \frac{\alpha \lhd_{\overline{X}} \alpha'}{\alpha \to \beta \lhd_X \alpha' \to \beta}$$

Fig. 6. O- and P-extension of types (here $X \in \{O, P\}$, $\overline{O} = P$ and $\overline{P} = O$)

the rules in Fig. 6. Extension of a type environment is obtained by extending a type of one variable, leaving others unchanged. So $\Gamma \lhd_O \Delta$ if there exists a variable x such that $\Gamma(x) \lhd_O \Delta(x)$ and $\Gamma(y) = \Delta(y)$ for all $y \neq x$. We write $\Gamma \vdash M : \tau \ \lhd_P \ \Delta \vdash M : \sigma$ if (1) $\Gamma = \Delta$ and $\tau \lhd_P \sigma$, or (2) $\Gamma \lhd_O \Delta$ and $\tau = \sigma$ (recall that the position of Γ in $\Gamma \vdash M : \tau$ is contravariant). In the same way, $\Gamma \lhd_P \Delta$ and $\Gamma \vdash M : \tau \ \lhd_O \ \Delta \vdash M : \sigma$ are defined.

Remark 2. Extension of types and type judgements corresponds to extending a play in game semantics [1, 4]: O- and P-extensions respectively correspond to the extensions of a play by adding an O-move and a P-move. For more details, see the full version of the paper. A variant of extension relations can be found in [9], which is also inspired by game semantics. □

An *incomplete typing rule* is obtained by replacing the conclusion of a typing rule to its O-extension (the resulting typing rules are called *O-rules*), or by replacing a premise of a typing rule to its P-extension (the resulting typing rules are called *P-rules*). The list of O- and P-rules is shown in Fig. 7. Typing rules in the previous subsection are now called *complete typing rules*. We use double lines for incomplete typing rules, in order to distinguish them from complete typing rules.

An *incomplete derivation* is one derived from complete typing rules with *exactly one* use of an incomplete rule. A derivation that uses only complete rules is called a *complete derivation*.

We introduce marks ▲ and ▼ to express which judgement has been extended. For example, since (O-App) rule is obtained by replacing the conclusion of (App) rule with its O-extension, we add ▲ the conclusion $\Gamma' \Cap \Delta' \vdash M\ N : \tau$ and write an instance of (O-App) rule as:

$$\frac{\Gamma \vdash M : \alpha \to \tau \qquad \Delta \vdash N : \alpha}{\Gamma' \Cap \Delta' \vdash M\ N : \tau' \ \blacktriangle}.$$

Note that these marks are used just for readability. Each instance of an incomplete typing rule has a unique annotation that can be easily reconstructed.

For the technical convenience, we introduce a symbol $\star$ indicating the root of the derivation, which has the following typing rules:

$$\frac{\Gamma \vdash M : \tau}{\star}\ (\star) \qquad \frac{\Gamma \vdash M : \tau \ \blacktriangledown}{\star}\ (\text{P-}\star)$$

We require that the conclusion of derivations must be $\star$. By this assumption, every judgement in a derivation has its parent; in other words, every judgement

$$\text{(O-Var1)}\quad \frac{\dagger}{x:\tau \vdash x:\tau'} \qquad [\tau \lhd_O \tau']$$

$$\text{(O-Var2)}\quad \frac{\dagger}{x:\tau' \vdash x:\tau} \qquad [\tau \lhd_P \tau']$$

$$\text{(O-}\Diamond\text{)}\quad \frac{\Diamond}{\emptyset \vdash M:\tau} \qquad [\emptyset \lhd_O \tau]$$

$$\text{(O-Fun)}\quad \frac{\Gamma, \widetilde{x}:\widetilde{\alpha} \vdash M:q}{\Gamma' \vdash F:\widetilde{\alpha}' \to q'} \qquad \left[\begin{array}{l}\mathcal{R}(F) = \lambda\widetilde{x}.M^o \\ \Gamma \vdash F:\widetilde{\alpha} \to q \ \lhd_O\ \Gamma' \vdash F:\widetilde{\alpha}' \to q'\end{array}\right]$$

$$\text{(P-Fun)}\quad \frac{\Gamma', \widetilde{x}:\widetilde{\alpha}' \vdash M:q'}{\Gamma \vdash F:\widetilde{\alpha} \to \tau} \qquad \left[\begin{array}{l}\mathcal{R}(F) = \lambda\widetilde{x}.M^o \\ \Gamma, \widetilde{x}:\widetilde{\alpha} \vdash M:q \ \lhd_P\ \Gamma', \widetilde{x}:\widetilde{\alpha}' \vdash M:q'\end{array}\right]$$

$$\text{(O-App)}\quad \frac{\Gamma \vdash M:\alpha \to \tau \qquad \Delta \vdash N:\alpha}{\Gamma' \oplus \Delta' \vdash M\ N:\tau'} \qquad \left[\begin{array}{l}\Gamma \oplus \Delta \vdash M\ N:\tau \\ \quad \lhd_O\ \Gamma' \oplus \Delta' \vdash M\ N:\tau'\end{array}\right]$$

$$\text{(P-AppL)}\quad \frac{\Gamma' \vdash M:\alpha' \to \tau' \qquad \Delta \vdash N:\alpha}{\Gamma \oplus \Delta \vdash M\ N:\tau} \qquad \left[\begin{array}{l}\Gamma \vdash M:\alpha \to \tau \\ \quad \lhd_P\ \Gamma' \vdash M:\alpha' \to \tau'\end{array}\right]$$

$$\text{(P-AppR)}\quad \frac{\Gamma \vdash M:\alpha \to \tau \qquad \Delta' \vdash N:\alpha'}{\Gamma \oplus \Delta \vdash M\ N:\tau} \qquad [\Delta \vdash N:\alpha \ \lhd_P\ \Delta' \vdash N:\alpha']$$

$$\text{(O-Int)}\quad \frac{\Gamma \vdash M:\alpha \qquad \Delta \vdash M:\beta}{\Gamma' \oplus \Delta' \vdash M:\alpha' \oplus \beta'} \qquad \left[\begin{array}{l}\Gamma \oplus \Delta \vdash M:\alpha \oplus \beta \\ \quad \lhd_O\ \Gamma' \oplus \Delta' \vdash M:\alpha' \oplus \beta'\end{array}\right]$$

$$\text{(P-IntL)}\quad \frac{\Gamma' \vdash M:\alpha' \qquad \Delta \vdash M:\beta}{\Gamma \oplus \Delta \vdash M:\alpha \oplus \beta} \qquad [\Gamma \vdash M:\alpha \ \lhd_P\ \Gamma' \vdash M:\alpha']$$

$$\text{(P-IntL)}\quad \frac{\Gamma \vdash M:\alpha \qquad \Delta' \vdash M:\beta'}{\Gamma \oplus \Delta \vdash M:\alpha \oplus \beta} \qquad [\Delta \vdash M:\beta \ \lhd_P\ \Delta' \vdash M:\beta']$$

Fig. 7. Incomplete typing rules

$$(1)\quad \frac{\dfrac{\dagger}{a:\emptyset \to q \vdash a:\emptyset \to q} \quad \dfrac{\Diamond}{\emptyset \vdash b:\emptyset}}{\dfrac{a:(p \to q) \oplus \emptyset \vdash a\ b:q}{\star}\ \blacktriangle} \qquad (2)\quad \frac{\dfrac{\dfrac{\dagger}{a:p \to q \vdash a:\emptyset \to q}\ \blacktriangle \quad \dfrac{\Diamond}{\emptyset \vdash b:\emptyset}}{a:(p \to q) \oplus \emptyset \vdash a\ b:q}}{\star}$$

$$(3)\quad \frac{\dfrac{\dagger}{a:p \to q \vdash a:p \to q}\ \blacktriangledown \quad \dfrac{\Diamond}{\emptyset \vdash b:\emptyset}}{\dfrac{a:(p \to q) \oplus \emptyset \vdash a\ b:q}{\star}} \qquad (4)\quad \frac{\dfrac{\dagger}{a:p \to q \vdash a:p \to q} \quad \dfrac{\Diamond}{\emptyset \vdash b:p}\ \blacktriangle}{\dfrac{a:(p \to q) \oplus \emptyset \vdash a\ b:q}{\star}}$$

$$(5)\quad \frac{\dfrac{\dagger}{a:p \to q \vdash a:p \to q} \quad \dfrac{\dagger}{b:p \vdash b:p}\ \blacktriangledown}{\dfrac{a:(p \to q) \oplus \emptyset \vdash a\ b:q}{\star}} \qquad (6)\quad \frac{\dfrac{\dagger}{a:p \to q \vdash a:p \to q} \quad \dfrac{\dagger}{b:p \vdash b:p}}{\dfrac{a:(p \to q) \oplus \emptyset, b:\emptyset \oplus p \vdash a\ b:q}{\star}\ \blacktriangledown}$$

Fig. 8. Incomplete derivations and local resolutions of type mismatches

is used as a premise of a rule. Moreover we can assume without loss of generality that all derivations have exactly one use of a double-lined rule: a complete derivation is regarded as a derivation with (P-$\star$).

Figure 8 gives examples of incomplete derivations.

3.4 Derivation Rewriting

Here we provide a method to make an incomplete derivation complete, by iteration of local resolutions of type mismatches. For example, let us consider the incomplete derivation (1) in Fig. 8. In derivation (1), the type mismatch is found at the type for a in the environment: in the judgement marked with ▲, a is assumed to be a function taking an argument of type p, but in its left premise, a is a function taking an arbitrary argument (i.e. one of type $\emptyset$). One way to resolve this type mismatch is to change the type environment of the premise. The resulting derivation is (2) in Fig. 8, in which the judgement $a : (p \to q) \uplus \emptyset \vdash a\ b : q$ is derived by a complete typing rule. However the change introduces another type mismatch, i.e. an incomplete derivation for $a : p \to q \vdash a : \emptyset \to q$. To resolve the type mismatch, we change the type for the subject a to $p \to q$ and obtain derivation (3). There is also a type mismatch in derivation (3), in which b is required to have type p but actually has type $\emptyset$, and a local resolution of the type mismatch results in derivation (4). Thus, we finally reaches a complete derivation, (6) in Fig. 8.

We formalise the above idea by defining *derivation rewriting*. For derivations $\mathcal{D}$ and $\mathcal{D}'$, we write $\mathcal{D} \longrightarrow \mathcal{D}'$ if $\mathcal{D}$ is rewritten to $\mathcal{D}'$. For derivations in Fig. 8, (1) $\longrightarrow$ (2) $\longrightarrow$ (3) $\longrightarrow$ (4) $\longrightarrow$ (5) $\longrightarrow$ (6) holds, as expected. In each step of rewriting, an incomplete typing rule is replaced with a complete one, and at the same time another use of an incomplete typing rule (that may be (P-$\star$)) is introduced to the derivation. At the end we reach a derivation using (P-$\star$) rule, i.e. a complete derivation.

Derivation rewriting rules are listed in Fig. 9 and Fig. 10. In these figures, we omit the context in which the incomplete typing rule is used, and subderivations of its premises. For example, it might be better to write (▲App1) as

$$\dfrac{\dfrac{\mathcal{D}_1}{\Gamma \vdash M : \alpha \to \tau} \quad \dfrac{\mathcal{D}_2}{\Delta \vdash N : \alpha}}{\dfrac{\Gamma' \uplus \Delta \vdash M\ N : \tau' \ \blacktriangle}{\mathcal{C}}} \longrightarrow \dfrac{\dfrac{\mathcal{D}_1}{\Gamma' \vdash M : \alpha \to \tau' \ \blacktriangle} \quad \dfrac{\mathcal{D}_2}{\Delta \vdash N : \alpha}}{\dfrac{\Gamma' \uplus \Delta \vdash M\ N : \tau'}{\mathcal{C}}}.$$

An instance of the rule is (1) $\longrightarrow$ (2) in Fig. 8, in which $\mathcal{C} = \star$ and $\mathcal{D}_1 = \mathcal{D}_2 = \Diamond$.

Theorem 2 (Progress and Determinacy of Rewriting). *For any incomplete derivation $\mathcal{D}$, there exists a derivation $\mathcal{D}'$ such that $\mathcal{D} \longrightarrow \mathcal{D}'$. Moreover if $\mathcal{D} \longrightarrow \mathcal{D}_1$ and $\mathcal{D} \longrightarrow \mathcal{D}_2$, then $\mathcal{D}_1 = \mathcal{D}_2$.* □

3.5 Connection to Higher-Order Model Checking

So far the goal has been to obtain a complete derivation. However, as we have seen in Section 3.1, the resulting complete derivation may not respect the tran-

(VAR1) $\dfrac{\dagger}{\dfrac{}{x : \tau \vdash x : \tau'}\,\blacktriangle} \longrightarrow \dfrac{\dagger}{\dfrac{}{x : \tau' \vdash x : \tau'}\,\blacktriangledown} \quad [\tau \lhd_O \tau']$

(VAR2) $\dfrac{\dagger}{\dfrac{}{x : \tau' \vdash x : \tau}\,\blacktriangle} \longrightarrow \dfrac{\dagger}{\dfrac{}{x : \tau' \vdash x : \tau'}\,\blacktriangledown} \quad [\tau \lhd_P \tau']$

(▲FUN) $\dfrac{\dfrac{\Gamma, \widetilde{x} : \widetilde{\alpha} \vdash M : q}{\Gamma' \vdash F : \widetilde{\alpha}' \to q'}\,\blacktriangle}{} \longrightarrow \dfrac{\dfrac{\Gamma', \widetilde{x} : \widetilde{\alpha}' \vdash M : q'}{\Gamma' \vdash F : \widetilde{\alpha}' \to q'}\,\blacktriangle}{}$

(▼FUN) $\dfrac{\dfrac{\Gamma', \widetilde{x} : \widetilde{\alpha}' \vdash M : q'}{\Gamma \vdash F : \widetilde{\alpha} \to q}\,\blacktriangledown}{} \longrightarrow \dfrac{\dfrac{\Gamma', \widetilde{x} : \widetilde{\alpha}' \vdash M : q'}{\Gamma' \vdash F : \widetilde{\alpha}' \to q'}\,\blacktriangledown}{}$

(▲APP1) $\dfrac{\dfrac{\Gamma \vdash M : \alpha \to \tau}{\Gamma' \oplus \Delta \vdash M\ N : \tau'}\,\blacktriangle \quad \dfrac{\Delta \vdash N : \alpha}{}}{} \longrightarrow \dfrac{\dfrac{\Gamma' \vdash M : \alpha \to \tau'}{\Gamma' \oplus \Delta \vdash M\ N : \tau'}\,\blacktriangle \quad \dfrac{\Delta \vdash N : \alpha}{}}{}$

(▲APP2) $\dfrac{\dfrac{\Gamma \vdash M : \alpha \to \tau}{\Gamma \oplus \Delta' \vdash M\ N : \tau}\,\blacktriangle \quad \dfrac{\Delta \vdash N : \alpha}{}}{} \longrightarrow \dfrac{\dfrac{\Gamma \vdash M : \alpha \to \tau}{\Gamma \oplus \Delta' \vdash M\ N : \tau} \quad \dfrac{\Delta' \vdash N : \alpha}{}\,\blacktriangle}{}$

(▼APP1) $\dfrac{\dfrac{\Gamma' \vdash M : \alpha \to \tau'}{\Gamma \oplus \Delta \vdash M\ N : \tau}\,\blacktriangledown \quad \dfrac{\Delta \vdash N : \alpha}{}}{} \longrightarrow \dfrac{\dfrac{\Gamma' \vdash M : \alpha \to \tau'}{\Gamma' \oplus \Delta \vdash M\ N : \tau'}\,\blacktriangledown \quad \dfrac{\Delta \vdash N : \alpha}{}}{}$

(▼APP2) $\dfrac{\dfrac{\Gamma \vdash M : \alpha' \to \tau}{\Gamma \oplus \Delta \vdash M\ N : \tau}\,\blacktriangledown \quad \dfrac{\Delta \vdash N : \alpha}{}}{} \longrightarrow \dfrac{\dfrac{\Gamma \vdash M : \alpha' \to \tau}{\Gamma \oplus \Delta \vdash M\ N : \tau} \quad \dfrac{\Delta \vdash N : \alpha'}{}\,\blacktriangle}{}$

(▼APP3) $\dfrac{\dfrac{\Gamma \vdash M : \alpha \to \tau}{\Gamma \oplus \Delta \vdash M\ N : \tau} \quad \dfrac{\Delta' \vdash N : \alpha}{}\,\blacktriangledown}{} \longrightarrow \dfrac{\dfrac{\Gamma \vdash M : \alpha \to \tau}{\Gamma \oplus \Delta' \vdash M\ N : \tau}\,\blacktriangledown \quad \dfrac{\Delta' \vdash N : \alpha}{}}{}$

(▼APP4) $\dfrac{\dfrac{\Gamma \vdash M : \alpha \to \tau}{\Gamma \oplus \Delta \vdash M\ N : \tau} \quad \dfrac{\Delta \vdash N : \alpha'}{}\,\blacktriangledown}{} \longrightarrow \dfrac{\dfrac{\Gamma \vdash M : \alpha' \to \tau}{\Gamma \oplus \Delta' \vdash M\ N : \tau}\,\blacktriangle \quad \dfrac{\Delta \vdash N : \alpha'}{}}{}$

(▲INT1) $\dfrac{\dfrac{\Gamma \vdash M : \tau}{\Gamma' \oplus \Delta \vdash M : \tau' \oplus \sigma}\,\blacktriangle \quad \dfrac{\Delta \vdash M : \sigma}{}}{} \longrightarrow \dfrac{\dfrac{\Gamma' \vdash M : \tau'}{\Gamma' \oplus \Delta \vdash M : \tau' \oplus \sigma}\,\blacktriangle \quad \dfrac{\Delta \vdash M : \sigma}{}}{}$

(▲INT2) $\dfrac{\dfrac{\Gamma \vdash M : \tau}{\Gamma \oplus \Delta' \vdash M : \tau \oplus \sigma'}\,\blacktriangle \quad \dfrac{\Delta \vdash M : \sigma}{}}{} \longrightarrow \dfrac{\dfrac{\Gamma \vdash M : \tau}{\Gamma \oplus \Delta' \vdash M : \tau \oplus \sigma'} \quad \dfrac{\Delta' \vdash M : \sigma'}{}\,\blacktriangle}{}$

(▼INT1) $\dfrac{\dfrac{\Gamma' \vdash M : \tau'}{\Gamma \oplus \Delta \vdash M : \tau \oplus \sigma}\,\blacktriangledown \quad \dfrac{\Delta \vdash M : \sigma}{}}{} \longrightarrow \dfrac{\dfrac{\Gamma' \vdash M : \tau'}{\Gamma' \oplus \Delta \vdash M : \tau' \oplus \sigma}\,\blacktriangledown \quad \dfrac{\Delta \vdash M : \sigma}{}}{}$

(▼INT2) $\dfrac{\dfrac{\Gamma \vdash M : \tau}{\Gamma \oplus \Delta \vdash M : \tau \oplus \sigma} \quad \dfrac{\Delta' \vdash M : \sigma'}{}\,\blacktriangledown}{} \longrightarrow \dfrac{\dfrac{\Gamma \vdash M : \tau}{\Gamma \oplus \Delta' \vdash M : \tau \oplus \sigma'}\,\blacktriangledown \quad \dfrac{\Delta' \vdash M : \sigma'}{}}{}$

Fig. 9. Derivation rewriting rules (part 1)

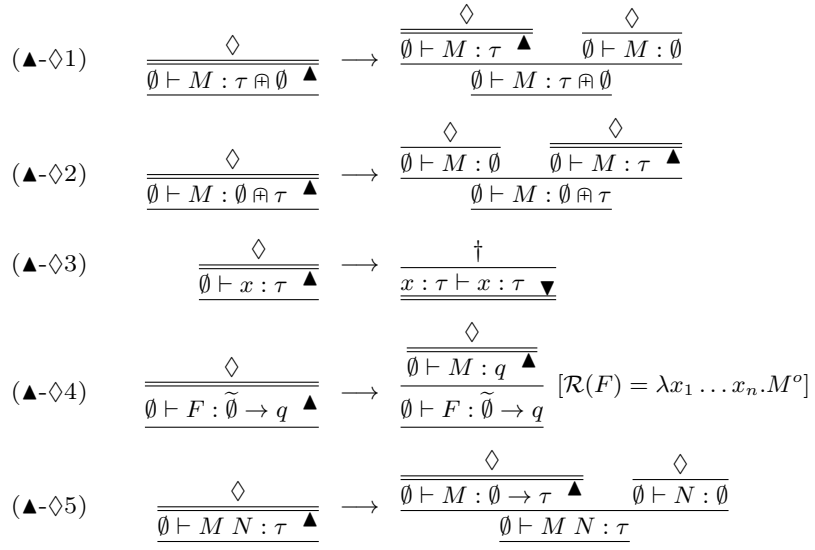

Fig. 10. Derivation rewriting rules (part 2)

sition rule of a given automaton. In that case, we need to continue the type inference process by updating the type environment:

$$(\blacktriangledown\text{-}\star) \qquad \frac{\overline{\Theta \vdash S : q_0 \ \blacktriangledown}}{\star} \longrightarrow \frac{\overline{\Theta' \vdash S : q_0 \ \blacktriangle}}{\star} \qquad \begin{bmatrix} \Theta \lhd_P \Theta' \\ \delta \vDash \Diamond\Theta' \end{bmatrix}$$

Here $\delta \vDash \Diamond\Theta$ if and only if there exists a sequence of P-extensions $\Theta \lhd_P \Theta_1 \lhd_P \ldots \lhd_P \Theta_n$ $(n \geq 0)$ such that $\delta \vDash \Theta_n$. In contrast to incomplete derivations, a complete derivation may have more than one successor, or no successor at all.

Now type inference by derivation rewriting can be represented by a (possibly infinite) rewriting sequence of the form

$$\frac{\Diamond}{\overline{\emptyset \vdash S : q_0 \ \blacktriangle}}{\star} \longrightarrow^* \frac{\vdots}{\overline{\Theta_1 \vdash S : q_0 \ \blacktriangledown}}{\star} \longrightarrow \frac{\vdots}{\overline{\Theta_2 \vdash S : q_0 \ \blacktriangle}}{\star} \longrightarrow^* \frac{\vdots}{\overline{\Theta_3 \vdash S : q_0 \ \blacktriangledown}}{\star} \longrightarrow \cdots$$

where $\emptyset = \Theta_0 \lhd_O \Theta_1 \lhd_P \Theta_2 \lhd_O \cdots$. We call the left-most derivation above the *initial derivation*, and write $\mathcal{D}_{\text{init}}$ for it. We say the sequence is *fair* if for all i, every contra-variant occurrence of $\emptyset$ in Θ_i is eventually replaced with some base type $q \in Q$. For example, if the sequence is fair and $a : \emptyset \to q \in \Theta_i$ for some i, then there exists $j > i$ such that $a : p \to q \in \Theta_j$ for some $p \in Q$.

Theorem 3. *Let $\mathcal{G}$ be a recursion scheme and $\mathcal{A}$ be a trivial tree automaton. Then $[\![\mathcal{G}]\!]$ is accepted by $\mathcal{A}$ if and only if there exists a rewriting sequence starting*

from $\mathcal{D}_{\mathrm{init}}$ such that (1) it is infinite and fair and satisfies $\delta \vDash \Diamond\Theta_i$ for every i, or (2) it is finite and ends with a complete derivation of $\Theta \vdash S : q_0$ such that $\delta \vDash \Theta$. □

Suppose that $[\![\mathcal{G}]\!]$ is accepted by $\mathcal{A}$. Theorem 3 intuitively says that by rewriting derivations in possibly infinite steps, one can reach a complete derivation that certifies the acceptance of $[\![\mathcal{G}]\!]$ by $\mathcal{A}$. Thus an over-approximation of a certificate can be intuitively computed from the set of all reachable derivations by collecting all type bindings for non-terminals.

In fact, for some technical reason, we need an additional rewriting rule

$$(\Diamond\text{-GiveUp}) \qquad \dfrac{\dfrac{}{\emptyset \vdash F : \emptyset \to \ldots \to \emptyset \to q}\,\Diamond}{\text{▲}} \longrightarrow \dfrac{\dfrac{}{\bot : q \vdash F : \emptyset \to \ldots \to \emptyset \to q}\,\Omega}{\text{▼}}$$

that gives up computing the subderivation any more and returns $\bot$ that is a sign of giving up (here Ω is just a symbol like † and $\Diamond$). With this rule, we can construct an over-approximation of a certificate by using the derivation rewriting system as follows. Let $\mathfrak{h}$ be the map from rigid intersection types to flexible intersection types that replaces $\pitchfork$ with $\wedge$, and Γ_{Conc} be a type environment in the flexible intersection type system defined by:

$$\Gamma_{\mathrm{Conc}} = \{F : \mathfrak{h}(\tau) \mid \exists \mathcal{D}\ \exists \Delta.\ \mathcal{D}_{\mathrm{init}} \longrightarrow^* \mathcal{D} \text{ and } \Delta \vdash F : \tau \text{ appears in } \mathcal{D}\}.$$

Theorem 4. *Suppose that $[\![\mathcal{G}]\!]$ is accepted by $\mathcal{A}$ (and thus there exists a certificate). Then Γ_{Conc} is an over-approximation of a certificate.* □

4 Abstraction and Model-Checking Algorithm

Theorem 4 leads to a simple process computing an over-approximation of a certificate.

1. Compute the set $\mathbb{D}$ of all derivations reachable from $\mathcal{D}_{\mathrm{init}}$.
   1-1. Let $\mathbb{D} := \{\mathcal{D}_{\mathrm{init}}\}$.
   1-2. Let $\mathbb{D}' := \mathbb{D} \cup \{\mathcal{D}' \mid \exists \mathcal{D} \in \mathbb{D}.\ \mathcal{D} \longrightarrow \mathcal{D}'\}$.
   1-3. If $\mathbb{D} \subsetneq \mathbb{D}'$, then set $\mathbb{D} := \mathbb{D}'$ and goto 1-2.
2. Return $\Gamma_{\mathrm{Conc}} := \{F : \mathfrak{h}(\tau) \mid \exists \mathcal{D} \in \mathbb{D}\ \exists \Delta.\ \Delta \vdash F : \tau \text{ appears at } \mathcal{D}\ \}$.

However Step 1 may not terminate, or may take too long a time even if it terminates. In this section, we overcome the problem by applying an abstraction to $\mathbb{D}$.

The *abstraction map* takes a set of derivations in the rigid type system and returns a set of instances of typing rules in the *flexible* type system. The abstraction map collects all the instances of rigid typing rules appearing at $\mathbb{D}$ and then replaces $\pitchfork$ with $\wedge$. For example, the set shown in Fig. 11 is the result of applying the abstraction map to the set $\{(4), (5)\}$ of derivations in Fig. 8. The definition of the abstraction map is straightforward and omitted.

$$\left\{ \begin{array}{l} \dfrac{a{:}p{\to}q \vdash a\,b{:}q}{\star}, \quad \dfrac{a{:}p{\to}q \vdash a{:}p{\to}q \quad \vdash b{:}p}{a{:}p{\to}q \vdash a\,b{:}q}, \quad \dfrac{\dagger}{a{:}p{\to}q \vdash a{:}q}, \quad \dfrac{\Diamond}{\vdash b{:}p\ \blacktriangle} \\ \dfrac{a{:}p{\to}q \vdash a{:}p{\to}q \quad b{:}p \vdash b{:}p\ \blacktriangledown}{a{:}p{\to}q \vdash a\,b{:}q}, \quad \dfrac{\dagger}{b{:}p \vdash b{:}p} \end{array} \right\}$$

Fig. 11. Abstraction of the set $\{(4), (5)\}$ of derivations in Fig. 8

Suppose $\dfrac{\Diamond}{\vdash x{:}\phi\ \blacktriangle} \in \hat{\mathbb{D}}$ for some $\omega \prec_O \phi$. Then $\dfrac{\dagger}{x{:}\phi \vdash x{:}\phi} \in \hat{\mathbb{D}}$. Moreover,

(i) if $\dfrac{J \quad \vdash x{:}\phi}{K} \in \hat{\mathbb{D}}$, then $\dfrac{J \quad x{:}\phi \vdash x{:}\phi\ \blacktriangledown}{K} \in \hat{\mathbb{D}}$;

(ii) if $\dfrac{\vdash x : \phi \quad J}{K} \in \hat{\mathbb{D}}$, then $\dfrac{x{:}\phi \vdash x{:}\phi\ \blacktriangledown \quad J}{K} \in \hat{\mathbb{D}}$; and

(iii) if $\dfrac{\vdash x{:}\phi}{J} \in \hat{\mathbb{D}}$, then $\dfrac{x{:}\phi \vdash x{:}\phi\ \blacktriangledown}{J} \in \hat{\mathbb{D}}$.

Fig. 12. Abstraction of rewriting rule (▲-◇3) (here J and K are judgements)

Derivation rewriting rules can be easily adapted for the abstraction. Figure 12 shows the abstraction of the rewriting rule (▲-◇). For example, (ii) says that for every judgements J and K, if the left-hand side of

$$\dfrac{\dfrac{\Diamond}{\vdash x : \phi\ \blacktriangle} \quad J}{K} \qquad \longrightarrow \qquad \dfrac{\dfrac{\dagger}{x : \phi \vdash x : \phi\ \blacktriangledown} \quad J}{K}$$

may be a part of a reachable derivation, then by applying (▲-◇3) rule, the right-hand side may also be a part of a reachable derivation. See Appendix H for abstractions of other rewriting rules.

The model-checking algorithm is described in Fig. 13. Step 1 and 2 are the abstraction of the process to compute all reachable derivations, and step 3 is the fixed-point computation described in Section 2. Correctness of the algorithm is a consequence of Theorem 4 and the correctness of the abstract rewriting rules.

Theorem 5. *The procedure in Fig. 13 always terminates and returns* `yes` *if and only if* $[\![\mathcal{G}]\!]$ *is accepted by* $\mathcal{A}$. □

We estimate the complexity of the algorithm. Let $|Q|$ be the number of states of $\mathcal{A}$, A be the arity of $\mathcal{G}$, n be the order of $\mathcal{G}$, and $|\mathcal{G}| = \sum_{F \in \mathcal{N}} \#\mathcal{R}(F)$ where $\#M$ is the number of subterms of M. Since the size of $\hat{\mathbb{D}}$ and Γ are bounded by $O(|\mathcal{G}| \cdot \mathbf{exp}_n(f(A \cdot |Q|)))$ for some polynomial f (where $\mathbf{exp}_0(x) = x$ and $\mathbf{exp}_{n+1}(x) = 2^{\mathbf{exp}_n(x)}$), the algorithm terminates in time linear (modulo certain optimisations) in the size of $|\mathcal{G}|$ provided that the other parameters are fixed. See the full version for details.

Theorem 6. *The algorithm in Figure 13 runs in time* $O(|\mathcal{G}| \cdot \mathbf{exp}_n(f(A \cdot |Q|)))$, *where* f *is a polynomial.* □

ModelCheck($\mathcal{G}$,$\mathcal{A}$):
1. Construct $\hat{\mathbb{D}}$ by:

   1-1. Let $\hat{\mathbb{D}}$:= $\left\{\dfrac{\vdash S : q_0}{\star},\ \dfrac{\Diamond}{\vdash S : q_0 \blacktriangle}\right\}$;

   1-2. Applying abstract rewriting rules to $\hat{\mathbb{D}}$ until reaching a fixed-point ;
2. Let Γ := $\{F : \phi \mid \exists \Delta.\ \Delta \vdash F : \phi$ appears in $\hat{\mathbb{D}}\}$;
3. Repeat Γ := $\mathbf{Shrink}_{\mathcal{G},\mathcal{A}}(\Gamma)$ until $\Gamma = \mathbf{Shrink}_{\mathcal{G},\mathcal{A}}(\Gamma)$;
4. If $S : q_0 \in \Gamma$ then output 'yes', otherwise output 'no' ;

Fig. 13. The model-checking algorithm

Table 1. Comparison of GTRecS2 with other model checkers

input	TRecS	GTRecS	TravMC	GTRecS2	input	TRecS	GTRecS	TravMC	GTRecS2
Ocamlc	0.005	1.70	0.075	0.096	$\mathcal{G}_{2,5}$	-	0.078	0.06	0.006
Order5	0.002	-	0.061	-	$\mathcal{G}_{3,5}$	-	0.136	-	0.084
Gapid	0.008	-	0.425	3.43	$\mathcal{G}_{4,5}$	-	5.71	-	1.93
Xhtmlf	0.596	13.0	-	46.3	Fibstr	-	-	-	0.168
Mc91	0.028	20.5	1.00	0.301	L	-	0.248	-	0.006

5 Experimental Results

We have implemented a model checker GTRecS2 based on the algorithm discussed in Section 4,[5] except that Step 1-2 and 3 are interleaved in the actual algorithm. We compared its performance with previous model checkers: TRecS [7], GTRecS [9], and TravMC [13]. Except for TravMC, the experiments were conducted on a machine with Intel(R) Xeon(R) CPU with 3Ghz and 8GB memory, with time-out set for 60 seconds. For TravMC, we have used the web interface at http://mjolnir.cs.ox.ac.uk/cgi-bin/horsc/travmc-horsc/input, with time-out set for 10 seconds. We have tested the model checkers for two groups of benchmark programs. The first group, shown on the left-hand side, has been obtained from program verification problems [7, 12, 11], for which TRecS works well. The second group, shown on the right-hand side, consists of recursion schemes that generate finite but huge trees [9, 10]. As expected, TRecS and TravMC work well for the first group, while they behave quite badly for the second group, since their type inference essentially relies on reductions of terms. GTRecS and GTRecS2 work well for the second group, but GTRecS2 outperforms GTRecS by an order of magnitude except for Xhtmlf. GTRecS2 timed out for Order5. This is because Order5 is an order-5 recursion scheme (while the others are at most order-4), and the number of possible types blows up. Further optimisations are necessary to make GTRecS2 scalable with respect to the order of recursion schemes.

[5] Actually, GTRecS2 has already been in use [10], but it has been formalised in the present paper for the first time.

6 Related Work

As discussed in Section 1, there are several algorithms and implementations for higher-order model checking. The first practical model checker is Kobayashi's TRecS [7]. As we have mentioned, it works in practice for many practical examples but suffers from hyper-exponential worst-case time complexity in the size of recursion schemes. GTRecS [9] is the first practical algorithm that runs in time linear in the size of recursion schemes. As confirmed by experiments, in most cases, GTRecS is slower than GTRecS2. Our derivation rewriting is influenced by GTRecS. For example, P- and O-extension relations originate from GTRecS [9].

Recently Neatherway et al. [13] have proposed an algorithm based on the notion of *traversals* [15]. We have provided empirical comparison of their model checker TravMC with GTRecS2 in Section 5. Here, we make theoretical comparison. A state of their algorithm is represented by a derivation possibly containing *open judgements*, i.e. judgements that have not yet been proved, and the main operation is *closing* of an open judgement by inferring the premises. These notions seem closely related to those in our derivation rewriting system. However there is an important difference: our derivation rewriting is defined locally, i.e. one-step rewriting only affects the neighbours of the incomplete typing rule; whereas in their algorithm, closing an open judgement may affect a judgement far from that. Due to this difference, their algorithm and ours showed quite different behaviours in the experiments.

Several studies of rigid intersection type systems have shown strong connections to semantics of λ-calculi. Neergaard and Mairson [14] have proved that for a given (untyped) λ-term, the principal typing for the term in a rigid intersection type system called System $\mathbb{I}$ [5] corresponds to the normal form of the term, and thus the type inference in System $\mathbb{I}$ is equivalent to normalisation. Di Gianantonio, Honsell and Lenisa [3] have independently proposed another rigid intersection type system and established a relation between their type system and game semantics [4, 1]. Our type system is essentially the same as theirs, and the new contribution is the notion of incomplete derivation and derivation rewriting, which give a new insights into semantic aspects of rigid intersection type systems. The connection to game semantics is used to prove important properties of derivation rewriting, such as Theorem 3.

Since the derivation rewriting can be regarded as an intersection type inference algorithm, it may be related to other inference algorithms for intersection types such as [2, 5]. Establishing the formal connection is left for future work. Since their algorithm terminates only for normalising λ-terms, it cannot be used in our setting, where reductions of recursion schemes do not terminate.

7 Conclusion

Higher-order model checking provides a promising approach to verification of higher-order programs, and development of an efficient higher-order model checker

is a significant challenge in this field. We have presented a new approach to developing algorithms for higher-order model-checking based on derivation rewriting and its abstraction. We have implemented the algorithm based on the abstraction and confirmed that it outperforms previous model checkers in certain cases. For future work, we aim to find a better abstraction than the one in the present paper.

References

1. Abramsky, S., Jagadeesan, R., Malacaria, P.: Full Abstraction for PCF. Information and Computation **163**(2) (December 2000) 409–470
2. Carlier, S., Polakow, J., Wells, J., Kfoury, A.: System E: Expansion variables for flexible typing with linear and non-linear types and intersection types. In: ESOP 2004. Volume 0113193. (2004) 294–309
3. Di Gianantonio, P., Honsell, F., Lenisa, M.: A type assignment system for game semantics. Theoretical Computer Science **398**(1-3) (May 2008) 150–169
4. Hyland, J.M.E., Ong, C.H.L.: On Full Abstraction for PCF: I, II, and III. Information and Computation **163**(2) (December 2000) 285–408
5. Kfoury, A.J., Wells, J.B.: Principality and decidable type inference for finite-rank intersection types. In: Proceedings of the 26th ACM SIGPLAN-SIGACT symposium on Principles of programming languages - POPL '99, New York, New York, USA, ACM Press (1999) 161–174
6. Knapik, T., Niwiński, D., Urzyczyn, P.: Higher-order pushdown trees are easy. In: FoSSaCS 2002. (2002) 205–222
7. Kobayashi, N.: Model-checking higher-order functions. In Porto, A., López-Fraguas, F.J., eds.: PPDP, ACM (2009) 25–36
8. Kobayashi, N.: Types and higher-order recursion schemes for verification of higher-order programs. In Shao, Z., Pierce, B.C., eds.: POPL, ACM (2009) 416–428
9. Kobayashi, N.: A Practical Linear Time Algorithm for Trivial Automata Model Checking of Higher-Order Recursion Schemes. In Hofmann, M., ed.: 14th International Conference on Foundations of Software Science and Computational Structures (FoSSaCS 2011). Volume 6604 of Lecture Notes in Computer Science., Springer (2011) 260–274
10. Kobayashi, N., Matsuda, K., Shinohara, A.: Functional programs as compressed data. In: Proceedings of the ACM SIGPLAN 2012 workshop on Partial evaluation and program manipulation - PEPM '12, New York, New York, USA, ACM Press (2012) 121–130
11. Kobayashi, N., Sato, R., Unno, H.: Predicate abstraction and CEGAR for higher-order model checking. In Hall, M.W., Padua, D.A., eds.: PLDI, ACM (2011) 222–233
12. Kobayashi, N., Tabuchi, N., Unno, H.: Higher-order multi-parameter tree transducers and recursion schemes for program verification. In Hermenegildo, M.V., Palsberg, J., eds.: POPL, ACM (2010) 495–508
13. Neatherway, R.P., Ong, C.H.L., Ramsay, S.J.: A traversal-based algorithm for higher-order model checking. In: Proceedings of the 17th ACM SIGPLAN international conference on Functional programming - ICFP '12, New York, New York, USA, ACM Press (2012) 353–364
14. Neergaard, P.M.l., Mairson, H.G.: Types, potency, and idempotency: why nonlinearity and amnesia make a type system work. In Okasaki, C., Fisher, K., eds.: ICFP, ACM (2004) 138–149

15. Ong, C.H.L.: On Model-Checking Trees Generated by Higher-Order Recursion Schemes. In: LICS, IEEE Computer Society (2006) 81–90
16. Ong, C.H.L., Ramsay, S.J.: Verifying higher-order functional programs with pattern-matching algebraic data types. In: POPL 2011. Volume 46. (January 2011) 587–598
17. Rehof, J., Mogensen, T.Æ.: Tractable constraints in finite semilattices. Science of Computer Programming **35**(2-3) (November 1999) 191–221

A Basic Definitions: AJM's Game Model

Here we define the game model proposed by Abramsky, Jagadeesan and Malacaria (called AJM game model). Definitions here are somewhat different from the standard ones.

- Unlike the standard definition, a game in our definition has a *justification relation* on moves as its component. The justification relation is used to give a connection to the derivation rewriting system, and does not affect the definition of play and composition.
- All moves are questions.
- The set of moves in an exponential is defined as an indexed product of moves, similar to the original definition. In the original definition, the set of indexes is the set of natural numbers, whereas we use another set as indexes (see Definition 5).

Except for the differences listed above, the contexts of this section is the standard presentation of AJM game model, and thus readers who are familiar with game semantics can safely skip this section.

A.1 Sequences

For a set X, the set of all finite sequences on X is written as X^*. For $s \in X^*$, $|s|$ is the length of s and s_i is the ith element ($1 \leq i \leq |s|$). For $s, t \in X^*$, we write $s \sqsubseteq t$ if s is a prefix of t. For $s, t \in X^*$, their concatenation is written as $s \cdot t$ and st. For a sequence s and a set of sequences T, we write $s \cdot T$ for $\{s \cdot t \mid t \in T\}$. For $x \in X$ and $s \in X^*$, we write $x \in s$ if x appears in s.

Let $Y \subseteq X$ be a subset of X and $s \in X^*$, the restriction $s \upharpoonright Y \in Y^*$ is inductively defined by

- $\epsilon \upharpoonright Y = \epsilon$,
- $(s \cdot x) \upharpoonright Y = (s \upharpoonright Y) \cdot x$ if $x \in Y$, and
- $(s \cdot x) \upharpoonright Y = s \upharpoonright Y$ if $x \notin Y$.

For sequences s and t, we write $s \sqsubseteq^{\mathrm{even}} t$ if $s \sqsubseteq t$ and $|s|$ is even (s is called an *even prefix* of t). For a set A of sequences, A^{even} is its restriction to sequences of even length, i.e. $A^{\mathrm{even}} = \{s \in A \mid |s| \text{ is even }\}$.

A.2 Games and Strategies

A *game* is a 5-tuple $\mathfrak{A} = (\mathcal{M}, \lambda, P, \approx, \vdash)$, where

- $\mathcal{M}$ is a set of *moves*. We use m and n for metavariables ranging over $\mathcal{M}$ and s and t for metavariables over $\mathcal{M}^*$.
- $\lambda : \mathcal{M} \to \{O, P\}$ is the *labelling function*. We define $\overline{O} = P$, $\overline{P} = O$. The function $\overline{\lambda}$ is defined by $\overline{\lambda}(x) = \overline{\lambda(x)}$. We define $\mathcal{M}^O = \{m \in \mathcal{M} \mid \lambda(m) = O\}$ and $\mathcal{M}^P = \{m \in \mathcal{M} \mid \lambda(m) = P\}$.

– $P \subseteq \mathcal{M}^*$ is a prefix-closed subset of $\mathcal{M}^*$ that satisfies if $s \in P$, then $\lambda(s_1) = O$ and for all $i < |s|$, $\lambda(s_i) \neq \lambda(s_{i+1})$. An element of P is called a *valid position* (or a *play*).
– $\approx$ is an equivalence relation on P that satisfies the following condition. For every $s, t \in \mathcal{M}^*$, $s \approx t$ implies (1) $|s| = |t|$, (2) for every $s' \sqsubseteq s$ and $t' \sqsubseteq t$, $|s'| = |t'|$ implies $s' \approx t'$, and (3) $sm \in P$ implies $\exists n.(tn \in P$ and $sm \approx tn)$.
– $\vdash$ is a binary relation on $\mathcal{M}$ that satisfies (1) if $m \vdash n$, then $\lambda(m) \neq \lambda(n)$ and (2) if $\neg\exists m.\ m \vdash n$, then $\lambda(n) = O$. We call $\vdash$ the *justification relation*.

A *strategy* is a non-empty subset $\mathfrak{s} \subseteq P$ such that (1) for every $s \in \mathfrak{s}$, $|s|$ is even, and (2) for every $s \in \mathfrak{s}$, $t \sqsubseteq^{\mathrm{even}} s$ implies $t \in \mathfrak{s}$ (i.e. $\mathfrak{s}$ is even-prefix closed). A strategy $\mathfrak{s}$ is *history-free* if it satisfies the following conditions.

1. If $smn \in \mathfrak{s}$ and $tmn' \in \mathfrak{s}$, then $n = n'$.
2. If $smn \in \mathfrak{s}$, $t \in \mathfrak{s}$ and $tm \in P$, then $tmn \in \mathfrak{s}$.

Let $\mathfrak{s}$ and $\mathfrak{s}'$ be strategies. We write $\mathfrak{s} \precsim \mathfrak{s}'$ if $smn \in \mathfrak{s}$, $s' \in \mathfrak{s}'$ and $sm \approx s'm'$ implies there exists n' such that $s'm'n' \in \mathfrak{s}'$ and $smn \approx s'm'n'$. The relation $\approx$ on strategies is defined by $\mathfrak{s} \approx \mathfrak{s}'$ if and only if $\mathfrak{s} \precsim \mathfrak{s}'$ and $\mathfrak{s}' \precsim \mathfrak{s}$. It is symmetric and transitive [1, Proposition 2.2], but not necessarily reflexive.

Let $\mathfrak{s}$ be a strategy for $\mathfrak{A}$. We write $\mathfrak{s} : \mathfrak{A}$ if $\mathfrak{s}$ is history-free and $\mathfrak{s} \approx \mathfrak{s}$. Now $\approx$ is an equivalence relation on $\{\mathfrak{s} \mid \mathfrak{s} : \mathfrak{A}\}$. Let $\mathfrak{s}$ be a strategy such that $\mathfrak{s} : \mathfrak{A}$. We define $[\mathfrak{s}]$ as the equivalence class that contains $\mathfrak{s}$, i.e. $[\mathfrak{s}] = \{\mathfrak{t} : \mathfrak{A} \mid \mathfrak{t} \approx \mathfrak{s}\}$. If $[\mathfrak{s}]$ is an equivalence class of strategies for $\mathfrak{A}$, we write $[\mathfrak{s}] : \mathfrak{A}$.

Given a strategy $\mathfrak{s} : \mathfrak{A}$, we can define a partial function $\mathbf{Fun}(\mathfrak{s}) : \mathcal{M}^O \rightharpoonup \mathcal{M}^P$ by

$$\mathbf{Fun}(\mathfrak{s})(m) = n \qquad \text{if } \exists s.\ smn \in \mathfrak{s}.$$

(The function $\mathbf{Fun}(\mathfrak{s})$ is well-defined because $\mathfrak{s}$ is history-free.) Conversely a partial function $f : \mathcal{M}^O \rightharpoonup \mathcal{M}^P$ induces an even-prefix closed set $\mathbf{trace}(f)$ defined inductively by:

$$\epsilon \in \mathbf{trace}(f)$$
$$s \cdot m \cdot f(m) \in \mathbf{trace}(f) \ \text{ if } \ s \in \mathbf{trace}(f) \text{ and } s \cdot m \in P.$$

Given a partial function f, $\mathbf{trace}(f)$ is not necessarily a strategy, since $s \cdot m \cdot f(m)$ may not be in P even if $s \cdot m \in P$. Note that $\mathbf{trace}(\mathbf{Fun}(\mathfrak{s})) = \mathfrak{s}$ for any strategy $\mathfrak{s}$, and $\mathbf{Fun}(\mathbf{trace}(f)) \sqsubseteq f$ if $\mathbf{trace}(f)$ is a strategy.[6] So for every strategy $\mathfrak{s}$, there exists a least partial function on moves that induces $\mathfrak{s}$. In the following, we identified a strategy with a partial function by this correspondence.

A.3 Constructions on Games

We give some constructions on games.

[6] For partial functions $f, g : X \rightharpoonup Y$, we write $f \sqsubseteq g$ if and only if $\mathrm{dom}(f) \subseteq \mathrm{dom}(g)$ and $f(x) = g(x)$ for every $x \in \mathrm{dom}(f)$, where $\mathrm{dom}(f) = \{x \in X \mid f(x) \text{ is defined}\}$.

Definition 1 (Tensor Product). *Given games* $\mathfrak{A}_1 = (\mathcal{M}_1, \lambda_1, P_1, \approx_1, \vdash_1)$ *and* $\mathfrak{A}_2 = (\mathcal{M}_2, \lambda_2, P_2, \approx_2, \vdash_2)$*, the game* $\mathfrak{A}_1 \otimes \mathfrak{A}_2$ *is defined as* $(\mathcal{M}, \lambda, P, \approx, \vdash)$ *where*

- $\mathcal{M} = \mathcal{M}_1 + \mathcal{M}_2$*, here* $+$ *means disjoint union of the sets.*
- λ *is defined by*

$$\lambda(m) = \begin{cases} \lambda_1(m) & \text{(if } m \in \mathcal{M}_1\text{)} \\ \lambda_2(m) & \text{(if } m \in \mathcal{M}_2\text{).} \end{cases}$$

- $P = \{s \in (\mathcal{M}_1 + \mathcal{M}_2)^* \mid s \upharpoonright \mathcal{M}_1 \in P_1 \text{ and } s \upharpoonright \mathcal{M}_2 \in P_2\}$.
- $s \approx t$ *if and only if* $s \upharpoonright \mathcal{M}_1 \approx_1 t \upharpoonright \mathcal{M}_1$ *and* $s \upharpoonright \mathcal{M}_2 \approx_2 t \upharpoonright \mathcal{M}_2$ *and* $\forall i \leq |s|.\ (s_i \in \mathcal{M}_1 \Leftrightarrow t_i \in \mathcal{M}_1)$.
- $m \vdash n$ *if and only if* $m \vdash_1 n$ *or* $m \vdash_2 n$. □

Definition 2 (Product). *Given games* $\mathfrak{A}_1 = (\mathcal{M}_1, \lambda_1, P_1, \approx_1, \vdash_1)$ *and* $\mathfrak{A}_2 = (\mathcal{M}_2, \lambda_2, P_2, \approx_2, \vdash_2)$*, the game* $\mathfrak{A}_1 \& \mathfrak{A}_2$ *is defined as* $(\mathcal{M}, \lambda, P, \approx, \vdash)$ *where*

- $\mathcal{M} = \mathcal{M}_1 + \mathcal{M}_2$.
- λ *is defined by*

$$\lambda(m) = \begin{cases} \lambda_1(m) & \text{(if } m \in \mathcal{M}_1\text{)} \\ \lambda_2(m) & \text{(if } m \in \mathcal{M}_2\text{).} \end{cases}$$

- $P = P_1 \cup P_2$.
- $s \approx t$ *if and only if* $s \approx_1 t$ *or* $s \approx_2 t$.
- $m \vdash n$ *if and only if* $m \vdash_1 n$ *or* $m \vdash_2 n$. □

Definition 3 (Linear Implication). *Given games* $\mathfrak{A}_1 = (\mathcal{M}_1, \lambda_1, P_1, \approx_1, \vdash_1)$ *and* $\mathfrak{A}_2 = (\mathcal{M}_2, \lambda_2, P_2, \approx_2, \vdash_2)$*, the game* $\mathfrak{A}_1 \multimap \mathfrak{A}_2$ *is defined as* $(\mathcal{M}, \lambda, P, \approx, \vdash)$ *where*

- $\mathcal{M} = \mathcal{M}_1 + \mathcal{M}_2$.
- λ *is defined by*

$$\lambda(m) = \begin{cases} \overline{\lambda_1(m)} & \text{(if } m \in \mathcal{M}_1\text{)} \\ \lambda_2(m) & \text{(if } m \in \mathcal{M}_2\text{).} \end{cases}$$

- $P = \{s \in (\mathcal{M}_1 + \mathcal{M}_2)^* \mid s \upharpoonright \mathfrak{A}_1 \in P_1 \text{ and } s \upharpoonright \mathfrak{A}_2 \in P_2\}$.
- $s \approx t$ *if and only if* $s \upharpoonright \mathfrak{A}_1 \approx_1 t \upharpoonright \mathfrak{A}_1$ *and* $s \upharpoonright \mathfrak{A}_2 \approx_2 t \upharpoonright \mathfrak{A}_2$ *and* $\forall i \leq |s|.\ (s_i \in \mathfrak{A}_1 \Leftrightarrow t_i \in \mathfrak{A}_1)$.
- $m \vdash n$ *if and only if one of the following conditions holds.*
  - $m, n \in \mathcal{M}_1$ *and* $m \vdash_1 n$.
  - $m, n \in \mathcal{M}_2$ *and* $m \vdash_2 n$.
  - $m \in \mathcal{M}_2$ *and* $n \in \mathcal{M}_1$ *and* $\neg \exists n'.\ n' \vdash_1 n$. □

Definition 4 (Index). *The set of* indexes *is defined by the following grammar:*

$$\textit{Indexes} \qquad \xi ::= \square \mid l \cdot \xi \mid r \cdot \xi \mid \langle \xi_1, \xi_2 \rangle,$$

where $\square$*,* l *and* r *are symbols. We write the set of all indexes as* $\mathcal{I}$. □

We use $\mathcal{I}$ as the index set of the exponentials.

Definition 5 (Exponential). *Given a game $\mathfrak{A} = (\mathcal{M}, \lambda, P, \approx, \vdash)$, the game $!\mathfrak{A}$ is defined as $(\mathcal{M}', \lambda', P', \approx', \vdash')$, where*

- $\mathcal{M}' = \mathcal{I} \times \mathcal{M}$.
- *λ' is defined by $\lambda'(\,(\xi, m)\,) = \lambda(m)$.*
- *$s \in P'$ if and only if $s \upharpoonright \xi \in P$ for every $\xi \in \mathcal{I}$. Here $(s \upharpoonright \xi) \in \mathcal{M}^*$ is defined inductively by: (1) $\epsilon \upharpoonright \xi = \epsilon$, (2) $(s \cdot (\xi, m)\,) \upharpoonright \xi = (s \upharpoonright \xi) \cdot m$, and (3) $(s \cdot (\xi', m)\,) \upharpoonright \xi = s \upharpoonright \xi$ if $\xi \neq \xi'$.*
- *$s \approx' t$ if and only if there exists a permutation χ of $\mathcal{I}$ such that*
  - *$\mathrm{proj}_{\mathcal{I}}(s_i) = \chi(\mathrm{proj}_{\mathcal{I}}(t_i))$ for every $i \leq |s|$, where $\mathrm{proj}_{\mathcal{I}} : \mathcal{I} \times \mathcal{M} \to \mathcal{I}$ is the projection, and*
  - *for every $\xi \in \mathcal{I}$, $s \upharpoonright \chi(\xi) \approx t \upharpoonright \xi$.*
- *$(\xi, m) \vdash' (\xi', m')$ if and only if $\xi = \xi'$ and $m \vdash m'$.* □

In the following, we implicitly assume that game $\mathfrak{A}$ consists of $(\mathcal{M}_{\mathfrak{A}}, \lambda_{\mathfrak{A}}, P_{\mathfrak{A}}, \approx_{\mathfrak{A}}, \vdash_{\mathfrak{A}})$. For example, $\mathcal{M}_{\mathfrak{A} \multimap \mathfrak{B}}$ is the set of moves in the game $\mathfrak{A} \multimap \mathfrak{B}$. We often omit the subscript $\mathfrak{A}$ of $\approx_{\mathfrak{A}}$ and $\vdash_{\mathfrak{A}}$, and simply write $\approx$ and $\vdash$.

Definition 6 (Well-justified). *Let $\mathfrak{A} = (\mathcal{M}, \lambda, P, \approx, \vdash)$ be a game. A set $X \subseteq \mathcal{M}$ of moves is* well-justified *if for every $m \in X$, the justifier of m is also in X. In other words, for every $m \in X$, one of the following conditions holds.*

- *There exists n such that $n \vdash m$ and $n \in X$.*
- *There is no n such that $n \vdash m$.* □

A.4 Category of Games

Let $\mathfrak{A}$, $\mathfrak{B}$ and $\mathfrak{C}$ be games and $\mathfrak{s} : \mathfrak{A} \multimap \mathfrak{B}$ and $\mathfrak{t} : \mathfrak{B} \multimap \mathfrak{C}$ be strategies. We define

$$\mathfrak{s} \parallel \mathfrak{t} = \{s \in (\mathcal{M}_{\mathfrak{A}} + \mathcal{M}_{\mathfrak{B}} + \mathcal{M}_{\mathfrak{C}})^* \mid s \upharpoonright \mathfrak{A}, \mathfrak{B} \in \mathfrak{s} \text{ and } s \upharpoonright \mathfrak{B}, \mathfrak{C} \in \mathfrak{s}'\}$$
$$\mathfrak{s}; \mathfrak{t} = \{s \upharpoonright \mathfrak{A}, \mathfrak{C} \mid s \in \mathfrak{s} \parallel \mathfrak{t}\}^{\mathrm{even}}$$

Lemma 1. [1, Proposition 2.4] *Let $\mathfrak{s}, \mathfrak{s}' : \mathfrak{A} \multimap \mathfrak{B}$ and $\mathfrak{t}, \mathfrak{t}' : \mathfrak{B} \multimap \mathfrak{C}$. If $\mathfrak{s} \precsim \mathfrak{s}'$ and $\mathfrak{t} \precsim \mathfrak{t}'$, then $(\mathfrak{s}; \mathfrak{t}) \precsim (\mathfrak{s}'; \mathfrak{t}')$.* □

We define the category $\mathfrak{G}$ of games and history-free strategies.

Objects Games.
Morphisms A morphism from $\mathfrak{A}$ to $\mathfrak{B}$ is an equivalence class $[\mathfrak{s}] : \mathfrak{A} \multimap \mathfrak{B}$.
Composition The composition of $[\mathfrak{s}] : \mathfrak{A} \multimap \mathfrak{B}$ and $[\mathfrak{t}] : \mathfrak{B} \multimap \mathfrak{C}$ is $[\mathfrak{s}; \mathfrak{t}] : \mathfrak{A} \multimap \mathfrak{C}$.
Identity Given a game $\mathfrak{A}$, $\mathrm{id}_{\mathfrak{A}} : \mathfrak{A} \multimap \mathfrak{A}$ is defined by:

$$\mathrm{id}_{\mathfrak{A}} = \{s \in P^{\mathrm{even}}_{\mathfrak{A}^{(1)} \multimap \mathfrak{A}^{(2)}} \mid s \upharpoonright \mathfrak{A}^{(1)} = s \upharpoonright \mathfrak{A}^{(2)}\}$$

where labels (1) and (2) are used to distinguish between different occurrences of $\mathfrak{A}$.

Well-definedness of composition is a consequence of Lemma 1.

The map ! on objects of $\mathfrak{G}$ can be extended to a map on strategies. Let $\mathfrak{s} : \mathfrak{A} \multimap \mathfrak{B}$ be a strategy and $f_{\mathfrak{s}} : (\mathcal{M}_{\mathfrak{A}}^P + \mathcal{M}_{\mathfrak{B}}^O) \rightharpoonup (\mathcal{M}_{\mathfrak{A}}^O + \mathcal{M}_{\mathfrak{B}}^P)$ be the partial function that corresponds to $\mathfrak{s}$. Then the partial function $f_{!\mathfrak{s}} : (\mathcal{I} \times \mathcal{M}_{\mathfrak{A}}^P + \mathcal{I} \times \mathcal{M}_{\mathfrak{B}}^O) \rightharpoonup (\mathcal{I} \times \mathcal{M}_{\mathfrak{A}}^O + \mathcal{I} \times \mathcal{M}_{\mathfrak{B}}^P)$ is defined by:

$$f_{!\mathfrak{s}}(\,(\xi, m)\,) = (\xi, f_{\mathfrak{s}}(m)).$$

We defined $!\mathfrak{s} = \mathbf{trace}(f_{!\mathfrak{s}})$.

Lemma 2. ! *is a functor, i.e. (1)* $\mathfrak{s} \approx \mathfrak{t}$ *implies* $!\mathfrak{s} \approx !\mathfrak{t}$*, and (2)* $(!\mathfrak{s}; !\mathfrak{t}) \approx !(\mathfrak{s}; \mathfrak{t})$.

We define some strategies related to !. Let $\mathfrak{A}$ be a game. We define a strategy $\mathrm{der}_{\mathfrak{A}} : !\mathfrak{A} \multimap \mathfrak{A}$ induced from the partial function $f : (\mathcal{I} \times \mathcal{M}^P + \mathcal{M}^O) \rightharpoonup (\mathcal{I} \times \mathcal{M}^O + \mathcal{M}^P)$ defined by:

$$\begin{array}{lll} f(\,(\square, m)\,) = m & & (\text{if } (\square, m) \in \mathcal{I} \times \mathcal{M}^P) \\ f(m) \quad\;\; = (\square, m) & & (\text{if } m \in \mathcal{M}^O) \end{array}$$

and otherwise f is undefined. For a given game $\mathfrak{A}$, we define a strategy $\delta_{\mathfrak{A}} : !\mathfrak{A} \multimap !!\mathfrak{A}$ induced from the partial function $f : (\mathcal{I} \times \mathcal{M}^P + \mathcal{I} \times (\mathcal{I} \times \mathcal{M}^O)) \rightharpoonup (\mathcal{I} \times \mathcal{M}^O + \mathcal{I} \times (\mathcal{I} \times \mathcal{M}^P))$ defined by:

$$\begin{array}{ll} f(\,(\langle \xi, \zeta \rangle, m)\,) = (\xi, (\zeta, m)) & (\text{if } (\langle \xi, \zeta \rangle, m) \in \mathcal{I} \times \mathcal{M}^P) \\ f(\,(\xi, (\zeta, m))\,) = (\langle \xi, \zeta \rangle, m) & (\text{if } (\xi, (\zeta, m)) \in \mathcal{I} \times (\mathcal{I} \times \mathcal{M}^O)) \end{array}$$

and otherwise $f(m)$ is undefined.

Then $(!, [\mathrm{der}], [\delta])$ is a comonad [1, Proposition 2.12]. Let $K_!(\mathfrak{G})$ be the co-Kleisli category over the comonad $(!, [\mathrm{der}], [\delta])$, i.e. $K_!(\mathfrak{G})$ is defined by:

Objects Games.

Morphisms A morphism $[\mathfrak{s}] : \mathfrak{A} \to \mathfrak{B}$ in $K_!(\mathfrak{G})$ is a morphism $[\mathfrak{s}] : !\mathfrak{A} \multimap \mathfrak{B}$ in $\mathfrak{G}$.

Composition Let $[\mathfrak{s}] : \mathfrak{A} \to \mathfrak{B}$ and $[\mathfrak{t}] : \mathfrak{B} \to \mathfrak{C}$ in $K_!(\mathfrak{G})$. Thus $[\mathfrak{s}] : !\mathfrak{A} \multimap \mathfrak{B}$ and $[\mathfrak{t}] : !\mathfrak{B} \multimap \mathfrak{C}$ in $\mathfrak{G}$. Their composition $[\mathfrak{s}] \fatsemi [\mathfrak{t}]$ is defined by $[\delta_{\mathfrak{A}}; !\mathfrak{s}; \mathfrak{t}]$, illustrated by:

$$!\mathfrak{A} \xrightarrow{\delta_{\mathfrak{A}}} !!\mathfrak{A} \xrightarrow{!\mathfrak{s}} !\mathfrak{B} \xrightarrow{\mathfrak{t}} \mathfrak{C}$$

where $\cdot;\cdot$ is the standard composition of strategies. For strategies $\mathfrak{s} : !\mathfrak{A} \multimap \mathfrak{B}$ and $\mathfrak{t} : !\mathfrak{B} \multimap \mathfrak{C}$, we define $\mathfrak{s} \fatsemi \mathfrak{t}$ as $\delta_{\mathfrak{A}}; !\mathfrak{s}; \mathfrak{t}$.

Identity Given a game $\mathfrak{A}$, the identity in $K_!(\mathfrak{G})$ is $\mathrm{der}_{\mathfrak{A}} : !\mathfrak{A} \multimap \mathfrak{A}$ in $\mathfrak{G}$.

Then $K_!(\mathfrak{G})$ is a Cartesian closed category in which $\mathfrak{A}\&\mathfrak{B}$ is the Cartesian product and $!\mathfrak{A} \multimap \mathfrak{B}$ is the function space. For more detail, see [1].

Lastly we define an operation that we call the *application*. Let $\mathfrak{s} : !\mathfrak{A} \multimap !\mathfrak{B} \multimap \mathfrak{C}$ and $\mathfrak{t} : !\mathfrak{A} \multimap !\mathfrak{B}$. We write $f_{\mathfrak{s}}$ and $g_{\mathfrak{t}}$ for associated partial functions. We write

$$m \overset{\mathfrak{s}}{\mapsto} m' \qquad (\text{resp. } m \overset{\mathfrak{t}}{\mapsto} m')$$

if $f_{\mathfrak{s}} = m'$ (resp. $g_{\mathfrak{t}}(m) = m'$). The application $\mathfrak{s}@\mathfrak{t} : !\mathfrak{A} \multimap \mathfrak{C}$ is a strategy induced by the partial function $h : (\mathcal{I} \times \mathcal{M}_{\mathfrak{A}}^P + \mathcal{M}_{\mathfrak{C}}^O) \rightharpoonup (\mathcal{I} \times \mathcal{M}_{\mathfrak{A}}^O + \mathcal{M}_{\mathfrak{C}}^P)$ defined by:

– $h(m) = m'$ for $m \in \mathcal{M}^O_{\mathfrak{C}}$, if there exists a sequence

$$m \overset{\mathfrak{s}}{\mapsto} n_1 \overset{\mathfrak{t}}{\mapsto} n_2 \overset{\mathfrak{s}}{\mapsto} \cdots \overset{\mathfrak{t}}{\mapsto} n_k \overset{\mathfrak{s}}{\mapsto} m' \in \mathcal{M}^P_{\mathfrak{C}}.$$

– $h(m) = (l \cdot \xi',\ m')$ for $m \in \mathcal{M}^O_{\mathfrak{C}}$, if there exists a sequence

$$m \overset{\mathfrak{s}}{\mapsto} n_1 \overset{\mathfrak{t}}{\mapsto} n_2 \overset{\mathfrak{s}}{\mapsto} \cdots \overset{\mathfrak{t}}{\mapsto} n_k \overset{\mathfrak{s}}{\mapsto} (\xi', m') \in \mathcal{I} \times \mathcal{M}^O_{\mathfrak{A}}.$$

– $h(m) = (r \cdot \xi',\ m')$ for $m \in \mathcal{M}^O_{\mathfrak{C}}$, if there exists a sequence

$$m \overset{\mathfrak{s}}{\mapsto} n_1 \overset{\mathfrak{t}}{\mapsto} n_2 \overset{\mathfrak{s}}{\mapsto} \cdots \overset{\mathfrak{s}}{\mapsto} n_k \overset{\mathfrak{t}}{\mapsto} (\xi', m') \in \mathcal{I} \times \mathcal{M}^O_{\mathfrak{A}}.$$

– $h(\,(l \cdot \xi,\ m)\,) = m'$ for $(l \cdot \xi,\ m) \in \mathcal{I} \times \mathcal{M}^P_{\mathfrak{A}}$, if there exists a sequence

$$(\xi, m) \overset{\mathfrak{s}}{\mapsto} n_1 \overset{\mathfrak{t}}{\mapsto} n_2 \overset{\mathfrak{s}}{\mapsto} \cdots \overset{\mathfrak{t}}{\mapsto} n_k \overset{\mathfrak{s}}{\mapsto} m' \in \mathcal{M}^P_{\mathfrak{C}}.$$

– $h(\,(l \cdot \xi,\ m)\,) = (l \cdot \xi',\ m')$ for $(l \cdot \xi,\ m) \in \mathcal{I} \times \mathcal{M}^P_{\mathfrak{A}}$, if there exists a sequence

$$(\xi, m) \overset{\mathfrak{s}}{\mapsto} n_1 \overset{\mathfrak{t}}{\mapsto} n_2 \overset{\mathfrak{s}}{\mapsto} \cdots \overset{\mathfrak{t}}{\mapsto} n_k \overset{\mathfrak{s}}{\mapsto} (\xi', m') \in \mathcal{I} \times \mathcal{M}^O_{\mathfrak{A}}.$$

– $h(\,(l \cdot \xi,\ m)\,) = (r \cdot \xi',\ m')$ for $(l \cdot \xi,\ m) \in \mathcal{I} \times \mathcal{M}^P_{\mathfrak{A}}$, if there exists a sequence

$$(\xi, m) \overset{\mathfrak{s}}{\mapsto} n_1 \overset{\mathfrak{t}}{\mapsto} n_2 \overset{\mathfrak{s}}{\mapsto} \cdots \overset{\mathfrak{s}}{\mapsto} n_k \overset{\mathfrak{t}}{\mapsto} (\xi', m') \in \mathcal{I} \times \mathcal{M}^O_{\mathfrak{A}}.$$

– $h(\,(r \cdot \xi,\ m)\,) = m'$ for $(r \cdot \xi,\ m) \in \mathcal{I} \times \mathcal{M}^P_{\mathfrak{A}}$, if there exists a sequence

$$(\xi, m) \overset{\mathfrak{t}}{\mapsto} n_1 \overset{\mathfrak{s}}{\mapsto} n_2 \overset{\mathfrak{t}}{\mapsto} \cdots \overset{\mathfrak{t}}{\mapsto} n_k \overset{\mathfrak{s}}{\mapsto} m' \in \mathcal{M}^P_{\mathfrak{C}}.$$

– $h(\,(r \cdot \xi,\ m)\,) = (l \cdot \xi',\ m')$ for $(r \cdot \xi,\ m) \in \mathcal{I} \times \mathcal{M}^P_{\mathfrak{A}}$, if there exists a sequence

$$(\xi, m) \overset{\mathfrak{t}}{\mapsto} n_1 \overset{\mathfrak{s}}{\mapsto} n_2 \overset{\mathfrak{t}}{\mapsto} \cdots \overset{\mathfrak{t}}{\mapsto} n_k \overset{\mathfrak{s}}{\mapsto} (\xi', m') \in \mathcal{I} \times \mathcal{M}^O_{\mathfrak{A}}.$$

– $h(\,(r \cdot \xi,\ m)\,) = (r \cdot \xi',\ m')$ for $(r \cdot \xi,\ m) \in \mathcal{I} \times \mathcal{M}^P_{\mathfrak{A}}$, if there exists a sequence

$$(\xi, m) \overset{\mathfrak{t}}{\mapsto} n_1 \overset{\mathfrak{s}}{\mapsto} n_2 \overset{\mathfrak{t}}{\mapsto} \cdots \overset{\mathfrak{s}}{\mapsto} n_k \overset{\mathfrak{t}}{\mapsto} (\xi', m') \in \mathcal{I} \times \mathcal{M}^O_{\mathfrak{A}}.$$

The application operation here coincides with the standard interpretation of the application in the Cartesian closed category by the following lemma.

Lemma 3. *Let* $\mathfrak{s} : !\mathfrak{A} \multimap !\mathfrak{B} \multimap \mathfrak{C}$ *and* $\mathfrak{t} : !\mathfrak{A} \multimap \mathfrak{B}$. *Thus* $\mathfrak{s} : \mathfrak{A} \longrightarrow (!\mathfrak{B} \multimap \mathfrak{C})$ *and* $\mathfrak{t} : \mathfrak{A} \longrightarrow \mathfrak{B}$ *in* $K_!(\mathfrak{G})$. *Then*

$$\mathfrak{s}@(\mathrm{der}; !\mathfrak{t}) \in (\langle [\mathfrak{s}], [\mathfrak{t}] \rangle_{\S}\, \mathbf{eval}) \quad : !\mathfrak{A} \multimap \mathfrak{C}$$

where $\langle \cdot, \cdot \rangle$ *is the paring and* **eval** *be the evaluation map associated with the Cartesian closed category.* □

B Game Semantic Interpretation of Recursion Schemes

We give an interpretation of recursion schemes in AJM game model. Sorts are interpreted as games and terms are interpreted as strategies in games determined by their sorts. The interpretation given in this section is almost the same as the standard interpretation of simply-typed lambda calculus (with recursion) in the Cartesian closed category $K_!(\mathfrak{G})$. The only difference is the domain of the interpretation of terms. In the standard interpretation, a term is interpreted as a morphism of the category, i.e. an equivalence class of strategies in this setting, whereas in the interpretation in this section, a terms is interpreted as a strategy.

Let $\mathcal{G} = (\Sigma, \mathcal{N}, \mathcal{R}, S)$ be a recursion scheme. Given non-terminal $F \in \mathcal{N}$ such that $\mathcal{R}(F) = \lambda x_1 \ldots x_k.M$, a set $\mathrm{fv}(\mathcal{R}(F)) \subseteq \Sigma \cup \mathcal{N}$ is defined by $\mathrm{fv}(M) \backslash \{x_1, \ldots, x_k\}$. We say $\mathcal{G}$ is *loop-free* if there exists a partial order $\preceq$ on $\mathcal{N}$ that satisfies $\forall F \in \mathcal{N}.\ \forall F' \in \mathrm{fv}(\mathcal{R}(F)).\ F' \prec F$ (here $F' \prec F$ if and only if $F' \preceq F$ and $F' \neq F$). There is a one-to-one correspondence between loop-free recursion schemes and simply-typed lambda terms in η-long form.

We first give the interpretation of the base sort o, written $\mathfrak{O}$. The game $\mathfrak{O}$ is defined as $(\mathcal{M}, \lambda, P, \approx, \vdash)$, where $\mathcal{M} = \{o\}$, $\lambda(o) = O$, $P = \{\epsilon, o\}$, $\approx = \{(\epsilon, \epsilon), (o, o)\}$ and $\vdash = \{\ \}$. The interpretation of sorts are given by:

$$\begin{aligned} \langle\!|o|\!\rangle &= \mathfrak{O} \\ \langle\!|A \to B|\!\rangle &= !\langle\!|A|\!\rangle \multimap \langle\!|B|\!\rangle \\ \langle\!|A_1, \ldots, A_n \vdash B|\!\rangle &= !\langle\!|A_1|\!\rangle \otimes \cdots \otimes !\langle\!|A_n|\!\rangle \multimap \langle\!|B|\!\rangle \end{aligned}$$

To define the interpretation of terms, we give some operations on strategies. Every strategy $\mathfrak{s} : \mathfrak{A}_i \multimap \mathfrak{B}$ can be regarded as a strategy of $\mathfrak{A}_1 \otimes \mathfrak{A}_2 \otimes \cdots \otimes \mathfrak{A}_k \multimap \mathfrak{B}$ (here $k \geq i$), which we write as $\mathit{Weaken}_{\mathfrak{A}_1 \otimes \mathfrak{A}_2 \otimes \cdots \otimes \mathfrak{A}_k \multimap \mathfrak{B}}(\mathfrak{s})$. Every strategy $\mathfrak{s} : \mathfrak{A} \otimes \mathfrak{B}_1 \otimes \cdots \otimes \mathfrak{B}_k \multimap \mathfrak{C}$ can be regarded as a strategy of $\mathfrak{A} \multimap (\mathfrak{B}_1 \multimap (\mathfrak{B}_2 \multimap \cdots \multimap (\mathfrak{B}_k \multimap \mathfrak{C})))$, which we write as $\Lambda_{\mathfrak{B}_1, \ldots, \mathfrak{B}_k}(\mathfrak{s})$.

Let $\mathcal{G}$ be a loop-free recursion scheme. The interpretation of a term M that possible contains non-terminals of $\mathcal{G}$ is given by:

$$\begin{aligned} \langle\!|\widetilde{a} :: \widetilde{A}, \widetilde{x} :: \widetilde{B} \vdash a_i :: A_i|\!\rangle &= \mathit{Weaken}_{\langle\!|\widetilde{A}, \widetilde{B} \vdash A_i|\!\rangle}(\mathrm{der}_{\langle\!|A_i|\!\rangle}) \\ \langle\!|\widetilde{a} :: \widetilde{A}, \widetilde{x} :: \widetilde{B} \vdash x_i :: B_i|\!\rangle &= \mathit{Weaken}_{\langle\!|\widetilde{A}, \widetilde{B} \vdash B_i|\!\rangle}(\mathrm{der}_{\langle\!|B_i|\!\rangle}) \\ \langle\!|\widetilde{a} :: \widetilde{A}, \widetilde{x} :: \widetilde{B} \vdash M\, N :: C|\!\rangle &= \langle\!|\widetilde{a} :: \widetilde{A}, \widetilde{x} :: \widetilde{B} \vdash M :: D \to C|\!\rangle @ \langle\!|\widetilde{a} :: \widetilde{A}, \widetilde{x} :: \widetilde{B} \vdash N :: D|\!\rangle \\ \langle\!|\widetilde{a} :: \widetilde{A}, \widetilde{x} :: \widetilde{B} \vdash F :: C|\!\rangle &= \mathit{Weaken}_{\langle\!|\widetilde{A}, \widetilde{B} \vdash C|\!\rangle}(\Lambda_{\langle\!|D_1|\!\rangle, \ldots, \langle\!|D_k|\!\rangle}(\langle\!|\widetilde{x} :: \widetilde{A}, \widetilde{y} :: \widetilde{D} \vdash M :: o|\!\rangle)) \\ & \qquad \begin{pmatrix} C = D_1 \to D_2 \to \cdots \to D_k \to o \\ \mathcal{R}(F) = \lambda y_1 \ldots y_k.M \end{pmatrix} \end{aligned}$$

Note that the inductive definition above is well-defined because the recursion scheme is loop-free.

For a recursion scheme that is not necessarily loop-free, its interpretation is defined as the union of its *loop-free approximations*. A recursion scheme is approximated by a loop-free recursion scheme with an additional term constructor

Ω of sort o that means divergence. The interpretation of Ω is given by:

$$\langle\!|\widetilde{a} :: \widetilde{A}, \widetilde{x} :: \widetilde{B} \vdash \Omega :: o|\!\rangle = \Omega_{\langle\!|\widetilde{A},\widetilde{B} \vdash o|\!\rangle}$$

where $\Omega_{\mathfrak{A}} = \{\ \}$ for every game $\mathfrak{A}$.

Definition 7 (Loop-free approximation). *Let $\mathcal{G} = (\Sigma, \mathcal{N}, \mathcal{R}, S)$ be a recursion scheme and suppose $\mathcal{N} = \{F_1, F_2, \ldots, F_n\}$. For every natural number k, we define $\mathcal{N}^{[k]}$ as the set of variables $\{{F_i}^{[k]} \mid F_i \in \mathcal{N}\}$ and $M^{[k]} = M[{F_1}^{[k]}/F_1, \ldots, {F_n}^{[k]}/F_n]$. The loop-free recursion scheme $\mathcal{G}^{[k]}$ is defined by $\mathcal{G}^{[k]} = (\Sigma, \bigcup_{0 \le j \le k} \mathcal{N}^{[k]}, \mathcal{R}', S^{[k]})$, where $\mathcal{R}'$ is given by:*

- $\mathcal{R}'({F_i}^{[0]}) = \lambda\widetilde{x}.\Omega.$
- $\mathcal{R}'({F_i}^{[j+1]}) = \lambda\widetilde{x}.M^{[j]}$ *if* $\mathcal{R}(F) = \lambda\widetilde{x}.M.$ □

For a term M that possibly contains non-terminals of $\mathcal{G}$, its approximation $M^{[k]}$ is a term $M[{F_1}^{[k]}/F_1, \ldots, {F_n}^{[k]}/F_n]$ that contains non-terminals of $\mathcal{G}^{[k]}$.

Definition 8 (Interpretation of Recursion Schemes). *Let $\mathcal{G}$ be a recursion scheme and M be a term possibly containing non-terminals of $\mathcal{G}$. We define $\langle\!|M|\!\rangle_{\mathrm{rec}} = \bigcup_{k \in \mathbb{N}} \langle\!|M^{[k]}|\!\rangle$.* □

A recursion scheme $\mathcal{G} = (\Sigma, \mathcal{N}, \mathcal{R}, S)$ is interpreted as the strategy $\langle\!|S|\!\rangle_{\mathrm{rec}}$. We define $\langle\!|\mathcal{G}|\!\rangle = \langle\!|S|\!\rangle_{\mathrm{rec}}$. The value tree of $\mathcal{G}$ can be constructed from $\langle\!|S|\!\rangle_{\mathrm{rec}}$ as follows.

Assume $\Sigma = \{a_i :: A_i \mid 1 \le i \le k\}$, where

$$A_i = \underbrace{o \to \cdots \to o}_{arity(a_i)} \to o.$$

Let

$$\mathfrak{A}_i = \langle\!|A_i|\!\rangle = !\mathfrak{O}_{i,1} \multimap !\mathfrak{O}_{i,2} \multimap \cdots \multimap !\mathfrak{O}_{i,arity(a_i)} \to \mathfrak{O}_i$$

where $\mathfrak{O}_{i,j}$ (resp. $\mathfrak{O}_i$) is the game same as $\mathfrak{O}$ except that its unique move is named $o_{i,j}$ (resp. o_i). Then $\langle\!|S|\!\rangle_{\mathrm{rec}}$ is a strategy for game $!\mathfrak{A}_1 \otimes \cdots \otimes !\mathfrak{A}_k \multimap \mathfrak{O}_\epsilon$, whose O- and P-moves are given by:

$$\begin{aligned}
\mathcal{M}^O = \{o_\epsilon\}
&\cup \mathcal{I} \times (\mathcal{I} \times \{o_{1,j} \mid 1 \le j \le arity(a_1)\}) \\
&\cup \mathcal{I} \times (\mathcal{I} \times \{o_{2,j} \mid 1 \le j \le arity(a_2)\}) \\
&\vdots \\
&\cup \mathcal{I} \times (\mathcal{I} \times \{o_{k,j} \mid 1 \le j \le arity(a_k)\})
\end{aligned}$$

$$\mathcal{M}^P = \mathcal{I} \times \{o_i \mid 1 \le i \le k\}$$

Intuitively $\mathcal{M}^P$ corresponds to nodes in the tree and $\mathcal{M}^O$ to queries to the tree: o_ϵ asks what is the root node, and $(\xi, (\Box, o_{i,j}))$ asks what is the jth child of the node (ξ, o_i). We formalise this idea. Let f be the partial function that induces $\langle\!|S|\!\rangle_{\mathrm{rec}}$. Recall that a Σ-labelled tree is a partial function from $\{1, 2, \ldots, K\}$ to Σ, where $K = \max_{1 \leq i \leq k} \mathit{arity}(a_i)$. We first define a partial function $\hat{T}_f : \{1, 2, \ldots, K\}^* \rightharpoonup \mathcal{M}^P$, and then construct a tree $T_f : \{1, 2, \ldots, K\}^* \rightharpoonup \Sigma$. The partial function $\hat{T}_f$ is inductively defined by:

$$\begin{aligned} \hat{T}_f(\epsilon) &= f(o_\epsilon) \\ \hat{T}_f(p \cdot d) &= f(\ (\xi, (\Box, o_{i,d}))\ ) \qquad \text{if } \hat{T}_f(p) = (\xi, o_i) \text{ and } 1 \leq d \leq \mathit{arity}(a_i). \end{aligned}$$

The tree T_f is defined by:

$$T_f(p) = a_i \qquad \text{if and only if} \qquad \hat{T}_f(p) = (\xi, o_i) \text{ for some } \xi.$$

Lemma 4. *Let $\mathfrak{s}$ and $\mathfrak{t}$ be strategies for $\langle\!|\Sigma \vdash o|\!\rangle$, and f and g be corresponding partial functions. If $\mathfrak{s} \approx \mathfrak{t}$, then $T_f = T_g$.*

Proof. Let $p = d_1 d_2 \ldots d_n \in \{1, 2, \ldots, K\}^*$ and assume that $\hat{T}_f(p) = (\xi_n, o_{i_n})$. By definition of $\hat{T}_f$, we know that

$$s = o_\epsilon \cdot (\xi_0, o_{i_0}) \cdot (\xi_0, (\Box, o_{i_0,d_1})) \cdot (\xi_1, o_{i_1}) \cdot \cdots \cdot (\xi_{n-1}, (\Box, o_{i_{n-1},d_n})) \cdot (\xi, o_{i_n}) \in \mathfrak{s}$$

for some ξ_ν and i_ν $(1 \leq \nu < n)$. Then by induction on n, one can prove that

$$t = o_\epsilon \cdot (\zeta_0, o_{j_0}) \cdot (\zeta_0, (\Box, o_{j_0,d_1})) \cdot (\zeta_1, o_{j_1}) \cdot \cdots \cdot (\zeta_{n-1}, (\Box, o_{j_{n-1},d_n})) \cdot (\zeta, o_{j_n}) \in \mathfrak{t}$$

for some ζ_ν and j_ν $(1 \leq \nu \leq n)$ such that $s \approx t$. Since $s \approx t$, $i_\nu = j_\nu$ for every ν $(1 \leq \nu \leq n)$. Thus $T_f(p) = T_g(p) = a_{i_n}$. □

Lemma 5. *Let $\mathcal{G}$ be a recursion scheme, M be a term that possibly contains non-terminals of $\mathcal{G}$, f be the partial function that induces $\langle\!|M|\!\rangle_{\mathrm{rec}}$, and $p \in \{1, 2, \ldots, K\}^*$, where $K = \max_{1 \leq i \leq k} \mathit{arity}(a_i)$. If $(M^\perp)(p) = a_i \neq \perp$, then $T_f(p) = a_i$.*

Proof. By induction on p. □

The tree constructed from the strategy $\langle\!|S|\!\rangle_{\mathrm{rec}}$ is the value tree.

Theorem 7. *Let $\mathcal{G} = (\Sigma, \mathcal{N}, \mathcal{R}, S)$ be a recursion scheme and f be the partial function that induces $\langle\!|S|\!\rangle_{\mathrm{rec}}$. Then $[\![\mathcal{G}]\!] = T_f$.*

Proof. Let $p \in \{1, 2, \ldots, K\}^*$, where $K = \max_{1 \leq i \leq k} \mathit{arity}(a_i)$. Suppose that $[\![\mathcal{G}]\!](p) = a_i$. As we have seen in Section 2, we can assume without loss of generality that $a_i \neq \perp$. By definition, $S \longrightarrow_{\mathcal{G}}^* M$ for some M such that $(M^\perp)(p) = a_i$. Let g be the partial function that induces $\langle\!|M|\!\rangle_{\mathrm{rec}}$. Then we have $T_g(p) = a_i$ by Lemma 5. Because $K_!(\mathfrak{G})$ is Cartesian closed, $S \longrightarrow_{\mathcal{G}}^* M$ implies $\langle\!|S|\!\rangle_{\mathrm{rec}} \approx \langle\!|M|\!\rangle_{\mathrm{rec}}$. Thus by Lemma 4, we have $T_f(p) = a_i$ as desired. □

For a given set of moves $\mathcal{M}$, its *colouring* is a partial function from $\mathcal{M}$ to Q. Colouring can be represented by a set of colour assignments to moves $m : q$, e.g. $\{m_1 : q_1, m_2 : q_2, \dots\}$ where $m_i \neq m_j$ if $i \neq j$. Let $\mathfrak{s} : \langle\!|\Sigma \vdash o|\!\rangle$ be a strategy representing a tree, f be the partial function that induces $\mathfrak{s}$, and $\theta = \{m_1 : q_1, m_2 : q_2, \dots\}$ be colouring. We say a pair $(\mathfrak{s}, \theta)$ *respects the transition relation* δ, written $\delta \vDash (\mathfrak{s}, \theta)$, if

1. for every $m : q \in \theta$, we have $f(m) : q \in \theta$, and
2. for every $(\xi, o_i) : q \in \theta$ and $(\xi, (\square, o_{i,d})) : q_d \in \theta$ $(1 \leq d \leq \mathit{arity}(a_i))$, we have $(q, a_i, q_1 q_2 \dots q_{\mathit{arity}(a_i)}) \in \delta$.

Lemma 6. *Let $\mathcal{G}$ be a recursion scheme and $\mathcal{A}$ be a trivial tree automaton. Then $[\![\mathcal{G}]\!]$ is accepted by $\mathcal{A}$ if and only if there exists colouring θ such that $\delta \vDash (\langle\!|\mathcal{G}|\!\rangle, \theta)$.*

Proof. Easy. □

We write $\delta \vDash \Diamond(\mathfrak{s}, \theta)$ if there exists $\theta' \supseteq \theta$ such that $\delta \vDash (\mathfrak{s}, \theta')$. Let $\theta_0 = \{o_\epsilon : q_0, f(o_\epsilon) : q_0\}$. A sequence $\theta_0 \subseteq \theta_1 \subseteq \cdots \subseteq \theta_i \subseteq \cdots$ of colouring is *fair* if for every $(\xi, o_j) : q \in \theta_i$ and d such that $1 \leq d \leq \mathit{arity}(a_j)$, there exists i' such that $(\xi, (\square, o_{j,d})) : q_d \in \theta_{i'}$. We say colouring θ is *well-justified* if $\mathrm{dom}(\theta) = \{m \mid \exists q.\ m : q \in \theta\}$ is well-justified.

Lemma 7. *Let $\mathcal{G}$ be a recursion scheme and $\mathcal{A}$ be a trivial tree automaton. Then $[\![\mathcal{G}]\!]$ is accepted by $\mathcal{A}$ if and only if there exists a (possibly infinite) fair sequence $\theta_0 \subseteq \theta_1 \subseteq \cdots \subseteq \theta_i \subseteq \cdots$ of well-justified colouring that satisfies that $\delta \vDash \Diamond(\mathfrak{s}, \theta_i)$ for every i and that $\theta_{i+1} \backslash \theta_i$ contains an O-move and a P-move for every i.*

Proof. ($\Rightarrow$) Suppose that $[\![\mathcal{G}]\!]$ is accepted by $\mathcal{A}$. Let f be the partial function that induces $\langle\!|\mathcal{G}|\!\rangle$. Then by Lemma 6, there exists θ such that $\delta \vDash (\langle\!|\mathcal{G}|\!\rangle, \theta)$. We define a sequence $\theta_0 \subseteq \cdots \subseteq \theta_i \subseteq \cdots$ such that $\theta_i \subseteq \theta$ for every i by induction on i. We define $\theta_0 = \{o_\epsilon : q_0, f(o_\epsilon) : q_0\} \subseteq \theta$. Suppose that θ_i is given. Let $\mathcal{O}$ be the set of all O-moves of the form $(\xi, (\square, o_{i,d}))$ that is justified by a move in $\mathrm{dom}(\theta_i)$ and not contained by $\mathrm{dom}(\theta_i)$. If $\mathcal{O}$ is empty, then θ_i is the last element of the sequence. If not, let $m \in \mathcal{O}$. Since $\delta \vDash (\langle\!|\mathcal{G}|\!\rangle, \theta)$ and $m \in \theta_i \subseteq \theta$, we know that $m : q \in \theta$ for some q. Then θ_{i+1} is defined by $\theta_i \cup \{m : q, f(m) : q\}$. Then the sequence satisfies the requirement.

($\Leftarrow$) Suppose that there exists a sequence $\theta_0 \subseteq \cdots \subseteq \theta_i \subseteq \cdots$ that satisfies the requirement. Let $\theta = \bigcup_i \theta_i$. Then θ satisfies the requirement of Lemma 6, and thus $[\![\mathcal{G}]\!]$ is accepted by $\mathcal{A}$. □

C System$^\triangleright$: Yet Another Rigid Intersection Type System

Here we introduce yet another rigid intersection type system, called *System*$^\triangleright$, that has more direct connection to game semantics than the rigid type system in Section 3.

The syntax of types and intersections are give by:

Types	τ, σ	$::= q \mid \alpha \to \tau$
Intersections	α, β	$::= \bigcap\{(\xi, \tau_\xi) \mid \xi \in X\}$

where $X \subseteq \mathcal{I}$. We often write $\bigcap_{\xi\in X}(\xi,\tau_\xi)$ for $\bigcap\{(\xi,\tau_\xi) \mid \xi \in X\}$. The intersection $\bigcap_{\xi\in X}(\xi,\tau_\xi)$ means the indexed intersection of τ_ξ. The equivalence on intersections is defined by

$$\bigcap_{\xi\in X}(\xi,\tau_\xi) = \bigcap_{\zeta\in Y}(\zeta,\sigma_\zeta) \quad \text{iff} \quad X = Y \text{ and } \forall\xi \in X.\ \tau_\xi = \sigma_\xi.$$

This can be considered as a variant of rigid intersections, since indexes are significant, e.g.

$$\bigcap_{\xi\in\{l\cdot\Box\}}(\xi,\tau_\xi) \neq \bigcap_{\zeta\in\{r\cdot\Box\}}(\zeta,\sigma_\zeta)$$

even if $\tau_{l\cdot\Box} = \sigma_{r\cdot\Box}$. Note that, unlike the rigid type system in Section 3, types and intersections are completely separated.

An intersection $\bigcap_{\xi\in X}(\xi,\tau_\xi)$ can be represented by a partial function from indexes to types that is defined on X and maps ξ to τ_ξ. For notational convenience, we introduce a symbol $\bullet$ that means undefined type of sort A. We write $(\zeta,\bullet) \in \{(\xi,\tau_\xi) \mid \xi \in X\}$ if $\zeta \notin X$. Then an intersection can be represented by a partial function from indexes to the union of types and $\{\bullet\}$. When we allow τ_ξ to be $\bullet$, an intersection $\bigcap_{\xi\in X}(\xi,\tau_\xi)$ can be written as $\bigcap_{\xi\in\mathcal{I}}(\xi,\sigma_\xi)$, where $\sigma_\xi = \tau_\xi$ if $\xi \in X$ and $\sigma_\xi = \bullet$ if $\xi \notin X$. We simply write $\bigcap_\xi(\xi,\tau_\xi)$ for $\bigcap_{\xi\in X}(\xi,\tau_\xi)$. We define the empty intersection $\odot$ by $\odot = \bigcap_\xi(\xi,\bullet)$.

Now the sorting relations are divided into two kinds of judgements, τA for a type τ and a sort A, and $\alpha :: !A$ for an intersection α and a sort A. Here the notation $!A$ comes from the interpretation of the intuitionistic implication $\to$ in the linear logic: $A \to B = !A \multimap B$. The sorting relations are defined by the following rules:

$$\frac{}{q :: o} \qquad \frac{\alpha :: !A \qquad \tau :: B}{\alpha \to \tau :: A \to B} \qquad \frac{\tau_\xi :: A \quad (\forall\xi \in X)}{\bigcap_{\xi\in X}(\xi,\tau_\xi) :: !A}\,.$$

By definition, $\sqcap\{\ \} :: !A$ for every sort A. We sometimes write τ^A and α^A to make clear that τ and α are types and intersections of sort A.

The definition of type environments is similar to the one in Section 3, i.e. a finite set of binding $\{x_1 : \alpha_1, \ldots, x_n : \alpha_n\}$. In the following, we implicitly assume well-sortedness of type environments, i.e. $x^A : \alpha \in \Gamma$ implies $\alpha :: !A$. We define $\mathrm{dom}(\Gamma)$ and $\Gamma(x)$ in the same way as in Section 3, except that if $x \notin \mathrm{dom}(\Gamma)$, then $\Gamma(x) = \bigcap\{\ \}$ $(= \bigcap_{\xi\in\{\ \}}(\xi,\tau_\xi)\,)$. An important operation on type environments is *indexed product*. Let $X \subseteq \mathcal{I}$ be a subset of indexes and Γ_ξ be type environment for every $\xi \in X$. Then $\bigcap_{\xi\in X}(\xi,\Gamma_\xi)$ is defined by

$$\Big(\bigcap_{\xi\in X}(\xi,\Gamma_\xi)\Big)(x) = \bigcap\Big\{\big(\langle\xi,\zeta\rangle,\tau_\zeta\big) \;\Big|\; \Gamma_\xi(x) = \bigcap_{\zeta\in Y_\xi}(\zeta,\tau_\zeta) \text{ and } \zeta \in Y_\xi\Big\}.$$

For two intersections, their *rigid intersection* is defined by

$$(\bigcap_{\xi\in X}(\xi,\tau_\xi)) \pitchfork (\bigcap_{\zeta\in Y}(\zeta,\sigma_\zeta)) = \bigcap\Big(\{(l\cdot\xi,\ \tau_\xi) \mid \xi \in X\} \cup \{(r\cdot\zeta,\ \sigma_\zeta) \mid \zeta \in Y\}\Big).$$

The *rigid intersection on type environments* is defined by point-wise intersection, i.e.

$$(\Gamma \pitchfork \Delta)(x) = \Gamma(x) \pitchfork \Delta(x).$$

Type judgements are also divided into two kinds, type judgements $\Gamma \vdash M : \tau$ and intersection judgements $\Gamma \vdash M : \alpha$, similar to sorting relations. Typing rules are listed as follows:

$$\frac{\Gamma, \widetilde{x} : \widetilde{\alpha} \vdash M : q}{\Gamma \vdash F : \widetilde{\alpha} \to q}\ [\mathcal{R}(F) = \lambda\widetilde{x}.M^o] \qquad \frac{\dagger}{x : \bigsqcap\{(\Box, \tau)\} \vdash x : \tau}\ [x :: A \text{ and } \tau :: A]$$

$$\frac{\Gamma \vdash M : \alpha \to \tau \qquad \Delta \vdash N : \alpha}{\Gamma \pitchfork \Delta \vdash M\ N : \tau} \qquad \frac{\Gamma_\xi \vdash M : \tau_\xi \quad (\forall \xi \in X)}{\bigsqcap_{\xi \in X}(\xi, \Gamma_\xi) \vdash M : \bigsqcap_{\xi \in X}(\xi, \tau_\xi)}$$

$$\frac{\Diamond}{\{\,\} \vdash M : \bullet}.$$

Note here that Ω has no typing rule. Thus $\Gamma \vdash \Omega : \tau$ is not derivable for any Γ and τ.

We then introduce the extension relation, incomplete typing rules and incomplete judgements as in Section 3.3. The extension relations are defined by the following rules. Here $X \in \{P, O\}$ and $\overline{O} = P$ and $\overline{P} = O$.

$$\overline{\bullet^{A_1 \to \ldots \to A_n \to q} \lhd_O \sqcap\{\,\}^{A_1} \to \ldots \to \sqcap\{\,\}^{A_n} \to q}$$

$$\frac{\beta \lhd_X \beta'}{\alpha \to \beta \lhd_X \alpha \to \beta'} \qquad \frac{\alpha \lhd_{\overline{X}} \alpha'}{\alpha \to \beta \lhd_X \alpha' \to \beta}$$

$$\frac{\zeta \in X \qquad \tau_\zeta \lhd_X \sigma_\zeta \qquad \tau_\xi = \sigma_\xi\ (\forall \xi \in X \backslash \{\zeta\})}{\bigsqcap_{\xi \in X}(\xi, \tau_\xi) \lhd_X \bigsqcap_{\xi \in X}(\xi, \sigma_\xi)}$$

$$\frac{\zeta \notin X \qquad \bullet \lhd_O \sigma_\zeta \qquad \tau_\xi = \sigma_\xi\ (\forall \xi \in X)}{\bigsqcap_{\xi \in X}(\xi, \tau_\xi) \lhd_O \bigsqcap_{\xi \in X \cup \{\zeta\}}(\xi, \sigma_\xi)}$$

The extension relation $\Gamma \lhd_X \Gamma'$ is defined as there exists x such that $\Gamma(x) \lhd_X \Gamma'(x)$ and $\Gamma(y) = \Gamma'(y)$ for every $y \neq x$. We define the extension relation $\Gamma \vdash M : \tau \lhd_X \Gamma' \vdash M : \tau'$ on type judgements as either (1) $\Gamma = \Gamma'$ and $\tau \lhd_X \tau'$, or (2) $\Gamma \lhd_{\overline{X}} \Gamma'$ and $\tau = \tau'$. The extension relation on intersection judgements is defined in the same way.

The incomplete typing rules are shown in Fig. 14. The derivation rewriting rules are shown in Fig. 15 and Fig. 16. Note that the incomplete derivation

$$\frac{\Diamond}{\{\,\} \vdash \Omega : q}$$

does not have any successor. Conversely every incomplete derivations except for the above one has a unique successor.

$$\text{(O-Var1-Y)}\quad \frac{\dagger}{x : \bigsqcap\{(\square, \tau)\} \vdash x : \tau' \ \blacktriangle\!\!\blacktriangle} \qquad [\tau \lhd_O \tau']$$

$$\text{(O-Var2-Y)}\quad \frac{\dagger}{x : \bigsqcap\{(\square, \tau')\} \vdash x : \tau \ \blacktriangle\!\!\blacktriangle} \qquad [\tau \lhd_P \tau']$$

$$\text{(O-}\Diamond\text{-Y)}\quad \frac{\Diamond}{\{\,\} \vdash M : \tau \ \blacktriangle\!\!\blacktriangle} \qquad [\bullet \lhd_O \tau]$$

$$\text{(O-Fun-Y)}\quad \frac{\Gamma, \widetilde{x} : \widetilde{\alpha} \vdash M : q}{\Gamma' \vdash F : \widetilde{\alpha}' \to q' \ \blacktriangle\!\!\blacktriangle} \qquad \begin{bmatrix} \mathcal{R}(F) = \lambda \widetilde{x}.M^o \\ \Gamma \vdash F : \widetilde{\alpha} \to q \ \lhd_O \ \Gamma' \vdash F : \widetilde{\alpha}' \to q' \end{bmatrix}$$

$$\text{(P-Fun-Y)}\quad \frac{\Gamma', \widetilde{x} : \widetilde{\alpha}' \vdash M : q' \ \blacktriangledown}{\Gamma \vdash F : \widetilde{\alpha} \to \tau} \qquad \begin{bmatrix} \mathcal{R}(F) = \lambda \widetilde{x}.M^o \\ \Gamma, \widetilde{x} : \widetilde{\alpha} \vdash M : q \ \lhd_P \ \Gamma', \widetilde{x} : \widetilde{\alpha}' \vdash M : q' \end{bmatrix}$$

$$\text{(O-App-Y)}\quad \frac{\Gamma \vdash M : \alpha \to \tau \qquad \Delta \vdash N : \alpha}{\Gamma' \uplus \Delta' \vdash M\ N : \tau' \ \blacktriangle\!\!\blacktriangle} \qquad \begin{bmatrix} \Gamma \uplus \Delta \vdash M\ N : \tau \\ \quad \lhd_O \Gamma' \uplus \Delta' \vdash M\ N : \tau' \end{bmatrix}$$

$$\text{(P-AppL-Y)}\quad \frac{\Gamma' \vdash M : \alpha' \to \tau' \ \blacktriangledown \qquad \Delta \vdash N : \alpha}{\Gamma \uplus \Delta \vdash M\ N : \tau} \qquad \begin{bmatrix} \Gamma \vdash M : \alpha \to \tau \\ \quad \lhd_P \Gamma' \vdash M : \alpha' \to \tau' \end{bmatrix}$$

$$\text{(P-AppR-Y)}\quad \frac{\Gamma \vdash M : \alpha \to \tau \qquad \Delta' \vdash N : \alpha' \ \blacktriangledown}{\Gamma \uplus \Delta \vdash M\ N : \tau} \qquad [\Delta \vdash N : \alpha \ \lhd_P \ \Delta' \vdash N : \alpha']$$

$$\text{(O-Int-Y)}\quad \frac{\Gamma'_\zeta \vdash M : \tau'_\zeta \qquad \Gamma_\xi \vdash M : \tau_\xi \ (\forall \xi \neq \zeta)}{\bigsqcap_\xi (\xi, \Gamma_\xi) \vdash M : \bigsqcap_\xi (\xi, \tau_\xi)} \qquad [\Gamma'_\zeta \vdash M : \tau'_\zeta \ \lhd_O \ \Gamma_\zeta \vdash M : \tau_\zeta]$$

$$\text{(P-Int-Y)}\quad \frac{\Gamma'_\zeta \vdash M : \tau'_\zeta \qquad \Gamma_\xi \vdash M : \tau_\xi \ (\forall \xi \neq \zeta)}{\bigsqcap_\xi (\xi, \Gamma_\xi) \vdash M : \bigsqcap_\xi (\xi, \tau_\xi)} \qquad [\Gamma_\zeta \vdash M : \tau_\zeta \ \lhd_P \ \Gamma'_\zeta \vdash M : \tau'_\zeta]$$

Fig. 14. Incomplete typing rules

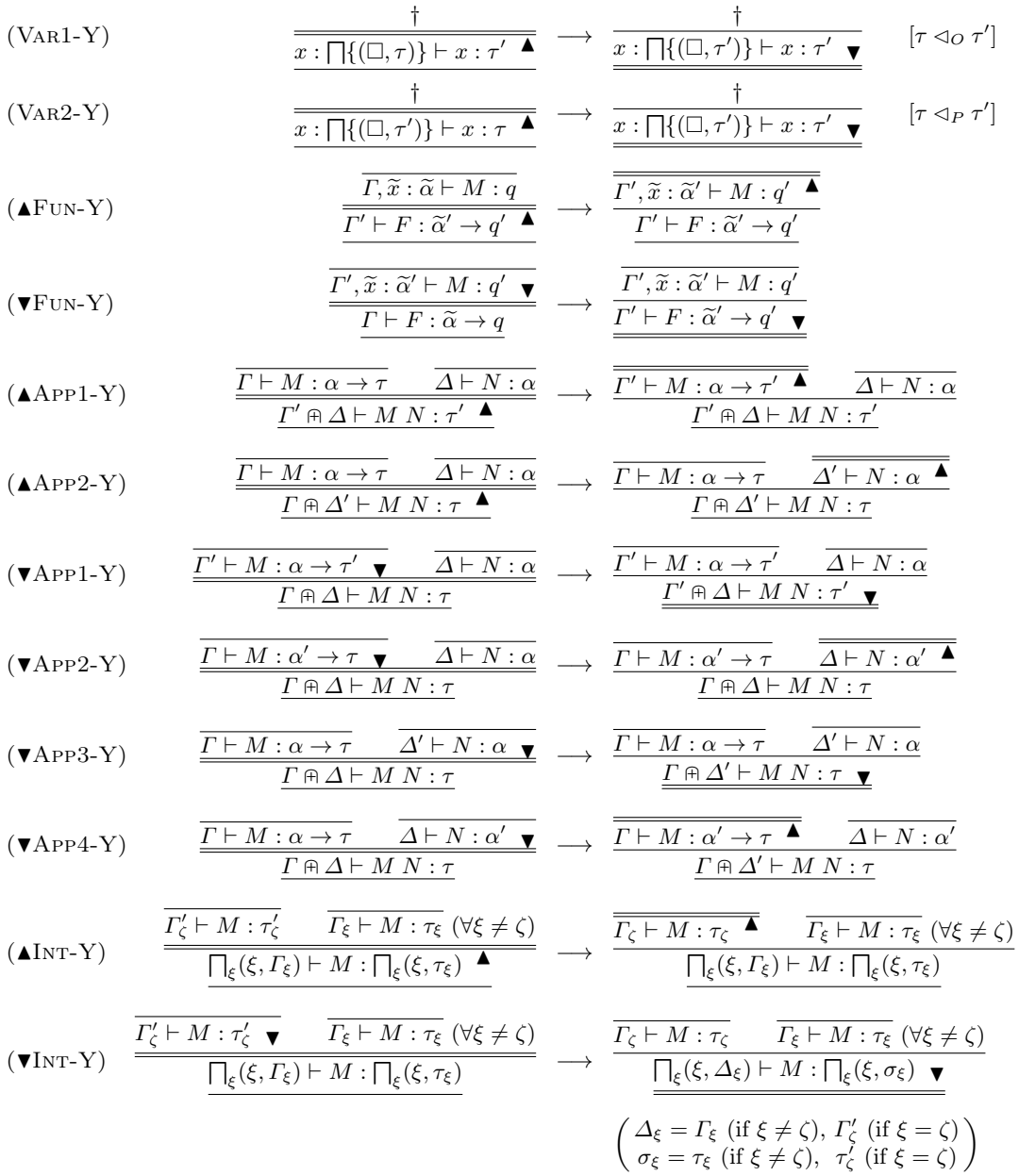

Fig. 15. Derivation rewriting rules in System$^\triangleright$ (part 1)

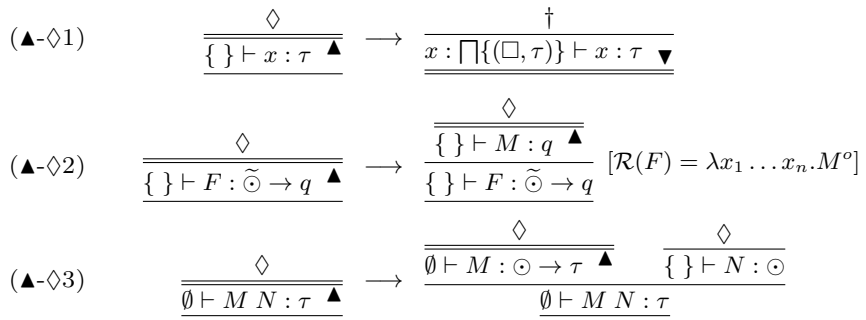

Fig. 16. Derivation rewriting rules in System$^{\triangleright}$ (part 2)

D Relation between System$^{\triangleright}$ and Game Semantics

This section gives a connection between System$^{\triangleright}$ and AJM game model. We interpret intersection types and judgements as a colouring. By this interpretation, O-extension (resp. P-extension) on types corresponds to adding an O-move (resp. a P-move) to the domain of the colouring. Then the derivation rewriting system induces a function from O-moves to P-moves, that coincides with the partial function determined by the strategy.

We define the map ϖ from types and intersections to colouring, i.e. sets of the form $\{m_1 : q_1, \ldots, m_k : q_k\}$ where $m_1, \ldots, m_k$ are pairwise distinct moves. The images of ϖ can be described more precisely:

- $\mathrm{dom}(\varpi(\tau)) \subseteq \mathcal{M}_{\llparenthesis A \rrparenthesis}$ for $\tau :: A$, and
- $\mathrm{dom}(\varpi(\alpha)) \subseteq \mathcal{M}_{\llparenthesis !A \rrparenthesis}$ for $\alpha :: !A$.

The map ϖ is defined by induction on the structure of sorts as follows:

$$\begin{aligned}
\varpi(q) &= \{o : q\} \\
\varpi(\bullet) &= \{\ \} \\
\varpi(\alpha \to \tau) &= \varpi(\alpha) \uplus \varpi(\tau) \\
&\qquad \left(\begin{array}{l}\text{Here } \uplus \text{ is the disjoint union associated with} \\ \mathcal{M}_{\llparenthesis A \to B \rrparenthesis} = \mathcal{M}_{\llparenthesis !A \multimap B \rrparenthesis} = \mathcal{M}_{\llparenthesis !A \rrparenthesis} + \mathcal{M}_{\llparenthesis B \rrparenthesis}.\end{array}\right) \\
\varpi\Big(\prod_{\xi \in X} (\xi, \tau_\xi)\Big) &= \bigcup_{\xi \in X} \{(\xi, m) : q \mid m : q \in \varpi(\tau_\xi)\}.
\end{aligned}$$

Lemma 8. *The map ϖ gives a bijective correspondence between types of sort A and non-empty and well-justified subsets of moves in $\llparenthesis A \rrparenthesis$ with colouring. Similarly, ϖ gives a bijective correspondence between intersections α of sort $!A$ and (possibly empty) well-justified subsets of moves in $\llparenthesis !A \rrparenthesis$ with colouring.*

Proof. By mutual induction on the structure of types and intersections, we can prove that the images of ϖ are compatible and well-justified. The injectivity of ϖ can also be proved by (mutual) induction on the structure of types and intersection. The surjectivity of ϖ can be proved by induction on the structure of sorts, in which we use induction on the derivation of compatibility of indexes for the case $!A$. □

Let $Z \in \{P, O\}$. The following lemma shows that Z-extension is equivalent to adding a Z-move.

Lemma 9. *Let $Z \in \{P, O\}$ and τ and τ' be types of the same sort A. Then $\tau \lhd_Z \tau'$ if and only if there exists a Z-move m such that $m \notin \mathrm{dom}(\varpi(\tau))$ and $\varpi(\tau) \cup \{m : q\} = \varpi(\tau')$ for some q. Similar statements hold for intersections.*

Proof. ($\Rightarrow$) By mutual induction on the derivation of $\tau \lhd_O \tau'$, $\tau \lhd_P \tau'$, $\alpha \lhd_O \alpha'$ and $\alpha \lhd_P \alpha'$.

($\Leftarrow$) By induction on the structure of sorts. □

The map ϖ can be extended to a map from judgements by

$$\varpi(\{x_i : \alpha_i \mid 1 \le i \le n\} \vdash M : \tau) = \biguplus_{1 \le i \le n} \varpi(\alpha_i) \uplus \varpi(\tau)$$

$$\varpi(\{x_i : \alpha_i \mid 1 \le i \le n\} \vdash^k_{\Uparrow} M : \beta) = \biguplus_{1 \le i \le n} \varpi(\alpha_i) \uplus \varpi(\beta)$$

where $\uplus$ means the disjoint union associated with$\mathcal{M}_{(\!|A_1,\ldots,A_n \vdash B|\!)} = \mathcal{M}_{(\!|!A_1|\!)} + \cdots + \mathcal{M}_{(\!|!A_n|\!)} + \mathcal{M}_{(\!|B|\!)}$. Extension relations on judgements enjoy a property similar to Lemma 9. Let $\Gamma \vdash M : \tau$ be a judgement and $\Gamma' \vdash M : \tau'$ be its extension. The there exists $m : q$ that satisfies a condition similar to Lemma 9. We write $(\Gamma \vdash M : \tau) + (m : q)$ for $\Gamma' \vdash M : \tau'$. Notations $(\Gamma \vdash M : \alpha) + (m : q)$ and $(\Gamma \vdash M : \tau) + (m : q) + (m' : q')$ are defined in the same way.

For a given sorted term $\widetilde{x} :: \widetilde{a} \vdash M :: B$, we define a partial function $\mathfrak{r}(M) : \mathcal{M}^O_{(\!|\widetilde{A} \vdash B|\!)} \rightharpoonup \mathcal{M}^P_{(\!|\widetilde{A} \vdash B|\!)}$ by $\mathfrak{r}(M)(m) = m'$ if and only if

$$\mathcal{D} = \dfrac{\vdots}{(\Gamma \vdash M : \tau) + (m : q)\ \blacktriangle} \longrightarrow^* \dfrac{\vdots}{(\Gamma \vdash M : \tau) + (m : q) + (m' : q')\ \blacktriangledown}$$

for some judgement $\Gamma \vdash M : \tau$ and incomplete derivation $\mathcal{D}$. Perhaps surprisingly, $\mathfrak{r}(M)$ is a well-defined partial function, i.e. m' and q' are determined independently from the judgement $\Gamma \vdash M : \tau$ and the derivation $\mathcal{D}$.

Lemma 10. *Let $\mathcal{G}$ be a recursion scheme and $\widetilde{x} :: \widetilde{A} \vdash M :: B$ be a term that possibly contains non-terminals of $\mathcal{G}$. Let $\mathcal{D}_1$ and $\mathcal{D}_2$ be complete derivations whose conclusions are $\Gamma_1 \vdash M : \tau_1$ and $\Gamma_2 \vdash M : \tau_2$. Let m be an O-move in $(\!|\widetilde{A} \vdash B|\!)$ and $q \in Q$. Suppose that $(\Gamma_1 \vdash M : \tau_1) + (m : q)$ and $(\Gamma_2 \vdash M : \tau_2) + (m : q)$ are well-defined. Then*

$$\mathcal{D}'_1 = \dfrac{\vdots}{(\Gamma_1 \vdash M : \tau_1) + (m : q)\ \blacktriangle} \longrightarrow^* \dfrac{\vdots}{(\Gamma_1 \vdash M : \tau_1) + (m : q) + (m' : q')\ \blacktriangledown} = \mathcal{D}''_1$$

for some $\mathcal{D}_1''$ if and only if

$$\mathcal{D}_2' = \dfrac{\vdots}{(\Gamma_2 \vdash M : \tau_2) + (m : q)}\ \blacktriangle \ \longrightarrow^* \ \dfrac{\vdots}{(\Gamma_2 \vdash M : \tau_2) + (m : q) + (m' : q')}\ \blacktriangledown = \mathcal{D}_2''$$

for some $\mathcal{D}_2''$. A similar statement holds for intersection judgements.

Proof. Both directions are equivalent. The claim can be proved by induction on the length of $\longrightarrow^*$ and case analysis on M.

If M is a variable, say x_i, then $B = A_i$. If $\Gamma_1 \vdash M : \tau_1$ is the empty judgement, then m is the unique move $m \in \mathcal{M}^O_{\langle\!|\widetilde{A} \vdash A_i|\!\rangle}$ such that $\neg\exists n.\ n \vdash_{\langle\!|\widetilde{A} \vdash A_i|\!\rangle} m$ and thus $\Gamma_2 \vdash M : \tau_2$ must also be the empty judgement, in which case the statement trivially holds. We assume that $\mathcal{D}_1$ and $\mathcal{D}_2$ are non-empty. Thus

$$\mathcal{D}_1' = \dfrac{\dagger}{(x_i : \bigcap\{(\square, \tau_1)\} \vdash x_i : \tau_1) + (m : q)}\ \blacktriangle$$

$$\longrightarrow^* \ \dfrac{\dagger}{(x_i : \bigcap\{(\square, \tau_1)\} \vdash x_i : \tau_1) + (m : q) + (m' : q')}\ \blacktriangledown = \mathcal{D}_1''$$

By definition, the justifier of m is in $\varpi(x_i : \bigcap\{(\square, \tau_1)\} \vdash x_i : \tau_1) = \varpi(\bigcap\{(\square, \tau_1)\}) \uplus \varpi(\tau_1)$. Assume the justifier is in $\varpi(\tau_1)$. Now the rewriting sequence is written as

$$\mathcal{D}_1' = \dfrac{\dagger}{x_i : \bigcap\{(\square, \tau_1)\} \vdash x_i : \tau_1'}\ \blacktriangle \ \longrightarrow^* \ \dfrac{\dagger}{x_i : \bigcap\{(\square, \tau_1')\} \vdash x_i : \tau_1'}\ \blacktriangledown = \mathcal{D}_1''$$

where τ_1' is the unique type such that $\varpi(\tau_1') = \varpi(\tau_1) \cup \{m : q\}$. So by definition,

$$\varpi\Big(\bigcap\{\ (\square, \tau_1')\ \}\Big) = \varpi\Big(\bigcap\{\ (\square, \tau_1)\ \}\Big) \cup \{\ (\square, m) : q\ \}.$$

Therefore we know that $m' = (\square, m)$. By a similar argument, we have

$$\mathcal{D}_2' = \dfrac{\dagger}{(x_i : \bigcap\{(\square, \tau_2)\} \vdash x_i : \tau_2) + (m : q)}\ \blacktriangle$$

$$\longrightarrow^* \ \dfrac{\dagger}{(x_i : \bigcap\{(\square, \tau_2)\} \vdash x_i : \tau_2) + (m : q) + (\ (\square, m) : q')}\ \blacktriangledown = \mathcal{D}_2''$$

as required. The other case can be proved in the same way.

Next we prove the case that $M = M_1\ M_2$. Then the rewriting sequence can be divided into three stages. Let us consider the following example in which $\xrightarrow{(1)}$

means the rewriting at the first stage and so on.

$$
\begin{array}{ll}
 & \dfrac{\dfrac{\vdots}{\Gamma_1 \vdash M_1 : \alpha_1 \to \tau_1} \quad \dfrac{\vdots}{\Delta_1 \vdash M_2 : \alpha_1}}{(\Gamma_1 \uplus \Delta_1 \vdash M_1\ M_2 : \tau_1) + (m:q)\ \blacktriangle} \\
 & \\
\xrightarrow{(1)} & \dfrac{\dfrac{\vdots}{(\Gamma_1 \vdash M_1 : \alpha_1 \to \tau_1) + (n_1 : q)\ \blacktriangle} \quad \dfrac{\vdots}{\Delta_1 \vdash M_2 : \alpha_1}}{(\Gamma_1 \uplus \Delta_1 \vdash M_1\ M_2 : \tau_1) + (m:q)} \\
 & \\
\xrightarrow{(2)^*} & \dfrac{\dfrac{\vdots}{(\Gamma_1 \vdash M_1 : \alpha_1 \to \tau_1) + (n_1 : q_1) + (n_2 : q_2)\ \blacktriangledown} \quad \dfrac{\vdots}{\Delta_1 \vdash M_2 : \alpha_1}}{(\Gamma_1 \uplus \Delta_1 \vdash M_1\ M_2 : \tau_1) + (m:q)} \\
 & \\
\xrightarrow{(2)} & \dfrac{\dfrac{\vdots}{(\Gamma_1 \vdash M_1 : \alpha_1 \to \tau_1) + (n_1 : q_1) + (n_2 : q_2)} \quad \dfrac{\vdots}{(\Delta_1 \vdash M_2 : \alpha_1) + (n_2 : q_2)\ \blacktriangle}}{(\Gamma_1 \uplus \Delta_1 \vdash M_1\ M_2 : \tau_1) + (m:q)} \\
 & \\
\xrightarrow{(2)^*} & \dfrac{\dfrac{\vdots}{(\Gamma_1 \vdash M_1 : \alpha_1 \to \tau_1) + (n_1 : q_1) + (n_2 : q_2)} \quad \dfrac{\vdots}{(\Delta_1 \vdash M_2 : \alpha_1) + (n_2 : q_2) + (n_3 : q_3)\ \blacktriangledown}}{(\Gamma_1 \uplus \Delta_1 \vdash M_1\ M_2 : \tau_1) + (m:q)} \\
 & \\
\xrightarrow{(2)} & \dfrac{\dfrac{\vdots}{(\Gamma_1 \vdash M_1 : \alpha_1 \to \tau_1) + (n_1 : q_1) + (n_2 : q_2) + (n_3 : q_3)\ \blacktriangle} \quad \dfrac{\vdots}{(\Delta_1 \vdash M_2 : \alpha_1) + (n_2 : q_2) + (n_3 : q_3)}}{(\Gamma_1 \uplus \Delta_1 \vdash M_1\ M_2 : \tau_1) + (m:q)} \\
 & \\
 & \vdots \\
 & \\
\xrightarrow{(2)^*} & \dfrac{\dfrac{\vdots}{(\Gamma_1 \vdash M_1 : \alpha_1 \to \tau_1) + (n_1 : q_1) + \ldots + (n_{2k+1} : q_{2k+1})\ \blacktriangledown} \quad \dfrac{\vdots}{(\Delta_1 \vdash M_2 : \alpha_1) + (n_2 : q_2) + \cdots + (n_{2k} : q_{2k})}}{(\Gamma_1 \uplus \Delta_1 \vdash M_1\ M_2 : \tau_1) + (m:q)} \\
 & \\
\xrightarrow{(3)} & \dfrac{\dfrac{\vdots}{(\Gamma_1 \vdash M_1 : \alpha_1 \to \tau_1) + (n_1 : q_1) + \ldots + (n_{2k+1} : q_{2k+1})} \quad \dfrac{\vdots}{(\Delta_1 \vdash M_2 : \alpha_1) + (n_2 : q_2) + \cdots + (n_{2k} : q_{2k})}}{(\Gamma_1 \uplus \Delta_1 \vdash M_1\ M_2 : \tau_1) + (m:q) + (m' : q')\ \blacktriangledown} = \mathcal{D}_1''
\end{array}
$$

In the first stage, the update $(m : q)$ is propagated to either M_1 or M_2. We can prove that n_1 is determined by m in the same way as in the case that $M = x$. In the second stage, M_1 and M_2 interact by updating α_1. Here by the induction hypothesis, we know that n_k is determined by n_{k-1} independently from the derivations and the judgements. In the third stage, the last update ($(n_{2k+1} : q_{2k+1})$ in the above example) is propagated to M. One can prove that m' is determined by the last update. Therefore the rewriting sequence starting from $\mathcal{D}_2'$ must have the same decomposition.

The case $M = F$ can be proved similarly. □

Thanks to Lemma 10, we know that $\mathfrak{r}(M)$ is a well-defined partial function for every M. Now we give the connection between System$^{\triangleright}$ and AJM's game

model. Although it is natural to expect that $\mathfrak{r}(M) = \mathbf{Fun}(\langle\!|M|\!\rangle)$, unfortunately, this statement is a bit too strong and the domain of $\mathfrak{r}(M)$ may be a proper subset of that of $\mathbf{Fun}(\langle\!|M|\!\rangle)$, because the definition of $\mathfrak{r}(M)$ restricts the domain of $\mathfrak{r}(M)$ to be included by

$$\{m \in \mathcal{M}^O \mid \exists \Gamma, \tau, q.\ \Gamma \vdash M : \tau \text{ and } (\Gamma \vdash M : \tau) + (m : q) \text{ is defined}\}.$$

We write the above set of O-moves as $RwDom(M)$. We prove that the restriction of $\mathbf{Fun}(\langle\!|M|\!\rangle)$ to $RwDom(M)$ (written as $\mathbf{Fun}(\langle\!|M|\!\rangle) \upharpoonright RwDom(M)$) is equivalent to $\mathfrak{r}(M)$.

Lemma 11. *Let $\mathcal{G}$ be a loop-free recursion scheme and M be a term that possibly contains non-terminals of $\mathcal{G}$. Then $\mathfrak{r}(M) = \mathbf{Fun}(\langle\!|M|\!\rangle) \upharpoonright RwDom(M)$.*

Proof. By induction on the structure of M. □

The same proposition holds for recursion schemes that are not necessarily loop-free. We need an auxiliary lemma that relates the rewriting of derivations for M and those for $M^{[k]}$. For partial functions $f, g : Y \rightharpoonup Z$, we write $f \leq g$ if $f(x) = y$ implies $g(x) = y$ for every x (or equivalently, if $\mathrm{dom}(f) \subseteq \mathrm{dom}(g)$ and $f(x) = g(x)$ for every $x \in \mathrm{dom}(f)$). For an infinite increasing chain of partial functions $f_i : Y \rightharpoonup Z$ $(i \geq 0)$, its limit $\bigsqcup_{i\geq 0} f_i$ is defined by:

$$(\bigsqcup_{i\geq 0} f_i)(x) = \begin{cases} y & (\exists k \geq 0.\ f_k(x) \text{ is defined and } f_k(x) = y) \\ \text{undefined} & (\text{otherwise}) \end{cases}$$

Lemma 12. *Let $\mathcal{G}$ be a recursion scheme and M be a term that possibly contains non-terminals of $\mathcal{G}$. Then $\mathfrak{r}(M^{[i]})$ is an infinite increasing chain and $\mathfrak{r}(M) = \bigsqcup_{i\geq 0} \mathfrak{r}(M^{[i]})$.*

Proof. Let $\mathcal{D}$ be a (complete or incomplete) derivation of $\Gamma \vdash M^{[k]} : \tau$ and $i \geq 0$ be an integer. We write $\mathcal{D}^{[+i]}$ for the derivation obtained by replacement of all occurrences of $F^{[k']}$ in $\mathcal{D}$ with $F^{[k'+i]}$. Then we can prove that $\mathcal{D}^{[+i]}$ is a valid derivation of $\Gamma \vdash F^{[k]} : \tau$ and $\mathcal{D} \longrightarrow \mathcal{D}'$ implies $\mathcal{D}^{[+i]} \longrightarrow \mathcal{D}'^{[+i]}$, by induction on k and M (in other words, on the structure of $M^{[k]}$). Thus $\mathfrak{r}(M^{[k]}) \leq \mathfrak{r}(M^{[k+i]})$ for every $k, i \geq 0$.

Similarly for a given derivation $\mathcal{D}$ of $\Gamma \vdash M^{[k]} : \tau$, we define $\mathcal{D}^{[+\infty]}$ as the one obtained by replacing all occurrences of $F^{[k']}$ with F. Then $\mathcal{D}^{[+\infty]}$ is the derivation of $\Gamma \vdash M : \tau$ and $\mathcal{D} \longrightarrow \mathcal{D}'$ implies $\mathcal{D}^{[+\infty]} \longrightarrow \mathcal{D}'^{[+\infty]}$. So $\mathfrak{r}(M^{[k]}) \leq \mathfrak{r}(M)$ for every k, and thus $\bigsqcup_{i\geq 0} \mathfrak{r}(M^{[i]}) \leq \mathfrak{r}(M)$.

We show that $\mathfrak{r}(M) \leq \bigsqcup_{i\geq 0} \mathfrak{r}(M^{[i]})$. Let $\mathcal{D}$ be a derivation of $\Gamma \vdash M : \tau$. For every k greater than the height of $\mathcal{D}$, one can construct a derivation of $\Gamma \vdash M^{[k]} : \tau$ by replacing each occurrence of F in $\mathcal{D}$ with $F^{[k']}$ for some appropriate integer $k' \leq k$. This derivation is written as $\mathcal{D}_{[\downarrow k]}$. Then $\mathcal{D} \longrightarrow \mathcal{D}'$ implies $\mathcal{D}_{[\downarrow k]} \longrightarrow \mathcal{D}'_{[\downarrow k]}$ for every k greater than the height of $\mathcal{D}'$.

Suppose that $\mathfrak{r}(M)(m) = m'$ for some O-move m and P-move m'. It suffices to show that $\mathfrak{r}(M^{[k]})(m) = m'$ for some k. By definition of $\mathfrak{r}(M)$, we have

$$\mathcal{D} = \dfrac{\begin{array}{c}\vdots\\ \Gamma \vdash M : \tau\end{array}}{(\Gamma \vdash M : \tau) + (m : q)\ \blacktriangle} \longrightarrow^* \dfrac{\begin{array}{c}\vdots\\ \Gamma \vdash M : \tau\end{array}}{(\Gamma \vdash M : \tau) + (m : q) + (m' : q')\ \blacktriangledown} = \mathcal{D}'$$

for some Γ, τ, q and q'. Let k be an integer greater than the height of $\mathcal{D}'$. Then we have

$$\mathcal{D}_{[k]} = \dfrac{\begin{array}{c}\vdots\\ \Gamma \vdash M^{[k]} : \tau\end{array}}{(\Gamma \vdash M^{[k]} : \tau) + (m : q)\ \blacktriangle} \longrightarrow^* \dfrac{\begin{array}{c}\vdots\\ \Gamma \vdash M^{[k]} : \tau\end{array}}{(\Gamma \vdash M^{[k]} : \tau) + (m : q) + (m' : q')\ \blacktriangledown} = \mathcal{D}'_{[k]}.$$

and thus $\mathfrak{r}(M^{[k]})(m) = m'$ as required. □

Theorem 8. *Let $\mathcal{G}$ be a recursion scheme and M be a term that possibly contains non-terminals of $\mathcal{G}$. Then $\mathfrak{r}(M) = \mathbf{Fun}(\langle\!|M|\!\rangle_{\mathrm{rec}}) \upharpoonright \mathit{RwDom}(M)$.*

Proof. Suppose $\widetilde{x} :: \widetilde{A} \vdash M :: B$ and let $\mathcal{M}^O = \mathcal{M}^O_{\langle\!|\widetilde{A} \vdash B|\!\rangle}$ and $\mathcal{M}^P = \mathcal{M}^P_{\langle\!|\widetilde{A} \vdash B|\!\rangle}$. For integers $i, j \geq 0$, we define

$$f_{i,j} = \mathbf{Fun}(\langle\!|M^{[i]}|\!\rangle) \upharpoonright \mathit{RwDom}(M^{[j]})$$

and $Z = \{f_{i,j} \mid i, j \geq 0\}$. Recall that the set of all partial functions of $\mathcal{M}^O \rightharpoonup \mathcal{M}^P$ is a complete partial order. Thus for every directed set Y, we have the least upper bound of Y, written as $\bigsqcup Y$. Z is a directed set since $i \leq i'$ and $j \leq j'$ implies $f_{i,j} \leq f_{i',j'}$. We prove that $\mathfrak{r}(M) = f = \mathbf{Fun}(\langle\!|M|\!\rangle_{\mathrm{rec}}) \upharpoonright \mathit{RwDom}(M)$.

We prove $\mathfrak{r}(M) = f$. By Lemma 11, we have $\mathfrak{r}(M^{[i]}) = f_{i,i}$. So by Lemma 12,

$$\mathfrak{r}(M) = \bigsqcup\{\mathfrak{r}(M^{[i]}) \mid i \geq 0\} = \bigsqcup\{f_{i,i} \mid i \geq 0\}.$$

Let $Y = \{f_{i,i} \mid i \geq 0\}$. We know that $\bigsqcup Y \leq \bigsqcup Z$ since $Y \subseteq Z$, and that $\bigsqcup Z \leq \bigsqcup Y$ since for every $f_{i,j} \in Z$, we have $f_{i,j} \leq f_{i+j,i+j} \in Y$.

We prove $f = \mathbf{Fun}(\langle\!|M|\!\rangle_{\mathrm{rec}}) \upharpoonright \mathit{RwDom}(M)$. We have the following equation:

$$\begin{aligned} f &= \bigsqcup\{f_{i,j} \mid i, j \geq 0\} \\ &= \bigsqcup\{\ \bigsqcup\{f_{i,j} \mid j \geq 0\}\ \mid i \geq 0\} \\ &= \bigsqcup\{\mathbf{Fun}(\langle\!|M^{[i]}|\!\rangle) \upharpoonright \mathit{RwDom}(M) \mid i \geq 0\} \\ &= \mathbf{Fun}(\langle\!|M|\!\rangle_{\mathrm{rec}}) \upharpoonright \mathit{RwDom}(M) \end{aligned}$$

Here we use the facts (1) $\mathit{RwDom}(M) = \bigcup_{i \geq 0} \mathit{RwDom}(M^{[i]})$, (2) $\cdot \upharpoonright \cdot$ is continuous in both arguments, and (3) $\mathbf{Fun}(\cdot)$ is continuous. □

Now we prove the counterpart of Theorem 3. Let $\Sigma = \{a_i :: A_i\}$. For a given type environment Θ, we write $\Theta :: \Sigma$ if $a : \alpha \in \Theta$ implies $a :: A \in \Sigma$ and $\alpha :: !A$. For a type $\tau :: A_i$, we write $\delta \vDash a_i : \tau$ if $\tau = \bigcap\{(\square, q_1)\} \to \bigcap\{(\square, q_2)\} \to \cdots \to \bigcap\{(\square, q_{\mathit{arity}(a_i)})\} \to q$ and $(q, a_i, q_1 \ldots q_{\mathit{arity}(a_i)}) \in \delta$. We write $\delta \vDash \Theta$

if $\delta \vDash a : \tau$ for every $a : \tau \in \Theta$. We write $\delta \vDash \Diamond\Theta$ if there exists a sequence $\Theta \lhd_P \Theta_1 \lhd_P \ldots \lhd_P \Theta_k$ such that $\delta \vDash \Theta_k$. Let $\{\ \} = \Theta_0 \lhd_O \Theta_1 \lhd_P \Theta_2 \lhd_O \ldots$ be a sequence of type environments such that $\Theta_i :: \Sigma$ for every i. We say the sequence is *fair* if for every i, a and ζ, if $a : \bigsqcap_{\xi \in X}(\xi, \tau_\xi) \in \Theta_i$ and $\tau_\zeta \neq \bullet$, then there exists $i' > i$ such that $a : \bigsqcap_{\xi \in X'}(\xi, \tau'_\xi) \in \Theta_{i'}$ where $\tau'_\xi = \bigsqcap\{(\Box, q_1)\} \to \cdots \to \bigsqcap\{(\Box, q_{arity(a)})\} \to q$.

Theorem 9. *Let $\mathcal{G}$ be a recursion scheme and $\mathcal{A}$ be a trivial tree automaton. Then $[\![\mathcal{G}]\!]$ is accepted by $\mathcal{A}$ if and only if there exists a sequence of type environments*

$$\{\ \} = \Theta_0 \lhd_O \Theta_1 \lhd_P \Theta_2 \lhd_O \Theta_3 \lhd_P \ldots$$

such that

$$\Theta_{2i} \vdash S : q_0 \longrightarrow^* \Theta_{2i+1} \vdash S : q_0$$

for every i and (1) it is infinite and fair and satisfies $\delta \vDash \Diamond\Theta_i$ for every i, or (2) it is finite and ends with a complete derivation of $\Theta \vdash S : q_0$ such that $\delta \vDash \Theta$.

Proof. By the same construction as in the proof of Lemma 7. □

E Proof of Theorem 2

Before proving Theorem 2, we prove some lemmas.

Lemma 13. *If $\tau \pitchfork \sigma \lhd_P \theta'$, then $\theta' = \tau' \pitchfork \sigma'$. If $\tau \pitchfork \sigma \lhd_O \theta'$ and $\tau \pitchfork \sigma \neq \emptyset$, then $\theta' = \tau' \pitchfork \sigma'$.*

Proof. Easy induction on the derivation of $\tau \pitchfork \sigma \lhd_P \theta'$ and $\tau \pitchfork \sigma \lhd_O \theta'$. □

Lemma 14. *Replacement of a conclusion in a complete typing rule with its O-extension results in an O-rule. Similarly replacement of a premise in a complete typing rule with its P-extension results in a P-rule.*

Proof. By case analysis on the rule before the replacement. The most interesting case is the replacement of the conclusion of the rule for intersections with its O-extension. By the replacement, the left rule below becomes the right rule:

$$\frac{\Gamma \vdash M : \tau \quad \Delta \vdash M : \sigma}{\Gamma \pitchfork \Delta \vdash M : \tau \pitchfork \sigma} \qquad \frac{\Gamma \vdash M : \tau \qquad \Delta \vdash M : \sigma}{\Xi' \vdash M : \theta'}$$

where $\Gamma \pitchfork \Delta \vdash M : \tau \pitchfork \sigma \ \lhd_O \ \Xi' \vdash M : \theta'$. We need to prove that Ξ' and θ' are intersections, i.e. $\Xi' = \Gamma' \pitchfork \Delta'$ and $\theta' = \tau' \pitchfork \sigma'$ for some Γ', Δ', τ' and σ'. By definition of $\Gamma \pitchfork \Delta \vdash M : \tau \pitchfork \sigma \ \lhd_O \ \Xi' \vdash M : \theta'$, either (i) $\Gamma \pitchfork \Delta = \Xi'$ and $\tau \pitchfork \sigma \lhd_O \theta'$, or (ii) $\Gamma \pitchfork \Delta \lhd_P \Xi'$ and $\tau \pitchfork \sigma = \theta'$. In case (i), by Lemma 13 and the side condition $\tau \pitchfork \sigma \neq \emptyset$ of the rule for intersections, we have $\theta' = \tau' \pitchfork \sigma'$. Thus by setting $\Gamma' = \Gamma$ and $\Delta' = \Delta$, the right rule above is an instance of (O-Int). Case (ii) can be proved in the same way. □

Determinacy of the derivation rewriting system (i.e. $\mathcal{D} \longrightarrow \mathcal{D}_1$ and $\mathcal{D} \longrightarrow \mathcal{D}_2$ implies $\mathcal{D}_1 = \mathcal{D}_2$) is a consequence of the facts that (1) every incomplete derivation contains exactly one instance of an incomplete typing rule, and (2) for every instance of an incomplete typing rule, there exists exactly one rewriting rule for it.

We prove progress (i.e. for every incomplete derivation $\mathcal{D}$, there exists a derivation $\mathcal{D}'$ such that $\mathcal{D} \longrightarrow \mathcal{D}'$). As we have seen above, every incomplete derivation has exactly one applicable rewriting rule. What we need to prove is that the result of the rewriting is a valid (complete or incomplete) derivation.

We prove it by cases on the rule used to derive $\mathcal{D} \longrightarrow \mathcal{D}'$. Here we only show the case that the rule is (▲App1) and other cases are omitted.

Suppose that $\mathcal{D} \longrightarrow \mathcal{D}'$ is derived by (▲App1). Then $\mathcal{D}$ is of the form

$$\mathcal{D} = \dfrac{\dfrac{\dfrac{\vdots}{\Gamma \vdash M : \alpha \to \tau}\ (1) \qquad \dfrac{\vdots}{\Delta \vdash N : \alpha}\ (2)}{\Gamma' \uplus \Delta \vdash M\ N : \tau' \ \blacktriangle}\ (3)}{\vdots}\ (4)$$

and $\mathcal{D}'$ is of the form

$$\mathcal{D}' = \dfrac{\dfrac{\dfrac{\vdots}{\Gamma' \vdash M : \alpha \to \tau' \ \blacktriangle}\ (1') \qquad \dfrac{\vdots}{\Delta \vdash N : \alpha}\ (2')}{\Gamma' \uplus \Delta \vdash M\ N : \tau'}\ (3')}{\vdots}\ (4') .$$

Note that subderivations over rule (2) in $\mathcal{D}$ is equivalent to those over rule (2') in $\mathcal{D}'$, subderivations over rule (1) is equivalent to those over rule (1'), and a part of derivation under (4) is equivalent to that of under (4'). All of these parts uses only complete typing rules, since $\mathcal{D}$ is an incomplete derivation using an incomplete typing rule at (3). Thus it suffices to show that rule (3') is a valid complete rule and rule (1') is a valid incomplete rule. Clearly (3') is a valid complete rule and we prove (1') is a valid incomplete rule. Since (3) in $\mathcal{D}$ is an instance of (O-App), we have $\Gamma \uplus \Delta \vdash M\,N : \tau \lhd_O \Gamma' \uplus \Delta \vdash M\,N : \tau'$. By definition, either (1) $\Gamma \lhd_P \Gamma'$ and $\tau = \tau'$; or (2) $\Gamma = \Gamma'$ and $\tau \lhd_O \tau'$. In both cases, $\Gamma \vdash M : \alpha \to \tau \lhd_O \Gamma' \vdash M : \alpha \to \tau'$. So by Lemma 14, rule (1') is a valid incomplete rule.

F Proof of Theorem 3

We have already proved the counterpart of Theorem 3 in System$^{\triangleright}$ (Theorem 9). Here we reduce Theorem 3 to Theorem 9 by giving a partial function $\natural$ from derivations in System$^{\triangleright}$ to those in the rigid type system that preserves the rewriting relation in a certain sense.

A set I of sequences on $\{l, r\}$ is *compatible* if elements in I are pairwise incomparable, i.e. for every $s, t \in I$, neither $s \sqsubseteq t$ nor $t \sqsubseteq s$, where $\sqsubseteq$ is the prefix ordering. Let us consider types given by the following grammar:

$$\begin{array}{ll} \textit{Types} & \tau, \sigma ::= q \mid \alpha \to \tau \\ \textit{Rigid Intersection} & \alpha, \beta ::= \bigsqcap_{i \in I} \tau_i \end{array}$$

where I is a compatible set of sequences on $\{l, r\}$ and τ_i means the mapping $i \mapsto \tau_i$ that is defined on I. The types and intersections given by the above syntax is "isomorphic" to those given by the syntax in Section 3, in the sense that there exists a structure-preserving bijective correspondence $\sim$ defined by:

$$\begin{array}{lll} \displaystyle\bigsqcap_{i \in \{\,\}} \tau_i \sim \emptyset & & \\ \displaystyle\bigsqcap_{i \in \{\epsilon\}} \tau_i \sim \tau & \text{iff} & \tau_\epsilon = \tau \\ \displaystyle\bigsqcap_{i \in I} \tau_i \sim \alpha \pitchfork \beta & \text{iff} & \displaystyle\bigsqcap_{i \in I^l} \tau_i^l \sim \alpha \text{ and } \bigsqcap_{i \in I^r} \tau_i^r \sim \beta \\ & & \text{where } I^l = \{j \mid l \cdot j \in I\}, \\ & & \qquad I^r = \{j \mid r \cdot j \in I\}, \\ & & \qquad \tau_i^l = \tau_{l \cdot i}, \text{ and} \\ & & \qquad \tau_i^r = \tau_{r \cdot i}. \end{array}$$

For example,

$$\bigsqcap_{i \in \{lll, lr, rr\}} \tau_i \quad \sim \quad ((\tau_{lll} \pitchfork \emptyset) \pitchfork \tau_{lr}) \pitchfork (\emptyset \pitchfork \tau_{rr}).$$

We give a mapping from objects in System$^\triangleright$ to those in the rigid type system through the above bijective correspondence.

The mapping $\natural$ is based on the mapping $\flat$ from indexes $\mathcal{I}$ in System$^\triangleright$ to sequences of $\{l, r\}$ defined inductively by

$$\begin{aligned} \flat(\square) &= \epsilon \\ \flat(l \cdot \xi) &= l \cdot \flat(\xi) \\ \flat(r \cdot \xi) &= r \cdot \flat(\xi) \\ \flat(\langle \xi_1, \xi_2 \rangle) &= \flat(\xi_1) \cdot \flat(\xi_2). \end{aligned}$$

Note that $\flat$ is not injective. For example,

$$\flat(\langle l \cdot \square,\ r \cdot \square \rangle) = \flat(\langle l \cdot r \cdot \square,\ \square \rangle) = \flat(l \cdot r \cdot \square) = l \cdot r.$$

We say a set X of indexes is *compatible* if (1) the restriction of $\flat$ to X is injective (i.e. $\flat(\xi_1) = \flat(\xi_2)$ implies $\xi_1 = \xi_2$ for every $\xi_1, \xi_2 \in X$), and (2) $\flat(X) = \{\flat(\xi) \mid \xi \in X\}$ is a compatible set of sequences on $\{l, r\}$.

Then we define the partial mapping from types in System$^{\triangleright}$ to those in the rigid type system by

$$\begin{aligned}
\natural(q) &= q \\
\natural(\alpha \to \tau) &= \natural(\alpha) \to \natural(\tau) \\
\natural(\prod_{\xi \in X} \tau_\xi) &= \prod_{i \in \flat(X)} \sigma_i \qquad \text{if } X \text{ is compatible and } \forall \xi \in X.\ \natural(\tau_\xi) = \sigma_{\flat(\xi)}.
\end{aligned}$$

The mapping $\natural$ is defined on types in which all index sets are compatible sets. By generalising the mapping $\natural$, one can define $\natural(\Gamma)$ for type environments Γ in System$^{\triangleright}$, $\natural(J)$ for judgements J in System$^{\triangleright}$ and $\natural(\mathcal{D})$ for derivations in System$^{\triangleright}$. All of these mappings are defined on those in which all index sets are compatible sets.

Lemma 15. *Let $\mathcal{D}$ and $\mathcal{D}'$ be derivations in System$^{\triangleright}$ and suppose that $\mathcal{D} \longrightarrow \mathcal{D}'$ and $\natural(\mathcal{D})$ is well-defined. Then $\natural(\mathcal{D}')$ is also well-defined and $\natural(\mathcal{D}) \longrightarrow^* \natural(\mathcal{D}')$ in the rigid type system.*

Proof. By induction on the structure of $\mathcal{D}$ and the case analysis on the rule used to derive $\mathcal{D} \longrightarrow \mathcal{D}'$. In most cases, one step rewriting in System$^{\triangleright}$ corresponds to one step rewriting in the rigid type system, except that the rule used to derive $\mathcal{D} \longrightarrow \mathcal{D}'$ is related to $\prod_{\xi \in X} \tau_\xi$. For example, if one uses the rule

$$\dfrac{\dfrac{\overline{\Gamma'_\zeta \vdash M : \tau'_\zeta} \qquad \overline{\Gamma_\xi \vdash M : \tau_\xi} \quad (\forall \xi \in X \backslash \{\zeta\})}{\prod_{\xi \in X} \langle \xi, \Gamma_\xi \rangle \vdash M : \prod_{\xi \in X} \langle \xi, \tau_\xi \rangle \ \blacktriangle}}{\longrightarrow \quad \dfrac{\overline{\Gamma_\zeta \vdash M : \tau_\zeta} \ \blacktriangle \qquad \overline{\Gamma_\xi \vdash M : \tau_\xi} \quad (\forall \xi \in X \backslash \{\zeta\})}{\prod_{\xi \in X} \langle \xi, \Gamma_\xi \rangle \vdash M : \prod_{\xi \in X} \langle \xi, \tau_\xi \rangle \ \blacktriangle}}$$

to derive $\mathcal{D} \longrightarrow \mathcal{D}'$ in System$^{\triangleright}$, then the corresponding rewriting sequence is $\natural(\mathcal{D}) \longrightarrow^k \natural(\mathcal{D}')$ where k is the length of $\flat(\zeta)$. □

So by induction on the length of the rewriting sequences $\mathcal{D}_{\text{init}} \longrightarrow^* \mathcal{D}$ in System$^{\triangleright}$, we have $\natural(\mathcal{D}_{\text{init}}) \longrightarrow^* \natural(\mathcal{D})$. Thus Theorem 3 is a consequence of Theorem 9.

G Proof of Theorem 4

Before proving Theorem 4, we define an approximation $\mathcal{G}^{\langle k \rangle}$ of the recursion scheme $\mathcal{G}$. The approximation $\mathcal{G}^{\langle k \rangle}$ is similar to the loop-free approximation $\mathcal{G}^{[k]}$ (see Definition 7), except that $\mathcal{G}^{\langle k \rangle}$ uses $\bot$ where $\mathcal{G}^{[k]}$ uses Ω (recall that Ω is a special term meaning divergence and $\bot$ is a distinguished terminal symbol).

Definition 9 (Approximation with $\bot$). *Let $\mathcal{G} = (\Sigma, \mathcal{N}, \mathcal{R}, S)$ be a recursion scheme and suppose $\mathcal{N} = \{F_1, \ldots, F_n\}$. For every integer k, we define $\mathcal{N}^{\langle k \rangle}$ as*

the set of variables $\{F_i^{\langle k \rangle} \mid F_i \in \mathcal{N}\}$ *and* $M^{\langle k \rangle} = M[F_1^{\langle k \rangle}/F_1, \ldots, F_n^{\langle k \rangle}/F_n]$. *The recursion scheme* $\mathcal{G}^{[k]}$ *is defined by* $\mathcal{G}^{\langle k \rangle} = (\Sigma^\perp, \bigcup_{0 \leq j \leq k} \mathcal{N}^{\langle k \rangle}, \mathcal{R}', S^{\langle k \rangle})$, *where* $\mathcal{R}'$ *is given by:*

- $\mathcal{R}'(F_i^{\langle 0 \rangle}) = \lambda \widetilde{x}.\perp$.
- $\mathcal{R}'(F_i^{\langle j+1 \rangle}) = \lambda \widetilde{x}.M^{\langle j \rangle}$ *if* $\mathcal{R}(F) = \lambda \widetilde{x}.M$. □

The key observation to prove Theorem 4 is that an over-approximation of a certificate can be constructed from a derivation for $\Theta \vdash S^{\langle k \rangle} : q_0$ if k is large enough. Specifically k must be greater than the number of all type bindings for non-terminals, i.e. the number of elements in

$$\{F : \phi \mid F^A \in \mathcal{N} \text{ and } \phi :: A\}.$$

Let k be the number that satisfies the above condition and fix it in the following. Then model-checking of $\mathcal{G}$ is equivalent to model-checking of $\mathcal{G}^{\langle k \rangle}$.

Theorem 10. *Let* $\mathcal{G}$ *be a recursion scheme,* $\mathcal{A}$ *be a trivial automaton and* k *be an integer that satisfies the above condition. Then* $[\![\mathcal{G}]\!]$ *is accepted by* $\mathcal{A}^\perp$ *if and only if* $[\![\mathcal{G}^{\langle k \rangle}]\!]$ *is accepted by* $\mathcal{A}^\perp$.

Proof. Let

$$\Gamma_0 = \{F : \phi \mid F^A \in \mathcal{N} \text{ and } \phi :: A\}$$

and

$$\Gamma_{i+1} = \mathbf{Shrink}_{\mathcal{G},\mathcal{A}}(\Gamma_i).$$

Since $\mathbf{Shrink}_{\mathcal{G},\mathcal{A}}(\Gamma) \subseteq \Gamma$ for every Γ, we have a decreasing chain

$$\Gamma_0 \supseteq \Gamma_1 \supseteq \Gamma_2 \supseteq \cdots \supseteq \Gamma_j \supseteq \ldots.$$

Because k is the number of elements in Γ_0, Γ_k is the greatest fixed-point of $\mathbf{Shrink}_{\mathcal{G},\mathcal{A}}$. Thus by the argument in Section 2, the model-checking problem is reduce to the problem to check if $S : q_0 \in \Gamma_k$.

By induction on i, we can prove that $F : \phi \in \Gamma_i$ if and only if $\Theta \vdash (F^{\langle i \rangle}, \mathcal{G}^{\langle i \rangle}) : (\phi, \Gamma)$ for some Γ and Θ such that $\delta^\perp \vDash \Theta$, where $\delta^\perp$ is the transition relation of $\mathcal{A}^\perp$. Hence $S : q_0 \in \Gamma_k$ if and only if $\Theta \vdash (S^{\langle k \rangle}, \Gamma^{\langle k \rangle}) : (q_0, \Gamma)$ for some Γ and Θ such that $\delta^\perp \vDash \Theta$. The latter condition is equivalent to that $[\![\Gamma^{\langle k \rangle}]\!]$ is accepted by $\mathcal{A}^\perp$. □

Moreover an over-approximation of a certificate can be constructed from the derivation of $\Theta \vdash (S^{\langle k \rangle}, \mathcal{G}^{\langle k \rangle}) : (q_0, \Gamma)$, given by

$$\{F : \phi \mid \exists i.\ F^{\langle i \rangle} : \phi \in \Gamma\}.$$

In the following lemma, we do the same thing using a derivation in the rigid intersection type system, instead of one in the flexible intersection type system.

Lemma 16. *Let $\mathcal{G}$ be a recursion scheme, $\mathcal{A}$ be a trivial automaton and k be a number that satisfies the condition above. Let $\mathcal{D}$ be a derivation of $\Theta \vdash S^{\langle k \rangle} : q_0$ in the rigid intersection type system such that $\delta^{\perp} \vDash \Theta$. Let Γ be the type environment in the flexible type system defined by:*

$$\Gamma = \{F : \mathfrak{h}(\tau) \mid \exists \Delta.\ \Delta \vdash F^{\langle l \rangle} : \tau \text{ appears in } \mathcal{D} \text{ for some } l\ \}$$

where $\mathfrak{h}$ is a function from rigid intersection types to flexible intersection types that replace $\owedge$ with $\wedge$. Then Γ is an over-approximation of a certificate.

Proof. By the same idea as the proof of [13, Theorem 2(ii)]. □

Now to prove Theorem 4, it suffices to show that the derivation of $\Theta \vdash S^{\langle k \rangle} : q_0$ is reachable from $\mathcal{D}_{\text{init}}$ by the derivation rewriting system (here we ignore the difference of labels, i.e. F is identified with $F^{\langle l \rangle}$ for any l). A judgement of the form $\Delta \vdash F : \tau$ is called a *non-terminal judgement*. For an occurrence of a non-terminal judgement in a derivation, its *depth* is defined as the number of non-terminal judgements appearing in the path from the root to the occurrence. A derivation of $\Theta \vdash S^{\langle k \rangle} : q_0$ is obtained in a way similar to that in the proof of Theorem 3, except that we use ($\Diamond$-GiveUp) rule when we reach a non-terminal judgement of depth k.

H List of Abstraction Rules and Correctness

This section gives the list of all abstraction rules mentioned in Section 4 and proves its correctness. Fix a recursion scheme $\mathcal{G}$ and a trivial tree automaton $\mathcal{A}$. We first define the abstraction function, and then the set *Rules* of all instances of typing rules including incomplete typing rules by using the abstraction function. The abstract rewriting relation $\Rrightarrow$ is a relation between $\mathcal{P}(\mathit{Rules})$ and *Rules* (here $\mathcal{P}(X)$ is the powerset of X).

Remark 3. The abstract rewriting relation $\Rrightarrow$ is somehow different from the rule in Fig. 12, which describes a rule that should be satisfied by a desirable set $\hat{\mathbb{D}}$ of instances of typing rules. Such a desirable set is defined as a fixed-point of $\Rrightarrow$, i.e. a set $\hat{\mathbb{D}}$ that satisfies $\forall R.(\hat{\mathbb{D}} \Rrightarrow R \text{ implies } R \in \hat{\mathbb{D}})$. □

We define the abstraction function $\mathfrak{h}$. The function $\mathfrak{h}$ is a function from types and intersections in the rigid type system to those in the flexible type system, defined by the replacement of $\owedge$ with $\wedge$. It is straightforward to extend $\mathfrak{h}$ to the function from type environments in the rigid type system to those in the flexible type system by $\mathfrak{h}(\Gamma) = \{x : \mathfrak{h}(\tau) \mid x : \tau \in \Gamma\}$, and to the function from type judgements in the rigid type system to those in the flexible type system by $\mathfrak{h}(\Gamma \vdash M : \tau) = \mathfrak{h}(\Gamma) \vdash M : \mathfrak{h}(\tau)$. Then the function $\mathfrak{h}$ is extended to a function from instances of typing rules (including incomplete rules) in the rigid type system to those in the flexible type system by judgement-wise application of $\mathfrak{h}$.

$$\frac{}{\omega \ \prec_O \ \omega \to \cdots \to \omega \to q} \qquad \frac{\phi \prec_X \phi'}{\bigwedge\{\phi, \psi_1, \ldots, \psi_n\} \prec_X \bigwedge\{\phi', \psi_1, \ldots, \psi_n\}}$$

$$\frac{\psi \prec_X \psi'}{\overline{\phi} \to \psi \prec_X \overline{\phi} \to \psi'} \qquad \frac{\overline{\phi} \prec_{\overline{X}} \overline{\phi'}}{\overline{\phi} \to \psi \prec_X \overline{\phi'} \to \psi}$$

Fig. 17. Extension relations in the flexibly type system

Instances of incomplete typing rules in the flexible type system are defined by using $\mathfrak{h}$: the set of all instances of incomplete typing rules in the flexible type system is *defined* as

$$\{\mathfrak{h}(R) \mid R \text{ is an instance of an incomplete typing rule in the rigid type system}\}.$$

We write *Rules* for the set of all rule instances (including instances of incomplete typing rules) in the flexible type system.

The abstraction map $\mathfrak{h}$ is extended to a function from (possibly incomplete) derivations in the rigid type system to *a set of instances of typing rules* in the flexible type system, defined by

$$\mathfrak{h}(\mathcal{D}) = \{\mathfrak{h}(R) \mid R \text{ is used in } \mathcal{D}\}.$$

We define the *abstract rewriting relation* $\Rrightarrow$ as a relation between sets of rule instances and rule instances. We need some auxiliary definitions. The relations $\phi \prec_O \phi'$, $\phi \prec_P \phi'$, $\overline{\phi} \prec_O \overline{\phi'}$ and $\overline{\phi} \prec_P \overline{\phi'}$ are defined by the rule in Fig. 17 (here $X \in \{O, P\}$, $\overline{O} = P$ and $\overline{P} = O$).

Lemma 17. *Let $X \in \{O, P\}$.*

- *If $\tau \lhd_X \sigma$, then $\mathfrak{h}(\tau) \prec_X \mathfrak{h}(\sigma)$.*
- *If $\alpha \lhd_X \beta$, then $\mathfrak{h}(\alpha) \prec_X \mathfrak{h}(\beta)$.*

Proof. By induction on the structures of derivations of $\tau \lhd_X \sigma$ and $\alpha \lhd_X \beta$. □

A *rule with a hole at premise* is a rule of the form

$$\frac{J \quad [\,]}{K}, \qquad \frac{[\,] \quad J}{K}, \qquad \frac{[\,]}{J}, \quad \text{or} \quad \frac{[\,]}{\star}$$

where $[\,]$ is the hole and J and K are judgements (in the flexible intersection type system). We define two kinds of hole filling operations. For every rule with a hole at premise $\mathbf{P}$, $\mathbf{P}[L]$ is a rule marked with a single line and $\mathbf{P}[\![L \blacktriangledown]\!]$ is a rule marked with a double line. For example, if

$$\mathbf{P} = \frac{J \quad [\,]}{K},$$

then

$$\mathbf{P}[L] = \frac{J \quad L}{K} \quad \text{and} \quad \mathbf{P}[\![L \blacktriangledown]\!] = \frac{J \quad L \blacktriangledown}{\overline{\overline{K}}}.$$

Similarly a *rule with a hole at conclusion* $\mathbf{C}$ is a rule of the form

$$\frac{J}{[\,]}, \qquad \frac{J \quad K}{[\,]}, \qquad \frac{\Diamond}{[\,]}, \quad \text{or} \quad \frac{\dagger}{[\,]}.$$

$\mathbf{C}[L]$ and $\mathbf{C}[\![L \blacktriangle]\!]$ are defined similarly.

The abstract rewriting rules are shown in Fig. 18-22 and the following rule of monotonicity:

$$\frac{\hat{\mathbb{E}} \Rrightarrow R}{\hat{\mathbb{E}} \cup \hat{\mathbb{E}}' \Rrightarrow R}.$$

By abuse of notation, for sets $\hat{\mathbb{E}}_1$ and $\hat{\mathbb{E}}_2$, we write $\hat{\mathbb{E}}_1 \Rrightarrow \hat{\mathbb{E}}_2$ if

- $\hat{\mathbb{E}}_1 \subseteq \hat{\mathbb{E}}_2$, and
- $\hat{\mathbb{E}}_1 \Rrightarrow R$ implies $R \in \hat{\mathbb{E}}_2$ for every R.

Theorem 11 (Correctness of Abstraction). *Let $\mathcal{D}_1$ and $\mathcal{D}_2$ be derivations in the rigid intersection type and $\hat{\mathbb{E}}_1$ and $\hat{\mathbb{E}}_2$ be sets of instances of typing rules in the flexible intersection type system. Suppose that*

- $\mathcal{D}_1 \longrightarrow \mathcal{D}_2$,
- $\hat{\mathbb{E}}_1 \Rrightarrow \hat{\mathbb{E}}_2$, *and*
- $\mathfrak{h}(\mathcal{D}_1) \subseteq \hat{\mathbb{E}}_1$.

Then $\mathfrak{h}(\mathcal{D}_2) \subseteq \hat{\mathbb{E}}_2$.

Proof. By case analysis on the rule used to derive $\mathcal{D}_1 \longrightarrow \mathcal{D}_2$. We prove the case that ($\blacktriangle$App1) is used. Other cases can be proved in the same way.

Since $\mathcal{D}_1 \longrightarrow \mathcal{D}_2$ is derived from ($\blacktriangle$App1) rule, we know that

$$\mathcal{D}_1 = \dfrac{\dfrac{\vdots}{\Gamma \vdash M : \alpha \to \tau} \quad \dfrac{\vdots}{\Delta \vdash N : \alpha}}{\dfrac{\Gamma' \uplus \Delta \vdash M\,N : \tau' \ \blacktriangle}{\vdots}} \qquad \mathcal{D}_2 = \dfrac{\dfrac{\dfrac{\vdots}{\Gamma' \vdash M : \alpha \to \tau' \ \blacktriangle} \quad \dfrac{\vdots}{\Delta \vdash N : \alpha}}{\Gamma' \uplus \Delta \vdash M\,N : \tau'}}{\vdots}.$$

We prove the claim by the case analysis on the shape of M.

Case $M = M_1\,M_2$ for some M_1 and M_2: Now we know that

$$\mathcal{D}_1 = \dfrac{\dfrac{\dfrac{\dfrac{\vdots}{\Gamma_1 \vdash M_1 : \beta \to \alpha \to \tau}\ (1) \quad \dfrac{\vdots}{\Gamma_1 \vdash M_2 : \beta}\ (2)}{\Gamma_1 \uplus \Gamma_2 \vdash M_1\,M_2 : \alpha \to \tau}\ (3) \quad \dfrac{\vdots}{\Delta \vdash N : \alpha}\ (4)}{(\Gamma_1' \uplus \Gamma_2') \uplus \Delta \vdash (M_1\,M_2)\,N : \tau' \ \blacktriangle}\ (5)}{\vdots}\ (6)$$

Suppose $\phi \prec_O \phi'$.

(i) $$\left\{ \dfrac{\dagger}{\dfrac{}{x{:}\phi \vdash x{:}\phi'}\,\blacktriangle} \right\} \Rrightarrow \dfrac{\dagger}{x{:}\phi' \vdash x{:}\phi'}$$

(ii) $$\left\{ \dfrac{\dagger}{\dfrac{}{x{:}\phi \vdash x{:}\phi'}\,\blacktriangle} \;,\; \mathbf{P}[x{:}\phi \vdash x{:}\phi'] \right\} \Rrightarrow \mathbf{P}[\![x{:}\phi' \vdash x{:}\phi'\ \blacktriangledown]\!]$$

(1) Abstraction of rewriting rule (VAR1)

Suppose $\phi \prec_P \phi'$.

(i) $$\left\{ \dfrac{\Diamond}{\dfrac{}{x{:}\phi' \vdash x{:}\phi}\,\blacktriangle} \right\} \Rrightarrow \dfrac{\dagger}{x{:}\phi' \vdash x{:}\phi'}$$

(ii) $$\left\{ \dfrac{\Diamond}{\dfrac{}{x{:}\phi' \vdash x{:}\phi}\,\blacktriangle} \;,\; \mathbf{P}[x{:}\phi \vdash x{:}\phi'] \right\} \Rrightarrow \mathbf{P}[\![x{:}\phi' \vdash x{:}\phi'\ \blacktriangledown]\!]$$

(2) Abstraction of rewriting rule (VAR2)

(i) $$\left\{ \dfrac{\Gamma, \{x_i{:}\overline{\phi}_i \mid 1 \le i \le n\} \vdash M{:}q}{\dfrac{}{\Gamma' \vdash F{:}\overline{\phi'}_1 \to \ldots \overline{\phi'}_n \to q}\,\blacktriangle} \right\} \Rrightarrow \dfrac{\Gamma', \{x_i{:}\overline{\phi'}_i \mid 1 \le i \le n\} \vdash M{:}q}{\Gamma' \vdash F{:}\overline{\phi'}_1 \to \ldots \overline{\phi'}_n \to q}$$

(ii) $$\left\{ \begin{array}{c} \dfrac{\Gamma, \{x_i{:}\overline{\phi}_i \mid 1 \le i \le n\} \vdash M{:}q}{\dfrac{}{\Gamma' \vdash F{:}\overline{\phi'}_1 \to \ldots \overline{\phi'}_n \to q}\,\blacktriangle} \\ \\ \mathbf{C}[\Gamma, \{x_i{:}\overline{\phi}_i \mid 1 \le i \le n\} \vdash M{:}q] \end{array} \right\} \Rrightarrow \mathbf{C}[\![\Gamma', \{x_i{:}\overline{\phi'}_i \mid 1 \le i \le n\} \vdash M{:}q\ \blacktriangle]\!]$$

(3) Abstraction of rewriting rule (▲-FUN)

(i) $$\left\{ \dfrac{\Gamma', \{x_i{:}\overline{\phi'}_i \mid 1 \le i \le n\} \vdash M : q\ \blacktriangledown}{\Gamma \vdash F{:}\overline{\phi}_1 \to \ldots \to \overline{\phi}_n \to q} \right\} \Rrightarrow \dfrac{\Gamma', \{x_i{:}\overline{\phi'}_i \mid 1 \le i \le n\} \vdash M{:}q}{\Gamma' \vdash F{:}\overline{\phi'}_1 \to \ldots \overline{\phi'}_n \to q}$$

(ii) $$\left\{ \begin{array}{c} \dfrac{\Gamma', \{x_i{:}\overline{\phi'}_i \mid 1 \le i \le n\} \vdash M : q\ \blacktriangledown}{\Gamma \vdash F{:}\overline{\phi}_1 \to \ldots \to \overline{\phi}_n \to q} \\ \\ \mathbf{P}[\Gamma \vdash F{:}\overline{\phi}_1 \to \ldots \to \overline{\phi}_n \to q] \end{array} \right\} \Rrightarrow \mathbf{P}[\![\Gamma' \vdash F{:}\overline{\phi'}_1 \to \ldots \overline{\phi'}_n \to q\ \blacktriangledown]\!]$$

(4) Abstraction of rewriting rule (▼-FUN)

Fig. 18. Abstract rewriting rules 1

(i) $$\left\{ \frac{\Gamma \vdash M : \overline{\psi} \to \phi \quad \Delta \vdash N : \overline{\psi}}{\Gamma' \wedge \Delta \vdash M\ N : \phi' \ \blacktriangle} \right\} \Rrightarrow \frac{\Gamma' \vdash M : \overline{\psi} \to \phi' \quad \Delta \vdash N : \overline{\psi}}{\Gamma' \wedge \Delta \vdash M\ N : \phi'}$$

(ii) $$\left\{ \begin{array}{c} \dfrac{\Gamma \vdash M : \overline{\psi} \to \phi \quad \Delta \vdash N : \overline{\psi}}{\Gamma' \wedge \Delta \vdash M\ N : \phi' \ \blacktriangle} \\ \\ \mathbf{C}[\Gamma \vdash M : \overline{\psi} \to \phi] \end{array} \right\} \Rrightarrow \mathbf{C}[\![\Gamma' \vdash M : \overline{\psi} \to \phi' \ \blacktriangle]\!]$$

(5) Abstraction of rewriting rule (▲App1)

(i) $$\left\{ \frac{\Gamma \vdash M : \overline{\psi} \to \phi \quad \Delta \vdash N : \overline{\psi}}{\Gamma \wedge \Delta' \vdash M\ N : \phi \ \blacktriangle} \right\} \Rrightarrow \frac{\Gamma \vdash M : \overline{\psi} \to \phi \quad \Delta' \vdash N : \overline{\psi}}{\Gamma \wedge \Delta' \vdash M\ N : \phi}$$

(ii) $$\left\{ \begin{array}{c} \dfrac{\Gamma \vdash M : \overline{\psi} \to \phi \quad \Delta \vdash N : \overline{\psi}}{\Gamma \wedge \Delta' \vdash M\ N : \phi \ \blacktriangle} \\ \\ \mathbf{C}[\Delta \vdash N : \overline{\psi}] \end{array} \right\} \Rrightarrow \mathbf{C}[\![\Delta' \vdash N : \overline{\psi} \ \blacktriangle]\!]$$

(6) Abstraction of rewriting rule (▲App2)

(i) $$\left\{ \frac{\Gamma' \vdash M : \overline{\psi} \to \phi' \ \blacktriangledown \quad \Delta \vdash N : \overline{\psi}}{\Gamma \wedge \Delta \vdash M\ N : \phi} \right\} \Rrightarrow \frac{\Gamma' \vdash M : \overline{\psi} \to \phi' \quad \Delta \vdash N : \overline{\psi}}{\Gamma' \wedge \Delta \vdash M\ N : \phi'}$$

(ii) $$\left\{ \begin{array}{c} \dfrac{\Gamma' \vdash M : \overline{\psi} \to \phi' \ \blacktriangledown \quad \Delta \vdash N : \overline{\psi}}{\Gamma \wedge \Delta \vdash M\ N : \phi} \\ \\ \mathbf{P}[\Gamma \wedge \Delta \vdash M\ N : \phi] \end{array} \right\} \Rrightarrow \mathbf{P}[\![\Gamma' \wedge \Delta \vdash M\ N : \phi' \ \blacktriangledown]\!]$$

(7) Abstraction of rewriting rule (▼App1)

(i) $$\left\{ \frac{\Gamma \vdash M : \overline{\psi'} \to \phi \ \blacktriangledown \quad \Delta \vdash N : \overline{\psi}}{\Gamma \wedge \Delta \vdash M\ N : \phi} \right\} \Rrightarrow \frac{\Gamma \vdash M : \overline{\psi'} \to \phi \quad \Delta \vdash N : \overline{\psi'}}{\Gamma \wedge \Delta \vdash M\ N : \phi}$$

(ii) $$\left\{ \begin{array}{c} \dfrac{\Gamma \vdash M : \overline{\psi'} \to \phi \ \blacktriangledown \quad \Delta \vdash N : \overline{\psi}}{\Gamma \wedge \Delta \vdash M\ N : \phi} \\ \\ \mathbf{C}[\Delta \vdash N : \overline{\psi}] \end{array} \right\} \Rrightarrow \mathbf{C}[\![\Delta \vdash N : \overline{\psi'} \ \blacktriangle]\!]$$

(8) Abstraction of rewriting rule (▼App2)

Fig. 19. Abstract rewriting rules 2

(i) $$\left\{ \frac{\Gamma \vdash M : \overline{\psi} \to \phi \quad \Delta' \vdash N : \overline{\psi}\ \blacktriangledown}{\Gamma \wedge \Delta \vdash M\ N : \phi} \right\} \Rrightarrow \frac{\Gamma \vdash M : \overline{\psi} \to \phi \quad \Delta' \vdash N : \overline{\psi}}{\Gamma \wedge \Delta' \vdash M\ N : \phi}$$

(ii) $$\left\{ \begin{array}{c} \dfrac{\Gamma \vdash M : \overline{\psi} \to \phi \quad \Delta' \vdash N : \overline{\psi}\ \blacktriangledown}{\Gamma \wedge \Delta \vdash M\ N : \phi} \\ \\ \mathbf{P}[\Gamma \wedge \Delta \vdash M\ N : \phi] \end{array} \right\} \Rrightarrow \mathbf{P}[\![\Gamma \wedge \Delta' \vdash M\ N : \phi\ \blacktriangledown]\!]$$

(9) Abstraction of rewriting rule (▼App3)

(i) $$\left\{ \frac{\Gamma \vdash M : \overline{\psi} \to \phi \quad \Delta \vdash N : \overline{\psi'}\ \blacktriangledown}{\Gamma \wedge \Delta \vdash M\ N : \phi} \right\} \Rrightarrow \frac{\Gamma \vdash M : \overline{\psi'} \to \phi \quad \Delta \vdash N : \overline{\psi'}}{\Gamma \wedge \Delta \vdash M\ N : \phi}$$

(ii) $$\left\{ \begin{array}{c} \dfrac{\Gamma \vdash M : \overline{\psi} \to \phi \quad \Delta \vdash N : \overline{\psi'}\ \blacktriangledown}{\Gamma \wedge \Delta \vdash M\ N : \phi} \\ \\ \mathbf{C}[\Gamma \vdash M : \overline{\psi} \to \phi] \end{array} \right\} \Rrightarrow \mathbf{C}[\![\Gamma \vdash M : \overline{\psi'} \to \phi\ \blacktriangle]\!]$$

(10) Abstraction of rewriting rule (▼App3)

(i) $$\left\{ \frac{\Gamma \vdash M : \overline{\phi} \quad \Delta \vdash M : \overline{\psi}}{\Gamma' \wedge \Delta \vdash M : \overline{\phi'} \wedge \overline{\psi}\ \blacktriangle} \right\} \Rrightarrow \frac{\Gamma' \vdash M : \overline{\phi'} \quad \Delta \vdash M : \overline{\psi}}{\Gamma' \wedge \Delta \vdash M : \overline{\phi'} \wedge \overline{\psi}}$$

(ii) $$\left\{ \begin{array}{c} \dfrac{\Gamma \vdash M : \overline{\phi} \quad \Delta \vdash M : \overline{\psi}}{\Gamma' \wedge \Delta \vdash M : \overline{\phi'} \wedge \overline{\psi}\ \blacktriangle} \\ \\ \mathbf{C}[\Gamma \vdash M : \overline{\phi}] \end{array} \right\} \Rrightarrow \mathbf{C}[\![\Gamma' \vdash M : \overline{\phi'}\ \blacktriangle]\!]$$

(11) Abstraction of rewriting rule (▲Int1)

(i) $$\left\{ \frac{\Gamma \vdash M : \overline{\phi} \quad \Delta \vdash M : \overline{\psi}}{\Gamma \wedge \Delta' \vdash M : \overline{\phi} \wedge \overline{\psi'}\ \blacktriangle} \right\} \Rrightarrow \frac{\Gamma \vdash M : \overline{\phi} \quad \Delta' \vdash M : \overline{\psi'}}{\Gamma \wedge \Delta' \vdash M : \overline{\phi} \wedge \overline{\psi'}}$$

(ii) $$\left\{ \begin{array}{c} \dfrac{\Gamma \vdash M : \overline{\phi} \quad \Delta \vdash M : \overline{\psi}}{\Gamma \wedge \Delta' \vdash M : \overline{\phi} \wedge \overline{\psi'}\ \blacktriangle} \\ \\ \mathbf{C}[\Delta \vdash M : \overline{\psi}] \end{array} \right\} \Rrightarrow \mathbf{C}[\![\Delta' \vdash M : \overline{\psi'}\ \blacktriangle]\!]$$

(12) Abstraction of rewriting rule (▲Int2)

Fig. 20. Abstract rewriting rules 3

(i) $$\left\{ \frac{\Gamma' \vdash M : \overline{\phi'} \blacktriangledown \quad \Delta \vdash M : \overline{\psi}}{\Gamma \wedge \Delta \vdash M : \overline{\phi} \wedge \overline{\psi}} \right\} \Rrightarrow \frac{\Gamma' \vdash M : \overline{\phi'} \quad \Delta \vdash M : \overline{\psi}}{\Gamma' \wedge \Delta \vdash M : \overline{\phi'} \wedge \overline{\psi}}$$

(ii) $$\left\{ \begin{array}{c} \dfrac{\Gamma' \vdash M : \overline{\phi'} \blacktriangledown \quad \Delta \vdash M : \overline{\psi}}{\Gamma \wedge \Delta \vdash M : \overline{\phi} \wedge \overline{\psi}} \\ \\ \mathbf{P}[\Gamma \wedge \Delta \vdash M : \overline{\phi} \wedge \overline{\psi}] \end{array} \right\} \Rrightarrow \mathbf{P}[\![\Gamma' \wedge \Delta \vdash M : \overline{\phi'} \wedge \overline{\psi} \blacktriangledown]\!]$$

(13) Abstraction of rewriting rule (▼Int1)

(i) $$\left\{ \frac{\Gamma \vdash M : \overline{\phi} \quad \Delta' \vdash M : \overline{\psi'} \blacktriangledown}{\Gamma \wedge \Delta \vdash M : \overline{\phi} \wedge \overline{\psi}} \right\} \Rrightarrow \frac{\Gamma \vdash M : \overline{\phi} \quad \Delta' \vdash M : \overline{\psi'}}{\Gamma \wedge \Delta' \vdash M : \overline{\phi} \wedge \overline{\psi'}}$$

(ii) $$\left\{ \begin{array}{c} \dfrac{\Gamma \vdash M : \overline{\phi} \quad \Delta' \vdash M : \overline{\psi'} \blacktriangledown}{\Gamma \wedge \Delta \vdash M : \overline{\phi} \wedge \overline{\psi}} \\ \\ \mathbf{P}[\Gamma \wedge \Delta \vdash M : \overline{\phi} \wedge \overline{\psi}] \end{array} \right\} \Rrightarrow \mathbf{P}[\![\Gamma \wedge \Delta' \vdash M : \overline{\phi} \wedge \overline{\psi'} \blacktriangledown]\!]$$

(14) Abstraction of rewriting rule (▼Int2)

(i) $$\left\{ \frac{\Diamond}{\omega \vdash M : \phi \blacktriangle} \right\} \Rrightarrow \frac{\omega \vdash M : \phi \quad \omega \vdash M : \omega}{\omega \vdash M : \phi}$$

(ii) $$\left\{ \frac{\Diamond}{\omega \vdash M : \phi \blacktriangle} \right\} \Rrightarrow \frac{\omega \vdash M : \omega \quad \omega \vdash M : \phi}{\omega \vdash M : \phi}$$

(iii) $$\left\{ \frac{\Diamond}{\omega \vdash M : \phi \blacktriangle} \right\} \Rrightarrow \frac{\Diamond}{\omega \vdash M : \omega}$$

(15) Abstraction of rewriting rule (▲-◊1) and (▲-◊2)

(i) $$\left\{ \frac{\Diamond}{\omega \vdash x : \phi \blacktriangle} \right\} \Rrightarrow \frac{\dagger}{x : \phi \vdash x : \phi}$$

(ii) $$\left\{ \begin{array}{c} \dfrac{\Diamond}{\omega \vdash x : \phi \blacktriangle} \\ \\ \mathbf{P}[\omega \vdash x : \phi] \end{array} \right\} \Rrightarrow \mathbf{P}[\![x : \phi \vdash x : \phi \blacktriangledown]\!]$$

(16) Abstraction of rewriting rule (▲-◊3)

Fig. 21. Abstract rewriting rules 4

Suppose $\mathcal{R}(F) = \lambda x_1 \ldots x_n.M^o$.

(i) $$\left\{ \dfrac{\Diamond}{\omega \vdash F : \omega \to \cdots \to \omega \to q \ ^{\blacktriangle}} \right\} \Rrightarrow \dfrac{\omega \vdash M : q}{\omega \vdash F : \omega \to \cdots \to \omega \to q}$$

(ii) $$\left\{ \dfrac{\Diamond}{\omega \vdash F : \omega \to \cdots \to \omega \to q \ ^{\blacktriangle}} \right\} \Rrightarrow \dfrac{\Diamond}{\omega \vdash M : q \ ^{\blacktriangle}}$$

(17) Abstraction of rewriting rule (▲-◇4)

(i) $$\left\{ \dfrac{\Diamond}{\omega \vdash M\,N : \phi \ ^{\blacktriangle}} \right\} \Rrightarrow \dfrac{\omega \vdash M : \omega \to \phi \quad \omega \vdash N : \omega}{\omega \vdash M\,N : \phi}$$

(ii) $$\left\{ \dfrac{\Diamond}{\omega \vdash M\,N : \phi \ ^{\blacktriangle}} \right\} \Rrightarrow \dfrac{\Diamond}{\omega \vdash M : \omega \to \phi \ ^{\blacktriangle}}$$

(iii) $$\left\{ \dfrac{\Diamond}{\omega \vdash M\,N : \phi \ ^{\blacktriangle}} \right\} \Rrightarrow \dfrac{\Diamond}{\omega \vdash N : \omega}.$$

(18) Abstraction of rewriting rule (▲-◇5)

Suppose $\Theta_0 \prec_P \Theta_1$ and $\delta \vDash \Diamond\Theta_1$.
Here $\delta \vDash \Diamond\Theta_1$ if $\Theta_1 \prec_P \ldots \prec_P \Theta_n$ and $\delta \vDash \Theta_n$ for some $n \geq 1$ and Θ_i $(1 < i \leq n)$.

(i) $$\left\{ \dfrac{\Theta_0 \vdash S : q_0 \ ^{\blacktriangledown}}{\star} \right\} \Rrightarrow \dfrac{\Theta_1 \vdash S : q_0}{\star}$$

(ii) $$\left\{ \begin{array}{c} \dfrac{\Theta_0 \vdash S : q_0 \ ^{\blacktriangledown}}{\star} \\ \\ \mathbf{C}[\Theta_0 \vdash S : q_0] \in \hat{\mathbb{E}} \end{array} \right\} \Rrightarrow \mathbf{C}[\![\Theta_1 \vdash S : q_0 \ ^{\blacktriangle}]\!]$$

(19) Abstraction of rewriting rule (▲-⋆)

Fig. 22. Abstract rewriting rules 5

and

$$\mathcal{D}_2 = \dfrac{\dfrac{\dfrac{\dfrac{\vdots}{\Gamma_1 \vdash M_1 : \beta \to \alpha \to \tau}\,(1') \quad \dfrac{\vdots}{\Gamma_1 \vdash M_2 : \beta}\,(2')}{\Gamma_1' \oplus \Gamma_2' \vdash M_1\, M_2 : \alpha \to \tau}\,(3') \quad \dfrac{\vdots}{\Delta \vdash N : \alpha}\,(4')}{(\Gamma_1' \oplus \Gamma_2') \oplus \Delta \vdash (M_1\, M_2)\, N : \tau'}\,(5')}{\vdots}\,(6')$$

where the numbers will be used to indicate the rules. Assume that $R \in \mathfrak{h}(\mathcal{D}_2)$. It suffices to show that $R \in \hat{\mathbb{E}}_2$. By definition of $\mathfrak{h}(\mathcal{D}_2)$, there exists an instance R' of a rule in the rigid type system such that R' is used in $\mathcal{D}_2$ and $R = \mathfrak{h}(R')$. There are six subcaces.

- Case that R' is above $(1')$ or $R' = (1')$: Then R' is above (1) in $\mathcal{D}_1$ or $R' = (1)$. By the assumption, $R = \mathfrak{h}(R') \in \hat{\mathbb{E}}_1$. By definition of $\hat{\mathbb{E}}_1 \Rrightarrow \hat{\mathbb{E}}_2$, we have $\hat{\mathbb{E}}_1 \subseteq \hat{\mathbb{E}}_2$. Hence $R \in \hat{\mathbb{E}}_2$ as desired.
- Case that R' is above $(2')$ or $R' = (2')$: Similar to the above case.
- Case that R' is above $(4')$ or $R' = (4')$: Similar to the above case.
- Case that R' is below $(6')$ or $R' = (6')$: Similar to the above case.
- Case that $R' = (5')$: Since $\mathfrak{h}(\mathcal{D}_1) \subseteq \hat{\mathbb{E}}_1$, we have

  $$\dfrac{\mathfrak{h}(\Gamma_1 \oplus \Gamma_2) \vdash M_1\, M_2 : \mathfrak{h}(\alpha) \to \mathfrak{h}(\tau) \quad \mathfrak{h}(\Delta) \vdash N : \mathfrak{h}(\alpha)}{\mathfrak{h}(\Gamma_1' \oplus \Gamma_2') \wedge \mathfrak{h}(\Delta) \vdash (M_1\, M_2)\, N : \mathfrak{h}(\tau')\ \blacktriangle} \in \hat{\mathbb{E}}_1.$$

  So by rule (5) in Fig. 19, we have

  $$\hat{\mathbb{E}}_1 \Rrightarrow \dfrac{\mathfrak{h}(\Gamma_1' \oplus \Gamma_2') \vdash M_1\, M_2 : \mathfrak{h}(\alpha) \to \mathfrak{h}(\tau') \quad \mathfrak{h}(\Delta) \vdash N : \mathfrak{h}(\alpha)}{\mathfrak{h}(\Gamma_1' \oplus \Gamma_2') \wedge \mathfrak{h}(\Delta) \vdash (M_1\, M_2)\, N : \mathfrak{h}(\tau')}.$$

  Since the right-hand side of the above expression is equivalent to $\mathfrak{h}(R')$ and thus to R, the above expression can be rephrased as $\hat{\mathbb{E}}_1 \Rrightarrow R$. So by definition of $\hat{\mathbb{E}}_1 \Rrightarrow \hat{\mathbb{E}}_2$, we have $R \in \hat{\mathbb{E}}_2$ as desired.
- Case that $R' = (3')$: Since $\mathfrak{h}(\mathcal{D}_1) \subseteq \hat{\mathbb{E}}_1$, we have

  $$\dfrac{\mathfrak{h}(\Gamma_1 \oplus \Gamma_2) \vdash M_1\, M_2 : \mathfrak{h}(\alpha) \to \mathfrak{h}(\tau) \quad \mathfrak{h}(\Delta) \vdash N : \mathfrak{h}(\alpha)}{\mathfrak{h}(\Gamma_1' \oplus \Gamma_2') \wedge \mathfrak{h}(\Delta) \vdash (M_1\, M_2)\, N : \mathfrak{h}(\tau')\ \blacktriangle} \in \hat{\mathbb{E}}_1$$

  and

  $$\dfrac{\mathfrak{h}(\Gamma_1) \vdash M_1 : \mathfrak{h}(\beta) \to \mathfrak{h}(\alpha) \to \mathfrak{h}(\tau) \quad \mathfrak{h}(\Gamma_2) \vdash M_2 : \mathfrak{h}(\beta)}{\mathfrak{h}(\Gamma_1 \oplus \Gamma_2) \vdash M_1\, M_2 : \mathfrak{h}(\alpha) \to \mathfrak{h}(\tau)} \in \hat{\mathbb{E}}_1.$$

  So by rule (5) in Fig. 19, we have

  $$\hat{\mathbb{E}}_1 \Rrightarrow \dfrac{\mathfrak{h}(\Gamma_1) \vdash M_1 : \mathfrak{h}(\beta) \to \mathfrak{h}(\alpha) \to \mathfrak{h}(\tau) \quad \mathfrak{h}(\Gamma_2) \vdash M_2 : \mathfrak{h}(\beta)}{\mathfrak{h}(\Gamma_1' \oplus \Gamma_2') \vdash M_1\, M_2 : \mathfrak{h}(\alpha) \to \mathfrak{h}(\tau')\ \blacktriangle}.$$

  Since the right-hand side of the above expression is $\mathfrak{h}(R')$ and by definition of $\hat{\mathbb{E}}_1 \Rrightarrow \hat{\mathbb{E}}_2$, we have $R = \mathfrak{h}(R') \in \hat{\mathbb{E}}_2$ as required.

Other cases such as $M = x$ and $M = F$ are similar to the above case. □

Corollary 1. *Let $\hat{\mathbb{D}}$ be a fixed-point of the abstract rewriting rules (i.e. $\hat{\mathbb{D}} \Rrightarrow R$ implies $R \in \hat{\mathbb{D}}$ for every R) such that*

$$\left\{\frac{\vdash S : q_0}{\star},\ \frac{\Diamond}{\vdash S : q_0 \blacktriangle}\right\} \subseteq \hat{\mathbb{D}}.$$

If $\mathcal{D}$ is reachable, i.e. $\mathcal{D}_{\mathrm{init}} \longrightarrow^* \mathcal{D}$, then $\mathfrak{h}(\mathcal{D}) \subseteq \hat{\mathbb{D}}$.

Proof. Note that the assumption that $\hat{\mathbb{D}}$ is a fixed-point of $\Rrightarrow$ means that $\hat{\mathbb{D}} \Rrightarrow \hat{\mathbb{D}}$.

We prove the corollary by induction on the length of $\mathcal{D}_{\mathrm{init}}$. The claim holds if the length is 0, because $\mathfrak{h}(\mathcal{D}_{\mathrm{init}}) = \left\{\frac{\vdash S : q_0}{\star},\ \frac{\Diamond}{\vdash S : q_0 \blacktriangle}\right\}$. Suppose that $\mathcal{D}_{\mathrm{init}} \longrightarrow^* \mathcal{D}' \longrightarrow \mathcal{D}$ and $\mathfrak{h}(\mathcal{D}') \subseteq \hat{\mathbb{D}}$. By Theorem 11 and the assumption that $\hat{\mathbb{D}} \Rrightarrow \hat{\mathbb{D}}$, we have $\mathfrak{h}(\mathcal{D}) \subseteq \hat{\mathbb{D}}$ as desired. □

I Proof of Therome 5

We first prove termination. Note that for every sort A, the set of all flexible types of sort A is bounded by a constant determined by A. Hence the number of elements of $\hat{\mathbb{D}}$ is bounded by a constant determined by the set of all sorts appearing in $\mathcal{G}$. Termination of step 1 is proved by the fact that the size of $\hat{\mathbb{D}}$, which is bounded by the constant, increases in each iteration of step 1-2. As we have seen in Section 2, step 3 always terminates because $\mathbf{Shrink}_{\mathcal{G},\mathcal{A}}(\Gamma) \supseteq \Gamma$ for every Γ.[7] Termination of step 2 and 4 is trivial. So the algorithm in Fig. 13 terminates for all input $(\mathcal{G}, \mathcal{A})$.

Next we prove that if the algorithm returns `yes`, then $[\![\mathcal{G}]\!]$ is accepted by $\mathcal{A}$. As we have seen in Section 2, $\mathbf{Shrink}_{\mathcal{G},\mathcal{A}}(\Gamma) = \Gamma$ and $S : q_0 \in \Gamma$ if and only if $\Theta \vdash (S, \mathcal{G}) : (q_0, \Gamma)$ for some Θ that satisfies $\delta \vDash \Theta$. Thus if the algorithm returns `yes`, $[\![\mathcal{G}]\!]$ is accepted by $\mathcal{A}$ by Theorem 1.

Lastly we show the converse. Suppose that $\mathcal{G}$ is accepted by $\mathcal{A}$. Let Γ be the type environment defined in step 2 of the algorithm and $\Gamma_{\mathbf{gfp}}$ be the one in step 4. By Corollary 1, we know that Γ is an over-approximation of Γ_{Conc} in Theorem 4. Hence by Theorem 4, there exists a certificate Δ such that $\Delta \subseteq \Gamma_{\mathrm{Conc}} \subseteq \Gamma$. Since Δ is a certificate, Δ is a fixed-point of $\mathbf{Shrink}_{\mathcal{G},\mathcal{A}}$. Since $\Gamma_{\mathbf{gfp}}$ is the greatest fixed-point contained by Γ, we have $\Delta \subseteq \Gamma_{\mathbf{gfp}}$. Since Δ is a certificate, $S : q_0 \in \Delta$ and thus $S : q_0 \in \Gamma_{\mathbf{gfp}}$ as desired.

J Proof of Therome 6

In order to prove Theorem 6, we need to give an algorithm to compute the fixed-point of $\Rrightarrow$, named $\hat{\mathbb{D}}$ in step 1-2 in Fig. 13. We propose an algorithm to compute $\hat{\mathbb{D}}$ in time $O(|\mathcal{G}| \cdot \mathbf{exp}_n(poly(A \cdot |Q|)))$ for some polynomial $poly$.

[7] Here $\bigwedge\{\phi_1, \ldots, \phi_n\} \subseteq \bigwedge\{\psi_1, \ldots, \psi_m\}$ if $\{\phi_1, \ldots, \phi_n\} \subseteq \{\psi_1, \ldots, \psi_n\}$, and $\Gamma \subseteq \Delta$ if $\Gamma(x) \subseteq \Delta(x)$ for every x.

We need some auxiliary definitions. Let $\mathcal{G}$ be a recursion scheme and M and N be terms over $\mathcal{G}$ (or $\star$). We write M/N if there exists a rule of which M appears in a premise and N appears in the conclusion. Formally M/N is defined by:

- M_1/N and M_2/N if $N = M_1\, M_2$,
- M/F if $F \in \mathcal{N}$ and $\mathcal{R}(F) = \lambda\widetilde{x}.M$, and
- $S/\star$.

For a given term M, $\mathit{Neighbour}(M) = \{N \mid N/M\} \cup \{N \mid M/N\}$. Let R_1 and R_2 be instances of typing rules, M_1 be the subject of the conclusion of R_1 and M_2 be that of R_2 (let $M_1 = \star$ if the conclusion of R_1 is $\star$, and similar for M_2). We write R_1/R_2 if M_1/M_2.

Lemma 18 (Locality of Rewriting Rules). *Let $\hat{\mathbb{E}}$ be a set of instances of typing rules and R be a rule instance. Suppose that $\hat{\mathbb{E}} \Rrightarrow R$. Then one of the following conditions holds.*

- *There exists $R_0 \in \hat{\mathbb{E}}$ such that $\{R_0\} \Rrightarrow R$.*
- *There exist $R_1, R_2 \in \hat{\mathbb{E}}$ such that $\{R_1, R_2\} \Rrightarrow R$ and R_1/R_2.*

Proof. By induction on the structure of the derivation of $\hat{\mathbb{E}} \Rrightarrow R$. □

The algorithm to compute a fixed-point of the abstract rewriting rules is shown in Fig. 23.

Theorem 12. *If the algorithm in Fig. 23 returns $\hat{\mathbb{D}}$, then $\hat{\mathbb{D}}$ is a fixed-point of the abstract rewriting rules, i.e. $\hat{\mathbb{D}} \Rrightarrow R'$ implies $R' \in \hat{\mathbb{D}}$.*

Proof. The following proposition is a loop invariant of step 2:

If $\hat{\mathbb{E}} \Rrightarrow R'$, then $R' \in \hat{\mathbb{E}} \cup \hat{\mathbb{E}}'$, where $\hat{\mathbb{E}} = \bigcup_M \hat{\mathbb{E}}(M)$ and $\hat{\mathbb{E}}' = \bigcup_M \hat{\mathbb{E}}'(M)$.

If the above proposition is a loop invariant, then $\hat{\mathbb{E}}' = \{\ \}$ implies $\hat{\mathbb{E}}$ is a fixed-point of the abstract rewriting rules. Thus $\hat{\mathbb{D}}$ is a fixed-point.

We prove that this proposition is actually a loop invariant. Assume $\hat{\mathbb{E}} \Rrightarrow R'$ implies $R' \in \hat{\mathbb{E}} \cup \hat{\mathbb{E}}'$. Let M be a term and $R \in \hat{\mathbb{E}}'(M)$. Suppose $(\hat{\mathbb{E}} \cup \{R\}) \Rrightarrow R'$ and $R' \notin \hat{\mathbb{E}} \cup \hat{\mathbb{E}}'$ at step 2-1. It suffices to show that $R' \in \hat{\mathbb{E}} \cup \hat{\mathbb{E}}'$ at step 2-6.

By Lemma 18, we have

1. there exists $R_0 \in \hat{\mathbb{E}} \cup \{R\}$ such that $\{R_0\} \Rrightarrow R'$, or
2. there exist $R_1, R_2 \in \hat{\mathbb{E}} \cup \{R\}$ such that $\{R_1, R_2\} \Rrightarrow R'$ and R_1/R_2.

Let us consider the latter case. We have $R_1 = R$ or $R_2 = R$ because $\{R_1, R_2\} \subseteq \hat{\mathbb{E}}$ contradicts to the assumption ($\{R_1, R_2\} \subseteq \hat{\mathbb{E}}$ implies $\hat{\mathbb{E}} \Rrightarrow R'$ and thus $R' \in \hat{\mathbb{E}} \cup \hat{\mathbb{E}}'$ at step 2-1).

Assume $R_1 = R$. Let N be the subject of the conclusion of R_2. We have M/N by definition. Hence $N \in \mathit{Neighbour}(M)$ and $R_2 \in \hat{\mathbb{E}}(N)$. So $R' \in \hat{\mathbb{X}}(N)$. So R' is added to $\hat{\mathbb{E}}'$ at step 2-3-2.

The case that $R_2 = R$ is similar.

Let us consider the farmer case: $\{R_0\} \subseteq \hat{\mathbb{E}} \cup \{R\}$ such that $\{R_0\} \Rrightarrow R'$. Similar to the above, we have $R_0 = R'$. Thus $R' \in \hat{\mathbb{Y}}$ in step 2-4. Therefore R' is added to $\hat{\mathbb{E}}'$ at step 2-5 as desired. □

ConstructD($\mathcal{G}$,$\mathcal{A}$):
1. Initialise $\hat{\mathbb{E}}$ and $\hat{\mathbb{E}}'$;
  1-1. Let $\hat{\mathbb{E}}(M) = \{\ \}$ for every term M appearing in $\mathcal{G}$;
  1-2. Let $\hat{\mathbb{E}}'(M) = \{\ \}$ for every term M appearing in $\mathcal{G}$;
  1-3. $\hat{\mathbb{E}}(\star)$:= $\left\{\dfrac{\vdash S : q_0}{\star}\right\}$;
  1-4. $\hat{\mathbb{E}}'(S)$:= $\left\{\dfrac{\Diamond}{\vdash S : q_0\ \blacktriangle}\right\}$;
2. Fixed-point computation ;
  2-1. Choose M and R such that $R \in \hat{\mathbb{E}}'(M)$;
      // Move R from $\hat{\mathbb{E}}'(M)$ to $\hat{\mathbb{E}}(M)$
  2-2. $\hat{\mathbb{E}}(M) := \hat{\mathbb{E}}(M) \cup \{R\}$ and $\hat{\mathbb{E}}'(M) := \hat{\mathbb{E}}'(M) \backslash \{R\}$;
  2-3. For each $N \in \mathit{Neighbour}(M)$ do
        // Apply abstract rules to $\hat{\mathbb{E}}(N) \cup \{R\}$
    2-3-1. Let $\hat{\mathbb{X}}(N) = \{R' \mid R' \notin \hat{\mathbb{E}}$ and $(\hat{\mathbb{E}}(N) \cup \{R\}) \Rrightarrow R'\}$;
    2-3-2. For each $R' \in \hat{\mathbb{X}}(N)$ do
          $\hat{\mathbb{E}}'(N') := \hat{\mathbb{E}}'(N') \cup \{R'\}$ where N' is the subject of R' ;
      // Apply abstract rules to $\{R\}$
  2-4. Let $\hat{\mathbb{Y}} = \{R' \mid R' \notin \hat{\mathbb{E}}$ and $\{R\} \Rrightarrow R'\}$;
  2-5. For each $R' \in \hat{\mathbb{Y}}$ do
          $\hat{\mathbb{E}}'(N') := \hat{\mathbb{E}}'(N') \cup \{R'\}$ where N' is the subject of R' ;
  2-6. If $\hat{\mathbb{E}}'(M) \neq \{\ \}$ for some M then goto 2-1 ;
3. Let $\hat{\mathbb{D}} = \bigcup_M \hat{\mathbb{E}}(M)$;
4. Return $\hat{\mathbb{D}}$;

Fig. 23. An algorithm to construct $\hat{\mathbb{D}}$

Now we estimate the time complexity of the algorithm. Let $\mathcal{A}$ be a trivial automaton and $\mathcal{G}$ be a recursion scheme whose order is n and arity is ar. For a sort A, we define $\mathbf{Types}(A) = \{\phi \mid \phi :: A\}$ and $|\mathbf{Types}(A)|$ be the number of elements in $\mathbf{Types}(A)$. Let $\Sigma = \{a_1 :: A_1, \ldots, a_n :: A_n\}$ and $|\mathbf{Types}| = \max\{|\mathbf{Types}(A_1 \to A_2 \to \cdots \to A_n \to B)| \mid F^B \in \mathcal{N}\}$. Then $|\mathbf{Types}|$ is bounded by $O(\mathbf{exp}_n(\mathit{poly}(ar \times |Q|)))$ for some polynomial poly. For each term M, the number of instances of typing rules in which conclusions have M as the subject is bounded by $3 \times |\mathbf{Types}|$, since an instance of a typing rule has at most three judgements.

The following lemma shows that the algorithm in Fig. 23 runs in time linear in the size of recursion schemes and in polynomial in the number of types.

Lemma 19. *The algorithm in Fig. 23 runs in time $O(|\mathcal{G}| \cdot \mathit{poly}(|\mathbf{Types}|))$ for some polynomial poly.*

Proof. We estimate how may times we reach step 2-3-1. An important observation is that for every term N, the set $\{M \mid N \in \mathit{Neighbour}(M)\}$ has at most three elements. Thus for each N, we need to compute step 2-3-1 for N at most $3 \times (3 \times |\mathbf{Types}|)$ times.

Since step 2-3-1 itself can run in time polynomial in the size of $\hat{\mathbb{E}}(N)$ that is bounded by $|\mathbf{Types}|$, the cost of step 2-3-1 for each N is bounded by $O(poly(|\mathbf{Types}|))$ for some polynomial $poly$.

Since the number of terms is bounded by $O(|\mathcal{G}|)$, the algorithm runs in time $O(|\mathcal{G}| \cdot poly(|\mathbf{Types}|))$. □

Now we prove Theorem 6 for the algorithm in Fig. 13 in which $\hat{\mathbb{D}}$ is computed by the algorithm in Fig. 23. By Lemma 19, we can construct $\hat{\mathbb{D}}$ in time $O(|\mathcal{G}| \cdot poly(|\mathbf{Types}|))$, and thus in time $O(|\mathcal{G}| \cdot \mathbf{exp}_n(poly'(ar \cdot |Q|)))$ for some polynomials $poly$ and $poly'$. The size of $\hat{\mathbb{D}}$ is also bounded by $O(|\mathcal{G}| \cdot \mathbf{exp}_n(poly'(ar \cdot |Q|)))$. Then the cost to compute Γ and the size of Γ in step 2 are bounded by the same function. The iteration in step 3 can be done in time linear in the size of Γ by using Rehof and Mogensen's algorithm [17] (see [8, 9]). Thus we obtain the claim.