Contextual Equivalence for State and Control via Nested Data

Benedict Bunting
University of Oxford
Oxford, UK

Andrzej S. Murawski
University of Oxford
Oxford, UK

ABSTRACT

We consider contextual equivalence in an ML-like language, where contexts have access to both general references and continuations. We show that in a finitary setting, i.e. when the base types are finite and there is no recursion, the problem is decidable for all programs with first-order references and continuations, assuming they have continuation- and reference-free interfaces. This is the best one can hope for in this case, because the addition of references to functions, to continuations or to references makes the problem undecidable.

The result is notable since, unlike earlier work in the area, we need not impose any restrictions on type-theoretic order or the use of first-order references inside terms. In particular, the programs concerned can generate unbounded heaps.

Our decidability argument relies on recasting the corresponding fully abstract trace semantics of terms as instances of automata with a decidable equivalence problem. The automata used for this purpose belong to the family of automata over infinite alphabets (aka data automata), where the infinite alphabet (dataset) has the shape of a forest.

ACM Reference Format:
Benedict Bunting and Andrzej S. Murawski. 2024. Contextual Equivalence for State and Control via Nested Data. In *39th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS '24), July 8–11, 2024, Tallinn, Estonia.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3661814.3662109

1 INTRODUCTION

Contextual equivalence is a fundamental program property, which has motivated programming language research for decades due to its expressivity, applicability and the challenge it poses to both manual and automated reasoning. This paper studies equivalence problems in an ML-like setting that combines functional programming (lambda calculus), state (general references) and control-flow manipulation (continuations). The prototypical language for studying this particular set of features is known as HOSC [14], and this is the language from which contexts will be drawn for the purpose of equivalence testing.

For the sake of avoiding an obvious source of undecidability, we shall assume that base types are finite, i.e. $\mathsf{Int} = \{0, \cdots, \max\}$, and there is no recursion. However, even with bounded base types, HOSC remains very expressive and, in particular, due to the presence of general references, has an undecidable termination problem. Consequently, in order to obtain decidability results for contextual equivalence, we will need to make restrictions to the terms that are being tested (the contexts remain unrestricted, though). Specifically, we are going to disallow references to functions, references to continuations and references to references, because in each case we can show that this would lead to undecidable equivalence. The resultant subset of HOSC, allowing for storage of values of ground types such as Unit, Bool and Int, will be called FOSC. Our main result then shows that contextual equivalence of FOSC terms (with respect to HOSC contexts) is decidable, assuming their boundary types do not contain reference or continuation types, although such types can occur inside the terms. We refer to the scenario as HOSC[FOSC].

The result should be contrasted with earlier undecidability results for the continuation-free subset of FOSC, called FOS. It is known that contextual equivalence between FOS terms (with respect to FOS contexts), i.e. FOS[FOS], is undecidable. Thus, our results show that changing the contexts to HOSC, i.e. considering HOSC[FOS], leads to decidability and that the result also extends to terms with continuations, i.e. HOSC[FOSC].

Example 1.1. Here is an inequivalence that could be detected using our methods. Suppose $\Gamma = \{f : ((\mathsf{Unit} \to \mathsf{Unit}) \to \mathsf{Unit}) \to \mathsf{Unit}\}$, $\sigma = \mathsf{Unit} \to \mathsf{Unit}$ and Ω represents divergence. Consider the terms $\Gamma \vdash M_1, M_2 : \mathsf{Unit}$ given below.

$$\begin{aligned} M_1 = \quad & \mathbf{let}\, b = \mathbf{ref\ ff\ in\ call/cc}(y.\\ & f(\lambda g^{\sigma}.b := \mathbf{tt}; g(); \mathbf{throw}\ ()\ \mathbf{to}\ y);\\ & \mathbf{if}\ !b\ \mathbf{then}\ ()\ \mathbf{else}\ \Omega)\\ M_2 = \quad & \mathbf{call/cc}(y. f(\lambda g^{\sigma}.g(); \mathbf{throw}\ ()\ \mathbf{to}\ y); \Omega) \end{aligned}$$

Intuitively, the two terms are not equivalent (with respect to HOSC contexts), because one can find a context that triggers the evaluation of the subterm '$\mathbf{if}\ !b\ \mathbf{then}\ ()\ \mathbf{else}\ \Omega$' of M_1 when b contains $\mathbf{tt}$ (yielding ()), whereas Ω will be reached in M_2. The simplest such context can instantiate f to a function that will apply its argument (i.e. $\lambda g.b := \mathbf{tt};\ g();\ \mathbf{throw}\ ()\ \mathbf{to}\ y$) to a g such that $g()$ triggers a return by f (rather than g). Note that this kind of behaviour really requires the context to use continuations. Indeed, the HOSC context $(\lambda f.\bullet)(\lambda h.\mathbf{call/cc}(z.\ h(\lambda x.\mathbf{throw}\ ()\ \mathbf{to}\ z))$ does the job.

We study HOSC[FOSC] using trace models, also known as *operational game semantics* [14]. The approach consists in modelling interactions between a term and a context as traces derived from a labelled transition system $\mathcal{L}_{\mathrm{HOSC}}$. The traces are alternating sequences of actions made by two players, called O (context) and P (term) respectively. The actions represent abstract calls and returns, which involve function and continuation names drawn from infinite sets of typed names. Overall, the traces make it possible to express the patterns of calls and returns arising in contextual interactions, so that contextual equivalence can be captured as trace equivalence.

Example 1.2. Here is a trace that distinguishes the two terms given above, in that it can be generated by the first term but not the second. We write P-actions in red and O-actions in blue; f, g, h are function names and c_1, c_2, c_3, c are continuation names. The names f, c (corresponding to the free identifier f and top-level continuation) are called initial and viewed as introduced by O. Other names introduced by O (i.e. in O-actions) are g, c_2. We refer to them as O-names. The remaining names (introduced in P-actions, such as h, c_1, c_3) are called P-names.

$$\begin{array}{ccccc} \bar{f}(h, c_1) & h(g, c_2) & \bar{g}((), c_3) & c_1(()) & \bar{c}(()) \\ \text{P} & \text{O} & \text{P} & \text{O} & \text{P} \end{array}$$

Here the first action (by P) represents a call to f, where h represents the argument '$\lambda g.b := \mathbf{tt}; g(); \mathbf{throw}\ ()\ \mathbf{to}\ y$', and c_1 is the corresponding continuation. The second action (by O) corresponds to applying h to an indeterminate function g, which is then called on $()$ (by P) in the third action. The following action $c_1(())$ then represents a return matching the initial call to f, because c_1 was introduced in $\bar{f}(h, c_1)$ at the time that f was called. Finally $\bar{c}(())$ means that $()$ is returned to the top-level continuation c.

In order to prove our decidability result, we shall relate $\mathcal{L}_{\text{HOSC}}$ to a class of automata with a decidable language equivalence problem. As it stands in [14], $\mathcal{L}_{\text{HOSC}}$ involves rather complicated configurations, far removed from classic automata theory. Among others, they may involve a term, an environment and a heap, each of which may contain occurrences of function, continuation or location names, drawn from infinite sets. In order to close the gap between $\mathcal{L}_{\text{HOSC}}$ and automata theory, we will first make a number of observations that will enable us to simplify $\mathcal{L}_{\text{HOSC}}$ for FOSC terms. The remaining complexity will be handled using an automata model over nested data, namely, *nested data class memory automata* (NDCMA) [6]. In the subsection below, we give a high-level overview of the steps that will ultimately take us to NDCMA.

Final result. Our main technical result states that the trace semantics of FOSC terms, as specified by $\mathcal{L}_{\text{HOSC}}$, can be represented faithfully using (weak deterministic) NDCMA. Since the associated language equivalence problem is decidable [6], it follows that so is contextual equivalence of FOSC terms with respect to HOSC contexts (HOSC[FOSC]). Note that the result does not restrict the type-theoretic order of terms. This is in stark contrast to classifications of decidable cases for FOS[FOS] [7], which rely on subtle differences in type shapes and where equivalence can be undecidable even for second-order types [21].

Related work. The concept of nested data is used in the automata literature to model nested parameterised systems, where a process can have subprocesses, which have subprocesses and so on, and the corresponding actions feature process identifiers [2, 8]. The papers considered respectively emptiness problems for automata and shuffle expressions, as well as satisfiability problems for temporal logics that can navigate the data. The first application of nested data to the study of higher-order computation was presented in [5], as part of a programme to classify decidable cases for contextual equivalence of finitary FOS terms in FOS contexts. The authors showed that NDCMA were suitable for modelling the game semantics of certain first-order FOS terms, which implied decidability of contextual equivalence in this case. For FOS contexts, the result could not be extended further due to the undecidability of the general case [20]. In view of this, our results demonstrate that NDCMA have much wider applicability and are capable of expressing unrestricted higher-order scenarios with state and control, if equivalence with respect to HOSC contexts is considered instead.

Contextual equivalence with respect to HOSC contexts has been researched through numerous approaches, e.g. eager normal-form bisimulations [25], Kripke Logical Relations [9], operational game semantics [14], and Kripke normal-form bisimulations [16], albeit with hardly any decidability results. A notable exception is a recent result for the $\lambda\mu\nu$-calculus (i.e. boolean functional programs with name creation, control and no references) with respect to $\lambda\mu_{\text{ref}}$ contexts (i.e. essentially HOSC) [11], where equivalence was characterised through equality of the induced Böhm-like trees called Lassen trees. As the trees are finite and computable, decidability follows. However, that approach crucially relies on the absence of state. Another related result, but for contexts without continuations, was obtained in [13] for terms that can create first-order references only at the top level, leading to bounded heaps during interaction. In contrast, our result applies to a setting where first-order references can be created anywhere inside the term, leading to interactions with potentially unbounded heaps. Conceptually, it follows the approach of *operational algorithmic game semantics* [4], which advocates using trace models (labelled transition systems) to uncover connections with automata.

Technical Overview

In this subsection, we outline the main technical innovations and observations necessary to derive a NDCMA presentation of $\mathcal{L}_{\text{HOSC}}$.

Bounded P-views and O-names. A *P-view* is a technical concept from game semantics, specifying a fragment of the trace that intuitively captures all resources currently in scope. For example, the P-view of the trace given above is $\bar{f}(h, c_1)\ c_1(())\ \bar{c}(())$, i.e. the $h(g, c_2)\ \bar{g}((), c_3)$ segment is ignored. It is known that traces generated by FOSC terms satisfy a condition called *P-visibility* [14]: when P calls a function introduced by O (resp. returns to a continuation introduced by O), the corresponding O-name must lie in the P-view at that point. This property would be violated if references to functions or to continuations were allowed.

Our first and most important observation is that *P-views* generated by FOSC terms are bounded, and the same applies to the length of reduction paths that generate them. This is a direct consequence of the absence of recursion. The boundedness of P-views implies that the number of O-names in any P-view will be bounded. As their use is restricted to the P-view (by P-visibility), we can obtain a faithful representation of traces by replacing O-names with (bounded) numerical indices that correspond to the order in which they are introduced in P-views. Thus, in the Example above, the O-names f, c, g, c_2 would become $1, 2, 3, 4$ respectively. Further, according to this idea, if the trace above were extended with $h(g', c_2')$, the O-names g', c_2' would also be given indices $3, 4$ (rather than $5, 6$). The replacement scheme proposed above allows us to circumvent the use of infinitely many O-names.

Bounded terms. Next we observe that the terms occurring inside configurations of $\mathcal{L}_{\text{HOSC}}$ may contain O-names only. This, in combination with the fact that the reduction paths corresponding to P-views are bounded, implies that the number of terms that occur in $\mathcal{L}_{\text{HOSC}}$ (for a given FOSC term) will also be bounded, once we have replaced O-names with the corresponding indices. This brings us closer to a finitary representation of $\mathcal{L}_{\text{HOSC}}$.

Unbounded P-names. The only infinitary ingredients that remain unbounded are the environments and the heaps. An environment in $\mathcal{L}_{\text{HOSC}}$ consists of bindings that link P-names to terms. We have already established that the number of terms occurring in $\mathcal{L}_{\text{HOSC}}$ (starting from a given FOSC term) can be bounded. However, we cannot afford to bound the number of P-names. This is because trace length is in general unbounded, so unboundedly many P-names may be introduced and O must be allowed to refer back to any of them. To address this aspect of $\mathcal{L}_{\text{HOSC}}$, we are going to rely on an infinite alphabet $\mathcal{D}$ to model P-names and an automata model over $\mathcal{D}$, namely an extension of *class memory automata* (CMA) [3], which can keep track of *memories* associated with each element of $\mathcal{D}$. This feature will be used to simulate environments. For the sake of uniformity, we will also use (a finite number of) fixed elements of $\mathcal{D}$ to act as indices, i.e. O-names.

CMA belong to the family of automata over infinite alphabets. As such, they read *data words*, i.e. sequences from $(\Sigma\times\mathcal{D})^*$, where Σ is a finite alphabet of *tags* and $\mathcal{D}$ is an infinite alphabet of *data values*, also called a *dataset*. As actions in our traces may contain several names, we will use sequences from $(\Sigma\times\mathcal{D})^*$ to represent actions, using tags to distinguish between calls/returns and their arguments, whereas elements from $\mathcal{D}$ will indicate the names involved.

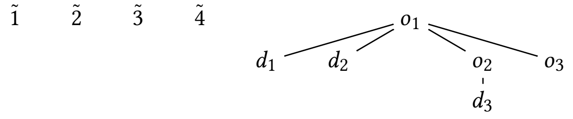
Nested data. In addition to using data values to represent names, we will use elements of $\mathcal{D}$ to keep track of the current P-view. To this end, it is convenient to assume that $\mathcal{D}$ has forest structure so that we can use branches of $\mathcal{D}$ to represent P-views. The data values used for this purpose will be called *ghost values* and in examples we will refer to them as o and variants. Ghost values will also help us deal with the heap. Note that the heap can in general become unbounded, e.g. if a term contains a subterm of the form '$\lambda f.\mathbf{let}\, x = \mathbf{ref}_{\mathsf{Int}}\,(0)\,\mathbf{in}\,\cdots$' and the subterm is called repeatedly by O. In order to tackle this source of unboundedness, we will represent the heap in a distributed fashion as memories associated with ghost values.

Nested data class memory automata (NDCMA) [6] take advantage of the forest structure of $\mathcal{D}$ and, when reading a letter $(a,d)\in\Sigma\times\mathcal{D}$, allow the automaton to access not only the memory associated with d but also that of its ancestors. Consequently, if we use memories of ghost values to store parts of the heap as they are being created in the P-view, the automaton will always be able to modify locations from the current P-view. This suffices to simulate heap changes in FOSC, but would be unsound for terms with references to references, as then chunks of heap could move freely between branches.

The use of ghost values will follow a certain discipline: initially and after each O-action, we create and *open* a new ghost value to reflect the change in the P-view and, once a P-action takes place, the name will be *closed*. Overall, the trace given above will be represented by the data word

$$
\begin{array}{l}
(\mathrm{Gopen},o_1)\,(\mathrm{Pcall}(\square),\tilde{1})\,(\mathrm{Parg},d_1)\,(\mathrm{Pcont},d_2)\,(\mathrm{Gclose},o_1)\\
\qquad(\mathrm{Ocall}(\square),d_1)\,(\mathrm{Oarg},\tilde{3})(\mathrm{Ocont},\tilde{4})\,(\mathrm{Gopen},o_2)\\
\qquad(\mathrm{Pcall}(()),\tilde{3})\,(\mathrm{Pcont},d_3)\,(\mathrm{Gclose},o_2)\\
\qquad(\mathrm{Oret}(()),d_2)(\mathrm{Gopen},o_3)\;(\mathrm{Pret}(()),\tilde{2})\,(\mathrm{Gclose},o_3)
\end{array}
$$

where we used fixed data values $\tilde{1},\tilde{2},\tilde{3},\tilde{4}$ to represent indices, and the data values involved are related in $\mathcal{D}$ as pictured below.



2 HOSC AND FOSC

HOSC is a call-by-value programming language equipped with general store and control flow operations [14]. We consider its finitary version with bounded base type Int and without recursion. The syntax is presented in Figure 1. HOSC typing judgments are of the form $\Sigma;\Gamma\vdash M:\tau$, where Σ and Γ are finite partial functions that assign types to locations and variables respectively. The operational semantics are defined over configurations of the form (M,h), where h is a heap mapping (at least) the locations in M to values. We write the reduction relation $(M,h)\longrightarrow(M',h')$, and write $(M,h)\Downarrow$ if there exist V,h' such that $(M,h)\longrightarrow^*(V,h')$ and V is a value. The full details of the typing rules and operational semantics can be found in [15]. We will be interested in deciding contextual equivalences with respect to HOSC contexts, denoted by C (K denotes evaluation contexts). Let us write $C[\tau]$ for contexts with a hole of type τ.

Definition 2.1 (Contextual Approximation and Equivalence). Given HOSC terms $\Gamma\vdash M_1,M_2:\tau$, we define $\Gamma\vdash M_1\lesssim M_2$ to hold, when for all HOSC contexts $C[\tau]$ such that $\vdash C[M_1],C[M_2]:\tau'$ for some τ', if $(C[M_1],\emptyset)\Downarrow$ then $(C[M_2],\emptyset)\Downarrow$. The terms $\Gamma\vdash M_1,M_2:\tau$ are called *contextually equivalent*, written $\Gamma\vdash M_1\simeq M_2$, when $\Gamma\vdash M_1\lesssim M_2$ and $\Gamma\vdash M_2\lesssim M_1$.

Although we consider finite base types and no recursion, the termination problem for HOSC is still undecidable. This is because higher-order references can be used to encode fixed-point combinators (Landin's knot) and the combination of higher-order recursion and finite state is expressive enough to encode Turing machines [17]. As a similar construction can be carried out using references to continuations, we shall consider a restriction of HOSC in which references to functions and continuations are disallowed, i.e. reference types are restricted to $\mathsf{Ref}\tau$ with τ generated by the grammar

$$\tau\triangleq\beta\,|\,\mathsf{Ref}\tau$$

where β ranges over $\{\mathsf{Unit},\mathsf{Int},\mathsf{Bool}\}$. The resultant language will be called GOSC. As $\simeq$ will also turn out undecidable for GOSC terms (for subtler reasons than undecidability of termination), we need to impose a further restriction on reference types and allow only $\mathsf{Ref}\beta$, i.e. references to values of base types. This language will be referred to as FOSC.

Example 2.2. Consider the FOSC terms $\vdash M_1,M_2:\mathsf{Int}\to(\mathsf{Int}\to\mathsf{Int})\to((\mathsf{Unit}\to\mathsf{Unit})\times(\mathsf{Unit}\to\mathsf{Int}))$ given by

$$
\begin{array}{rcl}
M_1 & = & \lambda x.\lambda f.\mathbf{let}\,r=\mathbf{ref}\,\widehat{0}\,\mathbf{in}\\
 & & \mathbf{let}\,inc=\lambda u.r:=!r+1\,\mathbf{in}\\
 & & \mathbf{let}\,get=\lambda u.\mathbf{if}\,!r<x\,\mathbf{then}\,f\,(!r)\,\mathbf{else}\,\Omega\,\mathbf{in}\,(inc,get),\\
M_2 & = & \lambda x.\lambda f.\mathbf{let}\,r=\mathbf{ref}\,x\,\mathbf{in}\\
 & & \mathbf{let}\,dec=\lambda u.r:=!r-1\,\mathbf{in}\\
 & & \mathbf{let}\,get=\lambda u.\mathbf{if}\,!r>\widehat{0}\,\mathbf{then}\,f\,(x-!r)\,\mathbf{else}\,\Omega\,\mathbf{in}\,(dec,get),
\end{array}
$$

$$
\begin{array}{rcl}
\sigma, \tau & \triangleq & \mathsf{Unit} \mid \mathsf{Int} \mid \mathsf{Bool} \mid \mathsf{Ref}\tau \mid \tau \times \sigma \mid \tau \to \sigma \mid \mathsf{Cont}\ \tau \\
U, V & \triangleq & () \mid \mathbf{tt} \mid \mathbf{ff} \mid \widehat{n} \mid x \mid \ell \mid \langle U, V \rangle \mid \lambda x^{\tau}.M \mid \mathbf{cont}_{\tau}\, K \\
M, N & \triangleq & V \mid \Omega \mid \langle M, N \rangle \mid \pi_i M \mid MN \mid \mathbf{ref}_{\tau}\, M \mid !M \mid M := N \mid \mathbf{if}\, M_1\, \mathbf{then}\, M_2\, \mathbf{else}\, M_3 \mid M \oplus N \mid M \boxdot N \mid M = N \mid \mathbf{call/cc}_{\tau}(x.M) \\
 & & \mid \mathbf{throw}_{\tau}\ \mathrm{M}\ \mathbf{to}\ N \\
K & \triangleq & \bullet \mid \langle V, K \rangle \mid \langle K, M \rangle \mid \pi_i K \mid VK \mid KM \mid \mathbf{ref}_{\tau}\, K \mid !K \mid V := K \mid K := M \mid \mathbf{if}\, K\, \mathbf{then}\, M\, \mathbf{else}\, N \mid K \oplus M \mid V \oplus K \mid K \boxdot M \mid V \boxdot K \\
 & & \mid K = M \mid V = K \mid \mathbf{throw}_{\tau}\ \mathrm{V}\ \mathbf{to}\ K \mid \mathbf{throw}_{\tau}\ \mathrm{K}\ \mathbf{to}\ M \\
C & \triangleq & \bullet \mid \langle M, C \rangle \mid \langle C, M \rangle \mid \pi_i C \mid \lambda x^{\tau}.C \mid MC \mid CM \mid \mathbf{ref}_{\tau}\, C \mid !C \mid C := M \mid M := C \mid \mathbf{if}\, C\, \mathbf{then}\, M\, \mathbf{else}\, N \mid \mathbf{if}\, M\, \mathbf{then}\, C\, \mathbf{else}\, N \\
 & & \mid \mathbf{if}\, M\, \mathbf{then}\, N\, \mathbf{else}\, C \mid C \oplus M \mid M \oplus C \mid C \boxdot M \mid M \boxdot C \mid C = M \mid M = C \mid \mathbf{call/cc}_{\tau}(x.C) \mid \mathbf{throw}_{\tau}\ \mathrm{C}\ \mathbf{to}\ M \mid \mathbf{throw}_{\tau}\ \mathrm{M}\ \mathbf{to}\ C
\end{array}
$$

Notational conventions: $x, y \in \mathbf{Var}$, $\ell \in \mathbf{Loc}$, $n \in \{0, \cdots, \max\}$, $i \in \{1, 2\}$, $\oplus \in \{+, -, *\}$, $\boxdot \in \{=, \neq, <\}$, Ω is the canonical divergent term without a reduction rule. Syntactic sugar: $\mathbf{let}\, x = M\, \mathbf{in}\, N$ stands for $(\lambda x.N)M$ (if x does not occur in N we also write $M; N$).

Figure 1: HOSC **syntax**

and assume $\widehat{\max} + \widehat{1} = \widehat{\max}$ (so addition does not overflow) and $\widehat{0} - \widehat{1} = \widehat{0}$. Then $\vdash M_1 \simeq M_2$, which is something that could be automatically proven using the technique presented in this paper. Counter programs like this have long been considered as challenging examples of contextual equivalence to establish [23], due to the fact they involve a pair of functions which encapsulated shared state. Notably, this example falls into a type fragment where contextual equivalence is undecidable when tested with contexts with first-order state only and without control, i.e. when the contexts come from the continuation-free fragment FOS of FOSC [5]. In contrast, $\simeq$ is defined using HOSC contexts.

We are going to study contextual equivalence through trace models proposed in [14]. They are applicable to *cr-free* terms.

Definition 2.3. A HOSC term $\Gamma \vdash M : \tau$ is ***cr-free*** if it does not contain occurrences of $\mathbf{cont}_{\sigma}\, K$ and locations, and its boundary types (i.e. the types occurring in Γ and τ) are cont- and ref-free.

Note that the restriction on $\mathbf{cont}_{\sigma}\, K$ and locations is insignificant, as both are run-time constructs, not to be used directly by programmers. Observe also that both Example 1.1 and 2.2 are cr-free. In fact, most examples studied in the literature are of this kind [1, 9, 23]. Our main result will show that $\simeq$ is decidable for *all cr-free* FOSC *terms*. In particular, the type-theoretic order of terms will remain unrestricted.

3 OPERATIONAL GAME SEMANTICS

We now review the trace models from [14]. They are based on traces, which can be thought of as exchanges of actions between two players, representing the context (O) and the term (P). Hence, the term *operational game semantics*.

Names and Abstract Values. In actions of the game, players pass (fresh) names to represent functions passed across the boundary. Continuation names are used to identify the question action being answered in an answer action.

Definition 3.1. Let $\mathrm{FNames} = \biguplus_{\sigma,\sigma'} \mathrm{FNames}_{\sigma \to \sigma'}$ be the set of ***function names***, partitioned into mutually disjoint countably infinite sets $\mathrm{FNames}_{\sigma \to \sigma'}$. We will use f, g to range over FNames, and write $f : \sigma \to \sigma'$ for $f \in \mathrm{FNames}_{\sigma \to \sigma'}$.

Analogously, let $\mathrm{CNames} = \biguplus_{\sigma} \mathrm{CNames}_{\sigma}$ be the set of ***continuation names***. We will use c, d to range over CNames, and write

$$
\begin{array}{lcl}
\mathbf{AVal}_{\sigma}(V) & \triangleq & \{(V, \emptyset)\} \quad \text{for } \sigma \in \{\mathsf{Unit}, \mathsf{Bool}, \mathsf{Int}\} \\
\mathbf{AVal}_{\sigma \to \sigma'}(V) & \triangleq & \{(f, [f \mapsto V]) \mid f \in \mathrm{FNames}_{\sigma \to \sigma'}\} \\
\mathbf{AVal}_{\sigma \times \sigma'}(\langle U, V \rangle) & \triangleq & \{(\langle A_1, A_2 \rangle, \gamma_1 \cdot \gamma_2) \mid (A, \gamma_1) \in \mathbf{AVal}_{\sigma}(U), \\
 & & \qquad (A_2, \gamma_2) \in \mathbf{AVal}_{\sigma'}(V)\}
\end{array}
$$

Figure 2: Value decomposition into abstract values and substitutions, and generation of abstract value sequences

$c : \sigma$ for $c \in \mathrm{CNames}_{\sigma}$. We assume that CNames, FNames are disjoint and let $\mathrm{Names} = \mathrm{FNames} \uplus \mathrm{CNames}$. Elements of Names will appear in structures throughout this work, and so $\nu(X)$ refers to the set of names used in some entity X.

Players take actions which consist of a name applied to some value. To handle functional arguments, ***abstract values*** are used. These are values with occurrences of functions replaced by names, generated by the grammar

$$A, B \triangleq f \mid () \mid \widehat{n} \mid \mathbf{tt} \mid \mathbf{ff} \mid \langle A, B \rangle,$$

where $n \in \{0, \cdots, \max\}$.

As names are intrinsically typed, abstract values can be typed in the obvious way, denoted $A : \sigma$. Given a value $V : \sigma$, we let $\mathbf{AVal}_{\sigma}(V)$ be the set of pairs (A, γ), where $A : \sigma$ is an abstract value and $\gamma : \nu(A) \to \mathrm{Vals}$ is a substitution (defined in Figure 2) such that $A\{\gamma\} = V$, i.e. A is an abstract value that matches V and γ is the corresponding matching.

Remark 3.2. *Note that $\cdot$ implicitly requires that function domains be disjoint, and $\uplus$ means the argument sets are disjoint.*

3.1 Play

Traces consist of actions that have a polarity, either P (Player) or O (Opponent), depending on whether they are made by the term being tested (P) or the context (O). Names are ***introduced*** either in an initial set N_O of names, or in values appearing in actions. Names are owned by the player whose action introduced the name (with names in N_O owned by O), and are referred to as O-names or P-names. The players take alternating actions, applying a name introduced by the other player to an abstract value. The types of actions are:

- ***Player Answer*** (PA) $\bar{c}(A)$, where $c : \sigma$ and $A : \sigma$. This corresponds to the term returning a value A using continuation name c.
- ***Player Question*** (PQ) $\bar{f}(A, c)$, where $f : \sigma \to \sigma'$, $A : \sigma$ and $c : \sigma'$. This corresponds to the term calling the function named by f, passing A as argument, and expecting the result with continuation name c.
- ***Opponent Answer*** (OA) $c(A)$, where $c : \sigma$ and $A : \sigma$. In this case, the context is producing a value A to the term, which is acting as a continuation with name c.
- ***Opponent Question*** (OQ) $f(A, c)$, where $f : \sigma \to \sigma'$, $A : \sigma$ and $c : \sigma'$. This action corresponds to the context calling the function named by f from the term, passing A as argument, and expecting the result with continuation name c.

In what follows, $\mathbf{a}$ is used to range over actions. We refer to f in $\bar{f}(A, c)$ and $f(A, c)$, and c in $\bar{c}(A)$ and $c(A)$ as the *head names* of $\mathbf{a}$.

Definition 3.3. Let $N_O \subseteq$ Names. An N_O-***trace*** is a sequence t of actions such that: actions alternate between P and O actions; no name is introduced twice; names from N_O need no introduction; any action $\mathbf{a}$ must have the form $\bar{f}(A, c)$, $f(A, c)$, $\bar{c}(A)$ or $c(A)$, where the head name of $\mathbf{a}$ was introduced by an earlier action $\mathbf{a}'$ of opposite polarity or $f \in N_O$ in the first case, or $c \in N_O$ in the third.

Example 3.4. We revisit the sequence from Example 1.2. Let $N_O = \{f, c\}$, where $f : ((\mathsf{Unit} \to \mathsf{Unit}) \to \mathsf{Unit}) \to \mathsf{Unit}$, $c : \mathsf{Unit}$. Then the sequence $t = \bar{f}(h, c_1)\ h(g, c_2)\ \bar{g}((), c_3)\ c_1(())\ \bar{c}(())$, where $h : (\mathsf{Unit} \to \mathsf{Unit}) \to \mathsf{Unit}$, $g : \mathsf{Unit} \to \mathsf{Unit}$ and $c_1, c_2, c_3 : \mathsf{Unit}$, is an N_O-trace.

3.2 Transition System

To generate traces corresponding to a term, one defines a special LTS, called $\mathcal{L}_{\mathrm{HOSC}}$. Its transition rules are presented in Figure 3. $\mathcal{L}_{\mathrm{HOSC}}$ contains terms built from HOSC syntax, extended with function names as values (with the obvious typing rule) and $\mathbf{cont}_\tau\ (K, c)$, which stands for $\mathbf{cont}_\tau\ K$ which, when thrown to, passes its result to c. The reduction $(M, h) \to (M', h')$ is extended to work on triples, taking the form $(M, c, h) \to (M', c', h')$, where c, c' are continuation names. This is done to ensure **call/cc** and **throw** keep track of the appropriate continuation name.

There are two types of configurations in $\mathcal{L}_{\mathrm{HOSC}}$: $\langle \gamma, \xi, \phi, h\rangle$ (*passive*, O to play) and $\langle M, c, \gamma, \xi, \phi, h\rangle$ (*active*, internal or P to play). In both, ϕ contains all names introduced so far by both players and h is the current heap. γ is an *environment* mapping function names introduced by P to functions, and continuation names to evaluation contexts (which represent the continuation at that point). ξ maps continuation names introduced by P to continuation names introduced by O, indicating continuation awaiting the result of $\gamma(c)$. In an active configuration, M is the *term* component, which captures the current behaviour of P, and c is the continuation name to return the result to. Transitions from active configurations are driven by the term components, while passive configurations 'choose' an action to take. Note that $\uplus$ stands for set-theoretic union on the understanding that the sets involved are disjoint. This guarantees freshness of names introduced by both players.

$$
\begin{array}{l|lll}
(P\tau) & \langle M, c, \gamma, \xi, \phi, h\rangle & \xrightarrow{\tau} & \langle N, c', \gamma, \xi, \phi, h'\rangle \\
 & \multicolumn{3}{l}{\text{when } (M, c, h) \to (N, c', h')} \\
(PA) & \langle V, c, \gamma, \xi, \phi, h\rangle & \xrightarrow{\bar{c}(A)} & \langle \gamma\cdot\gamma', \xi, \phi \uplus \nu(A), h\rangle \\
 & \multicolumn{3}{l}{\text{when } c : \sigma,\ (A, \gamma') \in \mathbf{AVal}_\sigma(V)} \\
(PQ) & \langle K[fV], c, \gamma, \xi, \phi, h\rangle & \xrightarrow{\bar{f}(A,c')} & \langle \gamma\cdot\gamma'\cdot[c' \mapsto K], \xi\cdot[c' \mapsto c], \\
 & & & \quad \phi \uplus \nu(A) \uplus \{c'\}, h\rangle \\
 & \multicolumn{3}{l}{\text{when } f : \sigma \to \sigma',\ (A, \gamma') \in \mathbf{AVal}_\sigma(V),\ c' : \sigma'} \\
(OA) & \langle \gamma, \xi, \phi, h\rangle & \xrightarrow{c(A)} & \langle K[A], c', \gamma, \xi, \phi \uplus \nu(A), h\rangle \\
 & \multicolumn{3}{l}{\text{when } c : \sigma,\ A : \sigma,\ \gamma(c) = K,\ \xi(c) = c'} \\
(OQ) & \langle \gamma, \xi, \phi, h\rangle & \xrightarrow{f(A,c)} & \langle VA, c, \gamma, \xi, \phi \uplus \nu(A) \uplus \{c\}, h\rangle \\
 & \multicolumn{3}{l}{\text{when } f : \sigma \to \sigma',\ A : \sigma,\ c : \sigma',\ \gamma(f) = V}
\end{array}
$$

Figure 3: $\mathcal{L}_{\mathrm{HOSC}}$ LTS

3.3 Trace semantics

Let $\Gamma \vdash M : \sigma$ be a cr-free HOSC term such that $\Gamma = \{x_1 : \sigma_1, \cdots, x_k : \sigma_k\}$. A Γ-***assignment*** ρ is a map from $\{x_1, \cdots, x_k\}$ to the set of abstract values such that, for all $1 \le i \ne j \le k$, we have $\rho(x_i) : \sigma_i$ and $\nu(\rho(x_i)) \cap \nu(\rho(x_j)) = \emptyset$. ρ simply creates a supply of names corresponding to the context. Let ρ be a Γ-assignment, $c : \sigma$ and $N_O = \nu(\rho) \cup \{c\}$. Then the active *initial configuration* $\mathsf{C}_M^{\rho,c}$ is defined to be $\langle M\{\rho\}, c, \emptyset, N_O, \emptyset\rangle$.

Definition 3.5. Given configurations C, C' and $\mathsf{t} = \mathbf{a}_1 \dots \mathbf{a}_n$, we write $\mathsf{C} \xrightarrow{\mathsf{t}} \mathsf{C}'$ if there exist $\mathsf{C}_1, \mathsf{C}_1', \cdots, \mathsf{C}_n, \mathsf{C}_n'$ such that $\mathsf{C} \xrightarrow{\tau}{}^* \mathsf{C}_1 \xrightarrow{\mathbf{a}_1} \mathsf{C}_1' \xrightarrow{\tau}{}^* \cdots \xrightarrow{\tau}{}^* \mathsf{C}_n \xrightarrow{\mathbf{a}_n} \mathsf{C}_n' \xrightarrow{\tau}{}^* \mathsf{C}'$. Define

$$\mathbf{Tr}_{\mathrm{HOSC}}(\mathsf{C}) \triangleq \{\mathsf{t} \mid \text{there exists } \mathsf{C}' \text{ such that } \mathsf{C} \xrightarrow{\mathsf{t}} \mathsf{C}'\}.$$

Remark 3.6. *Due to the freedom of name choice,* $\mathbf{Tr}_{\mathrm{HOSC}}(\mathsf{C})$ *is closed under type-preserving renamings that preserve names from* C.

Definition 3.7. The ***trace semantics*** of a cr-free HOSC term $\Gamma \vdash M : \sigma$ is defined to be

$$\mathbf{Tr}_{\mathrm{HOSC}}(\Gamma \vdash M : \sigma) \quad \triangleq \quad \{((\rho, c), t) \mid \rho \text{ is a } \Gamma\text{-assignment}, \; c : \sigma,\ t \in \mathbf{Tr}_{\mathrm{HOSC}}(\mathsf{C}_M^{\rho,c})\}.$$

Example 3.8. Consider the term $\Gamma \vdash M_1$ from Example 1.1 and the trace t from Example 3.4. Letting $\rho = \{f \mapsto f\}$, $c : \mathsf{Unit}$, we have $((\rho, c), t) \in \mathbf{Tr}_{\mathrm{HOSC}}(\Gamma \vdash M_1)$.

The full abstraction result from [14] then states that trace inclusion coincides exactly with contextual approximation and, thus, contextual equivalence is captured by trace equivalence.

Theorem 3.9 (Full Abstraction). *For any cr-free* HOSC *terms* $\Gamma \vdash M_1, M_2 : \sigma$, *we have* $\Gamma \vdash M_1 \lesssim M_2$ *iff* $\mathbf{Tr}_{\mathrm{HOSC}}(\Gamma \vdash M_1) \subseteq \mathbf{Tr}_{\mathrm{HOSC}}(\Gamma \vdash M_2)$.

Remark 3.10. *The full abstraction result [14, 15] was shown for the infinitary version of* HOSC *(with infinite* Int*). However, the unboundedness of* Int *was not relevant to the argument. In particular, in the definability argument (proof of Lemma 5), which shows that for every* finite trace *one can find a corresponding configuration, only integer values from the finite trace were actually used, i.e. the construction carries over to the finitary setting.*

Theorem 3.9 provides a powerful handle on reasoning about equivalence, which we will exploit in the remainder of the paper to conclude that $\simeq$ is decidable for cr-free FOSC terms.

Meanwhile, let us argue that $\simeq$ is not decidable for GOSC terms. To this end, we appeal to the undecidability result for GOS (GOSC without continuations) with respect to GOS contexts [22]. Note that it is not immediately clear that this implies undecidability for $\simeq$, which is defined using HOSC contexts. Indeed, in general, $\simeq$ is more discriminating, because there are more contexts available for testing. In [14], this is captured by imposing extra restrictions on O if GOS contexts are used (O-visibility, O-bracketing, trace completeness). However, the undecidability result in [22] uses closed GOS terms of type Unit $\rightarrow$ Unit $\rightarrow$ Unit and, for such terms, one can show that the restrictions on O hold vacuously. Consequently, we can conclude that such terms are contextually equivalent with respect to GOS contexts if and only if they are equivalent with respect to HOSC contexts (i.e. $\simeq$-equivalent). This implies the following result.

Theorem 3.11. *$\simeq$ is undecidable for* GOS *(and thus* GOSC*) terms.*

In the remainder of the paper we focus on developing the decidability result for FOSC. Recall that FOSC allows references of type Refβ ($\beta \in \{$Unit, Bool, Int$\}$) only, whereas in GOSC they could have type Ref$(\cdots(\text{Ref}\beta))$, i.e. references to references were allowed.

4 BOUNDS IN $\mathcal{L}_{\text{HOSC}}$

To allow us to construct an automaton from the LTS, we establish a series of (effectively computable) bounds on the behaviour of $\mathcal{L}_{\text{HOSC}}$ for GOSC terms. They are slightly technical in that they rely on the concept of a P-view from game semantics [12]. Intuitively, the P-view of a trace determines a segment of the trace that contains O-names which are available to the term. Although the results we discuss here and in the next section apply to GOSC, our ultimate decidability result will be for FOSC, because we will be unable to accommodate the storage of names in the heap while translating terms to automata.

Definition 4.1. The ***P-view*** of a trace is defined by

$$\begin{array}{lcl}
\text{View}_P(\epsilon) & \triangleq & \epsilon \\
\text{View}_P(t\ \bar{f}(A,c)\ t'\ c(A')) & \triangleq & \text{View}_P(t)\ \bar{f}(A,c)\ c(A') \\
\text{View}_P(t\ \bar{f}(A,c)\ t'\ g(A',c')) & \triangleq & \text{View}_P(t)\ \bar{f}(A,c)\ g(A',c') \\
 & & \text{where } g \in \nu(A) \\
\text{View}_P(t\ \bar{c}(A)\ t'\ g(A',c')) & \triangleq & \text{View}_P(t)\ \bar{c}(A)\ g(A',c') \\
 & & \text{where } g \in \nu(A) \\
\text{View}_P(t\ \mathbf{a}) & \triangleq & \text{View}_P(t)\ \mathbf{a} \\
 & & \text{where } \mathbf{a} \text{ is a P-action}
\end{array}$$

Note that View$_P(t)$ is defined by tracing the head name of an O-move to its point of introduction.

Example 4.2. Consider $t = \bar{f}(h,c_1)\ h(g,c_2)\ \bar{g}((),c_3)\ c_1(())\ \bar{c}(())$ from Example 3.4. Let us write $t_{\leq m}$ for the initial segment of t consisting of m actions. Here are the P-views arising in t:

$$\begin{array}{lcl}
\text{View}_P(t_{\leq 0}) & = & \epsilon, \\
\text{View}_P(t_{\leq 1}) & = & \bar{f}(h,c_1), \\
\text{View}_P(t_{\leq 2}) & = & \bar{f}(h,c_1)\ h(g,c_2), \\
\text{View}_P(t_{\leq 3}) & = & \bar{f}(h,c_1)\ h(g,c_2)\ \bar{g}((),c_3), \\
\text{View}_P(t_{\leq 4}) & = & \bar{f}(h,c_1)\ c_1(()), \\
\text{View}_P(t_{\leq 5}) & = & \bar{f}(h,c_1)\ c_1(())\ \bar{c}(()).
\end{array}$$

Next we single out the sequences of transitions that are responsible for generating P-views.

Definition 4.3. Let $\Gamma \vdash M : \sigma$ be a cr-free GOSC term, ρ a Γ-assignment, and $c : \sigma$ a continuation name. A sequence s is a ***view path*** if there exists a trace t such that $\mathrm{C}_M^{\rho,c} \xrightarrow{t} \mathbf{C}$, and s is the subsequence of transitions witnessing $\mathrm{C}_M^{\rho,c} \xrightarrow{t} \mathbf{C}$ consisting of

1. all transitions that contribute actions to View$_P(t)$, and
2. all consecutive τ-transitions starting at $\mathrm{C}_M^{\rho,c}$ or following O-actions selected in (1).

Our main bound, from which the other will be derived, is stated below.

Theorem 4.4. *Let $\Gamma \vdash M : \sigma$ be a cr-free* GOSC *term, ρ a Γ-assignment, and $c : \sigma$ a continuation name. Then there exists a bound* PathBound(M) *on the length of view paths from $\mathrm{C}_M^{\rho,c}$.*

The proof of this theorem is achieved using techniques espoused by Loader [18, 19] in the context of the lambda calculus, which he used to measure lengths of reduction paths in the simply-typed lambda calculus. We adapt them to establish a bound on the length of reduction sequences for a modified reduction relation, designed to capture the behaviour of $\mathcal{L}_{\text{HOSC}}$. An immediate consequence of Theorem 4.4 is a bound on P-views.

Corollary 4.5. *For any cr-free* GOSC *term $\Gamma \vdash M : \sigma$, there exists a bound* ViewLen(M) *such that, for any t in* $\mathbf{Tr}_{\text{HOSC}}(\Gamma \vdash M : \sigma)$*, the length of* View$_P(t)$ *is bounded by* ViewLen(M).

The theorem also gives rise to a bound on the size of terms generated by $\mathcal{L}_{\text{HOSC}}$, due to the fact that such terms occur in some view path. Only transitions in the view path can contribute to the growth of such terms, and can at most square the size of the term (in the case of τ-reductions) or increase it by a fixed size (in O-actions). Thus Corollary 4.5 implies a bound.

Definition 4.6. Let $\Gamma \vdash M : \sigma$ be a cr-free GOSC term. Let Reach$(M) \triangleq \{N \mid \rho$ is a Γ-assignment, $\mathrm{C}_M^{\rho,c} \xrightarrow{t} \mathbf{C}$, $\mathbf{C}$ is active and N is its term component$\}$.

Corollary 4.7. *Let $\Gamma \vdash M : \sigma$ be a cr-free* GOSC *term. There there exists a bound* TermSize(M) *such that $N \in$* Reach(M) *implies $|N| \leq$* TermSize(M)*, where $|N|$ denotes the size of the derivation of N in the grammar.*

A final bound concerns the number of locations that can be generated during the contiguous reductions in $\mathcal{L}_{\text{HOSC}}$.

Lemma 4.8. *For any cr-free* GOSC *term $\Gamma \vdash M : \sigma$, there exists a bound* HeapBound(M) *such for any sequence of transitions s from $\mathrm{C}_M^{\rho,c}$, the number of locations generated in consecutive τ-transitions in s is bounded by* HeapBound(M).

Note that the bound identified above only concerns location generation during *consecutive* τ-transitions. In general, the heaps in $\mathcal{L}_{\mathrm{HOSC}}$ are unbounded.

5 CANONICAL REPRESENTATION

Traces generated by GOSC terms are known to satisfy a technical condition, called P-visibility, which states that P-actions may only involve O-names from the current P-view [14]. This, in conjunction with the bound on the length of P-views established in the previous section, gives us a way to eliminate O-names and replace them with bounded numerical indices.

To this end, we first define the notion of P-visible names by ordering the O-names in the corresponding P-view in the obvious way and adding names from N_O at the front.

Definition 5.1. Given an even-length N_O-trace t, the sequence $\mathsf{Vis}_P(t)$ of ***P-visible names*** is defined as follows:

$$\begin{array}{rcll} \mathsf{Vis}_P(\epsilon) & \triangleq & \underline{N_O} & \\ \mathsf{Vis}_P(t\ \bar{f}(A,c)\ t'\ c(A')) & \triangleq & \mathsf{Vis}_P(t)\ \underline{\nu(A')} & \\ \mathsf{Vis}_P(t\ \bar{f}(A,c')\ t'\ g(A',c')) & \triangleq & \mathsf{Vis}_P(t)\ \underline{\nu(A')}\ c' & g\in\nu(A) \\ \mathsf{Vis}_P(t\ \bar{c}(A)\ t'\ g(A',c')) & \triangleq & \mathsf{Vis}_P(t)\ \underline{\nu(A')}\ c' & g\in\nu(A) \end{array}$$

where $\underline{N_O}$ stands for some fixed sequence of all names from N_O and $\underline{\nu(A')}$ is the sequence of elements of $\nu(A')$ listed in order of appearance in A'.

We then let $\mathrm{FromInd}(t,i)$ be the ith element of $\mathsf{Vis}_P(t)$ and $\mathrm{ToInd}(t,f)$ be the index of the name f in $\mathsf{Vis}_P(t)$. By convention, the first element will be assigned index 1. We write $|\mathsf{Vis}_P(t)|$ to refer to the length of $\mathsf{Vis}_P(t)$.

Example 5.2. Consider the $\{f,c\}$-trace $t=\bar{f}(h,c_1)\ h(g,c_2)\ \bar{g}((),c_3)\ c_1(())\ \bar{c}(())$ from Example 3.4. The corresponding P-views were shown in Example 4.2. Here are the corresponding sequences of P-visible names: $\mathsf{Vis}_P(t_{\le 0})=fc$, $\mathsf{Vis}_P(t_{\le 2})=fcgc_2$, $\mathsf{Vis}_P(t_{\le 4})=fc$. We have $\mathrm{FromInd}(t_{\le 2},1)=f$ and $\mathrm{ToInd}(t_{\le 2},c_2)=4$.

Definition 5.3. An N_O-trace t is ***P-visible*** if both

- for any prefix $t'\ \bar{f'}(A',c')$ of t, the name f' occurs in $\mathsf{Vis}_P(t')$,
- for any prefix $t'\ \bar{c'}(A')$ of t, the name c' occurs in $\mathsf{Vis}_P(t')$.

Example 5.4. The $\{f,c\}$-trace t from Example 5.2 is P-visible: f occurs in $\mathsf{Vis}_P(t_{\le 0})$, g occurs in $\mathsf{Vis}_P(t_{\le 2})$, and c occurs in $\mathsf{Vis}_P(t_{\le 4})$.

Next we will incorporate indices into traces. For a start, we need a notion of abstract value with indices.

Definition 5.5. An ***indexed abstract value*** is an abstract value in which indices were substituted for names, we denote them A^I. $\mathbf{AVal}_\sigma(A,i)$, defined below, specifies a decomposition (A^I,γ,j) of $A:\sigma$ into an indexed value A^I, a corresponding substitution (for indices) γ, and $j\ge i$ such that all indices in A^I are from $[i..j)$ (counting left to right) and j is the smallest index unused in A^I.

$$\begin{array}{lcl} \mathbf{AVal}_\sigma(A,i) & \triangleq & (A,\emptyset,i) \quad \text{for } \sigma\in\{\mathsf{Unit},\mathsf{Bool},\mathsf{Int}\} \\ \mathbf{AVal}_{\sigma\to\sigma'}(A,i) & \triangleq & (i,[i\mapsto A],i+1) \\ \mathbf{AVal}_{\sigma\times\sigma'}(\langle A_1,A_2\rangle,i) & \triangleq & (\langle A_1^I,A_2^I\rangle,\gamma_1\cdot\gamma_2,i'') \\ & & \ \text{where } (A_1^I,\gamma_1,i')=\mathbf{AVal}_\sigma(A_1,i) \\ & & \ \text{and } (A_2^I,\gamma_2,i'')=\mathbf{AVal}_{\sigma'}(A_2,i') \end{array}$$

We can now define the translation of P-visible traces into their canonical forms. They will be traces in which all occurrences of O-names are replaced with indices in a way guided by the number of O-names in the current P-view.

Definition 5.6. The ***canonical form*** $\mathbf{Can}(t)$ of a P-visible N_O-trace t is defined by:

$$\begin{array}{lcll} \mathbf{Can}(\epsilon) & \triangleq & \epsilon & \\ \mathbf{Can}(t'\ \mathbf{a}\ t''\ c(A)) & \triangleq & \mathbf{Can}(t'\ \mathbf{a}\ t'')\ c(A^I) & c \text{ introduced in } \mathbf{a}, \\ & & \multicolumn{2}{l}{\quad i=|\mathsf{Vis}_P(t')|{+}1,\ (A^I,\gamma,i')=\mathbf{AVal}(A,i)} \\ \mathbf{Can}(t'\ \mathbf{a}\ t''\ f(A,c)) & \triangleq & \mathbf{Can}(t'\ \mathbf{a}\ t'')\ f(A^I,i') & f \text{ introduced in } \mathbf{a}, \\ & & \multicolumn{2}{l}{\quad i=|\mathsf{Vis}_P(t')|{+}1,\ (A^I,\gamma,i')=\mathbf{AVal}(A,i)} \\ \mathbf{Can}(t'\ \bar{c}(A)) & \triangleq & \mathbf{Can}(t')\ \bar{i}(A) & i=\mathrm{ToInd}(t',c) \\ \mathbf{Can}(t'\ \bar{f}(A,c)) & \triangleq & \mathbf{Can}(t')\ \bar{i}(A,c) & i=\mathrm{ToInd}(t',f) \end{array}$$

We extend $\mathbf{Can}(t)$ to sets of N_O-traces in the obvious way.

Example 5.7. Suppose $\underline{N_O}=fc$ and $t=\bar{f}(h,c_1)\ h(g,c_2)\ \bar{g}((),c_3)\ c_1(())\ \bar{c}(())\ h(g',c')$. Then $\mathbf{Can}(t)=\bar{1}(h,c_1)\ h(3,4)\ \bar{3}((),c_3)\ c_1(())\ \bar{2}(())\ h(3,4)$.

Lemma 5.8. *Let t,t' be P-visible N_O-traces. Then $\mathbf{Can}(t)=\mathbf{Can}(t')$ if and only if t and t' are equal up to a permutation of O-names which preserves N_O.*

Corollary 5.9. *Let T,T' be sets of P-visible N_O-traces closed under permutation of names not in N_O. Then $\mathbf{Can}(T)=\mathbf{Can}(T')$ if and only if $T=T'$.*

A simple consequence of the bound on P-views from Corollary 4.5 is that finitely many indices suffice.

Lemma 5.10. *For any cr-free* GOSC *term $\Gamma\vdash M:\sigma$, there exists a bound* $\mathrm{MaxInd}(M)$ *such that, for any Γ-assignment ρ and continuation name $c:\sigma$, the largest index used in $\mathbf{Can}(\mathbf{Tr}(\mathsf{C}_M^{\rho,c}))$ is at most* $\mathrm{MaxInd}(M)$.

We now define a modified LTS, called $\mathcal{L}_{\mathrm{HOSC}}^{\mathbf{can}}$, which will generate the canonical representations of traces. Firstly, we will need to extend the syntax and extended reduction to allow indices annotated with a type $(i:\sigma)$ to be used in place of names. The configurations are extended so that they contain a mapping η from P-names to the number of O-names that were P-visible when the P-name was introduced *plus* 1. Similarly, in active configurations, k holds the number of currently P-visible names incremented by 1, i.e. the next available index. This allows the LTS to keep track of $|\mathsf{Vis}_P(t)|+1$ as it progresses and ensure that the generated traces are in canonical form. The transitions are presented in Figure 4. We take $\mathbf{Tr}_{\mathbf{can}}(\mathsf{C})$ to be the set of traces generated by $\mathcal{L}_{\mathrm{HOSC}}^{\mathbf{can}}$ from configuration C.

Definition 5.11. Given a cr-free GOSC-term $\Gamma\vdash M:\sigma$, Γ-assignment ρ and continuation name $c:\sigma$, let $\underline{\nu(\rho)}$ be a (canonical) sequence of names from $\nu(\rho)$, and $\underline{\rho}$ be ρ but with names replaced by (index:type) pairs according to the order in $\underline{\nu(\rho)}$. Let $i=|\nu(\rho)|$. The initial configuration $\mathsf{C}_{M,\mathbf{can}}^{\rho,\sigma}$ is then $\langle M\{\underline{\rho}\},(i{+}1:\sigma),\emptyset,\emptyset,\emptyset,i{+}2,\emptyset\rangle$.

The key result is that the new LTS produces exactly the canonical forms of traces from the original LTS.

Theorem 5.12. *Let $\Gamma\vdash M:\sigma$ be a cr-free* GOSC*-term, ρ a Γ-assignment and $c:\sigma$ a continuation name. Then $\mathbf{Tr}_{\mathbf{can}}(\mathsf{C}_{M,\mathbf{can}}^{\rho,\sigma})=\mathbf{Can}(\mathbf{Tr}(\mathsf{C}_M^{\rho,c}))$.*

Thus, $\mathcal{L}_{\mathrm{HOSC}}^{\mathbf{can}}$ provides a faithful representation of $\mathbf{Tr}_{\mathrm{HOSC}}(\Gamma\vdash M:\sigma)$ for cr-free GOSC-terms $\Gamma\vdash M:\sigma$.

$$
\begin{array}{l|lcl}
(P\tau) & \langle M,(i:\sigma),\gamma,\xi,\phi,h,k,\eta\rangle & \xrightarrow{\tau} & \langle N,(i':\sigma'),\gamma,\xi,\phi,h',k,\eta\rangle \\
& \text{when } (M,(i:\sigma),h)\to(N,(i':\sigma'),h') & & \\
(PA) & \langle V,(i:\sigma),\gamma,\xi,\phi,h,k,\eta\rangle & \xrightarrow{\bar{i}(A)} & \langle\gamma\cdot\gamma',\xi,\phi\uplus\nu(A),h,\eta\cdot[\nu(A)\mapsto k]\rangle \\
& \text{when } (A,\gamma')\in\mathbf{AVal}_\sigma(V) & & \\
(PQ) & \langle K[(i:\tau)V],(j:\tau'),\gamma,\xi,\phi,h,k,\eta\rangle & \xrightarrow{\bar{i}(A,c')} & \langle\gamma\cdot\gamma'\cdot[c'\mapsto K],\xi',\phi\uplus\nu(A)\uplus\{c'\},h,\eta'\rangle \\
& \multicolumn{3}{l}{\text{when } \tau=\sigma\to\sigma',\ (A,\gamma')\in\mathbf{AVal}_\sigma(V),\ c':\sigma',\ \xi'=\xi\cdot[c'\mapsto(j:\tau')],\ \eta'=\eta\cdot[\nu(A),c'\mapsto k]} \\
(OA) & \langle\gamma,\xi,\phi,h,\eta\rangle & \xrightarrow{c(A^I)} & \langle K[U],(j:\tau'),\gamma,\xi,\phi,h,i,\eta\rangle \\
& \multicolumn{3}{l}{\text{when } c:\sigma,\ A:\sigma,\ (A^I,\gamma',i)=\mathbf{AVal}(A,\eta(c)),\ U=\mathrm{Typed}(A^I,\gamma'),\ \gamma(c)=K,\ \xi(c)=(j:\tau')} \\
(OQ) & \langle\gamma,\xi,\phi,h,\eta\rangle & \xrightarrow{f(A^I,i)} & \langle VU,(i:\sigma'),\gamma,\xi,\phi,h,i+1,\eta\rangle \\
& \multicolumn{3}{l}{\text{when } f:\sigma\to\sigma',\ A:\sigma,\ (A^I,\gamma',i)=\mathbf{AVal}(A,\eta(f)),\ U=\mathrm{Typed}(A^I,\gamma'),\ \gamma(f)=V}
\end{array}
$$

Typed(A^I,γ) stands for A^I in which indices are annotated with types implied by γ.

Figure 4: $\mathcal{L}^{\mathbf{can}}_{\mathrm{HOSC}}$ LTS

6 AUTOMATA OVER NESTED DATA

Our aim is to represent $\mathcal{L}^{\mathbf{can}}_{\mathrm{HOSC}}$ using automata. As the action labels involve an infinite set of names, we will need an automata model that can process them. We shall rely on automata that can read words from $(\Sigma\times\mathcal{D})^*$, where Σ is finite and $\mathcal{D}$ is infinite. In the automata literature, such words are called *data words* and the elements of Σ and $\mathcal{D}$ are referred to as ***tags*** and ***data values*** respectively. Data values will be used to encode both names and indices in traces of $\mathcal{L}^{\mathbf{can}}_{\mathrm{HOSC}}$. We will also use data values in auxiliary roles, not corresponding to names in traces, to keep track of P-views and the heap. To this end, it turns out to be convenient to assume that $\mathcal{D}$ has the structure of a forest.

Definition 6.1 (Dataset). $\mathcal{D}$ is a countable infinite set equipped with a partial function pred : $\mathcal{D}\rightharpoonup\mathcal{D}$ (the parent function) satisfying the following conditions.

- Infinite branching and depth: $\mathrm{pred}^{-1}(\{d\})$ is infinite for any $d\in\mathcal{D}$.
- Well-foundedness: for any $d\in\mathcal{D}$, there exists $i\geq 1$, called the ***level*** of d, such that $\mathrm{pred}^{i-1}(d)$ is defined and $\mathrm{pred}^{i}(d)$ is undefined. We assume $\mathrm{pred}^0(d)=d$, i.e. the roots in $\mathcal{D}$ are defined to have level 1.

The forest structure of $\mathcal{D}$ makes it possible to view level-i data values as being nested inside the branch corresponding to its $i-1$ ancestors. Hence, the term *nested data.* To produce automata suitable for capturing $\mathcal{L}^{\mathbf{can}}_{\mathrm{HOSC}}$, we use a variant of *nested data class memory automata* [6]. Analogously to environments in $\mathcal{L}^{\mathbf{can}}_{\mathrm{HOSC}}$, the automata make it possible to associate *memories*, drawn from a finite set $\mathcal{M}$, with data values.

Let $\mathcal{M}_\perp=\mathcal{M}\uplus\{\perp\}$. A ***class memory function*** is a function $\mu:\mathcal{D}\to\mathcal{M}_\perp$ such that $\mu(d)\neq\perp$ for finitely many $d\in\mathcal{D}$ and, for all $d,d'\in\mathcal{D}$, if $\mu(d)=\perp$ and $d=\mathrm{pred}(d')$ then $\mu(d')=\perp$. Consequently, if we ignore data values d such that $\mu(d)=\perp$, we can view μ as a finite subforest of $\mathcal{D}$ with nodes labelled by memories. Note that $\mu(d)=\perp$ will represent the fact that the automaton has not encountered d yet, i.e. that d is *fresh.*

Definition 6.2. A ***nested data class memory automaton*** (NDCMA) is a tuple $\mathcal{A}=\langle Q,\Sigma,\mathcal{M},\mu_0,q_0,F,b,\delta\rangle$, where

- Q is a finite set of states;
- Σ is a finite alphabet of *tags*;
- $\mathcal{M}$ is a finite set of *memories*;
- μ_0 is an initial class memory function;
- $q_0\in Q$ is the initial state;
- $F\subseteq Q$ is the set of *accepting* states;
- $b\in\mathbb{N}$ is the *depth* of the automaton; and
- $\delta\subseteq\bigcup_{1\leq i\leq b}(Q\times\Sigma\times(\mathcal{M}_\perp)^i\times Q\times\mathcal{M}^i)\cup(Q\times Q)$ is the transition relation.

We write $q\xrightarrow[(m'_1,\cdots,m'_i)]{a,(m_1,\cdots,m_i)}q'$ or $q\xrightarrow[\overrightarrow{m'}]{a,\overrightarrow{m}}q'$ for

$$(q,a,(m_1,\cdots,m_i),q',(m'_1,\cdots,m'_i))\in\delta,$$

and $q\xrightarrow{\epsilon}q'$ for $(q,q')\in\delta$. Intuitively, when reading $(a,d)\in\Sigma\times\mathcal{D}$ with d of level i $(1\leq i\leq b)$, the automaton will have access to i memories $\overrightarrow{m}$ corresponding to the branch of $\mathcal{D}$ ending in d, which are then updated to $\overrightarrow{m'}$ according to δ.

$\mathcal{A}$ will be called ***deterministic*** if $q\xrightarrow{\epsilon}q'$ implies there are no other transitions from q, or there are no ϵ-transitions from q and $q,a,\overrightarrow{m}$ uniquely determine $q',\overrightarrow{m'}$ such that $(q,a,\overrightarrow{m},q',\overrightarrow{m'})\in\delta$. A ***configuration*** is a tuple (q,μ), where $q\in Q$ and μ is a class memory function. The initial configuration is (q_0,μ_0). A ***run*** of $\mathcal{A}$ on $(a_1,d_1)\cdots(a_p,d_p)\in(\Sigma\times\mathcal{D})^*$ is a sequence

$$(q_0,\mu_0)(q_1,\mu_1)\cdots(q_p,\mu_p)$$

such that, for all $1\leq j\leq p$, there exist $q'_{j-1},q''_j,\overrightarrow{m_j}=(m_{j1},\cdots,m_{ji_j})$, $\overrightarrow{m'_j}=(m'_{j1},\cdots,m'_{ji_j})$ such that $q_{j-1}\xrightarrow{\epsilon}{}^*q'_{j-1}$, $q'_{j-1}\xrightarrow[\overrightarrow{m'_j}]{a_j,\overrightarrow{m_j}}q''_j$, $q''_j\xrightarrow{\epsilon}{}^*q_j$, where i_j is the level of d_j, and

- memories are accessed: $\mu_{j-1}(\mathrm{pred}^{i_j-k}(d_j))=m_{jk}$ for all $1\leq k\leq i_j$;
- and updated: $\mu_j=\mu_{j-1}[\mathrm{pred}^{i_j-1}(d_j)\mapsto m'_{j1},\cdots,\mathrm{pred}(d_j)\mapsto m'_{j(i_j-1)},d_j\mapsto m'_{ji_j}]$.

A run is ***accepting*** if it ends in a configuration with an accepting state. We define $\mathcal{L}(\mathcal{A})$ to be the set of all data words w on which $\mathcal{A}$ has an accepting run.

Example 6.3. Consider the NDCMA $\langle\{q_0,q_1,q_2\},\{a_1,a_2,a_3,a_4\},\emptyset,$ $q_0,\{q_2\},2,\delta\rangle$ with δ defined by the following transitions.

$$q_0 \xrightarrow[m_1]{a_1,\perp} q_0 \quad q_0 \xrightarrow[(m_2,m_3)]{a_2,(m_1,\perp)} q_1 \quad q_1 \xrightarrow[(m_2,m_4)]{a_3,(m_2,\perp)} q_1 \quad q_1 \xrightarrow[(m_5,m_6)]{a_4,(m_2,m_3)} q_2$$

Below we present an accepting run on $(a_1,d_1)(a_1,d_2)(a_2,d_3)(a_3,d_4)$ (a_4,d_3), where d_1,d_2 are roots and $\mathrm{pred}(d_3)=\mathrm{pred}(d_4)=d_1$. The class memory functions μ inside configurations are shown as finite forests, where d^m stands for $\mu(d)=m$.

$$\begin{tikzcd}[column sep=small, row sep=small]
(q_0,\emptyset) & (q_0,\ d_1^{m_1}) & (q_0,\ d_1^{m_1}\ \ d_2^{m_1}) & (q_1, & d_1^{m_2} \arrow[d, dash, dashed] & d_2^{m_1}\,) \\
 & & & & d_3^{m_3} &
\end{tikzcd}$$

$$\begin{tikzcd}[column sep=small, row sep=small]
(q_1, & & d_1^{m_2} \arrow[dl, dash] \arrow[dr, dash] & & d_2^{m_1}\,) & (q_2, & & d_1^{m_5} \arrow[dl, dash] \arrow[dr, dash] & & d_2^{m_1}\,) \\
 & d_3^{m_3} & & d_4^{m_4} & & & d_3^{m_6} & & d_4^{m_4} &
\end{tikzcd}$$

The automata are a variation on the *weak NDCMA* defined in [6]. For convenience, our definition features an initial class memory function μ_0, whereas, for the original NDCMA, μ_0 was empty, i.e. $\mu_0(d)=\perp$ for all $d\in\mathcal{D}$. As one can use initial steps of an NDCMA to initialise the class memory to the same shape as μ_0, we inherit the decidability results for language containment from [6], provided the initial class memories are defined on the same data values.

Theorem 6.4. *Given two deterministic NDCMA $\mathcal{A}_1,\mathcal{A}_2$, whose initial class memory functions are defined on the same data values, the associated containment problem ($\mathcal{L}(\mathcal{A}_1)\subseteq\mathcal{L}(\mathcal{A}_2)$) is decidable.*

In [6], the above result was obtained via a reduction to well-structured transitions systems [10]. In particular, the problem is at least as hard as that of deciding coverability in Petri nets with reset arcs, i.e. it is Ackermann-hard [24].

7 FROM FOSC TO NDCMA (ENCODING)

In this section we explain how to simulate $\mathcal{L}^{\mathbf{can}}_{\mathrm{HOSC}}$ as an NDCMA, provided that the term $\Gamma\vdash M$ in question comes from FOSC. Recall that NDCMA read data words from $(\Sigma\times\mathcal{D})^*$, where Σ is a finite tag alphabet and $\mathcal{D}$ is an infinite forest-shaped data alphabet, as in Definition 6.1. Let us discuss how to represent traces of $\mathcal{L}^{\mathbf{can}}_{\mathrm{HOSC}}$ as data words. We will represent both indices and P-names using elements of $\mathcal{D}$. As the indices are bounded (Lemma 5.10), we fix a set $\mathcal{N}$ of data values at level 1 and refer to the corresponding data values as $\tilde{i}$:

$$\mathcal{N}=\{\,\tilde{i}\mid 1\le i\le \mathrm{MaxInd}(M)\}.$$

These fixed data values will be provided through the initial class memory function. In contrast, data values corresponding to P-names will be generated by the automaton at run time and their level will vary. Next, we describe the set of finite tags Σ that will be used in the encoding.

7.1 Tag alphabet

Since actions of the LTS may involve multiple names and NDCMA can process only a single name in each step, it is necessary to represent each action as a sequence from $(\Sigma\times\mathcal{D})^*$.

- The first tag in such a sequence will indicate the kind of action as well as the corresponding pattern of arguments. The tags will have one of the following shapes:

  $$\mathrm{Pcall}(A_\square),\mathrm{Pret}(A_\square),\mathrm{Ocall}(A_\square),\mathrm{Oret}(A_\square),$$

  where $A_\square$ stands for *value patterns* generated by the grammar

  $$A_\square \triangleq \square\mid()\mid\widehat{n}\mid\mathbf{tt}\mid\mathbf{ff}\mid\langle A_\square,A_\square\rangle.$$

  Note that they do not contain any names, which have been replaced with a placeholder $\square$. In our encoding of actions, the tags will be paired with a data value that corresponds to the head name of the action.
- Whenever $A_\square$ contains any occurrences of $\square$, the encoding of the action will further contain a sequence from $(\Sigma\times\mathcal{D})^*$ (of the same length as the number of $\square$s in $A_\square$) representing function arguments. Each element of that argument sequence will be tagged with Parg (resp. Oarg), and these tags will be paired up with data values representing P-names (resp. indices).
- Finally, if an action introduces a continuation name then this will be represented using the tag Pcont (resp. Ocont), which will be paired up with the corresponding data value.

Additionally, the encoding of each P-action will be preceded by (Gopen,o) and succeeded by (Gopen,o) for the same data value o, with different data values o used for different P-actions. In particular, the encoding of the whole trace will begin with (Gopen,o) for some o. The values o will be disjoint from those used to simulate indices or names, and we shall refer to them as *ghost data values* (or simply *ghost values*). The extra ghost tags and values will help us simulate administrative operations of $\mathcal{L}^{\mathbf{can}}_{\mathrm{HOSC}}$ and maintain a helpful relationship between the data values, used to handle storing the heap. To sum up, we will take Σ to be a finite subset of

$$\begin{aligned}\Sigma_{\mathrm{FOSC}} &= \{\mathrm{Gopen},\mathrm{Gclose},\mathrm{Parg},\mathrm{Pcont},\mathrm{Oarg},\mathrm{Ocont}\}\\ &\quad\cup\bigcup_{A_\square\in\mathrm{Vals}_\square}\{\mathrm{Pcall}(A_\square),\mathrm{Pret}(A_\square),\mathrm{Ocall}(A_\square),\mathrm{Oret}(A_\square)\}.\end{aligned}$$

where $\mathrm{Vals}_\square$ is the set of all value patterns. Although Σ_{FOSC} is infinite (because $\mathrm{Vals}_\square$ is infinite), we will only need a finite number of value patterns, because the types used in actions are syntactic subtypes of those occurring in the boundary types of terms.

7.2 Data alphabet

We have already mentioned that we will use fixed level-1 data values to correspond to indices. We will now explain the relationships between various other names used in the encoding.

- Recall that the encoding of each P-action will be preceded by (Gopen,o). If that P-action introduces any (function or continuation) names then to model such names we will use fresh data values d such that $\mathrm{pred}(d)=o$.
- For (Gopen,o) occurring at the very beginning of a trace encoding, we require that o be fresh and a root (level 1). For further occurrences of (Gopen,o), i.e. those following encodings of O-actions, we require that o be fresh and $\mathrm{pred}(o)=\mathrm{pred}(d)$, where d represents the head name of the O-action.

Note that, because $\mathrm{pred}(d)$ is a ghost value, the second condition will generate tree structure among ghost values, and data values used to model P-names will always be leaves. Moreover, as we require $\mathrm{pred}(o)=\mathrm{pred}(d)$, the induced tree on ghost values will have the same shape as the tree of P-views that are generated by $\mathcal{L}_{\mathrm{HOSC}}$. In particular, because of Corollary 4.5, its depth will be bounded, as required in NDCMA. For a ghost value, the class memory can be used to store locations generated in the sequence of τ-transitions in the corresponding portion of the view path induced by the P-view.

The following definition summarises our conventions on the use of tags and data values in trace encodings. Recall that $\mathcal{N}$ is the bounded set of data values used to represent indices, and $\tilde{i}$ is the data value corresponding to index i. We write $\#A_\square$ to refer to the number of placeholder $\square$s in $A_\square$.

Definition 7.1. A data word $w \in (\Sigma_{\mathrm{FOSC}} \times \mathcal{D})^*$ is ***valid*** if it has one of the shapes specified below.

(1) (Gopen, o), where $o \notin \mathcal{N}$ is a level-1 data value.
(2) $u(\mathrm{Pcall}(A_\square), \tilde{i})(\mathrm{Parg}, d_1) \cdots (\mathrm{Parg}, d_{\#A_\square})(\mathrm{Pcont}, d_{\#A_\square+1})(\mathrm{Gclose}, o)$, where u is a valid data word ending in (Gopen, o), the data values d_j are pairwise distinct, do not occur in $\mathcal{N}$ or u, and $\mathrm{pred}(d_j) = o$.
(3) $u(\mathrm{Pret}(A_\square), \tilde{i})(\mathrm{Parg}, d_1) \cdots (\mathrm{Parg}, d_{\#A_\square})(\mathrm{Gclose}, o)$, where u, d_j, o satisfy analogous conditions to (2).
(4) $u(\mathrm{Ocall}(A_\square), d)(\mathrm{Oarg}, \tilde{i_1}) \cdots (\mathrm{Oarg}, \tilde{i_{\#A_\square}})(\mathrm{Ocont}, \tilde{i_{\#A_\square+1}})(\mathrm{Gopen}, o)$, where u is a valid data word ending in (Gclose, o'), the data value d occurs in u but not in $\mathcal{N}$ or as a ghost value, the indices $i_1 \cdots i_{\#A_\square+1}$ form a contiguous range, the data value o does not occur in u or $\mathcal{N}$, and $\mathrm{pred}(o) = \mathrm{pred}(d)$.
(5) $u(\mathrm{Oret}(A_\square), d)(\mathrm{Oarg}, \tilde{i_1}) \cdots (\mathrm{Oarg}, \tilde{i_{\#A_\square}})(\mathrm{Gopen}, o)$, where u, d, i_j, o satisfy analogous conditions to (4).

Example 7.2. Consider the $\{f\}$-trace $\mathtt{t} = \bar{f}(\langle g, h\rangle, c_1)\ h(k, c_2)\ \bar{k}(\mathbf{tt}, c_3)\ c_1(\langle l, m\rangle)\ \bar{m}((), c_4)\ h(j, c_5)\ \bar{j}(\mathbf{tt}, c_6)\ c_3(\widehat{3})\ \bar{c_2}(\widehat{3})$. Observe that its canonical form $\mathbf{Can}(\mathtt{t})$ is $\bar{1}(\langle g, h\rangle, c_1)\ h(2,3)\ \bar{2}(\mathbf{tt}, c_3)\ c_1(\langle 2, 3\rangle)\ \bar{3}((), c_4)\ h(2,3)\ \bar{2}(\mathbf{tt}, c_6)\ c_3(\widehat{3})\ \bar{3}(\widehat{3})$, which will be encoded by the valid data word given below

$$
\begin{aligned}
&(\mathrm{Gopen}, o_1)(\mathrm{Pcall}(\langle\square,\square\rangle), \tilde{1})(\mathrm{Parg}, d_1)(\mathrm{Parg}, d_2)(\mathrm{Pcont}, d_3)(\mathrm{Gclose}, o_1)\\
&(\mathrm{Ocall}(\square), d_2)(\mathrm{Oarg}, \tilde{2})(\mathrm{Ocont}, \tilde{3})\ (\mathrm{Gopen}, o_2)\\
&(\mathrm{Pcall}(\mathbf{tt}), \tilde{2})(\mathrm{Pcont}, d_4)(\mathrm{Gclose}, o_2)\\
&(\mathrm{Oret}(\langle\square,\square\rangle), d_3)(\mathrm{Oarg}, \tilde{2})(\mathrm{Oarg}, \tilde{3})\ (\mathrm{Gopen}, o_3)\\
&(\mathrm{Pcall}(()), \tilde{3})(\mathrm{Pcont}, d_5)(\mathrm{Gclose}, o_3)\\
&(\mathrm{Ocall}(\square), d_2)(\mathrm{Oarg}, \tilde{2})(\mathrm{Ocont}, \tilde{3})\ (\mathrm{Gopen}, o_4)\\
&(\mathrm{Pcall}(\mathbf{tt}), \tilde{2})(\mathrm{Pcont}, d_6)(\mathrm{Gclose}, o_4)\\
&(\mathrm{Oret}(\widehat{3}), d_4)\ (\mathrm{Gopen}, o_5)(\mathrm{Pret}(\widehat{3}), \tilde{3})(\mathrm{Gclose}, o_5)
\end{aligned}
$$

where we have the following relations between data values:

$$\begin{tikzcd}
\tilde{1} & \tilde{2} & \tilde{3} & & o_1 \arrow[dl, dash] \arrow[d, dash] \arrow[dr, dash] \arrow[drr, dash] \arrow[drrrr, dash] \arrow[drrrrr, dash] & & & & & \\
& & & d_1 & d_2 & d_3 & o_2 \arrow[dl, dash] \arrow[dr, dash] & & o_3 \arrow[d, dash] & o_4 \arrow[d, dash] \\
& & & & & d_4 & & o_5 & d_5 & d_6
\end{tikzcd}$$

8 FROM FOSC TO NDCMA (AUTOMATON)

In this section, we show how to define NDCMA for a given cr-free FOSC term $\Gamma \vdash M : \sigma$. Let ρ be a Γ-assignment. Our aim will be to specify an NDCMA $\mathcal{A}_M^{\rho,\sigma}$ that will accept all data words representing traces from $\mathbf{Tr}_{\mathbf{can}}(\mathrm{C}_{M,\mathbf{can}}^{\rho,\sigma})$ according to the representation scheme specified in Section 7.

Recall that we use level-1 data values from a finite subset $\mathcal{N}$ of $\mathcal{D}$ to represent the bounded indices generated by $\mathcal{L}_{\mathrm{HOSC}}^{\mathbf{can}}$. Accordingly, we shall initialise the memory function μ_0 by $\mu_0(\tilde{i}) = i$ for $\tilde{i} \in \mathcal{N}$, and $\mu_0(d) = \perp$ otherwise.

8.1 State vs class memory

Next we take a closer look at configurations of $\mathcal{L}_{\mathrm{HOSC}}^{\mathbf{can}}$ and explain which parts will be represented using state and which will be delegated to class memory. The configurations have two kinds of shapes: $\langle N, (i : \tau), \gamma, \xi, \phi, h, k, \eta\rangle$ or $\langle \gamma, \xi, \phi, h, \eta\rangle$. By Corollary 4.7, we know that the size of N will be bounded. Together with the fact that O-names in N are replaced with pairs $(i : \tau)$ and that locations will be modelled by bounded natural numbers (see next subsection), this implies that the number of possible Ns will be bounded too and, consequently, we can store N in the state of the automaton. Similarly, we use state to store $(i : \tau)$ and k. Note that $1 \le i, k \le \mathrm{MaxInd}(M)$ and τ must be a syntactic subtype of a type from Γ or σ.

In contrast, to represent γ, ξ, η, we will take advantage of the class memory function. Recall that each of the maps is defined on P-names and the set of P-names can grow unboundedly in $\mathcal{L}_{\mathrm{HOSC}}^{\mathbf{can}}$, i.e. could not be stored in finite state space. We still need to argue that the number of memories can be bounded. For γ, this is the case because the terms and contexts stored in γ are subterms of terms N discussed in the previous paragraph. For ξ and η, we observe that their images contain indices bounded by $\mathrm{MaxInd}(M)$. In ξ they are paired up with types, but these are always syntactic subtypes of those occurring in the typing judgment of the original term. Consequently, the number of memories can also be bounded.

Finally, we discuss the heap, which presents a particular challenge. To handle it, we will take advantage of a special property of FOSC. Recall that traces of GOSC terms satisfy P-visibility. In [14], this is established by showing that terms in active configurations may only contain O-names introduced in the P-view. A similar argument can be used for FOSC terms to show an analogous property for heap locations.

Lemma 8.1. *Let $\Gamma \vdash M : \sigma$ be a cr-free* FOSC *and $N \in$* Reach(M), *i.e. there exists a Γ-assignment ρ, even-length trace t and c such that $\mathrm{C}_M^{\rho,c} \xrightarrow{t} \mathrm{C}$ and N is the term component of* C. *Then all locations in N have been introduced inside the corresponding view path.*

Example 8.2. We show that Lemma 8.1 fails in GOSC, i.e. when reference names can be stored. Let $\sigma = (\mathsf{Unit} \to \mathsf{Unit}) \times (\mathsf{Unit} \to \mathsf{Int})$ and consider the GOSC term $\vdash M : \sigma$ defined below.

$$M \triangleq \mathbf{let}\, r = \mathbf{ref}_{\mathsf{RefInt}}\, (\mathbf{ref}_{\mathsf{Int}}\, \widehat{0})\ \mathbf{in}\ \langle \lambda x.r := \mathbf{ref}_{\mathsf{Int}}\, \widehat{1}, \lambda y.!(!r)\rangle.$$

Assuming $c : \sigma$, the $\{c\}$-trace

$$\bar{c}(\langle f_1, f_2\rangle)\ f_1((), c_1)\ \bar{c_1}(())\ f_2((), c_2)\ \bar{c_2}(\widehat{1})$$

is in $\mathbf{Tr}_{\mathrm{HOSC}}(\mathrm{C}_M^{\emptyset,c})$. Note that the final P-action contains $\widehat{1}$, because

- between $f_1((), c_1)$ and $\bar{c_1}(())$, $\mathcal{L}_{\mathrm{HOSC}}$ generates a location, say ℓ, initialised to 1, which is then stored inside the top-level reference r of type $\mathsf{Ref}(\mathsf{Ref}(\mathsf{Int}))$;
- after $f_2((), c_2)$, $\mathcal{L}_{\mathrm{HOSC}}$ will reach a configuration with the term component equal to $!\ell$, which will evaluate to $\widehat{1}$.

Let $t = \bar{c}(\langle f_1, f_2\rangle)\ f_1((), c_1)\ \bar{c_1}(())\ f_2((), c_2)$ and observe that

$$\mathrm{View}_{\mathrm{P}}(t) = \bar{c}(\langle f_1, f_2\rangle)\ f_2((), c_2).$$

Thus, Lemma 8.1 fails in this case, because ℓ is introduced after $f_1((), c_1)$ and before $\bar{c_1}(())$, and the transitions between $f_1((), c_1)$ and $\bar{c_1}(())$ do *not* belong to the view path when $\bar{c_2}(\widehat{1})$ is about to be emitted. Intuitively, this is because ℓ escapes its P-view through the heap, due to r, which is a reference to another reference. This could not happen in FOSC.

8.2 Distributed storage

Thanks to Lemma 8.1, in order to simulate $\mathcal{L}^{\mathbf{can}}_{\mathrm{HOSC}}$, it suffices to be able to access and update locations introduced in the current view path. By Lemma 4.8, there is a bound HeapBound(M) on the number of locations that can be introduced between O- and P-actions and, thanks to Corollary 4.5, the length of P-views is bounded by ViewLen(M), meaning that at any given time it will suffice to keep track of a bounded fragment of the heap with up to HeapSize(M) = HeapBound(M) × ViewLen(M) locations. As the fragment is bounded, it will be possible to store it in the state. Thus, instead of the whole heap, we will only be storing the subheap that was created in the view path corresponding to the current configuration. To this end, we will rely on a concrete representation of locations, namely the natural numbers. Heaps will be finite mappings from $\mathbb{N}$ to base values. Furthermore, we will only care about heaps with the property that they are contiguous (i.e. if h is defined on i and $i+2$, it is defined on $i+1$). Given this, we let max h be the largest location h is defined on (and -1 for empty h). We then also need a subtle modification of the operational semantics:

$$(K[\mathbf{ref}\,V], h) \longrightarrow (K[l], h[l \mapsto V]) \quad \text{where } l = \max h + 1.$$

Even though at any given moment we need to refer to a bounded fragment of the heap, the whole heap can grow unboundedly and we cannot hope to use the state to store it in its entirety. Instead, we will store it in a distributed fashion as a tree using the class memory function for ghost values. Recall that the representation scheme is designed in such a way that a fresh ghost value follows each O-action (tag Gopen) as well as each P-action (tag Gclose). This gives us a chance to store new parts of the heap created between the two actions as the memory for the new ghost value. Additionally, recall that ghost values are linked in such a way that they generate branches corresponding to P-views. Since NDCMA have access to memories stored along the whole branch, the automaton will be able to read and modify parts of the heap associated with the current P-view, which is exactly what we need: the parts relevant to the current P-view can be extracted from class memory, moved into the state during an O-action, modified in state as necessary to simulate τ transitions, and eventually moved back into class memory when the corresponding ghost name is being closed (Gclose). Note that the same natural numbers may be used to correspond to different locations, but only if they arise in different branches.

Notation. In order to manage heap extraction from the class memory function, we define two operations that respectively combine parts of the heap stored on the same branch into a single heap and use a single heap to update a branch. Let $\overline{h} = (h_1, \cdots, h_k)$ be a sequence of heaps with pairwise disjoint domains. Then the union of the heaps will be written Union$(\overline{h}) \triangleq h_1 \cdot h_2 \cdot \ldots \cdot h_k$. In our case, the union will always be defined for a contiguous range starting from 0. On the other hand, Update$(\overline{h}, h')$, where h' is defined on at least the domain of Union$(\overline{h})$, produces a sequence that updates $\overline{h}$ with values from h'. We write $h'_{>m}$ (resp. $h'_{\leq m}$) for the restriction of h' to locations greater than (resp. at most) m.

$$\begin{array}{lcl}
\mathrm{Update}((h_1), h') & \triangleq & (h') \\
\mathrm{Update}((\overline{h}, h_k), h') & \triangleq & (\mathrm{Update}(\overline{h}, h'_{\leq m}), h'_{>m}) \\
& & \text{where } m = \max \mathrm{Union}(\overline{h})
\end{array}$$

Before we can present the transition function of $\mathcal{A}^{\rho,\sigma}_M$, we need to introduce another two operations for handling value patterns, which will respectively *decompose* values into a pattern along with a (possibly empty) sequence of function values, and *reconstruct* an abstract value with typed indices given a pattern and a sequence of typed indices.

$$\begin{array}{lcll}
\mathbf{Decom}(V) & \triangleq & (V, \epsilon) & V : \beta,\ \beta \in \{\mathsf{Unit}, \mathsf{Int}, \mathsf{Bool}\} \\
\mathbf{Decom}(V) & \triangleq & (\square, V) & V : \tau' \to \tau'' \\
\mathbf{Decom}(\langle V_1, V_2\rangle) & \triangleq & (\langle P_1, P_2\rangle, \overline{U_1}\ \overline{U_2}) & (P_i, \overline{U_i}) = \mathbf{Decom}(V_i)
\end{array}$$

$$\begin{array}{lcll}
\mathbf{Recon}(V, \epsilon) & \triangleq & V & V : \beta,\ \beta \in \{\mathsf{Unit}, \mathsf{Int}, \mathsf{Bool}\} \\
\mathbf{Recon}(\square, (i : \tau)) & \triangleq & (i : \tau) & \tau = \tau' \to \tau'' \\
\mathbf{Recon}((\langle P_1, P_2\rangle, \overline{I_1 I_2}) & \triangleq & \langle U_1, U_2\rangle & U_i = \mathbf{Recon}(P_i, \overline{I_i}), |\overline{I_i}| = \#P_i
\end{array}$$

Finally, we write $\sigma_1 \cdots \sigma_{\#A_\square} \vdash A_\square : \sigma'$, or simply $\overline{\sigma} \vdash A_\square : \sigma'$, when $A_\square : \sigma'$ provided the ith occurrence of $\square$ is typed as σ_i.

8.3 Transition function

The transition rules are listed in Figure 5.

Recall that $\mu_0(\tilde{i}) = i$ for all $\tilde{i} \in \mathcal{N}$. As the automaton runs and class memory evolves, the memory associated with $\tilde{i}$ will not change. This will let us force a transition on i whenever needed by referring to class memory. Below we walk the reader through various groups of transitions in Figure 5 and explain their role.

Initialisation. The first step of the automaton is specified by (Init). It will generate the first ghost name (the only one at level 1) and initialise the corresponding memory to $\emptyset^\dagger$ while reading the Gopen tag. Here $\emptyset$ refers to the empty heap and $\dagger$ is a marker indicating that this ghost name is currently *active*. At most one ghost name will be active at any given time.

P-actions. (Init) leads to $(M\{\underline{\rho}\}, (|\nu(\rho)|+1 : \sigma), \emptyset, |\nu(\rho)|+2)$, which corresponds to the initial configuration of $\mathcal{L}^{\mathbf{can}}_{\mathrm{HOSC}}$. In particular $|\nu(\rho)| + 1$ is the index of the initial continuation name, $\emptyset$ denotes the empty heap, and $|\nu(\rho)| + 2$ is the next index to be used. These components will then evolve through (ϵ) steps until the reduction gets stuck. If the reason is not Ω then the rules (Pcall) or (Pret) take over and the corresponding head name $\tilde{i}$ is read without changing the memory. Note the rules use tags containing a value pattern (Pcall$(A_\square)$ and Pret$(A_\square)$ respectively) and the associated sequence of values $\overline{V}$ is placed towards the end of the target state, where $(A_\square, \overline{V}) = \mathbf{Decom}(V)$. Additionally, in (Pcall), the context K is stored at the beginning of the target state. In both cases, the target state includes a Parg flag, indicating the start of a phase in which data values corresponding to arguments will be created.

The rules (PargC) and (PargR) generate new data values corresponding to each function in the argument, immediately under the currently active ghost value, which can be detected using $\dagger$. The memories for the data values are set to (V, k), storing the corresponding function and next index, provided the memories on the branch of the currently active ghost value have the form $(h_1, \cdots, h_j^\dagger)$. Note that at the very beginning we will have $j = 1$ (and $h_j = \emptyset$), but j will grow in subsequent steps as new ghost values are being created at levels deeper than 1. Note also that the transition does not change any memories other than that of the new data value, which becomes associated with (V, k) (analogously to how γ, η are updated in $\mathcal{L}^{\mathbf{can}}_{\text{HOSC}}$). For (Pcall), once all arguments have been processed and the argument component becomes empty, the (PargCϵ) rule transitions to a state tagged Pcont. In the (Pcont) rule, a new data value is similarly created for the continuation name associated with K, and its memory is initialised to $(K, (i : \tau), k)$ (analogously to how γ, ξ, η are updated in $\mathcal{L}^{\mathbf{can}}_{\text{HOSC}}$). Once the arguments and the context ((Pcall) only) have been dealt with, the automaton proceeds to (h, Gclose) with current heap h.

The (Gclose) rule deactivates the currently active ghost value by removing † from its memory and updating the memory function on the whole branch to Update$((h_1, \cdots, h_j), h)$, where $(h_1, \cdots, h_j)$ was the old content. The automaton then moves to Ostate and is ready to process O-actions.

O-actions. Recall that O-actions contain P-names as head names and that the memories of the data values corresponding to them have been set via (PargC), (PargR) to (V, k) or (Pcont) to $(K, (i : \tau), k)$. Accordingly, when (Ocall) or (Oret) read the data values, the memories can be used to continue the simulation of $\mathcal{L}^{\mathbf{can}}_{\text{HOSC}}$ by recovering the relevant term/context. The abstract value (with typed indices) $U = \mathbf{Recon}(A_\square, \overline{(i : \sigma_i)})$ is then synthesised from the associated value pattern $A_\square$ by filling $\square$s with typed indices from the range $\bar{i} = k..k' - 1$, where $k' = k + \#A_\square$. In (Ocall), k' then becomes the new continuation index, and $k' + 1$ is the next index to be used in the future. In (Oret), the next index is k', because there is no need to create a new continuation name. Following (Ocall), the arguments and continuation names (if applicable) are then announced with the rules (OargC), (OargCϵ) and (Ocont) by using the corresponding data values $\tilde{i}$. Similarly, after (Oret), rules (OargR), (OargRϵ) are used. Ultimately, this will take the automaton to a state whose last component is Gopen.

Recall that, when a data value corresponding to a P-name is created in (Pcall) or (Pret) it will have the *then* active ghost value as a parent. Consequently, when the data value for the P-name is read later, the automaton will also have access to the memory of the ghost name. The rules (Ocall) and (Oret) take this opportunity to activate the ghost value temporarily, by adding † to its memory. Once the automaton reaches the state marked with Gopen, it will create a new ghost name under the ghost name that was temporarily marked, in accordance with our convention for representing traces with data words. The old ghost value will be deactivated and the new one will become active. Indeed, note that in (Gopen) the memory on the corresponding branch will change from $(h_1, \cdots, h_j^\dagger, \perp)$ to $(h_1, \cdots, h_j, \emptyset^\dagger)$, i.e. † is passed to the new data value and the corresponding heap is initialised to $\emptyset$, as in (Init).

Acceptance. Finally, it suffices if the automaton accepts each time Gopen or Gclose is processed, as these mark the stages when action encodings have been completed. Overall, the construction yields a faithful simulator of $\mathcal{L}^{\mathbf{can}}_{\text{HOSC}}$.

LEMMA 8.3. *Let $\Gamma \vdash M : \sigma$ be a cr-free* FOSC*-term and ρ a Γ-assignment. Then $\mathcal{L}(\mathcal{A}^{\rho,\sigma}_M)$ consists of all data-word representations of elements of $\mathbf{Tr}_{\mathbf{can}}(\mathrm{C}^{\rho,\sigma}_{M,\mathbf{can}})$ according to the representation scheme from Section 7.*

Since there are only finitely many Γ-assignments up to name-equality, Lemma 8.3 (combined with Theorems 3.9, 5.12, 6.4) implies the following result.

THEOREM 8.4. *$\lesssim$ and (thus) $\simeq$ are decidable for cr-free* FOSC *terms.*

9 CONCLUSION

Theorem 8.4 (HOSC[FOSC]) should be contrasted with the undecidability result from Theorem 3.11 (HOSC[GOS]). This highlights how the availability of reference name storage can have a dramatic effect on contextual equivalence, even though the properties of traces remain the same (P-visibility). An interesting question to explore is whether Theorem 8.4 could be extended to programs with loops. Here our approach runs into problems: P-views are no longer unbounded and our trace encodings are no longer finitary. In particular, there is no bound on the number of indices or the level of data values. One could try to optimise the representation scheme somehow, e.g. by resetting the level after each iteration. However, this carries the risk of revealing the location of loops within the program, which cannot be reconciled with full abstraction - some programs with loops (e.g. bounded loops) may be equivalent to loop-free programs.

Another avenue for future research concerns equivalence with respect to contexts drawn from a subset of HOSC. Here the main question would be whether Theorem 8.4 still holds when only references to base types are allowed, i.e. FOSC[FOSC], or when continuations are omitted, let us call it the HOS[FOSC] case. We do not know the answers to these questions, though in the latter case we suspect that exploring the pushdown extension of class memory automata from [7] might be fruitful, given that traces corresponding to contexts without continuations obey a bracketing condition [14]. Still, even if this were the case, the status of the corresponding automata-theoretic problem is currently unknown and related to a long-standing open problem about branching vector addition systems.

However, we do know that when both continuations and higher-order references are forbidden in contexts, we obtain undecidability regardless of whether we allow references to names (GOS[GOS] [22]) or not (FOS[FOS] [21]). This illustrates the phenomenon that equivalence with respect to “simpler” contexts may be “harder”. Semantically, this is confirmed by the extra conditions of traces that need to be imposed to model restrictions of the HOSC case [14]. So, Theorem 8.4 could actually turn out to be a singular decidability result in this setting.

ACKNOWLEDGMENTS

This research was funded by EPSRC EP/T006579 and EPSRC Studentship 2742896.

(Init)	Initial	$\xrightarrow[\emptyset^\dagger]{\text{Gopen},\bot}$	$(M\{\underline{\rho}\}, (\lvert\nu(\rho)\rvert + 1 : \sigma), \emptyset, \lvert\nu(\rho)\rvert + 2)$	
(ϵ)	$(N, (i : \tau), h, k)$	$\xrightarrow{\epsilon}$	$(N', (i' : \tau'), h', k)$	when $(N, (i : \tau), h) \longrightarrow (N', (i' : \tau'), h')$
(Pcall)	$(K[(i : \tau)\ V], (j : \tau'), h, k)$	$\xrightarrow[i]{\text{Pcall}(A_\square),i}$	$(K, (j : \tau'), h, k, \overline{V}, \text{Parg})$	when $(A_\square, \overline{V}) = \mathbf{Decom}(V)$
(PargC)	$(K, (i : \tau), h, k, V\,\overline{V}, \text{Parg})$	$\xrightarrow[(h_1,\cdots,h_j^\dagger,(V,k))]{\text{Parg},(h_1,\cdots,h_j^\dagger,\bot)}$	$(K, (i : \tau), h, k, \overline{V}, \text{Parg})$	for $1 \le j \le B$
(PargCϵ)	$(K, (i : \tau), h, k, \epsilon, \text{Parg})$	$\xrightarrow{\epsilon}$	$(K, (i : \tau), h, k, \text{Pcont})$	for $1 \le j \le B$
(Pcont)	$(K, (i : \tau), h, k, \text{Pcont})$	$\xrightarrow[(h_1,\cdots,h_j^\dagger,(K,(i:\tau),k))]{\text{Pcont},(h_1,\cdots,h_j^\dagger,\bot)}$	(h, Gclose)	for $1 \le j \le B$
(Pret)	$(V, (i : \tau), h, k)$	$\xrightarrow[i]{\text{Pret}(A_\square),i}$	$(\overline{V}, h, k, \text{Parg})$	when $(A_\square, \overline{V}) = \mathbf{Decom}(V)$
(PargR)	$(V\,\overline{V}, h, k, \text{Parg})$	$\xrightarrow[(h_1,\cdots,h_j^\dagger,(V,k))]{\text{Parg},(h_1,\cdots,h_j^\dagger,\bot)}$	$(\overline{V}, h, k, \text{Parg})$	for $1 \le j \le B$
(PargRϵ)	$(\epsilon, h, k, \text{Parg})$	$\xrightarrow{\epsilon}$	(h, Gclose)	for $1 \le j \le B$
(Gclose)	(h, Gclose)	$\xrightarrow[(h'_1,\cdots,h'_j)]{\text{Gclose},(h_1,\cdots,h_j^\dagger)}$	Ostate	for $1 \le j \le B$
	when $(h'_1, \cdots, h'_j) = \text{Update}((h_1, \cdots, h_j), h)$			
(Ocall)	Ostate	$\xrightarrow[(h_1,\cdots,h_j^\dagger,(V,k))]{\text{Ocall}(A_\square),(h_1,\cdots,h_j,(V,k))}$	$(V\,U, (k' : \sigma''), h, \bar{i}, k', k' + 1, \text{Oarg})$	for $1 \le j \le B$
	when $V : \sigma' \longrightarrow \sigma''$, $\overline{\sigma} \vdash A_\square : \sigma'$, $\bar{i} = k \ldots k' - 1$, $k' = k + \#A_\square$, $U = \mathbf{Recon}(A_\square, \overline{(i : \sigma_i)})$, $h = \text{Union}((h_1, \cdots, h_j))$			
(OargC)	$(N, (k : \tau), h, i\,\bar{i}, k, k + 1, \text{Oarg})$	$\xrightarrow[i]{\text{Oarg},i}$	$(N, (k : \tau), h, \bar{i}, k, k + 1, \text{Oarg})$	
(OargCϵ)	$(N, (k : \tau), h, \epsilon, k, k + 1, \text{Oarg})$	$\xrightarrow{\epsilon}$	$(N, (k : \tau), h, k, k + 1, \text{Ocont})$	
(Ocont)	$(N, (k : \tau), h, k, k + 1, \text{Ocont})$	$\xrightarrow[k]{\text{Ocont},k}$	$(N, (k : \tau), h, k + 1, \text{Gopen})$	
(Oret)	Ostate	$\xrightarrow[(h_1,\cdots,h_j^\dagger,(K,(i:\tau),k))]{\text{Oret}(A_\square),(h_1,\cdots,h_j,(K,(i:\tau),k))}$	$(K[U], (i : \tau), h, \bar{i}, k', \text{Oarg})$	for $1 \le j \le B$
	$K[\sigma']$, $\overline{\sigma} \vdash A_\square : \sigma'$, $\bar{i} = k \ldots k' - 1$, $k' = k + \#A_\square$, $U = \mathbf{Recon}(A_\square, \overline{(i : \sigma_i)})$, $h = \text{Union}((h_1, \cdots, h_j))$			
(OargR)	$(N, (k : \tau), h, i\,\bar{i}, k', \text{Oarg})$	$\xrightarrow[i]{\text{Oarg},\ i}$	$(N, (k : \tau), h, \bar{i}, k', \text{Oarg})$	
(OargRϵ)	$(N, (k : \tau), h, \epsilon, k', \text{Oarg})$	$\xrightarrow{\epsilon}$	$(N, (k : \tau), h, k', \text{Gopen})$	
(Gopen)	$(N, (k : \tau), h, k', \text{Gopen})$	$\xrightarrow[(h_1,\cdots,h_j,\emptyset^\dagger)]{\text{Gopen},(h_1,\cdots,h_j^\dagger,\bot)}$	$(N, (k : \tau), h, k')$	for $1 \le j \le B$

$B = \lceil \text{ViewLen}(M)/2 \rceil + 1$ bounds the number of heaps appearing on a branch.

Figure 5: NDCMA transition rules for term M

REFERENCES

[1] Amal Ahmed, Derek Dreyer, and Andreas Rossberg. 2009. State-dependent representation independence. In *Proceedings of the 36th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2009, Savannah, GA, USA, January 21-23, 2009*, Zhong Shao and Benjamin C. Pierce (Eds.). ACM, 340–353. https://doi.org/10.1145/1480881.1480925

[2] Henrik Björklund and Mikołaj Bojańczyk. 2007. Shuffle Expressions and Words with Nested Data. In *Proceedings of MFCS (Lecture Notes in Computer Science, Vol. 4708)*. Springer, 750–761. https://doi.org/10.1007/978-3-540-74456-6_66

[3] Henrik Björklund and Thomas Schwentick. 2010. On notions of regularity for data languages. *Theor. Comput. Sci.* 411, 4-5 (2010), 702–715. https://doi.org/10.1016/j.tcs.2009.10.009

[4] Benedict Bunting and Andrzej S. Murawski. 2023. Operational Algorithmic Game Semantics. In *2023 38th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*. 1–13. https://doi.org/10.1109/LICS56636.2023.10175791

[5] Conrad Cotton-Barratt, David Hopkins, Andrzej S. Murawski, and C.-H. Luke Ong. 2015. Fragments of ML Decidable by Nested Data Class Memory Automata. In *Foundations of Software Science and Computation Structures - 18th International Conference, FoSSaCS 2015, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2015, London, UK, April 11-18, 2015. Proceedings (Lecture Notes in Computer Science, Vol. 9034)*, Andrew M. Pitts (Ed.). Springer, 249–263. https://doi.org/10.1007/978-3-662-46678-0_16

[6] Conrad Cotton-Barratt, Andrzej S. Murawski, and C.-H. Luke Ong. 2015. Weak and Nested Class Memory Automata. In *Language and Automata Theory and Applications*. Springer International Publishing, 188–199. https://doi.org/10.1007/978-3-319-15579-1_14

[7] Conrad Cotton-Barratt, Andrzej S. Murawski, and C.-H. Luke Ong. 2019. ML, Visibly Pushdown Class Memory Automata, and Extended Branching Vector Addition Systems with States. *ACM Trans. Program. Lang. Syst.* 41, 2 (2019), 11:1–11:38. https://doi.org/10.1145/3310338

[8] Normann Decker, Peter Habermehl, Martin Leucker, and Daniel Thoma. 2014. Ordered Navigation on Multi-attributed Data Words. In *Proceedings of CONCUR (Lecture Notes in Computer Science, Vol. 8704)*, Paolo Baldan and Daniele Gorla (Eds.). Springer, 497–511. https://doi.org/10.1007/978-3-662-44584-6_34

[9] Derek Dreyer, Georg Neis, and Lars Birkedal. 2012. The impact of higher-order state and control effects on local relational reasoning. *J. Funct. Program.* 22, 4-5 (2012), 477–528. https://doi.org/10.1017/S095679681200024X

[10] Alain Finkel and Philippe Schnoebelen. 2001. Well-structured transition systems everywhere! *Theor. Comput. Sci.* 256, 1-2 (2001), 63–92. https://doi.org/10.1016/S0304-3975(00)00102-X

[11] Daniel Hirschkoff, Guilhem Jaber, and Enguerrand Prebet. 2023. Deciding Contextual Equivalence of ν-Calculus with Effectful Contexts. In *Foundations of Software Science and Computation Structures - 26th International Conference, FoSSaCS 2023, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2023, Paris, France, April 22-27, 2023, Proceedings (Lecture Notes in Computer Science, Vol. 13992)*, Orna Kupferman and Paweł Sobociński (Eds.). Springer, 24–45. https://doi.org/10.1007/978-3-031-30829-1_2

[12] Martin Hyland and C.-H. Luke Ong. 2000. On Full Abstraction for PCF: I, II, and III. *Inf. Comput.* 163, 2 (2000), 285–408. https://doi.org/10.1006/inco.2000.2917

[13] Guilhem Jaber. 2019. SyTeCi: automating contextual equivalence for higher-order programs with references. *Proceedings of the ACM on Programming Languages* 4, POPL (dec 2019), 1–28. https://doi.org/10.1145/3371127

[14] Guilhem Jaber and Andrzej S. Murawski. 2021. Complete trace models of state and control. In *Proceedings of ESOP*. Springer International Publishing, 348–374. https://doi.org/10.1007/978-3-030-72019-3_13

[15] Guilhem Jaber and Andrzej S. Murawski. 2021. Complete trace models of state and control (full version). (Jan. 2021). https://hal.science/hal-03116698 Preprint.

[16] Guilhem Jaber and Andrzej S. Murawski. 2021. Compositional relational reasoning via operational game semantics. In *36th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2021, Rome, Italy, June 29 - July 2, 2021*. IEEE, 1–13. https://doi.org/10.1109/LICS52264.2021.9470524

[17] Neil D. Jones and Steven S. Muchnick. 1978. The Complexity of Finite Memory Programs with Recursion. *J. ACM* 25, 2 (1978), 312–321. https://doi.org/10.1145/322063.322074

[18] Ralph Loader. 1995. Normalisation by Calculation. (1995). https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.36.8157 Unpublished.

[19] Ralph Loader. 1998. *Notes on simply typed lambda calculus.* Technical Report ECS-LFCS-98-381. Laboratory for Foundations of Computer Science, University of Edinburgh. https://www.lfcs.inf.ed.ac.uk/reports/98/ECS-LFCS-98-381/

[20] Andrzej S. Murawski. 2003. On Program Equivalence in Languages with Ground-Type References. In *18th IEEE Symposium on Logic in Computer Science (LICS 2003), 22-25 June 2003, Ottawa, Canada, Proceedings*. IEEE Computer Society, 108. https://doi.org/10.1109/LICS.2003.1210050

[21] Andrzej S. Murawski. 2005. Functions with local state: Regularity and undecidability. *Theor. Comput. Sci.* 338, 1-3 (2005), 315–349. https://doi.org/10.1016/j.tcs.2004.12.036

[22] Andrzej S. Murawski and Nikos Tzevelekos. 2017. Algorithmic games for full ground references. *Formal Methods in System Design* 52, 3 (aug 2017), 277–314. https://doi.org/10.1007/s10703-017-0292-9

[23] Andrew M. Pitts and Ian Stark. 1999. *Operational Reasoning for Functions with Local State.* Cambridge University Press, USA, 227–274.

[24] Philippe Schnoebelen. 2002. Verifying lossy channel systems has nonprimitive recursive complexity. *Inf. Process. Lett.* 83, 5 (2002), 251–261. https://doi.org/10.1016/S0020-0190(01)00337-4

[25] Kristian Støvring and Søren B. Lassen. 2007. A complete, co-inductive syntactic theory of sequential control and state. In *Proceedings of the 34th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2007, Nice, France, January 17-19, 2007*, Martin Hofmann and Matthias Felleisen (Eds.). ACM, 161–172. https://doi.org/10.1145/1190216.1190244