Operational Algorithmic Game Semantics

Benedict Bunting
University of Oxford, UK

Andrzej S. Murawski
University of Oxford, UK

***Abstract*—We consider a simply-typed call-by-push-value calculus with state, and provide a fully abstract trace model via a labelled transition system (LTS) in the spirit of operational game semantics. By examining the shape of configurations and performing a series of natural optimisation steps based on name recycling, we identify a fragment for which the LTS can be recast as a deterministic visibly pushdown automaton. This implies decidability of contextual equivalence for the fragment identified and solvability in exponential time for terms in canonical form. We also identify a fragment for which these automata are finite-state machines.**

Further, we use the trace model to prove that translations of prototypical call-by-name (IA) and call-by-value (RML) languages into our call-by-push-value language are fully abstract. This allows our decidability results to be seen as subsuming several results from the literature for IA and RML. We regard our operational approach as a simpler and more intuitive way of deriving such results. The techniques we rely on draw upon simple intuitions from operational semantics and the resultant automata retain operational style, capturing the dynamics of the underlying language.

I. Introduction

The notion of contextual equivalence has been much studied in programming language theory, as it captures an especially natural form of equality between programs, widely applicable in verification. This has spurred efforts to produce denotational models in which equality in the model coincides exactly with contextual equivalence. These 'fully abstract' models represent the 'gold standard'. One particularly successful approach to this problem has arisen through *game semantics* [1], [2], [3], where a term is modelled as a strategy for a game of question and answer played between the term and its context. Over the past 25 years, game semantics has proven to be a powerful tool, providing fully abstract models for languages with a variety of features. A more recent development is *operational game semantics* [4], [5], in which strategies are represented as sets of traces generated by a labelled transition system (LTS) derived from the operational semantics of the language. This approach gives an intuitive and concrete representation of strategies, amenable to the application of operational techniques, as opposed to the abstract properties emphasised in earlier work. We summarise the main contributions of this paper below.

Firstly, we define a Call-By-Push-Value (CBPV) language equipped with dynamically generated first-order store. It can be seen as a canonical language for studying first-order state with arbitrary evaluation order.

Secondly, we present an operational game model for this language, and prove that it is fully abstract with respect to contextual equivalence. This means that terms have the same set of complete traces if and only if they are contextually equivalent.

We provide translations from the prototypical languages RML [6] and IA [7] into CBPV, and using our model, prove that these translations are fully abstract. This validates the assertion of CBPV subsuming CBV and CBN in this setting, and justifies studying CBPV as a vehicle for investigating first-order store.

Our final result is to identify two fragments of CBPV where contextual equivalence is decidable, and derive a decision procedure for these fragments. We achieve this by using operational intuitions to refine the corresponding LTS so that it produces traces over a finite alphabet. In particular, we identify a fragment for which the refined LTS can be viewed as a deterministic visibly pushdown automaton (VPA), and a smaller fragment where it is a deterministic finite automaton (DFA). The CBPV language is particularly suited to carrying out this transformation, as it provides a notion of thunks, which turns out to offer the appropriate level of abstraction to discuss when a passage to VPA and DFA should be possible.

The results follow the spirit of algorithmic game semantics by modelling strategies as automata [8], though our automata are more intuitive thanks to their close relation to the operational semantics. Because of the translations into RML and IA, our results unify and provide new proofs for several results from this school [9], [10], [11], [12]. At the technical level, we rely on purely operational techniques, and our work should be accessible to readers without a game semantics background. Overall, our approach represents a simpler way to obtain such decidability results and provides an operational understanding of why they hold.

Other Related Work: Typed Normal Form Bisimulations [13] are arguably closest to our setting. They were developed for Jump-With-Argument, a continuation passing-style version of CBPV without state. Although not made explicit, the transition system these bisimulations are defined on bears similarity to operational game semantics (OGS).

CBPV is already known to accommodate fully abstract translations from CBN and CBV, but their full abstraction in [14] was established in a different setting from ours (with printing as the side-effect). Consequently, we had to develop

This research was funded in whole or in part by EPSRC EP/T006579 and EPSRC Studentship 2742896.

979-8-3503-3587-3/23/$31.00

a separate argument.

We believe ours is the first work seeking to give decision procedures for contextual equivalence by refining operational game models into automata. However, OGS has been used as a foundation for other techniques, particularly bisimulation-based. An example of this strand of work is the equivalence checker SyTeCi [15] for automatically proving contextual equivalences with respect to contexts with general references. It is based on Symbolic Kripke Open Relations, which can be seen as an abstraction of OGS. The associated decidability result is incomparable to ours: it uses more powerful contexts and disallows reference creation inside functions, albeit without a type restriction. HOBBIT is a bounded-complete inequivalence checker, which exploits 'symbolic up-to' techniques to allow it to (semi-) automatically prove many equivalences [16]. Kripke Normal Form Bisimulations [17] are a sound and complete technique for showing contextual equivalence for a family of CBV languages with control and higher-order state, but the cited paper does not discuss decidability. Apart from OGS-inspired research, there has been much related work based on logical relations [18], [19] and normal form bisimulations [20], also without decision procedures.

II. The language

1) Syntax: The language being studied in this paper extends Levy's Call-by-Push-Value λ-calculus [21] with mutable state and basic data types. We will deal with the finitary fragment of the language, lacking infinite types and recursion. The types and terms are shown in Figure 1. The types are stratified into *values* (generated by σ) and *computations* (generated by $\underline{\tau}$). The slogan often associated with this division is 'A value is, a computation does'.

A ***typing judgement*** gives a type to a term given the types of locations and variables. In CBPV, we have separate judgments for value types and computation types. For values we write $\Sigma; \Gamma \vdash^v V : \sigma$ and for computations we write $\Sigma; \Gamma \vdash^v M : \underline{\tau}$, where Γ is a finite partial function that assigns types to variables, and Σ is a list of locations. We abbreviate $\emptyset; \Gamma \vdash M : \tau$ to $\Gamma \vdash M : \tau$ and $\Sigma; \emptyset \vdash M : \tau$ to $\Sigma \vdash M : \tau$. Typing judgements are derived using the rules in Figure 2.

We treat the type Ref as if it were a pair of thunks: a read thunk of type $U F \mathrm{Int}$ and a write thunk of type $U(\mathrm{Int} \to F\mathrm{Unit})$. This is an old idea, due to Reynolds [22], and is a feature of many game models of languages with references. As a consequence, we need the construct MkVar, which embeds a pair of thunks into the Ref type, allowing 'bad variables', which possibly return different values from those last written. When reading ($!V$) or writing ($V := U$) such a bad variable, we use the appropriate thunk. We include bad variables for comparability with existing results, although they can be avoided by incorporating parts of the heap into the trace semantics [4]. For space reasons, we will sometimes write $\{\!|V_1, V_2|\!\}$ for $\mathrm{MkVar}\ V_1\ V_2$.

2) The operational semantics: We present the operational semantics as the reduction relation $\to$ in Figure 3 on configurations, which are pairs of a computation and a heap h (a mapping of locations to values). We write $h : \Sigma$ for $\Sigma \subseteq \mathrm{dom}(h)$. $h[\ell \mapsto V]$ denotes updating ℓ in h.

3) Contextual equivalence: Contexts C can be seen as computations with a hole, $\bullet$, into which another computation may be substituted. We can also type a context, by saying $\Sigma; \Gamma \vdash^k C : \underline{\tau} \implies \underline{\tau'} :$ if $\Sigma; \Gamma, x : \underline{\tau} \vdash^c C[x] : \underline{\tau'} :$ for a fresh x. One key notion of terms being 'equal' is contextual equivalence, which is formalised in terms of termination. A ***terminal*** is a (closed) computation of the form $\mathrm{return}\ V$ or $\lambda x^{\sigma}.M$. Termination means that a term reduces to a terminal: we write $(M, h) \Downarrow_{ter}$ if there exist N, h' such that $(M, h) \to^* (N, h')$ and N is a terminal.

Definition 1. Given computations $\Gamma \vdash^c M_1, M_2 : \underline{\tau}$, we define $\Gamma \vdash^c M_1 \lesssim_{ter}^{\mathrm{CBPV}} M_2$ to hold, when for all contexts $\vdash^k C : \underline{\tau} \implies F\sigma$, we have $(C[M_1], \emptyset) \Downarrow_{ter}$ implies $(C[M_2], \emptyset) \Downarrow_{ter}$. We write $\cong_{ter}^{\mathrm{CBPV}}$ for the equivalence induced by $\lesssim_{ter}^{\mathrm{CBPV}}$.

A standard result is that contexts considered for contextual approximation can be restricted to evaluation contexts after instantiating the free variables of computations to closed values (*closed instances of use*, CIU). We write $\Sigma, \Gamma' \vdash \gamma : \Gamma$ for substitutions γ such that, for any $(x, \sigma_x) \in \Gamma$, the value $\gamma(x)$ satisfies $\Sigma; \Gamma' \vdash^v \gamma(x) : \sigma_x$. Then $M\{\gamma\}$ stands for the outcome of applying γ to M.

Definition 2 (CIU Approximation)**.** Let $\Gamma \vdash^c M_1, M_2 : F\sigma$ be CBPV computations. Then $\Gamma \vdash^c M_1 \lesssim_{ter}^{\mathrm{CBPV}(ciu)} M_2$, when for all Σ, h, K, γ, such that $h : \Sigma$, $\Sigma \vdash^k K : \underline{\tau} \implies F\sigma$, and $\Sigma \vdash \gamma : \Gamma$, we have $(K[M_1\{\gamma\}], h) \Downarrow_{ter}$ implies $(K[M_2\{\gamma\}], h) \Downarrow_{ter}$.

We obtain a CIU Lemma establishing the sufficiency of CIU testing following the framework of [23].

Lemma 3 (CIU Lemma)**.** *We have* $\Gamma \vdash^c M_1 \lesssim_{ter}^{\mathrm{CBPV}} M_2$ *iff* $\Gamma \vdash^c M_1 \lesssim_{ter}^{\mathrm{CBPV}(ciu)} M_2$.

We make the observation that the only contexts we really need to consider are those of type $F\sigma \implies F\sigma'$.

Lemma 4. *Let* $\Gamma \vdash^c M_1, M_2 : \sigma_1 \to \cdots \to \sigma_k \to F\sigma$. *Then* $\Gamma \vdash^c M_1 \lesssim_{ter}^{\mathrm{CBPV}} M_2$ *iff* $\Gamma, (x_1, \sigma_1), \cdots, (x_k, \sigma_k) \vdash^c M_1\, x_1 \cdots x_k \lesssim_{ter}^{\mathrm{CBPV}} M_2\, x_1 \cdots x_k$.

III. Labelled Transition System

We develop a labelled transition system (LTS) to capture the semantics of terms in the style of [24]. Its traces can be thought of as exchanges of moves between two players, representing the context and the term respectively. This way of modelling contextual interactions is often called *operational game semantics*.

1) Names and Abstract Values: In actions of this game, players pass (fresh) names to represent thunks passed between the two players. As these represent thunks, the names have a type $U\underline{\tau}$. In keeping with the Reynolds approach of embedding references as a pair of thunks, we will 'decompose' references into separate thunk names for reading and writing respectively.

Value Type	σ	$\triangleq$	$U\underline{\tau} \mid \text{Unit} \mid \text{Int} \mid \text{Ref}$	Computation Type	$\underline{\tau}$	$\triangleq$	$F\sigma \mid \sigma \to \underline{\tau}$
Value Term	V	$\triangleq$	$\text{thunk } M \mid x \mid () \mid \widehat{n} \mid \ell \mid \text{MkVar } V\ V$				
Computation Term	M	$\triangleq$	$\text{force } V \mid \text{return } V \mid \lambda x^\sigma.M \mid \text{let } x \text{ be } V.M \mid M \text{ to } x.M$ $\mid \text{case } V \text{ of } (M_i)_{i\in I} \mid MV \mid \text{ref } V \mid !V \mid V := V \mid \text{while } M \text{ do } M$				
Evaluation Context	K	$\triangleq$	$\bullet \mid K \text{ to } x.M \mid KV$				
Value Context	V_C	$\triangleq$	$\text{thunk } C \mid \text{MkVar } V_C\ V \mid \text{MkVar } V\ V_C$				
Context	C	$\triangleq$	$\bullet \mid \text{force } V_C \mid \text{return } V_C \mid \lambda x^\sigma.C \mid \text{let } x \text{ be } V_C.M \mid \text{let } x \text{ be } V.C \mid C \text{ to } x.M$ $\mid M \text{ to } x.C \mid \text{case } V \text{ of } (M_i)_{i<j}, C, (M_i)_{j<i} \mid CV \mid MV_C \mid !V_C \mid V_C := V$ $\mid \text{while } C \text{ do } M \mid \text{while } M \text{ do } C$				

Notational conventions: $x, y \in \mathbf{Var}$, $\ell \in \mathbf{Loc}$, $I = \{0, \cdots, \max\}$, $n \in I$
Syntactic sugar: If x does not occur free in N, we write $M; N$ for M to $x.N$, and Ω for $\text{while } (\text{return } \widehat{1}) \text{ do } (\text{return } ())$.
We write $V_1 + V_2$ for $\text{case } V_1 \text{ of } (\text{case } V_2 \text{ of } (\widehat{i+j})_{j\in I})_{i\in I}$

Fig. 1. CBPV syntax

$$\frac{}{\Sigma;\Gamma \vdash^v () : \text{Unit}} \qquad \frac{n \in \{0, \cdots, \max\}}{\Sigma;\Gamma \vdash^v \widehat{n} : \text{Int}} \qquad \frac{(x,\sigma) \in \Gamma}{\Sigma;\Gamma \vdash^v x : \sigma} \qquad \frac{\ell \in \Sigma}{\Sigma;\Gamma \vdash^v \ell : \text{Ref}} \qquad \frac{\Sigma;\Gamma \vdash^c M : \underline{\tau}}{\Sigma;\Gamma \vdash^v \text{thunk } M : U\underline{\tau}}$$

$$\frac{\Sigma;\Gamma \vdash^v V : \sigma}{\Sigma;\Gamma \vdash^c \text{return } V : F\sigma} \qquad \frac{\Sigma;\Gamma \vdash^v V : U\underline{\tau}}{\Sigma;\Gamma \vdash^c \text{force } V : \underline{\tau}} \qquad \frac{\Sigma;\Gamma \vdash^v V : \sigma \qquad \Sigma;\Gamma, x : \sigma \vdash^c M : \underline{\tau}}{\Sigma;\Gamma \vdash^c \text{let } x \text{ be } V.M : \underline{\tau}}$$

$$\frac{\Sigma;\Gamma \vdash^v V : \text{Int} \qquad \Sigma;\Gamma \vdash^c M_i : \underline{\tau}}{\Sigma;\Gamma \vdash^c \text{case } V \text{ of } (M_i)_{i\in I} : \underline{\tau}} \qquad \frac{\Sigma;\Gamma \vdash^c M : F\sigma \qquad \Sigma;\Gamma, x : \sigma \vdash^c N : \underline{\tau}}{\Sigma;\Gamma \vdash^c M \text{ to } x.N : \underline{\tau}} \qquad \frac{\Sigma;\Gamma, x : \sigma \vdash^c M : \underline{\tau}}{\Sigma;\Gamma \vdash^c \lambda x^\sigma.M : \sigma \to \underline{\tau}}$$

$$\frac{\Sigma;\Gamma \vdash^c M : \sigma \to \underline{\tau} \qquad \Sigma;\Gamma \vdash^v V : \sigma}{\Sigma;\Gamma \vdash^c MV : \underline{\tau}} \qquad \frac{\Sigma;\Gamma \vdash^v V : \text{Int}}{\Sigma;\Gamma \vdash^c \text{ref } V : F\text{Ref}}$$

$$\frac{\Sigma;\Gamma \vdash^v V_{\text{read}} : UF\text{Int} \qquad \Sigma;\Gamma \vdash^v V_{\text{write}} : U(\text{Int} \to F\text{Unit})}{\Sigma;\Gamma \vdash^v \text{MkVar } V_{\text{read}}\ V_{\text{write}} : \text{Ref}} \qquad \frac{\Sigma;\Gamma \vdash^v V : \text{Ref}}{\Sigma;\Gamma \vdash^c !V : F\text{Int}}$$

$$\frac{\Sigma;\Gamma \vdash^v V : \text{Ref} \qquad \Sigma;\Gamma \vdash^v U : \text{Int}}{\Sigma;\Gamma \vdash^c V := U : F\text{Unit}} \qquad \frac{\Sigma;\Gamma \vdash^c M : F\text{Int} \qquad \Sigma;\Gamma \vdash^c N : F\text{Unit}}{\Sigma;\Gamma \vdash^c \text{while } M \text{ do } N : F\text{Unit}}$$

Fig. 2. CBPV typing rules

$$\begin{array}{llll}
(K[\text{let } x \text{ be } V.M], h) \to (K[M\{V/x\}], h) & (K[\text{ref } V], h) \to (K[\text{return } \ell], h \cdot [\ell \mapsto V]) \\
(K[(\lambda x^\sigma.M)V], h) \to (K[M\{V/x\}], h) & (K[!\ell], h) \to (K[\text{return } h(\ell)], h) \\
(K[\text{case } i \text{ of } (M_i)], h) \to (K[M_i], h) & (K[\ell := V], h) \to (K[\text{return } ()], h[\ell \mapsto V]) \\
(K[\text{force thunk } M], h) \to (K[M], h) & (K[!(\text{MkVar } V_r\ V_w)], h) \to (K[\text{force } V_r], h) \\
(K[\text{return } V \text{ to } x.M], h) \to (K[M\{V/x\}], h) & (K[(\text{MkVar } V_r\ V_w) := U], h) \to (K[(\text{force } V_w)U], h) \\
(K[\text{while } M \text{ do } N], h) \to (K[M \text{ to } x.\text{case } x \text{ of return } (), (N \text{ to } y.\text{while } M \text{ do } N)_{i>0}], h) \qquad x, y \text{ are fresh}
\end{array}$$

Fig. 3. Operational semantics for CBPV

We will also use continuation names, to identify the question move being answered in an answer move. This is simply a technical convenience, as in the setting without control operators, all continuation names can be reconstructed due to the bracketing condition.

Definition 5. Let $\text{TNames} = \biguplus_{\underline{\tau}} \text{TNames}_{U\underline{\tau}}$ be the set of ***thunk names***, partitioned into mutually disjoint countably infinite sets $\text{TNames}_{U\underline{\tau}}$. We use f, g to range over TNames, and write $f : U\underline{\tau}$ for $f \in \text{TNames}_{U\underline{\tau}}$. Analogously, let $\text{CNames} = \biguplus_{\sigma} \text{CNames}_\sigma$ be the set of ***continuation names***. c ranges over CNames, and $c : \sigma$ denotes $c \in \text{CNames}_\sigma$. We assume CNames, TNames are disjoint and let $\text{Names} = \text{TNames} \uplus \text{CNames}$. 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 will take actions which consist of a name applied to some (sequence of) values. To handle passing thunks, we will use ***abstract values***. These are values with occurrences of thunks replaced by names, so are generated by the grammar:

$$A \triangleq f \mid () \mid \widehat{n} \mid \text{MkVar } A\ A$$

As names are intrinsically typed, abstract values can be typed in the obvious way, denoted $A : \sigma$. Given a value $V : \sigma$, $\mathbf{AVal}_\sigma(V)$ is the set of pairs (A, γ) such that A is an abstract value and $\gamma : \nu(A) \to \mathrm{Vals}$ is a substitution (defined in Figure 4). It is not quite the case that $A\{\gamma\} = V$ due to the treatment of locations. However, it is the case that $M\{V/x\} \Downarrow_{ter}$ iff $M\{A\{\gamma\}/x\} \Downarrow_{ter}$.

Remark 6. Note that $\cdot$ implicitly requires that function domains be disjoint, and $\uplus$ means the argument sets are disjoint.

Unlike in CBV, it does not suffice to consider passing only single abstract values. In CBPV, question actions occur when a thunk is forced (i.e. as we need a computation from the environment to occur), and answer actions correspond to a thunk returning a value (i.e reducing to $\mathrm{return}\ V$). Operationally, a CBPV thunk, when forced, will consume all of its arguments (the sequence of values it is applied to) before returning a value to a consumer (a context of form $\bullet\ \mathrm{to}\ x.M$). As these arguments are values, they cannot change during the reduction of the computation before they are consumed. Thus, from the view of contextual equivalence, a forced thunk behaves as if all arguments are consumed immediately when the thunk is forced.

To handle this, actions include sequences of abstract values, denoted by $\overrightarrow{A}$, and referred to as abstract argument sequences. Given a sequence of values (an argument sequence, written $\overrightarrow{V}$), we can find a decomposition into an abstract argument sequence and a substitution by applying $\mathbf{AVal}()$ to the values and combining the substitutions (ensuring names are disjoint). We abuse notation to write $\mathbf{AVal}(\overrightarrow{V})$ for the result.

For a given computation type, we can construct a sequence of abstract values using the function in Figure 4 (assuming name choices are fresh). We define the return type of a computation by $\mathbf{RType}(\sigma_1 \to \cdots \to \sigma_k \to F\sigma) \triangleq \sigma$.

2) Play: Our traces will 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 will be ***introduced*** either in a set N_O of names already introduced by O, a set N_P of names already introduced by P, 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, N_P by P), 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 argument sequence. 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 to the consumer corresponding to continuation name c.
- ***Player Question*** (PQ) $\bar{f}(\overrightarrow{A}, c)$, where $f : U\underline{\tau}$, $\overrightarrow{A} \in \mathbf{ASeq}(\underline{\tau})$, $c : \mathbf{RType}(\underline{\tau})$. This corresponds to the term forcing the thunk named by f, passing $\overrightarrow{A}$ as arguments, and expecting the result with the consumer corresponding to continuation name c.
- ***Opponent Answer*** (OA) $c(A)$, $c : \sigma$ then $A : \sigma$. In this case, the environment is producing a value A to the term, which is acting as a consumer with continuation name c.
- ***Opponent Question*** (OQ) $f(\overrightarrow{A}, c)$, where $f : U\underline{\tau}$, $\overrightarrow{A} \in \mathbf{ASeq}(\underline{\tau})$, $c : \mathbf{RType}(\underline{\tau})$. This action corresponds to the environment forcing the thunk named by f from the term, passing $\overrightarrow{A}$ as arguments, and expecting the result with the consumer corresponding to continuation name c.

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

Definition 7. Let $N_O, N_P \subseteq \mathrm{Names}$. An (N_O, N_P)-***trace*** is a sequence t of actions such that: the actions alternate between P and O actions; no name is introduced twice; names from N_O, N_P need no introduction; any action $\mathbf{a}$ must have the form $\bar{f}(\overrightarrow{A}, c)$, $f(\overrightarrow{A}, c)$, $\bar{c}(A)$ or $c(A)$, where the head name of $\mathbf{a}$ has been introduced by an earlier action $\mathbf{a}'$ of opposite polarity or (respectively) $f \in N_O$, $f \in N_P$, $c \in N_O$, $c \in N_P$.

Note that the first action in a $(N_O, \emptyset)$-trace must be by P.

Example 8. Let $N_O = \{f : U(\mathrm{Ref} \to \mathrm{Int} \to F\mathrm{Unit}), c : U(UF\mathrm{Int} \to F\mathrm{Int})\}$. Then $\mathtt{t} = \bar{f}([r, w]\,\widehat{4}, c_1)\ w(\widehat{1}, c_2)\ \bar{c_2}(())\ c_1(())\ \bar{c}(g)\ g(h, c_3)\ \bar{h}(\epsilon, c_4)\ c_4(\widehat{2})\ \bar{c_3}(\widehat{3})$. is an $(N_O, \emptyset)$-trace.

3) Visibility, bracketing, and completeness: It will turn out that the traces required to capture our CBPV language will have special properties, which correspond to the fact that our language lacks higher-order store (visibility) and control-flow operations (bracketing). The identification of these properties occurred in game semantics [3], [25], [26], and they are enforced by our transition system in the style of [24].

Definition 9. Given a prefix t of a (N_O, N_P)-trace:

- an OQ-action $f(\overrightarrow{A}, c)$ occurring in t is said to be ***unanswered*** in t if there does not exist a PA-move of the form $\bar{c}(A')$ in t;
- a PQ-action $\bar{f}(\overrightarrow{A}, c)$ occurring in t is said to be ***unanswered*** in t if there does not exist an OA-move of the form $c(A')$ in t.

To define the O-visibility and O-bracketing constraints, we will define a set $\mathsf{Vis}_O(t)$ of ***O-visible names*** and the ***top continuation name*** $\mathsf{Top}_O(t)$ in Figure 5. The function $\mathsf{Top}_O(t)$ essentially identifies the latest unanswered PQ-action, and returns the continuation name introduced then. $\mathsf{Vis}_O(t)$ is defined by tracing the head name of a move to its introduction, analogously to the game semantic notion of *view* [3].

Definition 10. Let t be a (N_O, N_P)-trace. t is ***O-visible*** if, for any prefix $t'\ f'(\overrightarrow{A'}, c')$ of t, we have $f' \in \mathsf{Vis}_O(t')$. t is ***O-bracketed*** if, for any prefix $t'\ c'(A)$ of t (i.e. any prefix ending with an O-answer), we have $c' \in \mathsf{Top}_O(t')$.

We now introduce some useful notation. Given a (N_O, N_P)-trace t, we write $t^\perp$ for the (N_P, N_O)-trace obtained by changing the polarity of each name: $f(\overrightarrow{A}, c')$ becomes $\bar{f}(\overrightarrow{A}, c')$ (and vice versa) and $c(A)$ becomes $\bar{c}(A)$ (and vice versa). Using this we can provide the dual notions to O-visibility and O-bracketing. We say an (N_O, N_P)-trace t is P-visible if $t^\perp$ is O-

$$
\begin{array}{lcl@{\qquad}lcl}
\mathbf{AVal}_\sigma(V) & \triangleq & \{(V,\emptyset)\} \quad \text{for } \sigma \in \{\mathrm{Unit},\mathrm{Int}\} & \mathbf{ASeq}(F\sigma) & \triangleq & \{\epsilon\} \\
\mathbf{AVal}_{U\underline{\tau}}(V) & \triangleq & \{(f,[f \mapsto V]) \mid f \in \mathrm{TNames}_{U\underline{\tau}}\} & \mathbf{ASeq}(\sigma \to \underline{\tau}) & \triangleq & \{A\ \ s \mid s \in \mathbf{ASeq}(\underline{\tau}), \\
\mathbf{AVal}_{\mathrm{Ref}}\ (\mathrm{MkVar}\ V_1\ V_2) & \triangleq & \{(\mathrm{MkVar}\ f\ g,[f \mapsto V_1,\ g \mapsto V_2])\} & & & A : \sigma \text{ an abstract value}\} \\
\mathbf{AVal}_{\mathrm{Ref}}\ (\ell) & \triangleq & \{(\mathrm{MkVar}\ f\ g,[f \mapsto \mathrm{thunk}\ (!\ell),\ g \mapsto \mathrm{thunk}\ (\lambda x.\ell := x)])\} & & &
\end{array}
$$

Fig. 4. Value decomposition into abstract values and substitutions, and generation of abstract value sequences

$$
\begin{array}{rcll}
\mathsf{Vis}_O(\epsilon) & = & N_P^T & \\
\mathsf{Vis}_O(t\ \bar{c}(A)) & = & (N_P^T) \cup \nu(A) & c \in N_O \\
\mathsf{Vis}_O(t\ \bar{f}(\overrightarrow{A'},c)\ t'\ \bar{c}(A)) & = & \mathsf{Vis}_O(t) \cup \nu(A) & \\
\mathsf{Vis}_O(t\ \bar{f}(\overrightarrow{A},c)) & = & \nu(\overrightarrow{A}) \cup \{c\} & f \in N_O \\
\mathsf{Vis}_O(t\ f'(\overrightarrow{A'},c')\ t'\ \bar{f}(\overrightarrow{A},c)) & = & \mathsf{Vis}_O(t) \cup \nu(\overrightarrow{A}) \cup \{c\} & f \in \nu(\overrightarrow{A'}) \\
\mathsf{Vis}_O(t\ c'(A')\ t'\ \bar{f}(\overrightarrow{A},c)) & = & \mathsf{Vis}_O(t) \cup \nu(\overrightarrow{A}) \cup \{c\} & f \in \nu(A'))
\end{array}
\qquad
\begin{array}{rcll}
\mathsf{Top}_O(\epsilon) & = & N_P^C & \\
\mathsf{Top}_O(t\ \bar{c}(A)) & = & N_P^C & c \in N_O \\
\mathsf{Top}_O(t\ f(\overrightarrow{A'},c)\ t'\ \bar{c}(A)) & = & \mathsf{Top}_O(t) & \\
\mathsf{Top}_O(t\ \bar{f}(\overrightarrow{A},c)) & = & \{c\} &
\end{array}
$$

Fig. 5. O-visible names $\mathsf{Vis}_O(t)$ and top continuation name $\mathsf{Top}_O(t)$ for (N_O, N_P)-trace t, where $N_P^T = N_P \cap \mathrm{TNames}$ and $N_P^C = N_P \cap \mathrm{CNames}$.

visible (and define $\mathsf{Vis}_P(t) = \mathsf{Vis}_O(t^\perp)$), and t is P-bracketed if $t^\perp$ is O-bracketed (and define $\mathsf{Top}_P(t) = \mathsf{Top}_O(t^\perp)$).

To capture termination, it will suffice to reason about complete traces [24].

Definition 11. An O-bracketed $(N_O, \emptyset)$-trace t starting with a P action is ***complete*** if $\mathsf{Top}_O(t) = \emptyset$. A P-bracketed $(\{\circ\}, N_P)$-trace t starting with an O action is ***complete*** if $\mathsf{Top}_P(t) = \{\circ\}$.

Example 12. The trace $\mathtt{t}$ from Example 8 is an O-bracketed, P-bracketed, O-visible, P-visible, complete trace.

4) Transition System: Using the above, we can define an LTS, called $\mathcal{L}_{\mathrm{CBPV}}$, which will generate the set of traces corresponding to a term. $\mathcal{L}_{\mathrm{CBPV}}$ will contain terms built from CBPV syntax, extended with all thunk names as values (with the obvious typing rule), with $\to$ behaving accordingly. $\mathcal{L}_{\mathrm{CBPV}}$ will contain configurations of the form $(\mathbf{S}, S)$, where $\mathbf{S}$ is a *state* and S a *stack*. There are two types of states in $\mathcal{L}_{\mathrm{CBPV}}$ (and so two kinds of configurations): $\langle \gamma, \phi, h, H, Fn\rangle$ (*passive*, O to play) and $\langle M, c, \gamma, \phi, h, H\rangle$ (*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 thunk names introduced by P to thunks. In an active configuration, M is the *term* component, which captures the current behaviour of P, and c is the continuation name to produce the result to. The stack is used to enforce O-bracketing. It consists of elements of the form $(c_P, (K, c_O))$ where c_P is a continuation P-name, K is an evaluation context in which to use the value produced to c_P, and c_O is an continuation O-name to return the result of K to. Fn represents the set of thunk P-names currently available to O (so after generating a prefix t, Fn contains thunk names from $\mathsf{Vis}_O(t)$), and H contains historical information about availability. This is used to enforce O-visibility, by maintaining the property that for a thunk O-name f, $H(f)$ contains the thunk P-names available to O when f was introduced. P-visibility is a consequence of the behaviour of the term.

We can now present the $\mathcal{L}_{\mathrm{CBPV}}$ transition rules in Figure 6. These are presented as labelled transitions between states, with $\mathbf{a}/m$ denoting pushing m to the stack when producing $\mathbf{a}$, and $\mathbf{a}, m$ denoting popping m when producing $\mathbf{a}$.

Let $\Gamma \vdash^c M : F\sigma$ be a CBPV computation 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 $c : \sigma$ and $N_O = \nu(\rho) \cup \{c\}$. Then the active initial configuration $\mathsf{C}_M^{\rho,c}$ is defined to be

$$(\langle M\{\rho\}, c, \emptyset, N_O, \emptyset, [N_O \mapsto \emptyset]\rangle, \perp)$$

Definition 13. Given two configurations $\mathbf{C}, \mathbf{C}'$, we write $\mathbf{C} \overset{\mathbf{a}}{\Longrightarrow} \mathbf{C}'$ if $\mathbf{C} \xrightarrow{\tau}{}^* \mathbf{C}'' \xrightarrow{\mathbf{a}} \mathbf{C}'$, with $\xrightarrow{\tau}{}^*$ representing multiple (possibly none) τ-actions. This notation is extended to sequences of actions: given $\mathtt{t} = \mathbf{a}_1 \ldots \mathbf{a}_n$, we write $\mathbf{C} \overset{\mathtt{t}}{\Longrightarrow} \mathbf{C}'$, if there exist $\mathbf{C}_1, \ldots, \mathbf{C}_{n-1}$ such that $\mathbf{C} \overset{\mathbf{a}_1}{\Longrightarrow} \mathbf{C}_1 \cdots \mathbf{C}_{n-1} \overset{\mathbf{a}_n}{\Longrightarrow} \mathbf{C}'$. We define $\mathbf{Tr}_{\mathrm{CBPV}}(\mathbf{C}) = \{\mathtt{t} \mid \text{there exists } \mathbf{C}' \text{ such that } \mathbf{C} \overset{\mathtt{t}}{\Longrightarrow} \mathbf{C}'\}$.

Remark 14. Due to the freedom of name choice, note that $\mathbf{Tr}_{\mathrm{CBPV}}(\mathbf{C})$ is closed under type-preserving renamings that preserve names from $\mathbf{C}$.

Definition 15. A ***path*** in an LTS from a configuration $\mathbf{C}$ to $\mathbf{C}'$ is a sequence of transitions and intermediate configurations by which $\mathbf{C}'$ can be reached from $\mathbf{C}$ when viewing the LTS as a directed graph.

Traces of $\mathcal{L}_{\mathrm{CBPV}}$ satisfy the following property.

Lemma 16. *Given $\Gamma \vdash^c M : F\sigma$ and a Γ-assignment ρ, for any $t \in \mathbf{Tr}_{\mathrm{CBPV}}(\mathsf{C}_M^{\rho,c})$, we have t is O-visible, O-bracketed, P-visible and P-bracketed.*

Definition 17. The ***trace semantics*** of a CBPV computation $\Gamma \vdash^c M : F\sigma$ is defined to be $\mathbf{Tr}_{\mathrm{CBPV}}(\Gamma \vdash^c M : F\sigma) \triangleq$

$$
\begin{array}{lll}
(P\tau) & \langle M, c, \gamma, \phi, h, H\rangle & \xrightarrow{\tau} \quad \langle N, c, \gamma, \phi, h', H\rangle \\
 & \text{when } (M,h) \to (N,h') & \\
(PA) & \langle \text{return } V, c, \gamma, \phi, h, H\rangle & \xrightarrow{\bar{c}(A)} \quad \langle \gamma\cdot\gamma', \phi \uplus \nu(A), h, H, H(c) \uplus \nu(A)\rangle \\
 & \text{when } c:\sigma,\ (A,\gamma') \in \mathbf{AVal}_\sigma(V) & \\
(PQ) & \langle K[(\text{force } f)\overrightarrow{V}], c, \gamma, \phi, h, H\rangle & \xrightarrow{\bar{f}(\overrightarrow{A},c')/(c',(K,c))} \quad \langle \gamma\cdot\gamma', \phi \uplus \phi', h, H, H(f) \uplus \nu(\overrightarrow{A})\rangle \\
 & \text{when } f : U\underline{\tau},\ (\overrightarrow{A},\gamma') \in \mathbf{AVal}(\overrightarrow{V}),\ \sigma = \mathbf{RType}(\underline{\tau}),\ c':\sigma \text{ and } \phi' = \nu(\overrightarrow{A}) \uplus \{c'\} & \\
(OA) & \langle \gamma, \phi, h, H, Fn\rangle & \xrightarrow{c(A),(c,(K,c'))} \quad \langle K[\text{return } A], c', \gamma, \phi \uplus \nu(A), h, H\cdot[\nu(A) \mapsto Fn]\rangle \\
 & \text{when } c:\sigma,\ A:\sigma & \\
(OQ) & \langle \gamma, \phi, h, H, Fn\rangle & \xrightarrow{f(\overrightarrow{A},c)} \quad \langle (\text{force } V)\overrightarrow{A}, c, \gamma, \phi \uplus \phi', h, H\cdot[\phi' \mapsto Fn]\rangle \\
 & \text{when } f \in Fn,\ f : U\underline{\tau},\ \overrightarrow{A} \in \mathbf{ASeq}(\underline{\tau}),\ \sigma = \mathbf{RType}(\underline{\tau}),\ c:\sigma,\ \gamma(f) = V \text{ and } \phi' = \nu(\overrightarrow{A}) \uplus \{c\} &
\end{array}
$$

Given $N \subseteq \text{Names}$, $[N \mapsto \mathcal{V}]$ stands for the map $[n \mapsto \mathcal{V} \mid n \in N]$.

Fig. 6. $\mathcal{L}_{\text{CBPV}}$ transition rules

$\{((\rho,c),t) \mid \rho \text{ is a } \Gamma\text{-assignment},\ c:\sigma,\ t \in \mathbf{Tr}_{\text{CBPV}}(\mathsf{C}_M^{\rho,c}),\ t \text{ is complete}\}$.

Example 18. Let $\Gamma = \{f : U(\text{Ref} \to \text{Int} \to F\text{Unit})\}$, $\sigma = U(UF\text{Int} \to F\text{Int})$, and

$$M = \text{ref}\,\widehat{0} \text{ to } x.(\text{force } f)\,x\,\widehat{4};$$
$$\text{return thunk } (\lambda h.\text{force } h \text{ to } y.!x \text{ to } z.y + z)$$

If $\rho = [f \mapsto f]$ and $\mathtt{t}$ is the trace in Example 8, then $((\rho,c),\mathtt{t}) \in \mathbf{Tr}_{\text{CBPV}}(\Gamma \vdash^c M : F\sigma)$.

IV. Full Abstraction

To establish the soundness of this model (trace inclusion implies contextual inclusion), we will wish to reason about a computation in a specific context. Let $\Gamma \vdash^c M : F\sigma$. Using the CIU lemma, we will consider testing using a heap $h : \Sigma$, evaluation context $\vdash^k K : F\sigma \implies F\sigma'$ and a substitution $\gamma : \Gamma$. Let us fix a continuation name $\circ : \sigma'$, which we use to determine when a context has returned.

Next we define the set $\mathbf{AVal}_\Gamma(\gamma)$ of all disjoint decompositions of values from γ into abstract values and the corresponding matchings by

$$\mathbf{AVal}_\Gamma(\gamma) = \{(\vec{A_i},\vec{\gamma_i}) \mid 1 \le i \le k, (A_i,\gamma_i) \in \mathbf{AVal}_{\sigma_i}(\gamma(x_i)),\ \nu(A_1),\cdots,\nu(A_k) \text{ mutually disjoint}\}$$

where $\Gamma = \{x_1 : \sigma_1,\ \cdots,\ x_k : \sigma_k\}$, $\vec{A_i}$ stands for $(A_1,\cdots,A_k)$, and $\vec{\gamma_i}$ for $(\gamma_1,\cdots,\gamma_k)$.

Definition 19 (Context configuration)**.** Given Σ, $h:\Sigma$, $\Sigma \vdash^c K : F\sigma \implies F\sigma'$, $\Sigma \vdash^c \gamma : \Gamma$, $(\vec{A_i},\vec{\gamma_i}) \in \mathbf{AVal}_\Gamma(\gamma)$ and $c : \sigma$ ($c \notin \circ$), the corresponding configuration $\mathsf{C}_{h,K,\gamma}^{\vec{\gamma_i},c}$ is defined by

$$\mathsf{C}_{h,K,\gamma}^{\vec{\gamma_i},c} = (\langle \biguplus_{i=1}^{k} \gamma_i,\ \phi' \uplus \{c\},\ h,\ [\circ \mapsto \emptyset],\ \phi'\rangle,\ (c,(K,\circ)) : \bot)$$

where $\phi' = \biguplus_{i=1}^{k} \nu(A_i)$

Intuitively, the names $\nu(A_i)$ correspond to thunks extracted from γ, whereas c corresponds to K. Note that traces in $\mathbf{Tr}_{\text{CBPV}}(\mathsf{C}_{h,K,\gamma}^{\vec{\gamma_i},c})$ will be $(\{\circ\}, \biguplus_{i=1}^{k} \nu(A_i) \uplus \{c\})$-traces.

For the next result, we introduce the following notation. Given $(\vec{A_i},\vec{\gamma_i}) \in \mathbf{AVal}_\Gamma(\gamma)$, we define a Γ-assignment $\rho_{\vec{A_i}}$ by $\rho_{\vec{A_i}}(x_i) = A_i$. Note that $\nu(\rho_{\vec{A_i}}) = \biguplus_{i=1}^{k} \text{dom}(\gamma_i)$. The key lemma we now need to prove will relate traces of a term to its ability to converge in a given evaluation context. From correctness, we can then obtain soundness.

Lemma 20 (Correctness)**.** *Let $\Gamma \vdash^c M : F\sigma$ be a* CBPV *computation, let Σ, h, K, γ be as above, $(\vec{A_i},\vec{\gamma_i}) \in \mathbf{AVal}_\Gamma(\gamma)$, and $c : \sigma$ ($c \neq \circ$). Then $(K[M\{\gamma\}], h) \Downarrow_{ter}$ iff there exist t, A such that $t \in \mathbf{Tr}_{\text{CBPV}}(\mathsf{C}_M^{\rho_{\vec{A_i}},c})$ and $t^\perp\,\bar{\circ}(A) \in \mathbf{Tr}_{\text{CBPV}}(\mathsf{C}_{h,K,\gamma}^{\vec{\gamma_i},c})$. Moreover, t satisfies $\nu(t) \cap \{\circ\} = \emptyset$.*

Theorem 21 (Soundness)**.** *For any* CBPV *computations $\Gamma \vdash^c M_1, M_2 : F\sigma$, $\mathbf{Tr}_{\text{CBPV}}(\Gamma \vdash^c M_1) \subseteq \mathbf{Tr}_{\text{CBPV}}(\Gamma \vdash^c M_2)$ then $\Gamma \vdash^c M_1 \lesssim_{ter}^{\text{CBPV}(ciu)} M_2$.*

For the opposite direction, we establish that any trace of a suitable shape corresponds to a context.

Lemma 22 (Definability)**.** *Suppose $\phi \subseteq \text{TNames}$ and t is an even-length O,P-visible, O,P-bracketed $(\{\circ\}, \phi \uplus \{c\})$-trace starting with an O-action, such that $t = t'\,\bar{\circ}(A)$ and t' is complete. There exists a passive configuration $\mathbf{C}$ such that $\mathbf{Tr}^{\text{even}}(\mathbf{C})$ is the even-length prefixes of t (along with their renamings via permutations on* Names *that fix $\phi \uplus \{\circ\}$). Moreover, $\mathbf{C} = \langle \gamma, \phi \uplus \{c\}, h, [\circ \mapsto \emptyset], \phi\rangle, (c,(K,\circ)) : \bot)$ for some h, K, γ.*

Completeness follows from definability and correctness.

Theorem 23 (Completeness)**.** *For any* CBPV *computations $\Gamma \vdash^c M_1, M_2 : F\sigma$, if $\Gamma \vdash^c M_1 \lesssim_{ter}^{\text{CBPV}(ciu)} M_2$ then $\mathbf{Tr}_{\text{CBPV}}(\Gamma \vdash^c M_1) \subseteq \mathbf{Tr}_{\text{CBPV}}(\Gamma \vdash^c M_2)$.*

Using soundness (Theorem 21), completeness (Theorem 23), and CIU lemma (Lemma 3), we have the following corollary.

Corollary 24 (Full Abstraction)**.** *For any* CBPV *computations $\Gamma \vdash^c M_1, M_2 : F\sigma$, then $\Gamma \vdash^c M_1 \lesssim_{ter}^{\text{CBPV}} M_2$ iff $\mathbf{Tr}_{\text{CBPV}}(\Gamma \vdash^c M_1) \subseteq \mathbf{Tr}_{\text{CBPV}}(\Gamma \vdash^c M_2)$.*

V. From LTS to Automata (alphabet)

In this section, we demonstrate how $\mathcal{L}_{\mathrm{CBPV}}$ gives rise directly to an automaton, when considering terms drawn from a particular fragment of CBPV. Initially, we will look for a fragment of CBPV for which (a faithful representation of) a term's traces can be captured using a (deterministic) ***Visibly Pushdown Automaton*** (VPA) [27]. A VPA is a type of pushdown automata in which the action on the stack is determined by the input symbol. The alphabet is partitioned into call (push), return (pop), and internal (noop) symbols. This ensures that inclusion (and so equivalence) of deterministic VPA is decidable in polynomial time.

In seeking to identify a fragment for which VPA's suffice, we need to ensure that the space of states and the alphabet of actions are finite. Primarily, this is an issue when it is not possible to bound the set of names visible to O, $\mathsf{Vis}_O(t)$ for a trace t generated by the LTS. As the configuration will require a map from (at least the visible) names to the corresponding thunks from P, not having a bound on visible names will also mean we cannot bound the size of this map. We will see how this consideration restricts the type of terms through examples.

Example 25. Let $N_O^1 = \{c : UFUF\mathrm{Int}\}$. Consider $(N_O^1, \emptyset)$-traces of the form $\bar{c}(g)\ g(\epsilon, c_1)\ \bar{c_1}(f_1)\ \cdots\ g(\epsilon, c_n)\ \bar{c_n}(f_n)\ f_i(\epsilon, c')$. To capture them, one needs to generate arbitrarily many fresh names, because the last action could refer to any f_i. This issue arises whenever we permit P to provide to O a thunk which returns a thunk (g in this case), as O can then obtain arbitrarily many, (potentially) distinct thunks.

Let $N_O^2 = \{c : U(U(UF\mathrm{Int} \to F\mathrm{Unit}) \to F\mathrm{Unit})\}$. Consider $(N_O^2, \emptyset)$-traces of the form $\bar{c}(g)\ g(f_1, c_1)\ \bar{f_1}(h_1, c_1')\ \cdots\ g(f_n, c_n)\ \bar{f_n}(h_n, c_n')\ h_i(\epsilon, c')$. Here we can see that the same issue arises when we allow an argument passed by O (f_i) to itself receive an argument thunk (h_i) from P.

The fragment defined below is designed precisely to circumvent the problems identified above.

Definition 26. A CBPV computation $\Gamma \vdash^c M : F\sigma^P$ is in the ***P-thunk-restricted*** (PTR) fragment when all types in Γ can be generated by σ^2 in the grammar below.

$$\begin{array}{llll} \sigma^2 & \triangleq & \sigma^1 \mid U\underline{\tau}^2 & \sigma^P \triangleq \sigma^0 \mid \mathrm{Ref} \mid U\underline{\tau}^P \\ \underline{\tau}^2 & \triangleq & F\sigma^2 \mid \sigma^P \to \underline{\tau}^2 & \underline{\tau}^P \triangleq F\sigma^0 \mid \sigma^1 \to \underline{\tau}^P \\ \sigma^1 & \triangleq & \sigma^0 \mid \mathrm{Ref} \mid U\underline{\tau}^1 & \underline{\tau}^1 \triangleq F\sigma^1 \mid \sigma^0 \to \underline{\tau}^1 \\ \sigma^0 & \triangleq & \mathrm{Int} \mid \mathrm{Unit} & \end{array}$$

Remark 27. Note that all thunk P-names in a trace generated by a computation in the PTR-fragment have the type $U\underline{\tau}^P$.

Remark 28. An alternative way to characterise the the PTR-fragment is by polarising the occurrences of U, which correspond to question actions. If one writes U^+ for occurrences of U that produce O-questions, and U^- for those producing P-questions, the problematic types in Example 25 are then $U^+FU^+F\mathrm{Int}$ and $U^+(U^-(U^+F\mathrm{Int} \to F\mathrm{Unit}) \to F\mathrm{Unit})$, both of which contain nested occurrences of U^+. The PTR-fragment is then obtained by forbidding nested occurrences of U^+, while allowing nested occurrences of U^-.

Definition 29. A $(N_O, \emptyset)$-trace is a PTR***-trace*** when it is O- and P-bracketed, O- and P-visible, and it starts with a P-action with $\{c\} = N_O \cap \mathrm{CNames}$ where $c : \sigma^P$, and for $f \in N_O \cap \mathrm{TNames}$, $f : \sigma^2$, where σ^P, σ^2 are as defined in Definition 26.

Observe that these are exactly the traces which $\mathcal{L}_{\mathrm{CBPV}}$ generates on PTR computations.

To provide a representation of traces without arbitrarily many fresh names, we develop the notion of a ***name scheme***. The idea is to associate each thunk type appearing in a typing judgment with a fixed name. Readers familiar with game semantics will find the definition similar to that of an arena, where thunk names and continuation names indicate the position of questions and answers respectively, and $\mathrm{Suc_T}, \mathrm{Suc_C}$ correspond to the enabling relation.

Definition 30. A $(\Gamma, F\sigma)$***-name scheme*** is a tuple $(\mathrm{TB}, \mathrm{CB}, \rho, c_0, \mathrm{Suc_T}, \mathrm{Suc_C})$ such that ρ is a Γ-assignment, $c_0 : \sigma$, and $\mathrm{TB} \subseteq \mathrm{TNames}$ and $\mathrm{CB} \subseteq \mathrm{CNames}$ are the smallest sets such that $\nu(\rho) \subseteq \mathrm{TB}$, $c_0 \in \mathrm{CB}$ and the conditions listed below are satisfied. We set $\mathrm{TB}_{U\underline{\tau}} \triangleq \mathrm{TB} \cap \mathrm{TNames}_{U\underline{\tau}}$ and $\mathrm{CB}_\sigma \triangleq \mathrm{CB} \cap \mathrm{CNames}_\sigma$.

- $\mathrm{Suc_T}$ is the least partial function from $(\mathrm{TB} \times \mathbb{N}) \uplus \mathrm{CB}$ to $\mathrm{TB} \cup (\mathrm{TB} \times \mathrm{TB})$ such that: if $c \in \mathrm{CB}_{U\underline{\tau}}$ then $\mathrm{Suc_T}(c) \in \mathrm{TB}_{U\underline{\tau}}$; if $c \in \mathrm{CB}_{\mathrm{Ref}}$ then $\mathrm{Suc_T}(c) \in \mathrm{TB}_{UF\mathrm{Int}} \times \mathrm{TB}_{U(\mathrm{Int} \to F\mathrm{Unit})}$; if $f \in \mathrm{TB}_{U(\sigma_1 \to \cdots \to \sigma_k \to FU\sigma')}$ and $1 \le i \le k$ then $\mathrm{Suc_T}(f, i) \in \mathrm{TB}_{U\underline{\tau_i}}$ for $\sigma_i = U\underline{\tau_i}$ and $\mathrm{Suc_T}(f, i) \in \mathrm{TB}_{UF\mathrm{Int}} \times \mathrm{TB}_{U(\mathrm{Int} \to F\mathrm{Unit})}$ for $\sigma_i = \mathrm{Ref}$.
- $\mathrm{Suc_C} : \mathrm{TB} \to \mathrm{CB}$ is a function such that if $f \in \mathrm{TB}_{U\underline{\tau}}$ then $\mathrm{Suc_C}(f) \in \mathrm{CB}_{\mathbf{RType}(\underline{\tau})}$.
- $\nu(\mathrm{Suc_X}(d)) \cap \nu(\mathrm{Suc_X}(d')) = \emptyset$ for $d \neq d'$ and $X \in \{\mathrm{T}, \mathrm{C}\}$ (which implies injectivity) and $(\mathrm{img}(\mathrm{Suc_T}) \cup \mathrm{img}(\mathrm{Suc_C})) \cap (\nu(\rho) \cup \{c_0\}) = \emptyset$.

Elements of TB and CB will be referred to as ***base thunk names*** and ***base continuation names*** respectively. Abstract values containing base names only will be called ***base abstract values***. We shall write $\Delta_{\Gamma, F\sigma}$ for a $(\Gamma, F\sigma)$-name scheme, and Δ when we leave $(\Gamma, F\sigma)$ implicit.

Example 31. Consider $\underline{\tau} = U(\underline{\tau'}) \to FUF\mathrm{Unit}$, where $\underline{\tau'} = UF\mathrm{Int} \to UF\mathrm{Unit} \to F\mathrm{Int}$, $\Gamma = \{f : U\underline{\tau}\}$, $c_0 : \mathrm{Unit}$. For simplicity, assume $f \in \mathrm{TB}_{U\underline{\tau}}$ and $\rho(f) = f$. Then $\Delta_{\Gamma, F\mathrm{Unit}} = (\mathrm{TB}, \mathrm{CB}, \rho, c_0, \mathrm{Suc_T}, \mathrm{Suc_C})$ is a name scheme, where

- $\mathrm{TB} = \{f : U\underline{\tau},\ g : UF\mathrm{Unit},\ h : U\underline{\tau'},\ i : UF\mathrm{Int},\ j : UF\mathrm{Unit}\}$;
- $\mathrm{CB} = \{c_0 : \mathrm{Unit},\ c_f : UF\mathrm{Unit},\ c_g : \mathrm{Unit},\ c_h : \mathrm{Int},\ c_i : \mathrm{Int},\ c_j : \mathrm{Unit}\}$;
- $\mathrm{Suc_T}(f, 1) = h$, $\mathrm{Suc_T}(h, 1) = i$, $\mathrm{Suc_T}(h, 2) = j$, $\mathrm{Suc_T}(c_f) = g$; and
- $\mathrm{Suc_C}(f) = c_f$, $\mathrm{Suc_C}(g) = c_g$, $\mathrm{Suc_C}(h) = c_h$, $\mathrm{Suc_C}(i) = c_i$, $\mathrm{Suc_C}(j) = c_j$.

Showing $\mathrm{Suc_T}$ with solid arrows, and $\mathrm{Suc_C}$ with dashed, this

$$\begin{array}{lcll}
\mathbf{BVals}^{\Delta}_{\sigma}(d) & \triangleq & \{V \mid V:\sigma\} & \text{where } \sigma \in \{\mathrm{Int}, \mathrm{Unit}\} \\
\mathbf{BVals}^{\Delta}_{U_\tau}(d) & \triangleq & \{f\} & \text{where } \mathrm{Suc_T}(d) = f \\
\mathbf{BVals}^{\Delta}_{\mathrm{Ref}}(d) & \triangleq & \{\{\!|f,g|\!\}\} & \text{where } \mathrm{Suc_T}(d) = (f,g)
\end{array}$$

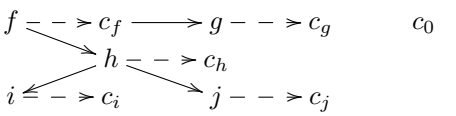
$$\begin{array}{lcl}
\mathbf{BValSeq}^{\Delta}(f) & \triangleq & \{\epsilon\} \text{ where } f : F\sigma \\
\mathbf{BValSeq}^{\Delta}(f) & \triangleq & \{B_1 \cdots B_k \mid B_i \in \mathbf{BVals}^{\Delta}_{\sigma_i}((f,i))\} \\
 & & \text{where } f : \sigma_1 \to \cdots \to \sigma_k \to F\sigma
\end{array}$$

Fig. 7. Definition of base abstract values for given base head names and $\Delta = (\mathrm{TB}, \mathrm{CB}, \rho, c_0, \mathrm{Suc_T}, \mathrm{Suc_C})$

$$\begin{array}{lcll}
\mathbf{Base}^{\Delta}_{t}(n) & \triangleq & n & \text{where } n \in N_O \\
\mathbf{Base}^{\Delta}_{t}(c) & \triangleq & \mathrm{Suc_C}(g) & \text{where } c \text{ is introduced in } f(A,c) \text{ or } \bar{f}(A,c) \text{ and } g = \mathbf{Base}^{\Delta}_{t}(f) \\
\mathbf{Base}^{\Delta}_{t}(f) & \triangleq & g & \text{where } f \text{ is introduced in } c(A) \text{ or } \bar{c}(A) \text{ with } c:\sigma,\ c' = \mathbf{Base}^{\Delta}_{t}(c) \\
 & & & \{B\} = \mathbf{BVals}^{\Delta}_{\sigma}(c) \text{ and } g = \mathbf{Match}(A,B,f) \\
\mathbf{Base}^{\Delta}_{t}(f) & \triangleq & g & \text{where } f \text{ is introduced in } f'(\overrightarrow{A},c) \text{ or } \bar{f}'(\overrightarrow{A},c) \text{ with } g' = \mathbf{Base}^{\Delta}_{t}(f'), \\
 & & & \overrightarrow{B} \in \mathbf{BValSeq}^{\Delta}(g') \text{ and } g = \mathbf{Match}(\overrightarrow{A},\overrightarrow{B},f) \\
\mathbf{Marked}(t) & \triangleq & \{t' \mid \mathbf{a} \text{ is an O-action in } t,\ f \text{ is introduced in } \mathbf{a}, \text{ and } t' \text{ is } t \text{ with } f \text{ replaced by } \hat{f}\} & \\
\mathbf{Rename}^{\Delta}(t) & \triangleq & \mathbf{Base}^{\Delta}(t) \cup \bigcup_{t' \in \mathbf{Marked}(t)} \mathbf{Base}^{\Delta}(t') &
\end{array}$$

Fig. 8. The function $\mathbf{Base}^{\Delta}_{t}()$ which converting names appearing in t to base names from $\Delta = (\mathrm{TB}, \mathrm{CB}, \rho, c_0, \mathrm{Suc_T}, \mathrm{Suc_C})$

can be visualised as a forest.



We can recast many definitions to use name schemes. For a start, we will redefine the notion of a trace so that it relies on base head names only, and base abstract values depend upon the name they are passed to via $\mathrm{Suc_T}$ or $\mathrm{Suc_C}$. To this end, in Figure 7 we define $\mathbf{BVals}^{\Delta}_{\sigma}(d)$ (for $d \in (\mathrm{TB} \times \mathbb{N}) \uplus \mathrm{CB}$) and $\mathbf{BValSeq}^{\Delta}(f)$ (for $f \in \mathrm{TB}$) to indicate the associated base abstract values and sequences thereof respectively. Note that they are determined uniquely up to numerical constants.

Definition 32. Let $\Delta = (\mathrm{TB}, \mathrm{CB}, \rho, c_0, \mathrm{Suc_T}, \mathrm{Suc_C})$ be a name scheme. A Δ-***trace*** is a sequence t of actions such that: the actions alternate between P and O actions; names from $\nu(\rho) \cup c_0$ need no introduction; and the possible actions are:

- $\bar{f}(\overrightarrow{A},c)$ where $f \in \mathrm{TB}$, $c = \mathrm{Suc_C}(f)$, $\overrightarrow{A} \in \mathbf{BValSeq}^{\Delta}(f)$ and f was introduced by an earlier O-action or $f \in \nu(\rho)$,
- $\bar{c}(A)$ where $c : \sigma \in \mathrm{CB}$, $A \in \mathbf{BVals}^{\Delta}_{\sigma}(c)$ and c was introduced by an earlier O-action or $c = c_0$,
- $f(\overrightarrow{A},c)$ where $f \in \mathrm{TB}$, $c = \mathrm{Suc_C}(f)$, $\overrightarrow{A} \in \mathbf{BValSeq}^{\Delta}(f)$ and f was introduced by an earlier P-action.
- $c(A)$ where $c : \sigma \in \mathrm{CB}$, $A \in \mathbf{BVals}^{\Delta}_{\sigma}(c)$ and c was introduced by an earlier P-action.

Given the structure on base names, we can now introduce some terminology to distinguish the different classes of names.

Definition 33. Let $\Delta = (\mathrm{TB}, \mathrm{CB}, \rho, c_0, \mathrm{Suc_T}, \mathrm{Suc_C})$. The names in $\nu(\rho)$ and c_0 are said to be ***initial***. A name $f \in \mathrm{TB}$ is said to be a ***level-***n name if $n = 0$ and the name is initial, or $n = 1$ and $f \in \nu(\mathrm{Suc_T}(c_0))$, or there exists a level-$(n-1)$ name g and $j \in \mathbb{N}$ such that $f \in \nu(\mathrm{Suc_T}(g,j))$, or

- there exists a level-n name g and $c \in \mathrm{CB}$ such that $f \in \nu(\mathrm{Suc_T}(c))$ and $c = \mathrm{Suc_C}(g)$.

For level-n name f, the sequence of thunk names induced by the repeated use of the last rule will be called an ***introduction chain***, and the first name in the chain (i.e. introduced by the earlier rules) is called the ***originator***.

Example 34. In $\Delta_{\Gamma, F\mathrm{Unit}}$ from Example 31, f, g are level 0, h is level 1, and i, j are level 2. f is the originator of g. These align with the position of names in the visualisation.

Remark 35. Observe that, when used in a trace, the level 0 names will be O-names, so the level 1 names will be P-names, and the level 2 names will be O-names. In the PTR-fragment, all thunk names have level at most 2.

We shall say that a name scheme $\Delta = (\mathrm{TB}, \mathrm{CB}, \rho, c_0, \mathrm{Suc_T}, \mathrm{Suc_C})$ ***agrees with*** $N_O \subseteq \mathrm{Names}$ if $N_O = \nu(\rho) \cup c_0$. Observe that, given a $(N_O, \emptyset)$-trace t and a Δ agreeing with N_O, we can construct a function $\mathbf{Base}^{\Delta}_{t}$ that maps names in $\nu(t)$ to base names by recursing on the introduction of names until we reach names in N_0. This is given formally in Figure 8, where $\mathbf{Match}(A,B,f)$ finds the base name in B in the same position as f in A.We can extend $\mathbf{Base}^{\Delta}_{t}$ to sets of names, abstract values, actions, and also to an entire trace, which we shall write simply as $\mathbf{Base}^{\Delta}(t)$. This is analogous to erasing justification pointers in game-semantic plays. We can now state a first useful result about base names. Intuitively, it means that, in PTR traces, base names suffice to distinguish O-visible thunk names, i.e. in such traces we can use base names to represent OQ actions faithfully.

Lemma 36. *Let t be a* PTR *$(N_O, \emptyset)$-trace ending in a P-action. Then $f, f' \in \mathsf{Vis}_O(t)$, we have $\mathbf{Base}^{\Delta}_{t}(f) \neq \mathbf{Base}^{\Delta}_{t}(f')$.*

It would be desirable if we could represent the traces of a PTR-computation using Δ-traces. However, it turns out that simply applying $\mathbf{Base}^{\Delta}$ to a trace loses information.

Example 37. Let $\Gamma = \{f : UFUF\text{Unit}\}$ and $\Delta = (\{ f : UFUF\text{Unit}, g : UF\text{Unit} \}, \{c_0 : \text{Unit}, d : UF\text{Unit}, e : \text{Unit}\}, \rho, c_0, [d \mapsto g], [f \mapsto d, g \mapsto e])$, where $\rho = [f \mapsto f]$. Consider the two computations $\Gamma \vdash^c M_1, M_2 : F\text{Unit}$, where $M_i = \text{force } f \text{ to } g_1.\text{force } f \text{ to } g_2.\text{force } g_i$. We have that the complete traces in $\mathbf{Tr}(\mathsf{C}^{\rho,c_0}_{M_i})$ all have the form

$$\mathtt{t}_i = \bar{f}(\epsilon, c_1)\ c_1(g_1)\ \bar{f}(\epsilon, c_2)\ c_2(g_2)\ \bar{g_i}(\epsilon, c_3)\ c_3(())\ \bar{c_0}(())$$

That is, the traces are distinguished by the use of either g_1 or g_2. However, we have $\mathbf{Base}^{\Delta}\mathbf{Tr}(\mathsf{C}^{\rho,c_0}_{M_i})$ as given in Figure 9. That is, we lose the distinction between g_1 and g_2, and so if we simply used Δ-traces to model terms, we would equate M_1 with M_2, which would mean losing the soundness property.

A natural solution to this would be to take inspiration from the way that $\mathtt{t}_i$ is presented in this example, and use traces in which base names can appear scripted by when they are introduced. We could also exploit Lemma 42 to allow us to reset these indices after a PA-action. However, the presence of while loops means that this is not viable. A while loop might cause a PQ-action to occur arbitrarily many times, seemingly requiring an unbounded number of indices. We could attempt to exploit the fact that the scopes in the language ensure that any name introduced during an iteration of a loop cannot escape that loop to reset the indices at the end of a loop. However, the end of a loop cannot be apparent in a trace. This makes it difficult to see how to equate the terms

$$\begin{array}{rcl} N_1 & = & (\text{force } f \text{ to } g.\text{force } g); \text{force } f \text{ to } g.\text{force } g \\ N_2 & = & \text{ref } 2 \text{ to } x.\text{while } !x \text{ do } ((\text{force } f \text{ to } g.\text{force } g); \\ & & \qquad !x \text{ to } v.v - 1 \text{ to } w.x := w\ ) \end{array}$$

This is because, for the first, we would want to have traces like $\bar{f}(\epsilon, d_1)\ d_1(g_1)\ \bar{g_1}(\epsilon, e_1)\ e_1(())\ \bar{f}(\epsilon, d_2)\ d_2(g_2)\ \bar{g_2}(\epsilon, e_2)\ e_2(())\ \bar{c_0}(())$, as it is not 'safe' to reset the index counter after the $e_1(())$, whereas for the second we would need to have traces like $\bar{f}(\epsilon, d_1)\ d_1(g_1)\ \bar{g_1}(\epsilon, e_1)\ e_1(())\ \bar{f}(\epsilon, d_1)\ d_1(g_1)\ \bar{g_1}(\epsilon, e_1)\ e_1(())\ \bar{c_0}(())$, as we would be resetting the index at the end of every iteration.

This issue also appears in the work of Hopkins et al. [12], which they resolve by encoding a single P-pointer in each word their automata generate, and then use the fact that a set of words, each with one pointer, can be used to uniquely represent a full play. We adopt the same approach, and adapt it to our name-based setting.

The key to this is the notion of a ***marked name***, which we shall write as $\hat{f}$, where f is said to be the underlying name, which can be either from TNames or TBNames for some Δ, depending on context. We introduce f into our structures (traces, abstract values, etc.) by permitting a marked name wherever the underlying name can occur, and will refer to these also as marked (e.g. marked trace). In particular, for base abstract values, a marked name can appear in place of its underlying name in $\mathbf{BVals}^{\Delta}_{\sigma}(d)$. We also extend the functions $\mathbf{Base}^{\Delta}$ and $\mathbf{Base}^{\Delta}_{t}$ so that they preserve marks.

With this notion, we can now define an appropriate mapping of PTR-traces to a set of marked, Δ-traces as $\mathbf{Rename}^{\Delta}(t)$ in Figure 8. Observe that $\mathbf{Rename}^{\Delta}(t)$ consists of traces whose underlying names are all the same, and for each trace, at most one O-base name introduced during t has been marked. We lift $\mathbf{Rename}()$ to sets in the obvious way.

Example 38. Let Δ, M_1, M_2 be as in Example 37. Then we have $\mathbf{Rename}^{\Delta}(\mathbf{Tr}(\mathsf{C}^{\rho,c_0}_{M_i}))$ as shown in Figure 9.

$\mathbf{Rename}^{\Delta}()$ turns out to be sufficiently informative to provide a faithful representation of PTR-traces.

Lemma 39. *Suppose t_1, t_2 are* PTR *$(N_O, \emptyset)$-traces, Δ agrees with N_O and $\mathbf{Rename}^{\Delta}(t_1) = \mathbf{Rename}^{\Delta}(t_2)$. Then t_1 and t_2 are equal up to a permutation of names that preserves N_O.*

Corollary 40. *For* PTR*-computations $\Gamma \vdash^c M_1, M_2 : F\sigma$, continuation name $c : \sigma$, a Γ-assignment ρ and $\Delta_{\Gamma,c} = (\text{TB}, \text{CB}, \rho, c, \text{Suc}_{\text{T}}, \text{Suc}_{\text{C}})$, we have $\mathbf{Tr}(\mathsf{C}^{M_1}_{\rho,c}) = \mathbf{Tr}(\mathsf{C}^{M_2}_{\rho,c})$ iff $\mathbf{Rename}^{\Delta_{\Gamma,c}}(\mathbf{Tr}(\mathsf{C}^{M_1}_{\rho,c})) = \mathbf{Rename}^{\Delta_{\Gamma,c}}(\mathbf{Tr}(\mathsf{C}^{M_2}_{\rho,c}))$.*

VI. From LTS to Automata (transitions)

Let $\Delta = (\text{TB}, \text{CB}, \rho, c_0, \text{Suc}_{\text{T}}, \text{Suc}_{\text{C}})$ be a name scheme. We wish to arrive at an LTS $\mathcal{L}^{\Delta}_{\text{PTR}}$ generating marked Δ-traces. In particular, the traces should be those arising from an application of $\mathbf{Rename}^{\Delta}(t)$, i.e. they need to include at most one introduction of a marked name, which must be an O-name. Although traces of $\mathcal{L}^{\Delta}_{\text{PTR}}$ will rely on base names, the configurations will distinguish their occurrences via indexing. An ***indexed name*** has the form b^i, where $b \in \text{TB} \cup \text{CB}$, and $i \in \mathbb{N}$ is an ***index***. Indexed names will appear in the domain of components like γ and in terms, but never in traces. Indexed names can also be marked in the same way as others.

To handle the generation of new indices, our configurations will contain a function $\eta : \text{TB} \cup \text{CB} \to \mathbb{N}$ mapping each base name to the next available index. In Figure 10 we define new versions of functions analogous to $\mathbf{AVal}_{\sigma}(V)$, $\mathbf{BVals}^{\Delta}_{\sigma}(d)$, and $\mathbf{BValSeq}^{\Delta}(f)$ but taking an additional argument η. They generate abstract values with indexed names and an updated η'. Given an abstract value or sequence with indexed names, we write $\beta(A)$ to denote the same abstract value with indices removed (but preserving marks).

Note that a typical update to η will make the values grow. In order to keep them bounded, we will implement a recycling scheme for indices. In order to formulate it, we need to transfer the notion of level to thunk names in a trace. Let t be an N_O-trace such that Δ agrees with N_O. Then f in t is a level-n name if $\mathbf{Base}^{\Delta}_{t}(f)$ is a level-n name, and g is the originator of f if g can be reached from f by following the introduction of head names, and $\mathbf{Base}^{\Delta}_{t}(g)$ is the originator of $\mathbf{Base}^{\Delta}_{t}(f)$.

Example 41. In trace $\mathtt{t}$ in Example 8, f is a level-0 name, r, w, g are level 1, and h is level 2.

Our recycling scheme is inspired by the Lemmata below.

Lemma 42. *Let $c : \sigma^0$ be a continuation name (one which corresponds to returning a value of a basic type). Then, for any O/P-visible, and O/P-bracketed trace $s =$*

$$
\begin{array}{rcl}
\mathbf{Base}^{\Delta}(\mathbf{Tr}(\mathsf{C}^{\rho,c_0}_{M_i})) & = & \{\mathtt{t} \text{ where } \mathtt{t} = \bar{f}(\epsilon,d)\ d(g)\ \bar{f}(\epsilon,d)\ d(g)\ \bar{g}(\epsilon,e)\ e(())\ \bar{c_0}(()) \\
\mathbf{Rename}^{\Delta}(\mathbf{Tr}(\mathsf{C}^{\rho,c_0}_{M_1})) & = & \{\mathtt{t}, \bar{f}(\epsilon,d)\ d(\hat{g})\ \bar{f}(\epsilon,d)\ d(g)\ \bar{\hat{g}}(\epsilon,e)\ e(())\ \bar{c_0}(()), \bar{f}(\epsilon,d)\ d(g)\ \bar{f}(\epsilon,d)\ d(\hat{g})\ \bar{g}(\epsilon,e)\ e(())\ \bar{c_0}(())\} \\
\mathbf{Rename}^{\Delta}(\mathbf{Tr}(\mathsf{C}^{\rho,c_0}_{M_2})) & = & \{\mathtt{t}, \bar{f}(\epsilon,d)\ d(\hat{g})\ \bar{f}(\epsilon,d)\ d(g)\ \bar{g}(\epsilon,e)\ e(())\ \bar{c_0}(()), \bar{f}(\epsilon,d)\ d(g)\ \bar{f}(\epsilon,d)\ d(\hat{g})\ \bar{\hat{g}}(\epsilon,e)\ e(())\ \bar{c_0}(())\}
\end{array}
$$

Fig. 9. Translation of traces for Examples 37 and 38

$$
\begin{array}{lcl}
\mathbf{IVal}^{\Delta}_{\sigma}(d,V,\eta) & \triangleq & (V,\emptyset,\eta) \quad \text{for } \sigma \in \{\mathrm{Unit},\mathrm{Int}\} \\
\mathbf{IVal}^{\Delta}_{U\tau}(d,V,\eta) & \triangleq & (f^i,[f^i \mapsto V],\eta[f \mapsto i+1]) \text{ where } \mathrm{Suc_T}(d) = f, \eta(f) = i \\
\mathbf{IVal}^{\Delta}_{\mathrm{Ref}}(d,\{\!|V_1,V_2|\!\},\eta) & \triangleq & (\{\!|f^{\eta(f)},g^{\eta(j)}|\!\},[f^{\eta(f)} \mapsto V_1,\ g^{\eta(g)} \mapsto V_2],\eta[f,g \mapsto \eta(f)+1,\eta(g)+1]) \text{ where } \mathrm{Suc_T}(d) = (f,g) \\
\mathbf{IVal}^{\Delta}_{\mathrm{Ref}}(d,\ell,\eta) & \triangleq & (\{\!|f^i,g^j|\!\},[f^i \mapsto \text{thunk } (!\ell),\ g^j \mapsto \text{thunk } (\lambda x.\ell := x)],\eta[f,g \mapsto i+1,j+1]) \\
 & & \text{where } \mathrm{Suc_T}(d) = (f,g), \eta(f) = i, \eta(g) = j \\
\mathbf{IVal}^{\Delta}(f,\overrightarrow{V},\eta_0) & \triangleq & (A_1 \cdots A_k, \gamma_1 \cdot \gamma_2 \cdots \gamma_k, \eta_k) \text{ where } f : U(\sigma_1 \to \cdots \to \sigma_k \to F\sigma), \overrightarrow{V} = V_1 \cdots V_k \\
 & & \text{and for } 1 \le i \le k, (A_i,\gamma_i,\eta_i) = \mathbf{IVal}^{\Delta}_{\sigma_i}((f,i),V_i,\eta_{i-1})
\end{array}
$$

$$
\begin{array}{lcll}
\mathbf{IVals}^{\Delta}_{\sigma}(d,\eta) & \triangleq & \{(V,\eta) \mid V : \sigma\} & \text{where } \sigma \in \{\mathrm{Int},\mathrm{Unit}\} \\
\mathbf{IVals}^{\Delta}_{U\tau}(d,\eta) & \triangleq & \{(f^i,\eta[f \mapsto i+1])\} & \text{where } \mathrm{Suc_T}(d) = f,\ \eta(f) = i \\
\mathbf{IVals}^{\Delta}_{\mathrm{Ref}}(d,\eta) & \triangleq & \{(\mathrm{MkVar}\ f^i\ g^j,\eta[f,g \mapsto i+1,j+1])\} & \text{where } \mathrm{Suc_T}(d) = (f,g), \eta(f) = i, \eta(g) = j \\
\mathbf{IValSeq}^{\Delta}(f,\eta_0) & \triangleq & \{(B_1 \cdots B_k,\eta_k) \mid (B_i,\eta_i) \in \mathbf{IVals}^{\Delta}_{\sigma_i}((f,i),\eta_{i-1})\} & \text{where } f : U(\sigma_1 \to \cdots \to \sigma_k \to F\sigma)
\end{array}
$$

Fig. 10. Functions for decomposing values in indexed abstract values and maps, for $\Delta = (\mathrm{TB},\mathrm{CB},\rho,c_0,\mathrm{Suc_T},\mathrm{Suc_C})$

$t\ f(\overrightarrow{A},c)\ t'\ \bar{c}(A')\ t''$, no names introduced in $f(\overrightarrow{A},c)\ t'$ appear in $\mathsf{Vis}_O(s)$ (if s ends in a P-action) or $\mathsf{Vis}_P(s)$ (if s ends in an O-action).

Lemma 43. *Let $s = t\ f(\overrightarrow{A},c)\ t'\ \bar{g}(\overrightarrow{A'},d)$ and $s' = s\ t''\ d(A)$ be* PTR *$(N_O,\emptyset)$-traces, where g is a level-2 name whose originator is introduced in $\overrightarrow{A}$. Let X be the names introduced in $f(\overrightarrow{A},c)\ t'\ \bar{g}(\overrightarrow{A'},d)$. Then if s'' is a proper prefix of s' at least as long as s, $\mathsf{Vis}_O(s'') \cap X = \emptyset$ (if s'' ends in a P-action) and $\mathsf{Vis}_P(s'') \cap X = \emptyset$ (if s'' ends in an O-action).*

Note that both Lemmata state that certain names become unavailable. In the first case this deactivation is permanent after $\bar{c}(A')$, whereas in the second case it is temporary: it starts after a level-2 name is used in $\bar{g}(\overrightarrow{A'},d)$ and ends after the corresponding $d(A)$. We will take advantage of the deactivation period to reuse the deactivated indices. In the first case, this will be done simply by resetting the relevant bounds. In the PTR fragment, all non-initial continuation names have type σ^0, so this recycling is actually widely applicable. In the second case, we will reset the parameters temporarily and, to be able to restore them, will push the information related to deactivated names on the stack (PQ). It can then be restored during the matching pop (OA).

Example 44. To better explain why this second scheme is necessary, recall Γ and name scheme $\Delta_{\Gamma,F\mathrm{Unit}}$ from Example 31, and consider the computation $\Gamma \vdash^c (\text{force } f)(\text{thunk } \lambda i.\lambda j.(\text{force } i))$ to $g.\text{force } g : F\mathrm{Unit}$. Some of the associated traces, written with base names and specific indices, have the following shape:

$$\bar{f^0}(h^0,c_f^0)\ h^0(i^0\ j^0,c_h^0)\ \bar{i^0}(\epsilon,c_i^0)\ h^0(i^1\ j^1,c_h^1)\ \bar{i^1}(\epsilon,c_i^1)\ \ldots.$$

What happens here is that, when P calls i^n, h^0 becomes visible to O. This allows O to call h^0 again with i^{n+1}. As this can repeat unboundedly many times, we must recycle the indices on i and j, which is what the second recycling scheme permits.

Before we can present $\mathcal{L}^{\Delta}_{\mathrm{PTR}}$ we will need one final element, modifications to the operational semantics given in Figure 3. Their purpose is to replace the generation of arbitrary new locations with locations drawn sequentially from $\mathbb{N}$, similarly to how we intend to use indexed names. This will enable us to exploit the fact that Lemmata 42 and 43 mean that available locations are also restricted. Instead of having configurations of the form (M,h), we have ones of the form (M,h,i_h,η), where i_h is the next available location (and η will be a function as above). The previous operational rules $\to$ (save those for $\mathrm{ref}\ V$ and $\mathrm{while}\ M\ \mathrm{do}\ N$) are embedded into the new reduction $\to_e$ using the rule

$$\frac{(M,h) \to (M',h')}{(M,h,i_h,\eta) \to_e (M',h',i_h,\eta)}.$$

The reduction rule for handling new references is replaced by $(K[\mathrm{ref}\ V],h,i_h,\eta) \to_e (K[i_h],h \cdot [i_h \mapsto V],i_h+1,\eta)$. It uses i_h as the location for the new reference, and then sets the next location to be i_h+1. This gives an operational semantics which is behaviorally the same as generating a fresh location, so long as i_h is larger than any name appearing in h.

We also make changes to handle the $\mathrm{while\ do}$ construct. The idea is to reset both i_h and η back to the value before the loop once we reach the end of the loop. This is due to the fact that, by the way the scopes work in the language, any name or location generated in the loop cannot be used outside of (that iteration of) the loop. In particular, we introduce a new construct, $\mathrm{end}(i_h,\eta).M$, to indicate the end of an iteration of a loop. We provide rules for while and end in Figure 11, where $h_{<i_h}$ denotes the heap h restricted to domain of location

smaller than i_h. Similarly, if ζ is a partial map from indexed names, and η maps base names to indices, we write ζ_η to mean ζ restricted to indexed names f^i for which $i < \eta(f)$. We will use $h_{\geq i_h}$ and $\zeta_{\geq \eta}$ analogously.

Finally, we present the LTS $\mathcal{L}^{\Delta}_{\mathrm{PTR}}$ in Figure 12. Active configurations of $\mathcal{L}^{\Delta}_{\mathrm{PTR}}$ have the form $\langle M, c, \gamma, h, H, i_h, \eta, \mu, l\rangle$ and passive ones $\langle \gamma, h, H, Fn, i_h, \eta, \mu, l\rangle$. As described above, η is a function from base names to the next available index, which we call the ***(next) index component***. i_h is the ***(next) location component***, the next available location. μ is the ***reset*** component, a partial map from (indexed) level-2 thunk names and O-continuation names to the value of (i_h, η) prior to the move that introduced the name. l is a binary flag used to indicate whether a marked name has been produced in the trace so far.

We now need to define initial configurations. Let $\Gamma \vdash^c M : F\sigma$ be a PTR computation and $\Delta = (\mathrm{TB}, \mathrm{CB}, \rho, c_0, \mathrm{Suc_T}, \mathrm{Suc_C})$ be a $(\Gamma, F\sigma)$-name scheme. Let $\rho^0 = [x_i \mapsto \rho(x_i)^0]$, $N_O = \nu(\rho) \cup \{c_0\}$ and $N_O^0 = \{n^0 \mid n \in N_0\}$. Then the active initial configuration $\mathsf{C}_M^{\mathrm{PTR},\Delta}$ is defined to be

$$(\langle M\{\rho^0\}, c_0^0, \emptyset, \emptyset, [N_O^0 \mapsto \emptyset], 0, \eta, \emptyset, 0\rangle, \bot)$$

where $\eta = [N_O, c_0 \mapsto 1] \cdot [(\mathrm{TB} \cup \mathrm{CB}) \setminus (N_O \cup \{c_0\}) \mapsto 0]$.

The main change to the LTS, is to 'recycle' the indices, so as to keep the space of reachable configurations finite. This is the role of the μ component, based on the properties identified in Lemmata 42 and 43. In particular, after a PA-action (other than on the initial continuation name), we 'prune' the domains of the components to the index names and locations introduced before the OQ-action being answered. Similarly, after a PQ-action on a level-2 name f, we split the components between the index names and locations introduced before the OQ-action introducing the originator of f, and those after. Those from before the OQ-action become part of the next configuration, whereas those from after are stored on the stack until they can be restored after the matching OA-action. Let $\mathbf{Tr}^{\Delta}_{\mathrm{PTR}}(\mathbf{C})$ be the set of base traces generated from $\mathbf{C}$ in $\mathcal{L}^{\Delta}_{\mathrm{PTR}}$.

Definition 45. The PTR***-trace semantics*** of a PTR-computation $\Gamma \vdash^c M : F\sigma$ is defined to be $\mathbf{Tr}_{\mathrm{PTR}}(\Gamma \vdash^c M : F\sigma) \triangleq \{\ (\Delta, t) \mid \Delta$ is a $(\Gamma, F\sigma)$-name scheme, $t \in \mathbf{Tr}^{\Delta}_{\mathrm{PTR}}(\mathsf{C}_M^{\mathrm{PTR},\Delta}),\ t$ is complete$\}$.

We can show that the new semantics agree with the full trace semantics on PTR-computations.

Lemma 46. *For any* PTR*-computation* $\Gamma \vdash M : F\sigma$*, a* $(\Gamma, F\sigma)$*-name scheme* $\Delta = (\mathrm{TB}, \mathrm{CB}, \rho, c_0, \mathrm{Suc_T}, \mathrm{Suc_C})$*,* $\mathbf{Tr}^{\Delta}_{\mathrm{PTR}}(\mathsf{C}_M^{\mathrm{PTR},\Delta}) = \mathbf{Rename}^{\Delta}(\mathbf{Tr}_{\mathrm{CBPV}}(\mathsf{C}_M^{\rho,c_0}))$.

Lemma 46 with Corollaries 24 and 40 imply the following.

Theorem 47 (PTR Full Abstraction)**.** *For any* PTR *computations* $\Gamma \vdash M_1, M_2 : F\sigma$*, then* $\Gamma \vdash M_1 \lesssim^{\mathrm{CBPV}}_{ter} M_2$ *iff* $\mathbf{Tr}_{\mathrm{PTR}}(\Gamma \vdash M_1) \subseteq \mathbf{Tr}_{\mathrm{PTR}}(\Gamma \vdash M_2)$.

$\mathcal{L}^{\Delta}_{\mathrm{PTR}}$ turns out to be a VPA for any PTR-computation. In general, it inherits the non-elementary bounds from the λ-calculus but, for terms in canonical form (see Figure 13), we obtain an exponential bound.

Lemma 48. *For* PTR*-computation* $\Gamma \vdash^c M : F\sigma$*,* (Γ, σ)*-name scheme* Δ*, the set of of states reachable from* $\mathsf{C}_M^{\mathrm{PTR},\Delta}$ *in* $\mathcal{L}^{\Delta}_{\mathrm{PTR}}$ *is finite. If* M *is in canonical form, it is exponential in the size of* M*.*

Lemma 49. *For* PTR*-computation* $\Gamma \vdash^c M : F\sigma$ *and* (Γ, σ)*-name scheme* Δ*, one can effectively construct a deterministic VPA accepting* $\mathbf{Tr}^{\Delta}_{\mathrm{PTR}}(\mathsf{C}_M^{\mathrm{PTR},\Delta})$*. If* M *is in canonical form, the construction can be carried out in exponential time.*

VII. Decidability, complexity and translations

Theorem 50. *Contextual approximation for the* PTR*-fragment of* CBPV *is decidable. For computations in canonical form, it is decidable in exponential time.*

Proof. From Theorem 47, testing two computations $\Gamma \vdash^c M_1, M_2 : F\sigma$ for contextual approximation can be done by comparing the complete traces generated by $\mathcal{L}^{\Delta}_{\mathrm{PTR}}$ for every possible name scheme Δ. As choice of base names in Γ, σ is arbitrary, we need only care about the Ints occurring in Γ, which gives exponentially many Δ. By Lemma 49, each comparison reduces to a language equivalence test. For canonical forms, the two VPA's are constructible in exponential time. In particular, they will be of exponential size. Because language equivalence is in P for deterministic VPA, the lemma follows. □

One can show that it is the use of level-2 names that forces us to make use of an unbounded stack. The computations that omit level-2 names are of the form $\Gamma \vdash^c M : F\sigma^1$, where each type in Γ is a σ^2 type according to the grammar given below.

$$\begin{array}{lcllcl} \sigma^2 & \triangleq & \sigma^1 \mid U\underline{\tau}^1 & \sigma^1 & \triangleq & \sigma^0 \mid \mathrm{Ref} \mid U\underline{\tau}^0 \\ \underline{\tau}^1 & \triangleq & F\sigma^2 \mid \sigma^1 \to \underline{\tau}^1 & \underline{\tau}^0 & \triangleq & F\sigma^0 \mid \sigma^0 \to \underline{\tau}^0 \\ \sigma^0 & \triangleq & \mathrm{Int} \mid \mathrm{Unit} & & & \end{array}$$

In this case one can show that the stack height is bounded and, for canonical forms, the bound is linear. Consequently, we can treat the (bounded) stack as part of the state space and convert the VPA to a finite-state machine.

The fact that our results are stated for CBPV makes it possible to specialise them to the CBN- and CBV-variants of the language, known in the literature as Idealised Algol [7] and RML [6] respectively. This can be done by translation provided it is fully abstract (preserves and reflects contextual equivalence). Our translations extend the standard translations from the CBN and CBV λ-calculus respectively [21]. The translations of types are given in the table below. For RML, a term $M : \sigma$ is translated into a computation $M^{\mathrm{RML}} : F\sigma^{\mathrm{RML}}$.

RML type	CBPV value types
$\mathrm{Int}, \mathrm{Unit}, \mathrm{Ref}$	$\mathrm{Int}, \mathrm{Unit}, \mathrm{Ref}$
$\sigma_1 \to \sigma_2$	$U({\sigma_1}^{\mathrm{RML}} \to F{\sigma_2}^{\mathrm{RML}})$

$(K[\text{while } M \text{ do } N], h, i_h, \eta) \to_e (K[M \text{ to } x.\text{case } x \text{ of return } (), (N \text{ to } y.\text{end}(i_h, \eta).\text{while } M \text{ do } N)_{j>0}], h, i_h, \eta)$
$(K[\text{end}(i_h, \eta').\text{while } M \text{ do } N], h, j_h, \eta) \to_e (K[M \text{ to } x.\text{case } x \text{ of return } (), (N \text{ to } y.\text{end}(i_h, \eta).\text{while } M \text{ do } N)_j], h_{<i_h}, i_h, \eta')$

Fig. 11. The modifications needed to produce reduction relation $\to_e$

$$
\begin{array}{ll|ll}
(P\tau) & \langle M, c^j, \gamma, h, H, i_h, \eta, \mu, l\rangle & \xrightarrow{\tau} & \langle N, c^j, \gamma_{<\eta'}, h', H_{<\eta'}, i'_h, \eta', \mu_{<\eta'}, l\rangle \\
 & \text{when } (M, h, i_h, \eta) \to_e (N, h', i'_h, \eta') & & \\
(PA) & \langle \text{return } V, c_0^0, \gamma, h, H, i_h, \eta, \mu, l\rangle & \xrightarrow{\bar{c_0}(\beta(A))} & \langle \gamma \cdot \gamma', h, H, H(c_0) \uplus \nu(A), i_h, \eta', \mu, l\rangle \\
 & \text{when } c_0 : \sigma,\ (A, \gamma', \eta') = \mathbf{IVal}_\sigma^\Delta(c_0, V, \eta) & & \\
(PA) & \langle \text{return } V, c^i, \gamma, h, H, i_h, \eta, \mu, l\rangle & \xrightarrow{\bar{c}(V)} & \langle \gamma_{<\eta'}, h_{<i'_h}, H_{<\eta'}, H(c^i), i'_h, \eta', \mu_{<\eta'}, l\rangle \\
 & \text{when } c \neq c_0 \text{ and } (i'_h, \eta') = \mu(c^i) & & \\
(PQ) & \langle K[(\text{force } f^i)\overrightarrow{V}], c'^j, \gamma, h, H, i_h, \eta, \mu, l\rangle & \xrightarrow{\bar{f}(\beta(\overrightarrow{A}), c)/(c^0, (K, c'^j))} & \langle \gamma \cdot \gamma', h, H, H(f^i) \uplus \nu(\overrightarrow{A}), i_h, \eta', \mu, l\rangle \\
 & \text{when } f \text{ is not a level 2 name, } (\overrightarrow{A}, \gamma', \eta') \in \mathbf{IVal}^\Delta(f, \overrightarrow{V}, \eta), \text{ and } \mathrm{Suc_C}(f) = c & & \\
(PQ) & \langle K[(\text{force } f^i)\overrightarrow{V}], c'^j, \gamma, h, H, i_h, \eta, \mu, l\rangle & \xrightarrow{\bar{f}(\overrightarrow{V}, c)/(c^0, (K, c'^j), P)} & \langle \gamma_{<\eta'}, h_{<i'_h}, H_{<\eta'}, H(f^i), i'_h, \eta', \mu_{<\eta'}, l\rangle \\
 & \text{when } f \text{ is a level 2 name, and } (i'_h, \eta') = \mu(f^i),\ \mathrm{Suc_C}(f) = c, \text{ and } P = (i_h, \eta, \gamma_{\geq\eta'}, h_{\geq i'_h}, H_{\geq\eta'}, \mu_{\geq\eta'}) & & \\
(OA) & \langle \gamma, h, H, Fn, i_h, \eta, \mu, l\rangle & \xrightarrow{c(\beta(A)), (c^0, (K, c'^j))} & \langle K[\text{return } A], c'^j, \gamma, h, H \cdot [\nu(A) \mapsto Fn], i_h, \eta', \mu, l'\rangle \\
 & \text{when } c : \sigma,\ (A', \eta') \in \mathbf{IVals}_\sigma^\Delta(c, \eta) \text{ and if } l = 1 \text{ then } A = A', l' = 1 \text{ else } A \in \mathbf{Select}(A'), \text{ and } l' = \mathbf{IsMark}(A) & & \\
(OA) & \langle \gamma, h, H, Fn, i_h, \eta, \mu, l\rangle & \xrightarrow{c(\beta(A)), (c^0, (K, c'^j), P)} & \langle K[\text{return } A], c'^j, \gamma \cdot \gamma', h, H', i'_h, \eta', \mu', l'\rangle \\
 & \text{when } c : \sigma,\ P = (i'_h, \eta'', \gamma', h', H'', \mu''),\ (A', \eta') \in \mathbf{IVals}_\sigma^\Delta(c, \eta'') \text{ and if } l = 1 \text{ then } A = A', l' = 1 & & \\
 & \text{else } A \in \mathbf{Select}(A'), \text{and } l' = \mathbf{IsMark}(A); \text{ and } H' = H \cdot H'' \cdot [\nu(A) \mapsto Fn], \text{ and } \mu' = \mu \cdot \mu'' \cdot [\nu(A) \mapsto (i_h, \eta)] & & \\
(OQ) & \langle \gamma, h, H, Fn, i_h, \eta, \mu, l\rangle & \xrightarrow{f(\beta(\overrightarrow{A}), c)} & \langle \text{force } V\overrightarrow{A}, c^j, \gamma, h, H \cdot [\nu(\overrightarrow{A}), c^j \mapsto Fn], i_h, \eta', \mu', l\rangle \\
 & \text{when } f^i \in Fn,\ (\overrightarrow{A'}, \eta'') \in \mathbf{IValSeq}^\Delta(f, \eta),\ \mathrm{Suc_C}(f) = c,\ \eta(c) = j, \eta' = \eta''[c \mapsto j+1], \gamma(f^i) = V, \text{ and} & & \\
 & \text{if } l = 1 \text{ then } A = A', l' = 1 \text{ else } A \in \mathbf{Select}(A'), \text{ and } l' = \mathbf{IsMark}(A); \text{ and } \mu' = \mu \cdot [\nu(\overrightarrow{A}), c^j \mapsto (i_h, \eta)] & &
\end{array}
$$

In the PQ rules, the name f can be either marked or unmarked. In the second PA (PQ), V ($\overrightarrow{V}$) does not contain thunks, so is an abstract value. The second OA rule is sound as γ', h', H'', μ'' are disjoint from γ, h, H, μ. $\mathbf{Select}(A)$ is the set of marked indexed abstract values obtained by marking at most one name in A. $\mathbf{IsMark}(A) = 1$ if a name in A is marked, 0 otherwise.

Fig. 12. $\mathcal{L}_{\mathrm{PTR}}^\Delta$ transition rules for name scheme $\Delta = (\mathrm{TBNames}, \mathrm{CBNames}, \rho, c_0, \mathrm{Suc_T}, \mathrm{Suc_C})$

$$
\begin{array}{lll}
\text{Ground Types} & \beta & \triangleq \ \text{Unit} \mid \text{Int} \\
\text{Restricted Values} & V_0 & \triangleq \ x \mid () \mid \widehat{n} \mid \ell \mid \text{MkVar } V_0\ V_0 \\
\text{Values} & V & \triangleq \ V_0 \mid \text{thunk } M \mid \text{MkVar (thunk } M\text{) (thunk } M\text{)} \\
\text{Restricted Computations} & M_0 & \triangleq \ \text{force } V_0 \mid \text{return } V_0 \mid M_0 V \mid \text{ref } V \mid !V_0 \mid V_0 := V_0 \\
\text{Computations} & M & \triangleq \ M_0 \mid \text{return } V \mid \lambda x^\sigma.M \mid \text{let } x^\beta \text{ be } V.M \\
 & & \quad \mid M \text{ to } x^\beta.M \mid M_0 \text{ to } x.M \mid \text{case } V \text{ of } (M_i)_{i\in I} \mid \text{while } M \text{ do } M
\end{array}
$$

Fig. 13. The grammar for terms in canonical form

IA type	CBPV computation types
$\underline{\text{expr}}, \underline{\text{com}}$	$F\text{Int}, F\text{Unit}$
$\underline{\text{var}}$	$\text{Int} \to \text{Int} \to F\text{Int}$
$\underline{\tau_1} \to \underline{\tau_2}$	$U\underline{\tau_1}^{\mathrm{IA}} \to \underline{\tau_2}^{\mathrm{IA}}$

Remark 51. The CBN translation of $\underline{\text{var}}$ into $\text{Int} \to \text{Int} \to F\text{Int}$ uses the first argument as a boolean flag to indicate whether reading or writing will take place. The term translation ensures that, during reading, the second parameter will be ignored. For writing, the translated term will always return 0. If the first argument is different from 0 or 1, the translated term will diverge.

That the translations turn out fully abstract is not completely surprising: there are several similar results in the literature, though none of them applies to the framework we are considering, e.g. the results from [21] are phrased for higher-order references and observing output instead of termination. Our fully abstract model $\mathcal{L}_{\mathrm{CBPV}}$ plays a crucial role in establishing the full abstraction of our translations.

Theorem 52. *Let* $\lesssim_{ter}^{\mathrm{IA}}$, $\lesssim_{ter}^{\mathrm{RML}}$ *be the notions of contextual approximation in* IA *and* RML *respectively. For IA terms* $\Gamma \vdash M_1 \lesssim_{ter}^{\mathrm{IA}} M_2$ *iff* $\Gamma^{\mathrm{IA}} \vdash^c {M_1}^{\mathrm{IA}} \lesssim_{ter}^{\mathrm{CBPV}} {M_2}^{\mathrm{IA}}$*, and for RML terms* $\Gamma \vdash M_1 \lesssim_{ter}^{\mathrm{RML}} M_2$ *iff* $\Gamma^{\mathrm{RML}} \vdash^c {M_1}^{\mathrm{RML}} \lesssim_{ter}^{\mathrm{CBPV}} {M_2}^{\mathrm{RML}}$.

The above result means that Theorem 50 subsumes existing decidability results for IA and RML from [10], [12], as the translations of third-order IA types [10] and O-strict RML types [12] belong to the PTR fragment. Consequently, the present results can be seen as an operational explanation of the earlier results for CBN and CBV.

VIII. Conclusion

We demonstrated an approach to proving decidability results for contextual equivalence by deriving decidable automata models from labelled transitions systems through a series of relatively easy optimisations. The configurations of these automata retain operational character, which makes them suitable for specification of further program analysis tasks. Since operational game models are in general easier to construct and understand, we believe the approach is likely to turn out fruitful when it comes to analyzing more complicated frameworks in the future.

References

[1] H. Nickau, "Hereditarily Sequential Functionals," in *Proceedings of LFCS*, ser. LNCS, vol. 813. Springer, 1994, pp. 253–264.

[2] S. Abramsky, R. Jagadeesan, and P. Malacaria, "Full Abstraction for PCF," *Inf. Comput.*, vol. 163, no. 2, pp. 409–470, 2000.

[3] J. M. E. Hyland and C. L. Ong, "On Full Abstraction for PCF: I, II, and III," *Inf. Comput.*, vol. 163, no. 2, pp. 285–408, 2000.

[4] J. Laird, "A Fully Abstract Trace Semantics for General References," in *Proceedings of ICALP*, ser. LNCS, vol. 4596. Springer, 2007, pp. 667–679.

[5] G. Jaber, "Operational Nominal Game Semantics," in *Proceedings of FoSSaCS*, ser. LNCS, vol. 9034. Springer, 2015, pp. 264–278.

[6] S. Abramsky and G. McCusker, "Call-by-Value Games," in *Proceedings of CSL*, ser. LNCS, vol. 1414. Springer, 1997, pp. 1–17.

[7] ——, "Linearity, Sharing and State: a fully abstract game semantics for Idealized Algol with active expressions," in *Algol-like languages*, P. W. O'Hearn and R. D. Tennent, Eds. Birkhaüser, 1997, pp. 297–329.

[8] S. Abramsky, D. R. Ghica, A. S. Murawski, and C.-H. L. Ong, "Applying Game Semantics to Compositional Software Modelling and Verification," in *Proceedings of TACAS*, ser. LNCS, vol. 2988. Springer, 2004, pp. 421–435.

[9] D. R. Ghica and G. McCusker, "The regular language semantics of second-order Idealized Algol," *Theor. Comput. Sci.*, vol. 309, pp. 469–502, 2003.

[10] A. S. Murawski and I. Walukiewicz, "Third-order Idealized Algol with iteration is decidable," *Theor. Comput. Sci.*, vol. 390, no. 2-3, pp. 214–229, Jan. 2008.

[11] A. S. Murawski, "Functions with local state: Regularity and Undecidability," *Theor. Comput. Sci.*, vol. 338, no. 1-3, pp. 315–349, 2005.

[12] D. Hopkins, A. S. Murawski, and C.-H. L. Ong, "A Fragment of ML Decidable by Visibly Pushdown Automata," in *Proceedings of ICALP*, ser. LNCS, vol. 6756. Springer, 2011, pp. 149–161.

[13] S. B. Lassen and P. B. Levy, "Typed Normal Form Bisimulation," in *Proceedings of CSL*, ser. LNCS, vol. 4646. Springer, 2007, pp. 283–297.

[14] P. B. Levy, "Call-by-push-value: Decomposing call-by-value and call-by-name," *High. Order Symb. Comput.*, vol. 19, no. 4, pp. 377–414, 2006.

[15] G. Jaber, "SyTeCi: automating contextual equivalence for higher-order programs with references," *Proc. ACM Program. Lang.*, vol. 4, no. POPL, pp. 59:1–59:28, 2019.

[16] V. Koutavas, Y. Y. Lin, and N. Tzevelekos, "From Bounded Checking to Verification of Equivalence via Symbolic Up-to Techniques," in *Proceedings of TACAS*, ser. LNCS, vol. 13244. Springer, 2022, pp. 178–195.

[17] G. Jaber and A. S. Murawski, "Compositional relational reasoning via operational game semantics," in *Proceedings of LICS*. IEEE, 2021, pp. 1–13.

[18] A. M. Pitts and I. D. B. Stark, "Operational Reasoning for Functions with Local State," in *Higher-Order Operational Techniques in Semantics*, A. D. Gordon and A. M. Pitts, Eds. CUP, 1998, pp. 227–273.

[19] D. Dreyer, G. Neis, and L. Birkedal, "The impact of higher-order state and control effects on local relational reasoning," *J. Funct. Program.*, vol. 22, no. 4-5, pp. 477–528, 2012.

[20] D. Biernacki, S. Lenglet, and P. Polesiuk, "A Complete Normal-Form Bisimilarity for State," in *Proceedings of FoSSaCS*, ser. LNCS, vol. 11425. Springer, 2019, pp. 98–114.

[21] P. B. Levy, *Call-By-Push-Value. A Functional/Imperative Synthesis*, ser. Semantics Structures in Computation. Springer, 2004, vol. 2.

[22] J. C. Reynolds, "The Essence of Algol," in *Algol-like languages*, P. W. O'Hearn and R. D. Tennent, Eds. Birkhaüser, 1997, pp. 67–88.

[23] C. Talcott, "Reasoning about Programs With Effects," *Electron. Notes Theor. Comput. Sci.*, vol. 14, pp. 301–314, 1998.

[24] G. Jaber and A. S. Murawski, "Complete trace models of state and control," in *Proceedings of ESOP*, ser. LNCS, vol. 12648. Springer, 2021, pp. 348–374.

[25] S. Abramsky, K. Honda, and G. McCusker, "A Fully Abstract Game Semantics for General References," in *Proceedings of LICS*. IEEE, 1998, pp. 334–344.

[26] J. Laird, "Full Abstraction for Functional Languages with Control," in *Proceedings of LICS*. IEEE, 1997, pp. 58–67.

[27] R. Alur and P. Madhusudan, "Visibly Pushdown Languages," in *Proceedings of STOC*. ACM, 2004, pp. 202–211.