A Bisimulation-Like Proof Method for Contextual Properties in Untyped λ-Calculus with References and Deallocation[☆]

Eijiro Sumii

Graduate School of Information Sciences, Tohoku University, Aoba-ku Aramki Aza-aoba 6-3-09, Sendai 980-8579, Japan

Abstract

We develop a general method of proving properties of programs *under arbitrary contexts*—including (but not limited to) observational equivalence, space improvement, and a form of memory safety of the programs—in untyped call-by-value λ-calculus with first-class, dynamically allocated, higher-order references and deallocation. The method generalizes Sumii et al.'s environmental bisimulation technique, and gives a sound and complete characterization of each proved property, in the sense that the "bisimilarity" (the largest set satisfying the bisimulation-like conditions) equals the set of terms with the property to be proved. We give examples of contextual properties concerning typical data structures such as linked lists, binary search trees, and directed acyclic graphs with reference counts, all with deletion operations that release memory. This shows the scalability of the environmental approach from contextual equivalence to other binary relations (such as space improvement) and unary predicates (such as memory safety), as well as to languages with non-monotone store.

1. Introduction

1.1. Background

Memory management is tricky, be it manual or automatic. Manual memory management is notoriously difficult, leading to memory leaks and segmentation faults (or, even worse, security holes). Automatic memory management is usually more convenient. Still, real programs often suffer from performance

---

problems—in terms of both memory and time—due to automatic memory management, and require manual tuning. In addition, implementing memory management routines—such as memory allocators and garbage collectors—is even harder than writing programs that use them.

To address these problems, various theories for safe memory management have been developed, including linear types (Wadler, 1990), regions (Tofte and Talpin, 1994), and the capability calculus (Crary et al., 1999), just to name a few. These approaches typically conduct a sound and efficient static analysis—often based on types—on programs, and guarantee their memory safety. However, since static analyses are necessarily incomplete in the sense that some safe programs are rejected, the programs usually have to be written in a style that is accepted by the analysis.

1.2. Our contributions

In this paper, we develop a different approach, originating from Sumii et al.'s environmental bisimulations (Sumii and Pierce, 2007a,b; Koutavas and Wand, 2006; Sangiorgi et al., 2007). Unlike most static analyses, our method is not fully automated, but is (sound and) complete in the sense that all (and only) safe programs can potentially be proved safe. Moreover, it guarantees a form of memory safety of the programs *under any context*, even if the context—or, in fact, the whole language—is untyped.

For instance, consider the triple *dag* of functions in Figure 1, which implements an abstract data type—a directed acyclic graph object, with addition and deletion operations and garbage collection by reference counting—using deallocation. (Details of this implementation are not important now and will be explained in Section 9. The formal syntax and semantics of our language will be given in Section 3.) To prove the memory safety of such an implementation, it makes no sense to evaluate the tuple of functions by itself, because they are just functions and do no harm (or good) unless applied. Rather, we must consider all possible uses of it, i.e., put it under arbitrary contexts. Our method gives such a proof.

Because our method is based on a relational technique (namely, bisimulations), we can also prove binary properties such as observational equivalence, in addition to unary properties such as memory safety. Furthermore, we can prove stronger binary properties than observational equivalence, like "the memory usage (i.e., number of locations) is the same on the left hand side and the right" or "the left hand side uses less memory than the right" (cf. Gustavsson and Sands (1999)). Again, our proof assures that such properties of programs are preserved by arbitrary contexts in the language, like contextual equivalence (Morris, 1968).

1.3. Our approach

1.3.1. Environmental bisimulations

Suppose that we want to prove the equivalence of two programs e and e'. (Throughout this paper, we often follow the notational convention that metavariables with $'$ are used for objects on the right hand side of binary relations,

$$
\begin{aligned}
dag \quad &= \quad \texttt{new } z := \texttt{null};\\
&\qquad \langle addn, deln, gc \rangle\\[1em]
addn \quad &= \quad \lambda \langle x, p \rangle.\\
&\qquad x + 0;\\
&\qquad map(\lambda y.\, y + 0)p;\\
&\qquad incr_x(!z)p;\\
&\qquad \texttt{new } n := \langle x, \texttt{true}, 0, p, !z \rangle;\\
&\qquad z := n\\
incr_x \quad &= \quad \texttt{fix } f(n).\, \lambda p.\\
&\qquad \texttt{ifnull } n \texttt{ then } \langle\rangle \texttt{ else}\\
&\qquad \texttt{if } \#_1(!n) \stackrel{int}{=} x \texttt{ then diverge else}\\
&\qquad \texttt{if } member(\#_1(!n))p \texttt{ then}\\
&\qquad\quad \#_3^5(!n) \leftarrow \#_3(!n) + 1;\\
&\qquad\quad f(n)(remove1\,(\#_1(!n))p)\\
&\qquad \texttt{else}\\
&\qquad\quad f(\#_5(!n))p\\[1em]
deln \quad &= \quad \lambda x.\, deln_x(!z)\\
deln_x \quad &= \quad \texttt{fix } g(n).\\
&\qquad \texttt{ifnull } n \texttt{ then } \langle\rangle \texttt{ else}\\
&\qquad \texttt{if } \#_1(!n) \stackrel{int}{=} x \texttt{ then}\\
&\qquad\quad \#_2^5(!n) \leftarrow \texttt{false}\\
&\qquad \texttt{else}\\
&\qquad\quad g(\#_5(!n))\\[1em]
gc \quad &= \quad \lambda x.\, z := decr(!z)[\,]\\
decr \quad &= \quad \texttt{fix } h(n).\, \lambda p.\\
&\qquad \texttt{ifnull } n \texttt{ then null else}\\
&\qquad \texttt{if } member(\#_1(!n))p \texttt{ then}\\
&\qquad\quad \#_3^5(!n) \leftarrow \#_3(!n) - 1;\\
&\qquad\quad h(n)(remove1\,(\#_1(!n))p)\\
&\qquad \texttt{else if } \#_2(!n) \vee \#_3(!n) > 0 \texttt{ then}\\
&\qquad\quad \#_5^5(!n) \leftarrow h(\#_5(!n))p;\\
&\qquad\quad n\\
&\qquad \texttt{else}\\
&\qquad\quad h(\#_5(!n))(append(\#_4(!n))p)\\
&\qquad\quad \texttt{before free}(n)
\end{aligned}
$$

Figure 1: Directed acyclic graph with garbage collection by reference counting

and ones without for the left hand side and for unary relations.) The basic idea of our approach is to consider the set X of every possible "configuration" of the programs. A configuration takes one of the two forms: $(\mathcal{R}, s \triangleright e, s' \triangleright e')$ and $(\mathcal{R}, s, s')$. The former means that the compared programs e and e' are running under stores s and s', respectively. The latter means that the programs have stopped with stores s and s'. In both forms, $\mathcal{R}$ is a binary relation on values and represents the *knowledge* of a context, called an environment. Informally, $(v, v') \in R$ means that the context has learned v from the program on the left hand side and v' on the right.

For instance, suppose that we have a configuration $(\mathcal{R}, s \triangleright e, s' \triangleright e')$ in X. (Typically, $\mathcal{R}$ is empty at first.) If $s \triangleright e$ reduces to $t \triangleright d$ in one step according to the operational semantics of the language, then it must be that $s' \triangleright e'$ also reduces to some $t' \triangleright d'$ in some number of steps, and the new configuration $(\mathcal{R}, t \triangleright d, t' \triangleright d')$ belongs to X again. Knowledge $\mathcal{R}$ does not change yet, because the context cannot learn anything from these internal transitions.

Now, suppose $(\mathcal{R}, s \triangleright e, s' \triangleright e') \in X$ and e has stopped running, i.e., e is a value v. Then $s' \triangleright e'$ must also converge to some $t' \triangleright w'$, and the context learns the resulting values v and w'. Thus, $\mathcal{R}$ is extended with the value pair (v, w'), and $(\mathcal{R} \cup \{(v, w')\}, s, t')$ must belong to X.

Once the compared programs have stopped, the context can make use of elements from its knowledge to make more observations. For example, suppose $(\mathcal{R}, s, s') \in X$ and $(\ell, \ell') \in \mathcal{R}$. This means that location ℓ (resp. ℓ') is known to the context on the left (resp. right) hand side. If $s = t \uplus \{\ell \mapsto v\}$ and $s' = t' \uplus \{\ell' \mapsto v'\}$ (where $\_ \uplus \{\_ \mapsto \_\}$ denotes store extension), then the context can read the contents v (resp. v') of ℓ (resp. ℓ') on the left (resp. right) hand side, and add them to its knowledge, requiring $(\mathcal{R} \cup \{(v, v')\}, s, s') \in X$.

Or, the contents can be updated with any values composed from the knowledge of the context. That is, for any $(w, w') \in \mathcal{R}^{\star}$, we require $(\mathcal{R}, t \uplus \{\ell \mapsto w\}, t' \uplus \{\ell' \mapsto w'\}) \in X$. Here, $\mathcal{R}^{\star}$ is the *context closure* of $\mathcal{R}$ and denotes the set of (pairs of) terms that can be composed from values in $\mathcal{R}$. Formally, it is defined as

$$\mathcal{R}^{\star} = \{([v_1, \ldots, v_n / x_1, \ldots, x_n]e, [v'_1, \ldots, v'_n / x_1, \ldots, x_n]e) \mid (v_1, v'_1), \ldots, (v_n, v'_n) \in \mathcal{R}, \ fv(e) \subseteq \{x_1, \ldots, x_n\}, \ loc(e) = \emptyset\}$$

where $fv(e)$ is the set of free variables in e and $loc(e)$ is the set of locations that appear in e. The context e above is required to be location-free so that it cannot "guess" locations that are not (yet) known to the context. Note that *known* locations can still be accessed, because they can be substituted into free variables of e.

The context can also deallocate known locations, or allocate fresh ones. For the former case, we require $(\mathcal{R}, t, t') \in X$ for any $(\mathcal{R}, t \uplus \{\ell \mapsto v\}, t' \uplus \{\ell' \mapsto v'\}) \in X$ with $(\ell, \ell') \in \mathcal{R}$. For the latter case, $(\mathcal{R} \cup \{(\ell, \ell')\}, t \uplus \{\ell \mapsto v\}, t' \uplus \{\ell' \mapsto v'\}) \in X$ is required for any $(\mathcal{R}, t, t') \in X$ with fresh ℓ, ℓ' and $(v, v') \in \mathcal{R}^{\star}$.

Of course, there are also conditions for observations on values other than locations. For instance, if $(\mathcal{R}, s, s') \in X$ and $(\lambda x. e, \lambda x. e') \in \mathcal{R}$, then $(\mathcal{R}, s \triangleright (\lambda x. e)v, s' \triangleright (\lambda x. e')v') \in X$ is required for any $(v, v') \in R^{\star}$, because the context

4

can apply any functions it knows $((\lambda x. e, \lambda x. e') \in \mathcal{R})$ to any arguments it can compose $((v, v') \in R^\star)$.

1.3.2. Congruence of environmental bisimilarity

As we shall prove, the largest set X satisfying the above conditions—which exists because all of them are monotone on X—is "contextual" in the following sense (where $\mathcal{R}^\star_{val}$ denotes the restriction of $\mathcal{R}^\star$ to values):

- If a configuration $(\mathcal{R}, s \triangleright e, s' \triangleright e')$ is in X, then its context-closed version $(\mathcal{R}^\star_{val}, s \triangleright E[e], s' \triangleright E[e'])$ is also in X, for any location-free evaluation context E.

- If a configuration $(\mathcal{R}, s, s')$ is in X, then its context-closed version $(\mathcal{R}^\star_{val}, s \triangleright e, s' \triangleright e')$ is also in X, for any $(e, e') \in \mathcal{R}^\star$.

The restriction to location-free evaluation contexts in the first item is *not* a limitation of our approach, as already shown in previous work (Sumii and Pierce, 2007b; Koutavas and Wand, 2006): if one wants to prove the equivalence of e and e' under non-evaluation contexts, it suffices to prove the equivalence of $\lambda x. e$ and $\lambda x. e'$ (for fresh x) under evaluation contexts only; if a context needs access to some locations $\ell_1, \ldots, \ell_n$, it suffices to require $(\ell_1, \ell_1), \ldots, (\ell_n, \ell_n) \in \mathcal{R}$. Programs with free variables are not a problem, either: instead of open e and e', it suffices to consider $\lambda x_1. \ldots \lambda x_n. e$ and $\lambda x_1. \ldots \lambda x_n. e'$ for $\{x_1, \ldots x_n\} \supseteq fv(e) \cup fv(e')$.

1.3.3. Generalization to contextual relations

The above approach is not limited to the proof of contextual equivalence, but can be generalized to other binary relations as well. For example, if we add a condition "$|dom(s)| \leq |dom(s')|$ for any $(\mathcal{R}, s \triangleright e, s' \triangleright e') \in X$," then one can conclude that e uses fewer locations than e' under arbitrary (evaluation) contexts. In general, any predicate P on configurations can be added to the conditions of X while keeping it contextual, as long as P itself is contextual (i.e., preserved by contexts). It does not have to be a congruence relation (or even a pre-congruence relation), hence the term "contextual" rather than "congruent" (or pre-congruent).

1.3.4. Contextual predicates and local memory safety

In fact, there is no reason why the proved contextual relations have to be binary. Rather, they can be of arbitrary arity. In particular, the arity can be 1, meaning unary predicates. To obtain conditions for the unary version of X, we just have to remove everything that belongs to the "right hand side." Again, the resulting X is contextual as long as the predicate P itself is contextual.

A prominent example of such unary properties is *local memory safety*. Let us first classify all locations into "local" and "public" ones. The intent is that local locations are kept secret from the context, whereas public locations can be directly manipulated by the context. (This restriction is a mere matter of a proof technique, and does not limit the observational power of contexts at runtime.

In other words, we can always divide locations so that all locations that are directly manipulated by the context are public.) Next, let $P(\mathcal{R}, s \triangleright e)$ be false if and only if e is immediately reading from, writing to, or deallocating a local location that is not in $dom(s)$. Then, just as in the binary case, we can prove that the largest X satisfying the bisimulation-like conditions is contextual. (Of course, we here are not considering a congruence or an equivalence relation—or even a binary relation at all!—but the set X is still "bisimulation-like" in the sense that it involves co-induction and is contextual.)

Another example of unary contextual properties is an upper bound on the number of local locations. To be concrete, let $P(\mathcal{R}, s \triangleright e)$ and $P(\mathcal{R}, s)$ be true if and only if the number of local locations in $dom(s)$ is at most a constant c. Then, again, we can use our approach to prove that a term e allocates at most c local locations under arbitrary evaluation contexts (and arbitrary non-evaluation contexts, if we consider $\lambda x.\, e$ for fresh x; see Section 1.3.2) that do not create local locations themselves.

1.4. Overview of the paper

The rest of this paper is structured as follows. Section 2 discusses related work. Section 3 defines our target language. Section 4 develops the binary version of our proof technique and Section 5 gives examples (contextual relations between two multiset implementations). In addition, Section 6 introduces an auxiliary "up-to" technique to simplify the proofs, with examples in Section 7. Section 8 defines the unary version of our approach and Section 9 gives an example (directed acyclic graphs with garbage collection with reference counting). Section 10 concludes with future work.

Throughout the paper, familiarity with induction, co-induction, traditional (i.e., non-environmental) bisimulations, λ-calculus (with state), and (small-step) operational semantics is assumed. Literature in these areas includes Milner (1999), Pierce (2002, Chapter 21.1 in particular), and Sangiorgi and Walker (2001).

2. Related work

As stated above, our technique is rooted in previous work on environmental bisimulations by Sumii and others (Sumii and Pierce, 2007a,b; Koutavas and Wand, 2006; Sangiorgi et al., 2007). Sumii and Pierce (2007a,b) published the first environmental bisimulations for higher-order languages (λ-calculi with encryption and type abstraction). Koutavas and Wand (2006) reformulated Sumii-Pierce's approach in λ-calculus with general references. Sangiorgi et al. (2007) re-reformulated these approaches in λ-calculi and higher-order π-calculus. The present work generalizes the *notion* of environmental bisimulation itself to non-equivalence properties, in λ-calculus with general references and deallocation.

Denotational semantics can be used to prove contextual equivalence of programs (see, for example, Mitchell, 1996, pp. 77 and 344). In short, two programs are contextually equivalent if their denotations are the same (provided

that the semantics is adequate, of course). However, it is known to be hard to develop "fully abstract"—i.e., sound and complete—denotational semantics for languages with local store (Meyer and Sieber, 1988), let alone general references or deallocation.

Logical relations are relations between (semantics of) programs defined by induction on their types, and can be used for proving properties like contextual equivalence and memory safety. Pitts and Stark (1998) defined (binary) syntactic logical relations—i.e., relations between the syntax of programs itself rather than their semantics—for a simply-typed call-by-value higher-order language with references to integers, and proved that they characterize contextual equivalence in this language. Ahmed et al. (2009) developed step-indexed logical relations—i.e., relations defined by induction on the number of reduction steps instead of types—for call-by-value λ-calculus with general references (references to arbitrary values, including functions and references themselves) and polymorphic (universal and existential) types. To our knowledge, no work has been published on (binary) logical relations in a language with general references and their deallocation.

Ahmed (2004, Chapter 7) defined unary step-indexed logical relations for a continuation-passing-style higher-order language with regions and their deallocation (like the capability calculus). Ahmed et al. (2005, 2007) defined unary step-indexed logical relations in languages with linear types and deallocation. None of these consider contextual equivalence or other binary properties.

3. The language

The syntax of our language is given in Figure 2. It is a standard call-by-value λ-calculus extended with references and deallocation, in addition to first-order primitives (such as Boolean values and integer arithmetic) and tuples, which are added solely for the sake of convenience. The operational semantics is also standard and given in Figure 3. It is parametrized by the semantics of primitives, given as a partial function $[\![\_]\!]$ to constants from operations on constants.

A location ℓ^π is an atomic symbol that models a reference in ML (though it is untyped and deallocatable in our language) or a pointer in C (although our language omits pointer arithmetic for simplicity, it can easily be added by modeling the store as a finite map from locations to *arrays* of values). It has a locality label $\top$ or $\bot$ to distinguish local and public locations, as outlined in the introduction. In what follows, we omit locality labels when they are unimportant. We assume that there exist a countably infinite number of locations, both local and public. A special location $\texttt{null}^\bot$ is reserved for representing a never allocated location. This treatment is just for the sake of simplicity of examples. We write $loc(e)$ for the set of locations that appear in e (except $\texttt{null}^\bot$), and $fv(e)$ for the set of free variables in e. Note that there is no binder for locations in the syntax of our language.

Allocation $\texttt{new}\ x^\pi := e_1; e_2$ creates a fresh location ℓ^π of the specified locality π, initializes the contents with the value of e_1, binds the location to x, and executes e_2. (It is just as easy to separate allocation $\texttt{new}\ x^\pi$ from initialization

$$
\begin{array}{lr}
\pi, \rho ::= & \text{locality} \\
\quad \top & \text{local} \\
\quad \bot & \text{public} \\
d, e, C, D ::= & \text{term} \\
\quad x & \text{variable} \\
\quad \lambda x.\, e & \text{function} \\
\quad e_1 e_2 & \text{application} \\
\quad c & \text{constant} \\
\quad op(e_1, \ldots, e_n) & \text{primitive} \\
\quad \texttt{if } e_1 \texttt{ then } e_2 \texttt{ else } e_3 & \text{conditional branch} \\
\quad \langle e_1, \ldots, e_n \rangle & \text{tupling} \\
\quad \#_i(e) & \text{projection} \\
\quad \ell^\pi & \text{location} \\
\quad \texttt{new } x^\pi := e_1; e_2 & \text{allocation} \\
\quad \texttt{free}(e) & \text{deallocation} \\
\quad e_1 := e_2 & \text{update} \\
\quad !e & \text{dereference} \\
\quad e_1 \overset{ptr}{=} e_2 & \text{pointer equality} \\
u, v, w ::= & \text{value} \\
\quad \lambda x.\, e & \text{function} \\
\quad c & \text{constant} \\
\quad \langle v_1, \ldots, v_n \rangle & \text{tuple} \\
\quad \ell^\pi & \text{location} \\
E, F ::= & \text{evaluation context} \\
\quad [\,] & \text{hole} \\
\quad E e & \text{application (left)} \\
\quad v E & \text{application (right)} \\
\quad op(v_1, \ldots, v_m, E, e_1, \ldots, e_n) & \text{primitive} \\
\quad \texttt{if } E \texttt{ then } e_1 \texttt{ else } e_2 & \text{conditional branch} \\
\quad \langle v_1, \ldots, v_m, E, e_1, \ldots, e_n \rangle & \text{tupling} \\
\quad \#_i(E) & \text{projection} \\
\quad \texttt{new } x^\pi := E; e & \text{allocation} \\
\quad \texttt{free}(E) & \text{deallocation} \\
\quad E := e & \text{update (left)} \\
\quad v := E & \text{update (right)} \\
\quad !E & \text{dereference} \\
\quad E \overset{ptr}{=} e & \text{pointer equality (left)} \\
\quad v \overset{ptr}{=} E & \text{pointer equality (right)} \\
\end{array}
$$

Figure 2: Syntax

$$
\begin{array}{lcll}
s \triangleright (\lambda x.\, e)v & \rightarrow & s \triangleright [v/x]e \\
s \triangleright op(c_1, \ldots, c_n) & \rightarrow & s \triangleright [\![op(c_1, \ldots, c_n)]\!] \\
s \triangleright \texttt{if true then } e_1 \texttt{ else } e_2 & \rightarrow & s \triangleright e_1 \\
s \triangleright \texttt{if false then } e_1 \texttt{ else } e_2 & \rightarrow & s \triangleright e_2 \\
s \triangleright \#_i(v_1, \ldots, v_i, \ldots, v_n) & \rightarrow & s \triangleright v_i \\
s \triangleright \texttt{new } x^\pi := v; e & \rightarrow & s \uplus \{\ell^\pi \mapsto v\} \triangleright [\ell^\pi/x]e & \text{if } \ell^\pi \neq \texttt{null}^\perp \\
s \triangleright \texttt{free}(\ell^\pi) & \rightarrow & s \setminus \ell^\pi \triangleright \langle\rangle \\
s \uplus \{\ell^\pi \mapsto v\} \triangleright \ell^\pi := w & \rightarrow & s \uplus \{\ell^\pi \mapsto w\} \triangleright \langle\rangle \\
s \triangleright {!}\ell^\pi & \rightarrow & s \triangleright s(\ell^\pi) \\
s \triangleright \ell^\pi \overset{ptr}{=} \ell^\pi & \rightarrow & \texttt{true} \\
s \triangleright \ell_1^\pi \overset{ptr}{=} \ell_2^\rho & \rightarrow & \texttt{false} & \text{if } \ell_1^\pi \neq \ell_2^\rho \\
s \triangleright E[d] & \rightarrow & t \triangleright E[e] & \text{if } s \triangleright d \rightarrow t \triangleright e
\end{array}
$$

Figure 3: Reduction

$x^\pi := e_1$, but the present form is slightly shorter. In addition, we simply prefer not to fix a single, arbitrary initial value of locations.) Our intent is to disallow contexts to allocate local locations. This is not a limitation, as explained in the introduction.

Deallocation $\texttt{free}(e)$ releases memory and lets it be reused later. Update $e_1 := e_2$ overwrites the contents of a location.

Pointer equality $e_1 \overset{ptr}{=} e_2$ compares locations themselves (not their contents). We do not use it in our examples (except for comparison with $\texttt{null}^\perp$), but it is necessary for contexts to have a realistic observational power. If both locations are live, their equality can be tested just by writing to one of the locations and reading from the other. However, this is not possible when either (or both) of them is "dead," i.e., already deallocated.

Throughout this paper, we focus on properties of closed terms and values only. (This is not a limitation, again as explained in the introduction.) Thus, we can model a (possibly multi-hole) context C just by a term e with free variables $x_1, \ldots, x_n$, and a context application $C[e_1, \ldots, e_n]$ by a variable substitution $[e_1, \ldots, e_n/x_1, \ldots, x_n]e$. For this reason, we use meta-variables C and D for terms that are used for representing contexts. By convention, we require that terms denoted by capital letters are location-free (except for $\texttt{null}^\perp$) and do not include local allocation $\texttt{new } x^\top$.

For brevity, we use various syntactic sugar. We write $\texttt{let } x = e_1 \texttt{ in } e_2$ for $(\lambda x.\, e_2)e_1$, and $e_1; e_2$ for $\texttt{let } x = e_1 \texttt{ in } e_2$ where x does not appear free in e_2. Recursive function $\texttt{fix } f(x).\, e$ is defined as (the value of) $Y(\lambda f.\, \lambda x.\, e)$ by using some call-by-value fixed-point operator Y as usual. As in Standard ML, $e_1 \texttt{ before } e_2$ denotes $\texttt{let } x = e_1 \texttt{ in } e_2; x$, again with x not free in e_2. We also write $e_1 \wedge e_2$ for $\texttt{if } e_1 \texttt{ then } e_2 \texttt{ else false}$ and $e_1 \vee e_2$ for $\texttt{if } e_1 \texttt{ then true else } e_2$. Note that these conjunction and disjunction operators are not symmetric, as in most programming languages with side effects or di-

vergence. As in Objective Caml, `if` e_1 `then` e_2 abbreviates `if` e_1 `then` e_2 `else` $\langle\rangle$, where $\langle\rangle$ is the nullary tuple. Moreover, `ifnull` e_1 `then` e_2 `else` e_3 abbreviates `if` $e_1 \overset{ptr}{=} \texttt{null}^\perp$ `then` e_2 `else` e_3. Pattern matching $\lambda\langle x_1, \ldots, x_n\rangle. e$ means $\lambda x. \texttt{let } x_1 = \#_1(x) \texttt{ in } \ldots \texttt{let } x_n = \#_n(x) \texttt{ in } e$, for fresh x. Finally, $\#_j^i(!e_1) \leftarrow e_2$ stands for $\texttt{let } x = e_1 \texttt{ in } x := \langle \#_1(!x), \ldots, \#_{j-1}(!x), e_2, \#_{j+1}(!x), \ldots, \#_i(!x)\rangle$.

We give higher precedence to ; and `before` than λ, `let`, and `if` forms. Thus, for instance, `if` e_1 `then` e_2 `else` $e_3; e_4$ and $\lambda x. e_1; e_2$ mean `if` e_1 `then` e_2 `else` $(e_3; e_4)$ and $\lambda x. (e_1; e_2)$, respectively, rather than $(\texttt{if } e_1 \texttt{ then } e_2 \texttt{ else } e_3); e_4$ or $(\lambda x. e_1); e_2$.

Our operational semantics is a standard small-step reduction semantics with evaluation contexts and stores. Here, a store s is a finite map from locations (except $\texttt{null}^\perp$) to closed values. We write $dom(s)$ for the domain of store s. We also write $s \uplus \{\ell \mapsto v\}$ for the extension of store s with location ℓ mapped to value v, with the assumption that $\ell \notin dom(s)$. It is undefined if $\ell \in dom(s)$. Similarly, $s_1 \uplus s_2$ is defined to be $s_1 \cup s_2$ if $dom(s_1) \cap dom(s_2) = \emptyset$, and undefined otherwise. $s \backslash \tilde{\ell}$ denotes the store obtained from s by removing $\tilde{\ell}$ from its domain. Again, it is undefined if $\tilde{\ell} \notin dom(s)$. We write $\twoheadrightarrow$ for the reflexive and transitive closure of $\rightarrow$. We also write $s \triangleright e \rightarrow$ if $s \triangleright e \rightarrow t \triangleright d$ for some t and d, and write $s \triangleright e \nrightarrow$ if not $s \triangleright e \rightarrow$. Furthermore, we write $s \triangleright e \Downarrow\!\!\!/$ if there exist no t and v such that $s \triangleright e \twoheadrightarrow t \triangleright v$.

Note that the reduction is non-deterministic, even up to renaming of locations. For instance, consider $e = (\texttt{new } x := \langle\rangle; x \overset{ptr}{=} \ell)$. Then, we have both $\emptyset \triangleright e \rightarrow \{\ell \mapsto \langle\rangle\} \triangleright (\ell \overset{ptr}{=} \ell) \rightarrow \{\ell \mapsto \langle\rangle\} \triangleright \texttt{true}$ and $\emptyset \triangleright e \rightarrow \{m \mapsto \langle\rangle\} \triangleright (m \overset{ptr}{=} \ell) \rightarrow \{m \mapsto \langle\rangle\} \triangleright \texttt{false}$. This is one of the characteristics of our language, where deallocation makes dangling pointers (like ℓ in the above example), which may or may not get reallocated later.

Throughout the paper, we often abbreviate sequences $A_1, \ldots, A_n$ to $\tilde{A}$, for any kind of meta-variables A_i. We also abbreviate sequences of tuples, like $(A_1, B_1), \ldots, (A_n, B_n)$, as $(\tilde{A}, \tilde{B})$. Thus, for example, $[\tilde{v}/\tilde{x}]e$ denotes $[v_1, \ldots, v_n / x_1, \ldots, x_n]e$.

4. Binary environmental relations

In this section, we develop our approach for *binary* relations including contextual equivalence, which is closer to (the small-step version of) the original environmental bisimulations (Sumii and Pierce, 2007a,b; Koutavas and Wand, 2006; Sangiorgi et al., 2007).

First, we establish the basic terminology for our developments. Intuitions behind the definitions are given in the introduction.

Definition 4.1 (state and binary configuration). *The pair $s \triangleright e$ of store s and closed term e is called a* state. *A* binary configuration *is a quintuple of the form $(\mathcal{R}, s \triangleright e, s' \triangleright e')$ or a triple of the form $(\mathcal{R}, s, s')$, where $\mathcal{R}$ is a binary relation on closed values.*

Note that we do not impose well-formedness conditions such as $loc(e) \subseteq dom(s)$ and $loc(e') \subseteq dom(s')$, because deallocation may (rightfully) make dangling pointers.

Definition 4.2 (context closure). *The* context closure $\mathcal{R}^\star$ *of a binary relation* $\mathcal{R}$ *on closed values, is defined by* $\mathcal{R}^\star = \{([\tilde{v}/\tilde{x}]C, [\tilde{v}'/\tilde{x}]C) \mid (\tilde{v}, \tilde{v}') \in \mathcal{R},\ fv(C) \subseteq \{\tilde{x}\}\}$.

We write $R^\star_{val}$ for the restriction of $\mathcal{R}^\star$ to values. Note $\mathcal{R} \subseteq \mathcal{R}^\star = (\mathcal{R}^\star_{val})^\star$.

Then, we give the main definitions in this section. For brevity, we omit some universal and existential quantifications on meta-variables in the conditions below. They should be clear from the context—or, more precisely, from the positions of the first occurrences of the meta-variables. For instance, when we say

> For every $(\mathcal{R}, s \triangleright d, s' \triangleright d') \in X$, if $s \triangleright d \to t \triangleright e$, then $s' \triangleright d' \twoheadrightarrow t' \triangleright e'$
> and $(\mathcal{R}, t \triangleright e, t' \triangleright e') \in X$

it means

> For every $(\mathcal{R}, s \triangleright d, s' \triangleright d') \in X$, and for any t and e, if $s \triangleright d \to t \triangleright e$ then for some t' and e' we have $s' \triangleright d' \twoheadrightarrow t' \triangleright e'$ and $(\mathcal{R}, t \triangleright e, t' \triangleright e') \in X$

because t and e first appear in the assumption, whereas t' and e' first appear in the conclusion.

Definition 4.3 (reduction closure). *A set X of binary configurations is* reduction-closed *if, for every* $(\mathcal{R}, s \triangleright d, s' \triangleright d') \in X$,

 i. *If* $s \triangleright d \to t \triangleright e$, *then* $s' \triangleright d' \twoheadrightarrow t' \triangleright e'$ *and* $(\mathcal{R}, t \triangleright e, t' \triangleright e') \in X$.
 ii. *If* $d = v$, *then* $s' \triangleright d' \twoheadrightarrow t' \triangleright v'$ *and* $(\mathcal{R} \cup \{(v, v')\}, s, t') \in X$.
 iii. *Symmetric versions of the two conditions above, that is:*
    (i') *If* $s' \triangleright d' \to t' \triangleright e'$, *then* $s \triangleright d \twoheadrightarrow t \triangleright e$ *and* $(\mathcal{R}, t \triangleright e, t' \triangleright e') \in X$.
    (ii') *If* $d' = v'$, *then* $s \triangleright d \twoheadrightarrow t \triangleright v$ *and* $(\mathcal{R} \cup \{(v, v')\}, t, s') \in X$.

Intuitively, reduction closure means that the property in question is preserved throughout the execution of the programs e and e' (including the returned values v and v', which are then learned by the context).

Definition 4.4 (consistency). *A predicate P on binary configurations is* consistent *if for any* $(\mathcal{R}, s \triangleright d, s' \triangleright d') \in P$ *and for any* $(\mathcal{R}, s, s') \in P$,

- *If* $(u, u') \in \mathcal{R}$, *then the outermost syntactic shape of u is the same as that of u'.*

- *If* $(u, u') \in \mathcal{R}$, *then* $u = c \iff u' = c$, *for any constant c.*

- *If* $(\ell_1^\perp, \ell_1'^\perp) \in \mathcal{R}$ *and* $(\ell_2^\perp, \ell_2'^\perp) \in \mathcal{R}$, *then* $\ell_1^\perp = \ell_2^\perp \iff \ell_1'^\perp = \ell_2'^\perp$.

- *If* $(\ell^\pi, \ell'^{\pi'}) \in \mathcal{R}$, *then* $\pi = \pi' = \perp$ *and* $\ell^\perp \in dom(s) \iff \ell'^\perp \in dom(s')$.

Informally, consistency is required for ensuring that

- the "forms" of values on the left and right hand sides are the same, including the equality of constants and locations (and whether the locations are allocated or deallocated), and

- all locations known to the context are indeed public (recall Section 1.3.4).

Note that any subset of a consistent predicate is again consistent. In the rest of the paper, we require that all the predicates P are consistent, often implicitly. This is a trivial restriction because none of them mention the environments $\mathcal{R}$ anyway. We also assume that our primitives include equality tests for all constants.

Definition 4.5 (environmental P-simulation). *Let P be a (consistent) predicate on binary configurations. A reduction-closed subset X of P is called an environmental P-simulation if, for every $(\mathcal{R}, s, s') \in X$ and $(u, u') \in \mathcal{R}$,*

1. *If $u = \lambda x.\,e$ and $u' = \lambda x.\,e'$, then $(\mathcal{R}, s \triangleright uv, t \triangleright u'v') \in X$ for any $(v, v') \in \mathcal{R}^\star$.[1]*

2. *If $u = \langle v_1, \ldots, v_i, \ldots, v_n \rangle$ and $u' = \langle v'_1, \ldots, v'_i, \ldots, v'_n \rangle$, then $(\mathcal{R} \cup \{(v_i, v'_i)\}, s, s') \in X$.*

3. *If $u = \ell^\perp$, $u' = \ell'^\perp$, $s = t \uplus \{\ell^\perp \mapsto v\}$ and $s' = t' \uplus \{\ell'^\perp \mapsto v'\}$, then*
   (a) *$(\mathcal{R}, t, t') \in X$.*
   (b) *$(\mathcal{R}, t \uplus \{\ell^\perp \mapsto w\}, t' \uplus \{\ell'^\perp \mapsto w'\}) \in X$ for any $(w, w') \in \mathcal{R}^\star$.*
   (c) *$(\mathcal{R} \cup \{(v, v')\}, s, s') \in X$.*

4. *For any $\ell^\perp \notin dom(s)$ and $(v, v') \in \mathcal{R}^\star$, we have $(\mathcal{R} \cup \{(\ell^\perp, \ell'^\perp)\}, s \uplus \{\ell^\perp \mapsto v\}, s' \uplus \{\ell'^\perp \mapsto v'\}) \in X$ for some $\ell'^\perp \notin dom(s')$.*

An environmental P-simulation X is called an *environmental P-bisimulation* if its inverse

$$
\begin{aligned}
X^{-1} \;\; = \;\; & \{(\mathcal{R}^{-1}, s' \triangleright e', s \triangleright e) \mid (\mathcal{R}, s \triangleright e, s' \triangleright e') \in X\} \\
\cup \;\; & \{(\mathcal{R}^{-1}, s', s) \mid (\mathcal{R}, s, s') \in X\}
\end{aligned}
$$

is also an environmental P-simulation (or, if X is an environmental P^{-1}-simulation—this is equivalent because all the other conditions are symmetric). An *environmental simulation* is defined as an environmental P^{obs}-simulation, where P^{obs} is the largest consistent predicate on binary configurations. Since all the conditions of environmental P-simulations (i.e., their generating function, to be precise) are monotone on X, the union of all environmental P-simulations is also an environmental P-simulation, called the *environmental P-similarity*. In what follows, we often omit the adjective "environmental" and just write "a simulation" to mean an environmental simulation. The same holds for all the combinations of P- and bi- simulations and similarity.

---

[1] Previous work (Sangiorgi et al., 2007) required $(\mathcal{R}, [v/x]e, [v'/x]e') \in X$ instead of $(\mathcal{R}, s \triangleright uv, t \triangleright u'v') \in X$ here. The latter is slightly more convenient for proving completeness in our non-deterministic language.

As outlined in the introduction, the conditions of P-simulation reflect observations made by contexts. In Definition 4.3 (reduction closure), Condition i (and the first half of Condition iii) mean reduction on the left hand side can be simulated by the right (and vice versa). Condition ii (and ii') adds the values returned by the programs to the knowledge of the context. In Definition 4.5 (P-simulation), Condition 1 corresponds to function application, and Condition 2 to element projection from tuples. Conditions 3a, 3b, 3c, and 4 represent deallocation of, writing to, reading from, and allocation of locations, respectively. Putting aside the generalization from contextual equivalence to arbitrary predicates, the major difference of the definition from previous work (Sangiorgi et al., 2007, Definition 4.1) is naturally Condition 3a, which corresponds to deallocation.

We are now going to prove the main result of this section: let $P_{\star\rightarrow}$ be the largest contextual, reduction-closed subset of P (which exists because the union of contextual, reduction-closed sets is again contextual and reduction-closed); then the P-similarity coincides with $P_{\star\rightarrow}$, provided that P itself is contextual in the following sense.

Definition 4.6 (contextuality). *A set P of binary configurations is* contextual *if its context closure*

$$
\begin{aligned}
P^{\star} \;=\; & \{(\mathcal{S}, s \triangleright [\tilde{v}/\tilde{x}]E[e], s' \triangleright [\tilde{v}'/\tilde{x}]E[e']) \mid \\
& \quad (\mathcal{R}, s \triangleright e, s' \triangleright e') \in P, \; \mathcal{S} \subseteq \mathcal{R}_{val}^{\star}, \; (\tilde{v}, \tilde{v}') \in \mathcal{R}, \; fv(E) \subseteq \{\tilde{x}\}\} \\
\cup \;& \{(\mathcal{S}, s \triangleright [\tilde{v}/\tilde{x}]C, s' \triangleright [\tilde{v}'/\tilde{x}]C) \mid \\
& \quad (\mathcal{R}, s, s') \in P, \; \mathcal{S} \subseteq \mathcal{R}_{val}^{\star}, \; (\tilde{v}, \tilde{v}') \in \mathcal{R}, \; fv(C) \subseteq \{\tilde{x}\}\} \\
\cup \;& \{(\mathcal{S}, s, s') \mid (\mathcal{R}, s, s') \in P, \; \mathcal{S} \subseteq \mathcal{R}_{val}^{\star}\}
\end{aligned}
$$

is included in P.

Note that $P \subseteq P^{\star} = (P^{\star})^{\star}$. An informal intuition for this definition has been given in Section 1.3.2. In short, contextuality means that P is preserved under contexts. Once again, the restriction to location-free (evaluation) contexts does not limit the applicability of our approach.

The inclusion $\mathcal{S} \subseteq \mathcal{R}_{val}^{\star}$ is necessary for the following technical reason: suppose we have a configuration $(\mathcal{R}, s \triangleright d, s' \triangleright d') \in X$ and put it under an evaluation context E, like $(\mathcal{R}, s \triangleright E[d], s' \triangleright E[d']) \in X$. If d and d' reduce to values v and v', respectively, then the context learns these values and adds them to its knowledge, like $(\mathcal{R} \cup \{(v, v')\}, s \triangleright E[v], s' \triangleright E[v']) \in X$. However, according to the conditions of reduction closure, we need $(\mathcal{R}, s \triangleright E[v], s' \triangleright E[v']) \in X$, where the knowledge $\mathcal{R}$ is *smaller* than $\mathcal{R} \cup \{(v, v')\}$. A similar case occurs when the context by itself allocates a fresh location.

This is not a real problem because smaller knowledge means fewer observations. In fact, instead of taking $\mathcal{S} \subseteq \mathcal{R}_{val}^{\star}$ here, it is also possible to generalize the definition of simulation to allow the increase of knowledge in the middle of an evaluation. This amounts to an up-to environment technique (Sangiorgi et al., 2007). In this paper, it is subsumed by the up-to context technique (Section 6) because of the inclusion above.

13

Lemma 4.7 (context closure preserves consistency). *If P is consistent, so is $P^\star$.*

PROOF. Immediate from Definition 4.4 with Definition 4.2 and 4.6. ☐

Lemma 4.8 (value contexts). *For any C, $\tilde{x}$, $\tilde{v}$ and $\tilde{v}'$, if $[\tilde{v}/\tilde{x}]C$ is a value, then so is $[\tilde{v}'/\tilde{x}]C$.*

PROOF. Straightforward induction on the syntax of C. ☐

Lemma 4.9 (soundness of P-similarity). *For any P, the $P^\star$-similarity is included in $(P^\star)_{\star\rightarrow}$.*

PROOF. Let X be the $P^\star$-similarity. By Definition 4.5, $X \subseteq P^\star$. Since $(P^\star)_{\star\rightarrow}$ is defined as the *largest* contextual and reduction-closed subset of $P^\star$, if we prove that $X^\star$ is reduction-closed (and contextual—but the latter is obvious since $(X^\star)^\star = X^\star$ by Definition 4.6), then $X \subseteq (P^\star)_{\star\rightarrow}$. We carry out this proof by case analysis on elements of $X^\star$ along Definition 4.6.

Case $(\mathcal{S}, s \triangleright [\tilde{v}/\tilde{x}]E[e], s' \triangleright [\tilde{v}'/\tilde{x}]E[e']) \in X^\star$ with $(\mathcal{R}, s \triangleright e, s' \triangleright e') \in X$ and $\mathcal{S} \subseteq \mathcal{R}^\star_{val}$ and $(\tilde{v}, \tilde{v}') \in \mathcal{R}$ and $fv(E) \subseteq \{\tilde{x}\}$.

We need to prove the conditions of reduction closure (Definition 4.3) for the element $(\mathcal{S}, s \triangleright [\tilde{v}/\tilde{x}]E[e], s' \triangleright [\tilde{v}'/\tilde{x}]E[e'])$ of $X^\star$.

To prove Condition i (of Definition 4.3), suppose $s \triangleright [\tilde{v}/\tilde{x}]E[e] \rightarrow$. Since $(\mathcal{R}, s \triangleright e, s' \triangleright e') \in X$ and X is reduction-closed (by Definition 4.5), if e is a value, then e' also reduces to some value (by Condition ii of Definition 4.3) and the rest of the proof amounts to the next case. Suppose thus that e is not a value. Since $s \triangleright [\tilde{v}/\tilde{x}]E[e] \rightarrow$ and E is an evaluation context, we have $s \triangleright e \rightarrow t \triangleright d$ for some t and d. Again since $(\mathcal{R}, s \triangleright e, s' \triangleright e') \in X$ and X is reduction-closed (by Definition 4.5), we have $s' \triangleright e' \twoheadrightarrow t' \triangleright d'$ for some t' and d' with $(\mathcal{R}, t \triangleright d, t' \triangleright d') \in X$ (by Condition i of Definition 4.3). Hence $(\mathcal{S}, t \triangleright [\tilde{v}/\tilde{x}]E[d], t' \triangleright [\tilde{v}'/\tilde{x}]E[d']) \in X^\star$ by Definition 4.6.

To prove Condition ii, suppose $[\tilde{v}/\tilde{x}]E[e]$ is a value, which we call w. Since E is an evaluation context, this can be the case only if e is also a value v. Since $(\mathcal{R}, s \triangleright v, s' \triangleright e') \in X$ and X is reduction-closed (by Definition 4.5), we have $s' \triangleright e' \twoheadrightarrow t' \triangleright v'$ for some t' and v' with $(\mathcal{R} \cup \{(v, v')\}, s, t') \in X$ (by Condition ii of Definition 4.3). Since $[\tilde{v}/\tilde{x}]E[v]$ is a value, so is $[\tilde{v}'/\tilde{x}]E[v']$ (by Lemma 4.8), which we call w'. Since $(w, w') \in (\mathcal{R} \cup \{(v, v')\})^\star_{val}$ by Definition 4.2, we obtain $(\mathcal{S} \cup \{(w, w')\}, s, t') \in X^\star$ by Definition 4.6.

The proof of Condition iii is symmetric to the proofs above.

Case $(\mathcal{S}, s \triangleright [\tilde{v}/\tilde{x}]C, s' \triangleright [\tilde{v}'/\tilde{x}]C) \in X^\star$ with $(\mathcal{R}, s, s') \in X$ and $\mathcal{S} \subseteq \mathcal{R}^\star_{val}$ and $(\tilde{v}, \tilde{v}') \in \mathcal{R}$ and $fv(C) \subseteq \{\tilde{x}\}$.

Again, we prove the conditions of reduction closure for $(\mathcal{S}, s \triangleright [\tilde{v}/\tilde{x}]C, s' \triangleright [\tilde{v}'/\tilde{x}]C) \in X^\star$.

First, we prove Condition i (of Definition 4.3) by induction on C. Suppose $s \triangleright [\tilde{v}/\tilde{x}]C \rightarrow$.

If C is of the form $E[D]$ for some $E \neq [\,]$, and if $s \triangleright [\tilde{v}/\tilde{x}]D \to t \triangleright d$ for some t and d, then $s' \triangleright [\tilde{v}'/\tilde{x}]D \twoheadrightarrow t' \triangleright d'$ for some t' and d' with $(\mathcal{S}, t \triangleright d, t' \triangleright d') \in X^\star$ by the induction hypothesis. Hence $s' \triangleright [\tilde{v}'/\tilde{x}]C \twoheadrightarrow t' \triangleright E[d']$, with $(\mathcal{S}, t \triangleright E[d], t' \triangleright E[d']) \in (X^\star)^\star = X^\star$ by Definition 4.6.

Otherwise, we proceed by case analysis on C.

Subcase $C = C_1 C_2$. Then $[\tilde{v}/\tilde{x}]C_1$ is a λ-abstraction and $[\tilde{v}/\tilde{x}]C_2$ (resp. $[\tilde{v}'/\tilde{x}]C_2$, by Lemma 4.8) is a value w (resp. w').

If C_1 itself is a λ-abstraction $\lambda x. C_0$, then the only possible reduction on the "left hand side" (of the bisimulation) is $s \triangleright [\tilde{v}/\tilde{x}]C \to s \triangleright [\tilde{v}/\tilde{x}]([C_2/x]C_0)$, which corresponds to $s' \triangleright [\tilde{v}'/\tilde{x}]C \to s' \triangleright [\tilde{v}'/\tilde{x}]([C_2/x]C_0)$ on the right hand side, with $(\mathcal{S}, s \triangleright [\tilde{v}/\tilde{x}]([C_2/x]C_0), s' \triangleright [\tilde{v}'/\tilde{x}]([C_2/x]C_0)) \in X^\star$ by Definition 4.6.

Otherwise, C_1 is a variable x_i and v_i is a λ-abstraction. Since $(v_i, v_i') \in \mathcal{R}$ and $(\mathcal{R}, s, s') \in X$ and X is a $P^\star$-simulation, v_i' is also a λ-abstraction by Definition 4.4, and therefore $(\mathcal{R}, s \triangleright v_i w, s' \triangleright v_i' w') \in X$ by Condition 1 of Definition 4.5. Since X is reduction-closed (by Definition 4.5), if $s \triangleright v_i w \to t \triangleright e$ for some t and e, then $s' \triangleright v_i' w' \twoheadrightarrow t' \triangleright e'$ for some t' and e' with $(\mathcal{R}, t \triangleright e, t' \triangleright e') \in X$ (by Condition i of Definition 4.3). Hence $(\mathcal{S}, t \triangleright e, t' \triangleright e') \in X^\star$ by Definition 4.6.

Subcase $C = op(C_1, \ldots, C_n)$. Then $[\tilde{v}/\tilde{x}]C_i$ is a constant c_i, for $i = 1, \ldots, n$, and $s \triangleright [\tilde{v}/\tilde{x}]C \to s \triangleright c$ for $c = [\![op(c_1, \ldots, c_n)]\!]$. If C_i itself is c_i, then $[\tilde{v}'/\tilde{x}]C_i = c_i$. Otherwise, C_i is a variable x_i and $v_i = c_i$. By Definition 4.4, $v_i' = c_i$. Therefore, $[\tilde{v}'/\tilde{x}]C_i = c_i$ anyway. Hence $s' \triangleright [\tilde{v}'/\tilde{x}]C \to s' \triangleright c$, with $(\mathcal{S}, s \triangleright c, s' \triangleright c) \in X^\star$ by Definition 4.6.

Subcase $C = \texttt{if } C_1 \texttt{ then } C_2 \texttt{ else } C_3$. Then $[\tilde{v}/\tilde{x}]C_1$ is a Boolean constant b and so is $[\tilde{v}'/\tilde{x}]C_1$ (for the same reason as in the previous subcase). If $b = \texttt{true}$, then the only possible reduction on the left hand side is $s \triangleright [\tilde{v}/\tilde{x}]C \to s \triangleright [\tilde{v}/\tilde{x}]C_2$, which corresponds to $s' \triangleright [\tilde{v}'/\tilde{x}]C \to s' \triangleright [\tilde{v}'/\tilde{x}]C_2$ on the right hand side, with $(\mathcal{S}, s \triangleright [\tilde{v}/\tilde{x}]C_2, s' \triangleright [\tilde{v}'/\tilde{x}]C_2) \in X^\star$ by Definition 4.6. The case $b = \texttt{false}$ is similar.

Subcase $C = \#_i(C_0)$. Then $[\tilde{v}/\tilde{x}]C_0$ is a tuple $\langle w_1, \ldots, w_n \rangle$ and $s \triangleright [\tilde{v}/\tilde{x}]C \to s \triangleright w_i$.

If C_0 itself is a tuple $\langle C_1, \ldots, C_n \rangle$, then $[\tilde{v}/\tilde{x}]C_i = w_i$, so $[\tilde{v}'/\tilde{x}]C_i$ is also a value w_i' (Lemma 4.8), for $i = 1, \ldots, n$. Hence $s' \triangleright [\tilde{v}'/\tilde{x}]C \to s' \triangleright w_i'$, with $(\mathcal{S}, s \triangleright w_i, s' \triangleright w_i') \in X^\star$ by Definition 4.6.

Otherwise, C_0 is a variable x_i and $v_i = \langle w_1, \ldots, w_n \rangle$. Since $(\langle w_1, \ldots, w_n \rangle, v_i') \in \mathcal{R}$ and $(\mathcal{R}, s, s') \in X$ and X is a $P^\star$-simulation, v_i' is also a tuple $\langle w_1', \ldots, w_n' \rangle$ by Definition 4.4, and therefore $(\mathcal{R} \cup \{(w_i, w_i')\}, s, s') \in X$ by Condition 2 of Definition 4.5. Hence $s' \triangleright [\tilde{v}'/\tilde{x}]C \to s' \triangleright w_i'$, with $(\mathcal{S}, s \triangleright w_i, s' \triangleright w_i') \in X^\star$ by Definition 4.6.

Subcase $C = (\texttt{new } x^\perp := C_1; C_2)$. Then $[\tilde{v}/\tilde{x}]C_1$ (resp. $[\tilde{v}'/\tilde{x}]C_1$, by Lemma 4.8) is a value w (resp. w') and the only possible reductions on the left hand side are of the form $s \triangleright [\tilde{v}/\tilde{x}]C \to s \uplus \{\ell^\perp \mapsto w\} \triangleright [\tilde{v}, \ell^\perp/\tilde{x}, x]C_2$ for some $\ell^\perp \notin dom(s)$, which corresponds to $s' \triangleright [\tilde{v}'/\tilde{x}]C \to s' \uplus \{\ell'^\perp \mapsto w'\} \triangleright [\tilde{v}', \ell'^\perp/\tilde{x}, x]C_2$ for some

$\ell'^\perp \notin dom(s')$ on the right hand side. Since $(\mathcal{R}, s, s') \in X$ and X is a $P^\star$-simulation, we have $(\mathcal{R} \cup \{(\ell^\perp, \ell'^\perp)\}, s \uplus \{\ell^\perp \mapsto w\}, s' \uplus \{\ell'^\perp \mapsto w'\}) \in X$ by Condition 4 of Definition 4.5. Hence $(\mathcal{S}, s \uplus \{\ell^\perp \mapsto w\} \rhd [\tilde{v}, \ell^\perp/\tilde{x}, x]C_2, s' \uplus \{\ell'^\perp \mapsto w'\} \rhd [\tilde{v}', \ell'^\perp/\tilde{x}, x]C_2) \in X^\star$ by Definition 4.6.

Subcase $C = \mathtt{free}(C_1)$. Then $[\tilde{v}/\tilde{x}]C_1$ is a location ℓ^π with $s = t \uplus \{\ell^\pi \mapsto w\}$ for some t and w, so $s \rhd [\tilde{v}/\tilde{x}]C \to t \rhd \langle\rangle$. Since contexts are location-free, it must be that C_1 is a variable x_i and $v_i = \ell^\pi$. By Definition 4.4, we have $\pi = \perp$ and v'_i is also a public location $\ell'^\perp$ with $s' = t' \uplus \{\ell'^\perp \mapsto w'\}$ for some t' and w', so $s' \rhd [\tilde{v}'/\tilde{x}]C \to t' \rhd \langle\rangle$. Furthermore, Condition 3a of Definition 4.5 implies $(\mathcal{R}, t, t') \in X$. Hence $(\mathcal{S}, t \rhd \langle\rangle, t' \rhd \langle\rangle) \in X^\star$ by Definition 4.6.

Subcase $C = (C_1 := C_2)$. Then $[\tilde{v}/\tilde{x}]C_1$ is a location ℓ^π, and $[\tilde{v}/\tilde{x}]C_2$ (resp. $[\tilde{v}'/\tilde{x}]C_2$, by Lemma 4.8) is a value w (resp. w'), with $s = t \uplus \{\ell^\pi \mapsto u\}$ for some t and u, so $s \rhd [\tilde{v}/\tilde{x}]C \to t \uplus \{\ell^\pi \mapsto w\} \rhd \langle\rangle$. Since contexts are location-free, it must be that C_1 is a variable x_i and $v_i = \ell^\pi$. By Definition 4.4, we have $\pi = \perp$ and v'_i is also a public location $\ell'^\perp$, with $s' = t' \uplus \{\ell'^\perp \mapsto u'\}$ for some t' and u', so $s' \rhd [\tilde{v}'/\tilde{x}]C \to t' \uplus \{\ell'^\perp \mapsto w'\} \rhd \langle\rangle$. Furthermore, since $(w, w') \in \mathcal{R}^\star$, Condition 3b of Definition 4.5 implies $(\mathcal{R}, t \uplus \{\ell^\perp \mapsto w\}, t' \uplus \{\ell'^\perp \mapsto w'\}) \in X$. Hence $(\mathcal{S}, t \uplus \{\ell^\perp \mapsto w\} \rhd \langle\rangle, t' \uplus \{\ell'^\perp \mapsto w'\} \rhd \langle\rangle) \in X^\star$ by Definition 4.6.

Subcase $C = {!}C_1$. Then $[\tilde{v}/\tilde{x}]C_1$ is a location ℓ^π with $s = t \uplus \{\ell^\pi \mapsto w\}$ for some t and w, so $s \rhd [\tilde{v}/\tilde{x}]C \to s \rhd w$. Since contexts are location-free, it must be that C_1 is a variable x_i and $v_i = \ell^\pi$. By Definition 4.4, we have $\pi = \perp$ and v'_i is also a public location $\ell'^\perp$ with $s' = t' \uplus \{\ell'^\perp \mapsto w'\}$ for some t' and w', so $s' \rhd [\tilde{v}'/\tilde{x}]C \to s' \rhd w'$. Furthermore, Condition 3c of Definition 4.5 implies $(\mathcal{R} \cup \{(w, w')\}, s, s') \in X$. Hence $(\mathcal{S}, s \rhd w, s' \rhd w') \in X^\star$ by Definition 4.6.

Subcase $C = (C_1 \overset{ptr}{=} C_2)$. Then $[\tilde{v}/\tilde{x}]C_1$ and $[\tilde{v}/\tilde{x}]C_2$ are locations $\ell_1^{\pi_1}$ and $\ell_2^{\pi_2}$, respectively, so $s \rhd [\tilde{v}/\tilde{x}]C \to s \rhd b$, where b is $\mathtt{true}$ if $\ell_1^{\pi_1} = \ell_2^{\pi_2}$ and $\mathtt{false}$ otherwise. Since contexts are location-free, it must be that C_1 and C_2 are variables x_i and x_j, respectively, with $v_i = \ell_1^{\pi_1}$ and $v_j = \ell_2^{\pi_2}$. By Definition 4.4, we have $\pi_1 = \pi_2 = \perp$, and v'_i and v'_j are also public locations $\ell_1'^\perp$ and $\ell_2'^\perp$, respectively, with $(\ell_1^\perp = \ell_2^\perp) \iff (\ell_1'^\perp = \ell_2'^\perp)$. Hence $s' \rhd [\tilde{v}'/\tilde{x}]C \to s' \rhd b$, with $(\mathcal{S}, s \rhd b, s' \rhd b) \in X^\star$ by Definition 4.6.

This concludes the proof of Condition i (of Definition 4.3).

To prove Condition ii, suppose $[\tilde{v}/\tilde{x}]C$ is a value w. Then $[\tilde{v}'/\tilde{x}]C$ is also a value w' (Lemma 4.8) and $(w, w') \in \mathcal{R}^\star$ by Definition 4.2. Hence $(\mathcal{S} \cup \{(w, w')\}, s, s') \in X^\star$ by Definition 4.6.

The proof of Condition iii is symmetric to those of Condition i and ii.

Case $(\mathcal{S}, s, s') \in X^\star$ with $(\mathcal{R}, s, s') \in P$ and $\mathcal{S} \subseteq \mathcal{R}_{val}^\star$. Definition 4.3 requires no condition for elements of this form. $\qquad\square$

Lemma 4.10 (completeness of P-similarity). *For any P, $(P^\star)_{\star\to}$ is included in the $P^\star$-similarity.*

PROOF. Let $X = (P^\star)_{\star\to}$ for the sake of readability. Since the $P^\star$-similarity is defined as the *largest* $P^\star$-simulation, it suffices to prove that X is a $P^\star$-simulation. We carry out this proof by checking each condition of Definition 4.5. Take $(\mathcal{R}, s, s') \in X$ and $(u, u') \in \mathcal{R}$.

To prove Condition 1, suppose $u = \lambda x.\, e$ and $u' = \lambda x.\, e'$, and take $(v, v') \in \mathcal{R}^\star$. Since $(u, u') \in \mathcal{R}$, we have $(\mathcal{R}, s \triangleright uv, s' \triangleright u'v') \in X^\star$ by Definition 4.6. Since X is contextual by definition, we also have $X^\star = X$. Hence $(\mathcal{R}, s \triangleright uv, s' \triangleright u'v') \in X$.

To prove Condition 2, suppose $u = \langle v_1, \ldots, v_i, \ldots, v_n \rangle$ and $u' = \langle v'_1, \ldots, v'_i, \ldots, v'_n \rangle$, and consider $(\mathcal{R}, s \triangleright \#_i(u), s' \triangleright \#_i(u')) \in X^\star = X$. Since X is reduction-closed by definition, and since $s \triangleright \#_i(u) \to s \triangleright v_i$ and $s' \triangleright \#_i(u') \to s \triangleright v'_i$, we obtain $(\mathcal{R} \cup \{(v_i, v'_i)\}, s, s') \in X$ by Definition 4.3.

To prove Condition 3a, 3b and 3c, suppose $u = \ell^\perp$, $u' = \ell'^\perp$, $s = t \uplus \{\ell^\perp \mapsto v\}$ and $s' = t' \uplus \{\ell'^\perp \mapsto v'\}$.

- For Condition 3a, consider $(\mathcal{R}, s \triangleright \texttt{free}(u), s' \triangleright \texttt{free}(u')) \in X^\star = X$. Again, since X is reduction-closed by definition, and since $s \triangleright \texttt{free}(u) \to t \triangleright \langle \rangle$ and $s' \triangleright \texttt{free}(u') \to t' \triangleright \langle \rangle$, we obtain $(\mathcal{R} \cup \{(\langle \rangle, \langle \rangle)\}, t, t') \in X$ by Definition 4.3. Hence $(\mathcal{R}, t, t') \in X^\star = X$ by Definition 4.6 (since $\mathcal{R} \subseteq \mathcal{R}^\star \subseteq (\mathcal{R} \cup \{(\langle \rangle, \langle \rangle)\})^\star$).

- For Condition 3b, suppose $(w, w') \in \mathcal{R}^\star$, and consider $(\mathcal{R}, s \triangleright u := w, s' \triangleright u' := w') \in X^\star = X$. Once again, since X is reduction-closed by definition, and since $s \triangleright u := w \to t \uplus \{\ell^\perp \mapsto w\} \triangleright \langle \rangle$ and $s' \triangleright u' := w' \to t' \uplus \{\ell'^\perp \mapsto w'\} \triangleright \langle \rangle$, we obtain $(\mathcal{R} \cup \{(\langle \rangle, \langle \rangle)\}, t \uplus \{\ell^\perp \mapsto w\}, t' \uplus \{\ell'^\perp \mapsto w'\}) \in X$ by Definition 4.3. Hence $(\mathcal{R}, t \uplus \{\ell^\perp \mapsto w\}, t' \uplus \{\ell'^\perp \mapsto w'\}) \in X^\star = X$ by Definition 4.6 (again since $\mathcal{R} \subseteq \mathcal{R}^\star \subseteq (\mathcal{R} \cup \{(\langle \rangle, \langle \rangle)\})^\star$).

- For Condition 3c, consider $(\mathcal{R}, s \triangleright !u, s' \triangleright !u') \in X^\star = X$. Twice again, since X is reduction-closed by definition, and since $s \triangleright !u \to s \triangleright v$ and $s' \triangleright !u' \to s' \triangleright v'$, we obtain $(\mathcal{R} \cup \{(v, v')\}, s, s') \in X$ by Definition 4.3.

To prove Condition 4, take $\ell^\perp \notin dom(s)$ and $(v, v') \in \mathcal{R}^\star$, and consider $(\mathcal{R}, s \triangleright (\texttt{new}\ x^\perp := v; x), s' \triangleright (\texttt{new}\ x^\perp := v'; x)) \in X^\star = X$. Thrice again, since X is reduction-closed by definition, and since $s \triangleright (\texttt{new}\ x^\perp := v; x) \to s \uplus \{\ell^\perp \mapsto v\} \triangleright \ell^\perp$ and $s' \triangleright (\texttt{new}\ x^\perp := v'; x) \to s' \uplus \{\ell'^\perp \mapsto v'\} \triangleright \ell'^\perp$ for some $\ell'^\perp \notin dom(s')$, we obtain $(\mathcal{R} \cup \{(\ell^\perp, \ell'^\perp)\}, s \uplus \{\ell^\perp \mapsto v\}, s' \uplus \{\ell'^\perp \mapsto v'\}) \in X$ by Definition 4.3. $\square$

From the two lemmas above, we obtain our main theorem:

Theorem 4.11 (characterization). *For any P, the $P^\star$-similarity coincides with $(P^\star)_{\star\to}$. In particular, if P is contextual, then the P-similarity coincides with $P_{\star\to}$.*

By Definition 4.4 and 4.6, the largest consistent predicate P^{obs} is trivially contextual. Thus:

Corollary 4.12 (bisimilarity equals contextual equivalence). *The bisimilarity coincides with the contextual equivalence $P^{obs}_{\star\to}$.*

17

5. Examples of P-bisimulations

We first show an example of contextual equivalence between two implementations of integer multisets, one with linked lists and the other with binary search trees.

5.1. Linked lists

We implement (mutable) linked lists in our language as follows.

Definition 5.1 (linked list).

$$
\begin{aligned}
set &= \texttt{new } z := \texttt{null}; \langle add, mem, del \rangle \\
add &= \lambda x.\, x + 0; \texttt{new } y := \langle x, !z \rangle; z := y \\
mem &= \lambda x.\, x + 0; mem_x(!z) \\
mem_x &= \texttt{fix } f(y).\, y \overset{ptr}{\neq} \texttt{null} \wedge (\#_1(!y) \overset{int}{=} x \vee f(\#_2(!y))) \\
del &= \lambda x.\, x + 0; z := del_x(!z) \\
del_x &= \texttt{fix } g(y). \\
&\qquad \texttt{ifnull } y \texttt{ then } y \texttt{ else} \\
&\qquad \texttt{if } \#_1(!y) \overset{int}{=} x \texttt{ then } \#_2(!y) \texttt{ before free}(y) \texttt{ else} \\
&\qquad \#_2^2(!y) \leftarrow g(\#_2(!y)); y
\end{aligned}
$$

Here, z is bound to the location of the present list, which is kept local to prevent direct (and unsafe) access. An empty list is represented by $\texttt{null}$. A non-empty list is represented by the location of the pair (e, r) of its first element e and the rest r of the list.

The list is equipped with three operations: addition, membership, and deletion. All of them are simple and standard (perhaps except for the integer addition $x + 0$, which serves as an assertion to ensure that x is indeed an integer, assuming that $v + 0$ is undefined for all non-integers v). For example, the recursive function del_x takes a list y, searches it for the element x, deletes it from y, and returns the updated list. (The syntactic sugar used above is defined in Section 3.)

Let $S, T, \dots$ denote multisets of integers. We write $+$ and $-$ for multiset union and difference. The predicate $Set(\ell, S, s)$, read "ℓ represents S under s," is defined by induction as follows.

- $Set(\texttt{null}, \emptyset, \emptyset)$.

- $Set(\ell, S_0 + \{i\}, s_0 \uplus \{\ell \mapsto \langle i, \ell_0 \rangle\})$ if $\ell \neq \texttt{null}$ and $Set(\ell_0, S_0, s_0)$.

The predicate $Set(\ell, S, s)$ is "precise" in the sense that it allows no extra locations in the store s other than those required for representing the set S. It is also possible to consider its "imprecise" version by replacing the axiom $Set(\texttt{null}, \emptyset, \emptyset)$ with $Set(\texttt{null}, \emptyset, s)$. However, it is always possible to state imprecise properties by using precise predicates, like $\exists s_0 \subseteq s.\, Set(\ell, S, s_0)$. Moreover, precise predicates are often useful for reasoning about memory leaks (or lack thereof), as we will see in examples.

The following lemmas follow by straightforward induction on the derivation of $Set(\ell, S, s)$ or $Set(\ell_0, S, s_0)$.

Lemma 5.2. *Suppose $Set(\ell_0, S, s_0)$. Then, for any t,*

$$s_0 \uplus \{m \mapsto \ell_0\} \uplus t \triangleright [m/z]\, add(i) \quad \twoheadrightarrow \quad s \uplus \{m \mapsto \ell\} \uplus t \triangleright \langle\rangle$$

with $Set(\ell, S + \{i\}, s)$.

Lemma 5.3. *Suppose $Set(\ell, S, s)$. Then, for any t, we have $s \uplus t \triangleright mem_i(\ell) \twoheadrightarrow s \uplus t \triangleright b$ with $b = \texttt{true}$ if $i \in S$, and $b = \texttt{false}$ otherwise.*

Corollary 5.4. *Suppose $Set(\ell, S, s)$. Then, for any t,*

$$s \uplus \{m \mapsto \ell\} \uplus t \triangleright [m/z]\, mem(i) \quad \twoheadrightarrow \quad s \uplus \{m \mapsto \ell\} \uplus t \triangleright b$$

with $b = \texttt{true}$ if $i \in S$, and $b = \texttt{false}$ otherwise.

Lemma 5.5. *Suppose $Set(\ell, S, s)$. Then, for any t, we have $s \uplus t \triangleright del_i(\ell) \twoheadrightarrow s_0 \uplus t \triangleright \ell_0$ with $Set(\ell_0, S - \{i\}, s_0)$.*

Like the other lemmas above, the last lemma is proved by induction on $Set(\ell, S, s)$. However, since it is the most important of these lemmas, we detail its proof.

PROOF. Trivial if $\ell = \texttt{null}$. Suppose $\ell \neq \texttt{null}$, $S = S_1 + \{j\}$, $s = s_1 \uplus \{\ell \mapsto \langle j, \ell_1 \rangle\}$, and $Set(\ell_1, S_1, s_1)$. If $i = j$, then $s \uplus t \triangleright del_i(\ell) \twoheadrightarrow s_1 \uplus t \triangleright \ell_1$. Thus, it suffices to take $s_0 = s_1$ and $\ell_0 = \ell_1$. Suppose $i \neq j$. By induction, $s_1 \uplus \{\ell \mapsto \langle j, \ell_1 \rangle\} \uplus t \triangleright del_i(\ell_1) \twoheadrightarrow s_2 \uplus \{\ell \mapsto \langle j, \ell_1 \rangle\} \uplus t \triangleright \ell_2$ with $Set(\ell_2, S_1 - \{i\}, s_2)$. Therefore, $s_1 \uplus \{\ell \mapsto \langle j, \ell_1 \rangle\} \uplus t \triangleright del_i(\ell) \twoheadrightarrow s_2 \uplus \{\ell \mapsto \langle j, \ell_2 \rangle\} \uplus t \triangleright \ell$. Since $Set(\ell_2, S_1 - \{i\}, s_2)$ and $i \neq j$, we have $Set(\ell, S_1 + \{j\} - \{i\}, s_2 \uplus \{\ell \mapsto \langle j, \ell_2 \rangle\})$. Thus, it suffice to take $s_0 = s_2 \uplus \{\ell \mapsto \langle j, \ell_2 \rangle\}$ and $\ell_0 = \ell$. □

Corollary 5.6. *Suppose $Set(\ell, S, s)$. Then, for any t,*

$$s \uplus \{m \mapsto \ell\} \uplus t \triangleright [m/z]\, del(i) \quad \twoheadrightarrow \quad s_0 \uplus \{m \mapsto \ell_0\} \uplus t \triangleright \langle\rangle$$

with $Set(\ell_0, S - \{i\}, s_0)$.

One may notice that all the lemmas above have the form

for any t, we have $s_1 \uplus t \triangleright e_1 \twoheadrightarrow s_2 \uplus t \triangleright e_2$

and might perhaps wonder why we do not establish a general "lemma" like

if $s_1 \triangleright e_1 \twoheadrightarrow s_2 \triangleright e_2$, then $s_1 \uplus t \triangleright e_1 \twoheadrightarrow s_2 \uplus t \triangleright e_2$.

However, because of dangling pointers (which can be created by deallocation), this property does not hold in general. For example, consider

$$\emptyset \triangleright \texttt{new } x := 0; x \stackrel{ptr}{=} \ell \quad \rightarrow \quad \{\ell \mapsto 0\} \triangleright \ell \stackrel{ptr}{=} \ell$$
$$\rightarrow \quad \{\ell \mapsto 0\} \triangleright \texttt{true}$$

and let $t = \{\ell \mapsto 1\}$. Then m cannot equal ℓ in

$$t \triangleright \texttt{new } x := 0; x \stackrel{ptr}{=} \ell \quad \rightarrow \quad t \uplus \{m \mapsto 0\} \triangleright m \stackrel{ptr}{=} \ell$$

so

$$t \triangleright \mathtt{new}\ x := 0; x \overset{ptr}{=} \ell \quad \not\rightarrow \quad t \uplus \{\ell \mapsto 0\} \triangleright \mathtt{true}.$$

The above property would hold if we restrict $dom(t)$ to be fresh, i.e., $dom(t) \cap loc(e_1) = \emptyset$. However, as we shall see, this restriction is too strong for many examples.

5.2. Binary search trees

We give another implementation—by (mutable) binary search trees—of integer multisets.

Definition 5.7 (binary search tree).

$$
\begin{aligned}
set' \quad &= \quad \mathtt{new}\ z := \mathtt{null}; \langle add', mem', del' \rangle \\[6pt]
add' \quad &= \quad \lambda x.\, x + 0; z := add'_x(!z) \\
add'_x \quad &= \quad \mathtt{fix}\ f(y). \\
&\qquad \mathtt{ifnull}\ y\ \mathtt{then}\ \mathtt{new}\ y' := \langle x, \mathtt{null}, \mathtt{null} \rangle; y'\ \mathtt{else} \\
&\qquad \mathtt{if}\ x \overset{int}{<} \#_1(!y)\ \mathtt{then}\ \#_2^3(!y) \leftarrow f(\#_2(!y)); y\ \mathtt{else} \\
&\qquad \#_3^3(!y) \leftarrow f(\#_3(!y)); y \\[6pt]
mem' \quad &= \quad \lambda x.\, x + 0; mem'_x(!z) \\
mem'_x \quad &= \quad \mathtt{fix}\ g(y). \\
&\qquad \mathtt{ifnull}\ y\ \mathtt{then}\ \mathtt{false}\ \mathtt{else} \\
&\qquad \mathtt{if}\ x \overset{int}{<} \#_1(!y)\ \mathtt{then}\ g(\#_2(!y))\ \mathtt{else} \\
&\qquad \mathtt{if}\ x \overset{int}{>} \#_1(!y)\ \mathtt{then}\ g(\#_3(!y))\ \mathtt{else} \\
&\qquad \mathtt{true} \\[6pt]
min \quad &= \quad \mathtt{fix}\ m(y). \\
&\qquad \mathtt{ifnull}\ \#_2(!y)\ \mathtt{then}\ \#_1(!y)\ \mathtt{else}\ m(\#_2(!y)) \\
del' \quad &= \quad \lambda x.\, x + 0; z := del'_-(!z)x \\
del'_- \quad &= \quad \mathtt{fix}\ h(y).\, \lambda x. \\
&\qquad \mathtt{ifnull}\ y\ \mathtt{then}\ y\ \mathtt{else} \\
&\qquad \mathtt{if}\ x \overset{int}{<} \#_1(!y)\ \mathtt{then}\ \#_2^3(!y) \leftarrow h(\#_2(!y))x; y\ \mathtt{else} \\
&\qquad \mathtt{if}\ x \overset{int}{>} \#_1(!y)\ \mathtt{then}\ \#_3^3(!y) \leftarrow h(\#_3(!y))x; y\ \mathtt{else} \\
&\qquad \mathtt{ifnull}\ \#_3(!y)\ \mathtt{then}\ \#_2(!y)\ \mathtt{before}\ \mathtt{free}(y)\ \mathtt{else} \\
&\qquad \mathtt{let}\ x' = min(\#_3(!y))\ \mathtt{in} \\
&\qquad \#_1^3(!y) \leftarrow x'; \#_3^3(!y) \leftarrow h(\#_3(!y))x'; y
\end{aligned}
$$

Similarly to the case of linked lists, z is bound to the location of the present tree. A tree is either a leaf or a node. A leaf is represented by $\mathtt{null}$. A node is represented by the location of the tuple $\langle x, y_1, y_2 \rangle$ of its element x, left sub-tree y_1, and right sub-tree y_2. The recursive function add'_x takes a tree y, inserts x into an appropriate place, and returns the updated tree. Function mem'_x searches a given tree for element x. Function del'_- is a little trickier: it looks for

a given element x in a given tree y, and replaces x with the minimum element in the right sub-tree.

We define a predicate $Set'(\ell, S, s)$ for binary search tree ℓ representing multiset S under store s, by induction as follows.

- $Set'(\texttt{null}, \emptyset, \emptyset)$.

- $Set'(\ell, \{i\} + S_1 + S_2, s_1 \uplus s_2 \uplus \{\ell \mapsto \langle i, \ell_1, \ell_2 \rangle\})$ if $\ell \neq \texttt{null}$, $Set'(\ell_1, S_1, s_1)$ and $Set'(\ell_2, S_2, s_2)$, with $i > j$ for any $j \in S_1$ and $i \leq k$ for any $k \in S_2$.

The following lemmas are proved by induction on the derivation of $Set'(\ell_0, S, s_0)$ or $Set'(\ell, S, s)$.

Lemma 5.8. *Suppose $Set'(\ell_0, S, s_0)$. Then, for any t, we have $s_0 \uplus t \triangleright add'_i(\ell_0) \twoheadrightarrow s \uplus t \triangleright \ell$ with $Set'(\ell, S + \{i\}, s)$*

Corollary 5.9. *Suppose $Set'(\ell_0, S, s_0)$. Then, for any t,*

$$s_0 \uplus t \uplus \{m \mapsto \ell_0\} \triangleright [m/z] add'(i) \quad \twoheadrightarrow \quad s \uplus t \uplus \{m \mapsto \ell\} \triangleright \langle\rangle$$

with $Set'(\ell, S + \{i\}, s)$

Lemma 5.10. *Suppose $Set'(\ell, S, s)$. Then, for any t, we have $s \uplus t \triangleright mem'_i(\ell) \twoheadrightarrow s \uplus t \triangleright b$ with $b = \texttt{true}$ if $i \in S$, and $b = \texttt{false}$ otherwise.*

Corollary 5.11. *Suppose $Set'(\ell, S, s)$. Then, for any t,*

$$s \uplus \{m \mapsto \ell\} \uplus t \triangleright [m/z] mem'(i) \quad \twoheadrightarrow \quad s \uplus \{m \mapsto \ell\} \uplus t \triangleright b$$

with $b = \texttt{true}$ if $i \in S$, and $b = \texttt{false}$ otherwise.

Lemma 5.12. *Suppose $Set'(\ell, S, s)$. If $\ell \neq \texttt{null}$, then for any t, we have $s \uplus t \triangleright min(\ell) \twoheadrightarrow s \uplus t \triangleright i$ and i is the minimum element of S.*

Lemma 5.13. *Suppose $Set'(\ell, S, s)$. Then, for any t, we have $s \uplus t \triangleright del'_-(\ell)i \twoheadrightarrow s_0 \uplus t \triangleright \ell_0$, with $Set'(\ell_0, S - \{i\}, s_0)$.*

Again, we give a detailed proof for the last lemma only.

PROOF. By induction on the size of S. Trivial if $\ell = \texttt{null}$. Suppose

- $\ell \neq \texttt{null}$,

- $S = \{i'\} + S_1 + S_2$,

- $s = s_1 \uplus s_2 \uplus \{\ell \mapsto \langle i', \ell_1, \ell_2 \rangle\}$,

- $Set'(\ell_1, S_1, s_1)$,

- $Set'(\ell_2, S_2, s_2)$,

- $i' > j$ for any $j \in S_1$, and

- $i' \leq k$ for any $k \in S_2$.

Case $i' = i$ and $\ell_2 = \texttt{null}$. Then $S_2 = \emptyset$ and $s_2 = \emptyset$, and we have $s \uplus t \triangleright del'_-(\ell)i \twoheadrightarrow s_1 \uplus t \triangleright \ell_1$, so it suffices to take $s_0 = s_1$ and $\ell_0 = \ell_1$.

21

Case $i' = i$ and $\ell_2 \neq$ null. Then by Lemma 5.12 we have $s \triangleright min(\ell_2) \twoheadrightarrow s \triangleright k'$ and k' is the minimum element of S_2. By induction, we obtain

$$s \uplus t \triangleright del'_\_(\ell)i \quad \twoheadrightarrow \quad s_1 \uplus s_3 \uplus \{\ell \mapsto \langle k', \ell_1, \ell_3 \rangle\} \uplus t \triangleright \ell$$

with $Set'(\ell_3, S_2 - \{k'\}, s_3)$. Thus, it suffices to take $s_0 = s_1 \uplus s_3 \uplus \{\ell \mapsto \langle k', \ell_1, \ell_3 \rangle\}$ and $\ell_0 = \ell$.

Case $i' < i$. Then, by induction,

$$s \uplus t \triangleright del'_\_(\ell)i \quad \twoheadrightarrow \quad s_3 \uplus s_2 \uplus \{\ell \mapsto \langle i', \ell_3, \ell_2 \rangle\} \uplus t \triangleright \ell$$

with $Set'(\ell_3, S_1 - \{i\}, s_3)$. Thus, it suffices to take $s_0 = s_3 \uplus s_2 \uplus \{\ell \mapsto \langle i', \ell_3, \ell_2 \rangle\}$ and $\ell_0 = \ell$.

Case $i' > i$. Similar to the case $i' < i$. □

Corollary 5.14. *Suppose $Set'(\ell, S, s)$. Then, for any t,*

$$s \uplus \{m \mapsto \ell\} \uplus t \triangleright [m/z] del'(i) \quad \twoheadrightarrow \quad s_0 \uplus \{m \mapsto \ell_0\} \uplus t \triangleright \langle \rangle$$

with $Set(\ell_0, S - \{i\}, s_0)$.

5.3. The bisimulation

We now prove the bisimulation between the multiset implementations by linked lists and binary search trees, roughly as follows:

- We first define a binary relation $\mathcal{R}_{m,m'}$ consisting of the three pairs of functions, taken from the two implementations. The parameters m and m' represent the (local) locations of the multiset data structures.

- Second, we define an environmental relation X consisting of configurations where no programs are running (the first and second subsets of X in the proof below) or the functions are about to start (the third and fourth subsets).

- We then define another environmental relation $Y \supseteq X$ that accounts for (public) locations allocated (and deallocated) by the context, as well as for reducts of the started functions.

- Finally, we prove that the context closure $Y^\star$ is an environmental bisimulation. This concludes the entire proof because $(\emptyset, \emptyset \triangleright set, \emptyset \triangleright set') \in Y^\star$.

Obviously, this proof is rather lengthy and burdensome. These burdens will be removed by the up-to techniques in Section 6.

Theorem 5.15. *set and set' are bisimilar. That is, $(\emptyset, \emptyset \triangleright set, \emptyset \triangleright set')$ belongs to the bisimilarity.*

PROOF. Take:

$$\mathcal{R}_{m,m'} = \{([m^\top/z]add, [m'^\top/z]add'),$$
$$([m^\top/z]mem, [m'^\top/z]mem'),$$
$$([m^\top/z]del, [m'^\top/z]del')\}$$

$$X = \{(\emptyset, \emptyset \triangleright set, \emptyset \triangleright set')\}$$
$$\cup \;\{(\mathcal{R}_{m,m'}, s \uplus \{m^\top \mapsto \ell^\top\}, s' \uplus \{m'^\top \mapsto \ell'^\top\}) \mid$$
$$Set(\ell^\top, S, s), \; Set'(\ell'^\top, S, s')\}$$
$$\cup \;\{(\mathcal{R}_{m,m'}, s \uplus \{m^\top \mapsto \ell^\top\} \triangleright d(i), s' \uplus \{m'^\top \mapsto \ell'^\top\} \triangleright d'(i)) \mid$$
$$Set(\ell^\top, S, s), \; Set'(\ell'^\top, S, s'), \; (d, d') \in \mathcal{R}_{m,m'}\}$$
$$\cup \;\{(\mathcal{R}_{m,m'}, s \uplus \{m^\top \mapsto \ell^\top\} \triangleright d(v), s' \uplus \{m'^\top \mapsto \ell'^\top\} \triangleright d'(v')) \mid$$
$$Set(\ell^\top, S, s), \; Set'(\ell'^\top, S, s'), \; (d, d') \in \mathcal{R}_{m,m'},$$
$$v \text{ and } v' \text{ are not integers}\}$$

$$Y = \{(\mathcal{S}, s \uplus \{\tilde{\ell}^\perp \mapsto \tilde{w}\} \setminus \tilde{m}^\perp, s' \uplus \{\tilde{\ell}'^\perp \mapsto \tilde{w}'\} \setminus \tilde{m}'^\perp) \mid$$
$$(\mathcal{R}, s, s') \in X,$$
$$\mathcal{S} = \mathcal{R} \cup \{(\tilde{\ell}^\perp, \tilde{\ell}'^\perp)\},$$
$$(\tilde{w}, \tilde{w}') \in \mathcal{S}^\star,$$
$$(\tilde{m}^\perp, \tilde{m}'^\perp) \in \mathcal{S}\}$$
$$\cup \;\{(\mathcal{S}, t \triangleright e, t' \triangleright e') \mid$$
$$(\mathcal{R}, s \triangleright d, s' \triangleright d') \in X,$$
$$\mathcal{S} = \mathcal{R} \cup \{(\tilde{\ell}^\perp, \tilde{\ell}'^\perp)\},$$
$$(\tilde{w}, \tilde{w}') \in \mathcal{S}^\star,$$
$$(\tilde{m}^\perp, \tilde{m}'^\perp) \in \mathcal{S},$$
$$s \uplus \{\tilde{\ell}^\perp \mapsto \tilde{w}\} \setminus \tilde{m}^\perp \triangleright d \twoheadrightarrow t \triangleright e,$$
$$s' \uplus \{\tilde{\ell}'^\perp \mapsto \tilde{w}'\} \setminus \tilde{m}'^\perp \triangleright d' \twoheadrightarrow t' \triangleright e'\}$$

We then prove that $Y^\star$ is a bisimulation, by case analysis and induction on the context in the definition of $Y^\star$. Since a simpler proof will be given in Section 7 by using a more general theorem in Section 6, we only sketch the outline here.

First, we prove that $Y^\star$ is reduction-closed. The cases for non-evaluation contexts are straightforward. Let the context be E. Cases where $E \neq [\,]$ follow by induction. The case $E = [\,]$ follows from the reduction-closed construction of Y with the lemmas in previous subsections. (Note that, if the arguments v and v' of d and d' are not integers, reductions get stuck both on the left hand side and on the right.) The inclusion of *all* reducts of d and d' works here because X involves only a deterministic fragment of our language with no dangling pointers (though the context can still create dangling pointers and be non-deterministic).

Then, we check each condition of bisimulation for each $(u, u') \in \mathcal{S}$ with $(\mathcal{S}, t, t') \in Y^\star$ by induction on the context. Again, the inductive cases are easy. As for the base case,

- Condition 1 and Condition 2 are satisfied by the (application-closed and projection-closed) construction of X and $\mathcal{R}_{m,m'}$, respectively. Note that $(v, v') \in \mathcal{S}^\star$ means either $v = v' = i$, or else v and v' are not integers (easy case analysis on contexts).

23

- Condition 3 and 4 are also satisfied by construction, i.e., by the inclusion of arbitrary $\ell^\perp$ and $\ell'^\perp$ (and the exclusion of arbitrary $m^\perp$ and $m'^\perp$) in Y.

$\square$

5.4. Beyond contextual equivalence

In fact, we can prove a stronger property than contextual equivalence by reusing the same bisimulation as above: $Set(\ell, S, s)$ and $Set(\ell', S, s')$ imply that the *number of locations*, both in s and in s', is equal to the size of S (this can be checked by easy induction on the size of S). Thus, we can define

$$\begin{aligned} P^{size} &= \{(\mathcal{R}, s \triangleright e, s' \triangleright e') \mid |dom(s)| = |dom(s')|\} \\ &\cup \{(\mathcal{R}, s, s') \mid |dom(s)| = |dom(s')|\} \end{aligned}$$

and prove that $(Y \cap P^{size})^\star$ is a P^{size}-bisimulation. (The intersection with P^{size} is for excluding intermediate states with mismatched numbers of locations on the left hand side and on the right.) In this particular example, even the previous *proof* (Section 5.3) remains valid only by replacing P^{obs} with the above P^{size} and Y with $Y \cap P^{size}$. The P^{size}-bisimulation then means that the number of locations allocated on the left hand side and on the right are the same *under arbitrary contexts*.

Similarly, we can easily prove (local) memory safety of the two implementations under arbitrary (public) contexts.

Definition 5.16 (local memory safety). *State $s \triangleright e$ is* local memory unsafe *if e is either $E[\mathtt{free}(\ell^\top)]$, $E[\ell^\top := v]$, or $E[!\ell^\top]$, with $\ell^\top \notin dom(s)$. It is* local memory safe *if not local memory unsafe. We often omit "local" and "memory," saying just "safe" or "unsafe," and write $safe(s \triangleright e)$ when $s \triangleright e$ is safe.*

Note that the definition above does not imply so-called "type safety," which is a more general property. For instance, *safe* does not preclude stuck states such as $\emptyset \triangleright (\ell\ 3)$. Note also that local memory safety only concerns local locations, so it is satisfied by a function like $\lambda x.\mathtt{free}(x); !x$ even though this function is (very) unsafe when applied to a public location by an (innocent) context like $\mathtt{new}\ z^\perp := \langle\rangle; [\,]z$. (In this work, we intend to protect programs' memory from contexts, but not vice versa. Alternatively, we could attach a localty $\perp$ or $\top$ to each read from or write to locations—instead of the locations themselves—in order to distinguish contexts' accesses from programs'.)

Then, we take

$$\begin{aligned} P^{safe} &= \{(\mathcal{R}, s \triangleright e, s' \triangleright e') \mid safe(s \triangleright e), safe(s' \triangleright e')\} \\ &\cup \{(\mathcal{R}, s, s') \mid \mathtt{true}\} \end{aligned}$$

and prove that $(Y \cap P^{safe})^\star$ is a P^{safe}-bisimulation, which is again straightforward: the only essential work is to check the predicate *safe* against each reduct of $add(v)$, $add'(v')$, etc. Note that the arguments v and v' can be non-integers, in which case the reducts get stuck at the integer addition $x + 0$. According to the definition, this is still considered (local) memory safe.

6. Up-to technique

As stated in Section 5.3, proof of bisimulation by using only its definition (Definition 4.5) is still tedious and bureaucratic. Specifically,

- the deallocation of arbitrary public locations $\tilde{m}^{\perp}$ and $\tilde{m}'^{\perp}$,

- the allocation of arbitrary public locations $\tilde{\ell}^{\perp}$ and $\tilde{\ell}'^{\perp}$ (with contents $\tilde{w}$ and $\tilde{w}'$), and

- the context closure $Y^{\star}$ instead of Y (or even X) itself

seem inessential by intuition.

To remove such bureaucracy, up-to techniques—as found in the bisimulation theory of concurrent calculi (see, e.g., Sangiorgi and Milner, 1992)—are useful in our case as well. There can be many up-to techniques and their combinations; we only present one of the most useful combinations below. Note that combination of up-to techniques is known to be subtle in general (Sangiorgi and Milner, 1992), so it is not straightforward to derive the soundness of a combination only from the soundness of each of the combined techniques.

Definition 6.1 (allocation closure). *The (binary) allocation closure of X is defined as:*

$$
\begin{aligned}
X^{\nu} \;=\;\; & \{(\mathcal{R}, s \triangleright e, s' \triangleright e') \mid (\mathcal{R}, s \triangleright e, s' \triangleright e') \in X\} \\
\cup\;\; & \{(\mathcal{S}, s \uplus \{\tilde{\ell}^{\perp} \mapsto \tilde{w}\} \setminus \tilde{m}^{\perp}, s' \uplus \{\tilde{\ell}'^{\perp} \mapsto \tilde{w}'\}) \setminus \tilde{m}'^{\perp} \mid \\
& \quad (\mathcal{R}, s, s') \in X,\; \mathcal{S} = \mathcal{R} \cup \{(\tilde{\ell}^{\perp}, \tilde{\ell}'^{\perp})\},\; (\tilde{w}, \tilde{w}') \in \mathcal{S}^{\star},\; (\tilde{m}^{\perp}, \tilde{m}'^{\perp}) \in \mathcal{S}\}
\end{aligned}
$$

where $s \uplus \{\tilde{\ell}^{\perp} \mapsto \tilde{w}\} \setminus \tilde{m}^{\perp}$ denotes extensions and restrictions of store s by locations $\tilde{\ell}$ and $\tilde{m}$, respectively, in any order.

Trivially, $X \subseteq X^{\nu} = (X^{\nu})^{\nu}$ for any X. Also, if P is consistent, then so is P^{ν}.

Definition 6.2 (environmental P-simulation up-to). *Let P be a (consistent) predicate on binary configurations. A subset X of P is called an* environmental P-simulation up-to context and allocation, *or just a P-simulation up-to in short, if all of the following conditions hold. (Differences from Definition 4.5 are underlined.)*

A. *For every $(\mathcal{R}, s \triangleright d, s' \triangleright d') \in X$,*
 (i) *If $s \triangleright d \to t \triangleright e$, then $s' \triangleright d' \twoheadrightarrow t' \triangleright e'$ and $(\mathcal{R}, t \triangleright e, t' \triangleright e') \in \underline{(X^{\nu})^{\star}}$.*
 (ii) *If $d = v$, then $s' \triangleright d' \twoheadrightarrow t' \triangleright v'$ and $(\mathcal{R} \cup \{(v, v')\}, s, t') \in \underline{(X^{\nu})^{\star}}$.*
 (iii) *Symmetric versions of the two conditions above, that is:*
  i'. *If $s' \triangleright d' \to t' \triangleright e'$, then $s \triangleright d \twoheadrightarrow t \triangleright e$ and $(\mathcal{R}, t \triangleright e, t' \triangleright e') \in \underline{(X^{\nu})^{\star}}$.*
  ii'. *If $d' = v'$, then $s \triangleright d \twoheadrightarrow t \triangleright v$ and $(\mathcal{R} \cup \{(v, v')\}, t, s') \in \underline{(X^{\nu})^{\star}}$.*
B. *For every $(\mathcal{R}, s, s') \in X$ and $(u, u') \in \mathcal{R}$,*
 (1) *If $u = \lambda x.\, e$ and $u' = \lambda x.\, e'$, then for any $\underline{(\mathcal{S}, t, t') \in \{(\mathcal{R}, s, s')\}^{\nu}\ and}$ $(v, v') \in \mathcal{S}^{\star}$, we have $(\mathcal{S}, t \triangleright uv, t' \triangleright u'v') \in X$.*

(2) If $u = \langle v_1, \ldots, v_i, \ldots, v_n \rangle$ and $u' = \langle v_1', \ldots, v_i', \ldots, v_n' \rangle$, then $(\mathcal{R} \cup \{(v_i, v_i')\}, s, s') \in \underline{(X^\nu)^\star}$.

(3) If $u = \ell^\perp$, $u' = \overline{\ell'^\perp}$, $\ell^\perp \in dom(s)$ and $\ell'^\perp \in dom(s')$, then $(\mathcal{R} \cup \{(s(\ell^\perp), s'(\ell'^\perp))\}, s, s') \in \underline{(X^\nu)^\star}$.

Intuitively, these conditions are a sound simplification of the original conditions of P-simulation by allowing to omit elements of the P-simulation. Specifically,

- The context closure $\star$ allow us to omit smaller knowledge ($\mathcal{S} \subseteq \mathcal{R}^\star_{val}$ in Definition 4.6) as well as configurations that can be reconstructed by the context from other configurations (again see Definition 4.6). Technically, all the positive occurrences of X are replaced with $(X^\nu)^\star$ except in Condition B1 (because it does not correspond to any reduction step; it does so only in combination with Condition Ai and its symmetric version).

- The allocation closure ν allow us to omit allocation of, writing to, and deallocation of public locations *as long as* their contents can be reconstructed by the context from its knowledge (see Definition 6.1). Note that the closure $\{(\mathcal{R}, s, s')\}^\nu$ before function application in Condition B1 is still essential for soundness: consider, for example, a function that takes n locations and checks whether they are pairwise distinct.

Theorem 6.3 (soundness of P-simulation up-to). *Let X be a P-simulation up-to. Then $(X^\nu)^\star$ is a $(P^\nu)^\star$-simulation.*

PROOF. First, we prove that $(X^\nu)^\star$ is reduction-closed. The proof is similar to that of Lemma 4.9 (soundness of P-similarity) except for the cases $C = \mathtt{free}(C_1)$ or $C_1 := C_2$ or $(\mathtt{new}\ x := C_1; C_2)$, where Condition 3a, 3b and 4 of Definition 4.5 (P-simulation) are no longer available in Definition 6.2 (P-simulation up-to); instead, the required conditions follow from Definition 6.1 (allocation closure).

To be concrete, we carry out the proof by case analysis on elements of $(X^\nu)^\star$ along Definition 4.6, as follows. Key differences from the proof of Lemma 4.9 are underlined below.

Case $(\mathcal{S}, s \triangleright [\tilde{v}/\tilde{x}]E[e], s' \triangleright [\tilde{v}'/\tilde{x}]E[e']) \in \underline{(X^\nu)^\star}$ with $(\mathcal{R}, s \triangleright e, s' \triangleright e') \in \underline{X^\nu}$ and $\mathcal{S} \subseteq \mathcal{R}^\star_{val}$ and $(\tilde{v}, \tilde{v}') \in \mathcal{R}$ and $fv(E) \subseteq \{\tilde{x}\}$. $\underline{\text{By Definition 6.1, } (\mathcal{R}, s \triangleright e, s' \triangleright e') \in X^\nu}$ $\underline{\text{implies } (\mathcal{R}, s \triangleright e, s' \triangleright e') \in X}$.

To prove Condition i (of Definition 4.3), suppose $s \triangleright [\tilde{v}/\tilde{x}]E[e] \rightarrow$. Since $(\mathcal{R}, s \triangleright e, s' \triangleright e') \in X$ and X is a P-simulation up-to, if e is a value, then e' also reduces to some value (by Condition Aii of Definition 6.2) and the rest of the proof amounts to the next case. Suppose thus that e is not a value. Since $s \triangleright [\tilde{v}/\tilde{x}]E[e] \rightarrow$ and E is an evaluation context, we have $s \triangleright e \rightarrow t \triangleright d$ for some t and d. Again since $(\mathcal{R}, s \triangleright e, s' \triangleright e') \in X$ and X is a P-simulation up-to, we have $s' \triangleright e' \twoheadrightarrow t' \triangleright d'$ for some t' and d' with $(\mathcal{R}, t \triangleright d, t' \triangleright d') \in \underline{(X^\nu)^\star}$ (by Condition Ai of Definition 6.2). Hence $(\mathcal{S}, t \triangleright [\tilde{v}/\tilde{x}]E[d], t' \triangleright [\tilde{v}'/\tilde{x}]E[d']) \in \underline{((X^\nu)^\star)^\star = (X^\nu)^\star}$ by Definition 4.6.

To prove Condition ii, suppose $[\tilde{v}/\tilde{x}]E[e]$ is a value, which we call w. Since E is an evaluation context, this can be the case only if e is also a value v. Then,

26

since $(\mathcal{R}, s \triangleright v, s' \triangleright e') \in X$ and X is a P-simulation up-to, we have $s' \triangleright e' \twoheadrightarrow t' \triangleright v'$ for some t' and v' with $(\mathcal{R} \cup \{(v, v')\}, s, t') \in (X^\nu)^\star$ (by Condition Aii of Definition 6.2). Since $[\tilde{v}/\tilde{x}]E[v]$ is a value, so is $[\tilde{v}'/\tilde{x}]E[v']$ (by Lemma 4.8), which we call w'. Since $(w, w') \in (\mathcal{R} \cup \{(v, v')\})^\star_{val}$ by Definition 4.2, we obtain $(\mathcal{S} \cup \{(w, w')\}, s, t') \in ((X^\nu)^\star)^\star = (X^\nu)^\star$ by Definition 4.6.

The proof of Condition iii is symmetric to the proofs above.

Case $(\mathcal{S}, s \triangleright [\tilde{v}/\tilde{x}]C, s' \triangleright [\tilde{v}'/\tilde{x}]C) \in (X^\nu)^\star$ with $(\mathcal{R}, s, s') \in X^\nu$ and $\mathcal{S} \subseteq \mathcal{R}^\star_{val}$ and $(\tilde{v}, \tilde{v}') \in \mathcal{R}$ and $fv(C) \subseteq \{\tilde{x}\}$. By Definition 6.1, $(\mathcal{R}, s, s') \in X^\nu$ implies $(\mathcal{R}, s, s') \in \{(\mathcal{R}_0, s_0, s_0')\}^\nu$ for some $(\mathcal{R}_0, s_0, s_0') \in X$.

First, we prove Condition i (of Definition 4.3) by induction on C. Suppose $s \triangleright [\tilde{v}/\tilde{x}]C \rightarrow$.

If C of the form $E[D]$ for some $E \neq [\,]$, and if $s \triangleright [\tilde{v}/\tilde{x}]D \rightarrow t \triangleright d$ for some t and d, then $s' \triangleright [\tilde{v}'/\tilde{x}]D \twoheadrightarrow t' \triangleright d'$ for some t' and d' with $(\mathcal{S}, t \triangleright d, t' \triangleright d') \in (X^\nu)^\star$ by the induction hypothesis. Hence $s' \triangleright [\tilde{v}'/\tilde{x}]C \twoheadrightarrow t' \triangleright E[d']$, with $(\mathcal{S}, t \triangleright E[d], t' \triangleright E[d']) \in ((X^\nu)^\star)^\star = (X^\nu)^\star$ by Definition 4.6.

Otherwise, we proceed by case analysis on C.

Subcase $C = C_1 C_2$. Then $[\tilde{v}/\tilde{x}]C_1$ is a λ-abstraction and $[\tilde{v}/\tilde{x}]C_2$ (resp. $[\tilde{v}'/\tilde{x}]C_2$, by Lemma 4.8) is a value w (resp. w').

If C_1 itself is a λ-abstraction $\lambda x. C_0$, then the only possible reduction on the "left hand side" (of the bisimulation) is $s \triangleright [\tilde{v}/\tilde{x}]C \rightarrow s \triangleright [\tilde{v}/\tilde{x}]([C_2/x]C_0)$, which corresponds to $s' \triangleright [\tilde{v}'/\tilde{x}]C \rightarrow s' \triangleright [\tilde{v}'/\tilde{x}]([C_2/x]C_0)$ on the right hand side, with $(\mathcal{S}, s \triangleright [\tilde{v}/\tilde{x}]([C_2/x]C_0), s' \triangleright [\tilde{v}'/\tilde{x}]([C_2/x]C_0)) \in (X^\nu)^\star$ by Definition 4.6.

Otherwise, C_1 is a variable x_i and v_i is a λ-abstraction. Since $(v_i, v_i') \in R$ and $(\mathcal{R}, s, s') \in \{(\mathcal{R}_0, s_0, s_0')\}^\nu$ and v_i is not a location, we have $(v_i, v_i') \in \mathcal{R}_0$ by Definition 6.1. Since X is a $P^\star$-simulation up-to, v_i' is also a λ-abstraction by Definition 4.4, and therefore $(\mathcal{R}, s \triangleright v_i w, s' \triangleright v_i' w') \in X$ by Condition B1 of Definition 6.2. Again since X is a $P^\star$-simulation up-to, if $s \triangleright v_i w \rightarrow t \triangleright e$ for some t and e, then $s' \triangleright v_i' w' \twoheadrightarrow t' \triangleright e'$ for some t' and e' with $(\mathcal{R}, t \triangleright e, t' \triangleright e') \in (X^\nu)^\star$ (by Condition Ai of Definition 6.2). Hence $(\mathcal{S}, t \triangleright e, t' \triangleright e') \in ((X^\nu)^\star)^\star = (X^\nu)^\star$ by Definition 4.6.

Subcase $C = op(C_1, \ldots, C_n)$. Then $[\tilde{v}/\tilde{x}]C_i$ is a constant c_i, for $i = 1, \ldots, n$, and $s \triangleright [\tilde{v}/\tilde{x}]C \rightarrow s \triangleright c$ for $c = [\![op(c_1, \ldots, c_n)]\!]$. If C_i itself is c_i, then $[\tilde{v}'/\tilde{x}]C_i = c_i$. Otherwise, C_i is a variable x_i and $v_i = c_i$. By Definition 4.4, $v_i' = c_i$. Therefore, $[\tilde{v}'/\tilde{x}]C_i = c_i$ anyway. Hence $s' \triangleright [\tilde{v}'/\tilde{x}]C \rightarrow s' \triangleright c$, with $(\mathcal{S}, s \triangleright c, s' \triangleright c) \in (X^\nu)^\star$ by Definition 4.6.

Subcase $C = \texttt{if } C_1 \texttt{ then } C_2 \texttt{ else } C_3$. Then $[\tilde{v}/\tilde{x}]C_1$ is a Boolean constant b and so is $[\tilde{v}'/\tilde{x}]C_1$ (for the same reason as in the previous subcase). If $b = \texttt{true}$, then the only possible reduction on the left hand side is $s \triangleright [\tilde{v}/\tilde{x}]C \rightarrow s \triangleright [\tilde{v}/\tilde{x}]C_2$, which corresponds to $s' \triangleright [\tilde{v}'/\tilde{x}]C \rightarrow s' \triangleright [\tilde{v}'/\tilde{x}]C_2$ on the right hand side, with $(\mathcal{S}, s \triangleright [\tilde{v}/\tilde{x}]C_2, s' \triangleright [\tilde{v}'/\tilde{x}]C_2) \in (X^\nu)^\star$ by Definition 4.6. The case $b = \texttt{false}$ is similar.

Subcase $C = \#_i(C_0)$. Then $[\tilde{v}/\tilde{x}]C_0$ is a tuple $\langle w_1, \ldots, w_n \rangle$ and $s \triangleright [\tilde{v}/\tilde{x}]C \rightarrow s \triangleright w_i$.

27

If C_0 itself is a tuple $\langle C_1, \ldots, C_n \rangle$, then $[\tilde{v}/\tilde{x}]C_i = w_i$, so $[\tilde{v}'/\tilde{x}]C_i$ is also a value w_i' (Lemma 4.8), for $i = 1, \ldots, n$. Hence $s' \triangleright [\tilde{v}'/\tilde{x}]C \to s' \triangleright w_i'$, with $(\mathcal{S}, s \triangleright w_i, s' \triangleright w_i') \in (X^\nu)^\star$ by Definition 4.6.

Otherwise, C_0 is a variable x_i and $v_i = \langle w_1, \ldots, w_n \rangle$. Since $(v_i, v_i') \in R$ and $(\mathcal{R}, s, s') \in \{(\mathcal{R}_0, s_0, s_0')\}^\nu$ and v_i is not a location, we have $(v_i, v_i') \in \mathcal{R}_0$ by Definition 6.1. Since X is a $P^\star$-simulation up-to, v_i' is also a tuple $\langle w_1', \ldots, w_n' \rangle$ by Definition 4.4, and therefore $(\mathcal{R}_0 \cup \{(w_i, w_i')\}, s_0, s_0') \in (X^\nu)^\star$ by Condition B2 of Definition 6.2. Hence $s' \triangleright [\tilde{v}'/\tilde{x}]C \to s' \triangleright w_i'$, with $(\mathcal{S}, s \triangleright w_i, s' \triangleright w_i') \in ((X^\nu)^\star)^\star = (X^\nu)^\star$ by Definition 4.6.

Subcase $C = (\texttt{new } x^\perp := C_1; C_2)$. Then $[\tilde{v}/\tilde{x}]C_1$ (resp. $[\tilde{v}'/\tilde{x}]C_1$, by Lemma 4.8) is a value w (resp. w') and the only possible reductions on the left hand side are of the form $s \triangleright [\tilde{v}/\tilde{x}]C \to s \uplus \{\ell^\perp \mapsto w\} \triangleright [\tilde{v}, \ell^\perp/\tilde{x}, x]C_2$ for some $\ell^\perp \notin dom(s)$, which corresponds to $s' \triangleright [\tilde{v}'/\tilde{x}]C \to s' \uplus \{\ell'^\perp \mapsto w'\} \triangleright [\tilde{v}', \ell'^\perp/\tilde{x}, x]C_2$ for some $\ell'^\perp \notin dom(s')$ on the right hand side. Since $(\mathcal{R}, s, s') \in X^\nu$, we have $(\mathcal{R} \cup \{(\ell^\perp, \ell'^\perp)\}, s \uplus \{\ell^\perp \mapsto w\}, s' \uplus \{\ell'^\perp \mapsto w'\}) \in (X^\nu)^\nu = X^\nu$ by Definition 6.1. Hence $(\mathcal{S}, s \uplus \{\ell^\perp \mapsto w\} \triangleright [\tilde{v}, \ell^\perp/\tilde{x}, x]C_2, s' \uplus \{\ell'^\perp \mapsto w'\} \triangleright [\tilde{v}', \ell'^\perp/\tilde{x}, x]C_2) \in (X^\nu)^\star$ by Definition 4.6.

Subcase $C = \texttt{free}(C_1)$. Then $[\tilde{v}/\tilde{x}]C_1$ is a location ℓ^π with $s = t \uplus \{\ell^\pi \mapsto w\}$ for some t and w, so $s \triangleright [\tilde{v}/\tilde{x}]C \to t \triangleright \langle\rangle$. Since contexts are location-free, it must be that C_1 is a variable x_i and $v_i = \ell^\pi$. By Definition 4.4, we have $\pi = \perp$ and v_i' is also a public location $\ell'^\perp$ with $s' = t' \uplus \{\ell'^\perp \mapsto w'\}$ for some t' and w', so $s' \triangleright [\tilde{v}'/\tilde{x}]C \to t' \triangleright \langle\rangle$. Since $(\ell^\perp, \ell'^\perp) \in \mathcal{R}$ and $(\mathcal{R}, s, s') \in X^\nu$, we have $(\mathcal{R}, t, t') \in (X^\nu)^\nu = X^\nu$ by Definition 6.1. Hence $(\mathcal{S}, t \triangleright \langle\rangle, t' \triangleright \langle\rangle) \in (X^\nu)^\star$ by Definition 4.6.

Subcase $C = (C_1 := C_2)$. Then $[\tilde{v}/\tilde{x}]C_1$ is a location ℓ^π, and $[\tilde{v}/\tilde{x}]C_2$ (resp. $[\tilde{v}'/\tilde{x}]C_2$, by Lemma 4.8) is a value w (resp. w'), with $s = t \uplus \{\ell^\pi \mapsto u\}$ for some t and u, so $s \triangleright [\tilde{v}/\tilde{x}]C \to t \uplus \{\ell^\pi \mapsto w\} \triangleright \langle\rangle$. Since contexts are location-free, it must be that C_1 is a variable x_i and $v_i = \ell^\pi$. By Definition 4.4, we have $\pi = \perp$ and v_i' is also a public location $\ell'^\perp$, with $s' = t' \uplus \{\ell'^\perp \mapsto u'\}$ for some t' and u', so $s' \triangleright [\tilde{v}'/\tilde{x}]C \to t' \uplus \{\ell'^\perp \mapsto w'\} \triangleright \langle\rangle$. Since $(\ell^\perp, \ell'^\perp) \in \mathcal{R}$ and $(\mathcal{R}, s, s') \in X^\nu$, and since $(w, w') \in \mathcal{R}^\star$, we have $(\mathcal{R}, t \uplus \{\ell^\perp \mapsto w\}, t' \uplus \{\ell'^\perp \mapsto w'\}) \in (X^\nu)^\nu = X^\nu$ by Definition 6.1. Hence $(\mathcal{S}, t \uplus \{\ell^\perp \mapsto w\} \triangleright \langle\rangle, t' \uplus \{\ell'^\perp \mapsto w'\} \triangleright \langle\rangle) \in (X^\nu)^\star$ by Definition 4.6.

Subcase $C = {!}C_1$. Then $[\tilde{v}/\tilde{x}]C_1$ is a location ℓ^π with $s = t \uplus \{\ell^\pi \mapsto w\}$ for some t and w, so $s \triangleright [\tilde{v}/\tilde{x}]C \to s \triangleright w$. Since contexts are location-free, it must be that C_1 is a variable x_i and $v_i = \ell^\pi$. By Definition 4.4, we have $\pi = \perp$ and v_i' is also a public location $\ell'^\perp$ with $s' = t' \uplus \{\ell'^\perp \mapsto w'\}$ for some t' and w', so $s' \triangleright [\tilde{v}'/\tilde{x}]C \to s' \triangleright w'$. Since $(\ell^\perp, \ell'^\perp) \in \mathcal{R}$ and $(\mathcal{R}, s, s') \in X^\nu$, the locations $\ell^\perp$ and $\ell'^\perp$ are either introduced by X^ν or else taken from $\mathcal{R}_0$. In the former case, we have $(w, w') \in R^\star$ by Definition 6.1. In the latter case, we have $(\mathcal{R}_0 \cup \{(w, w')\}, s_0, s_0') \in (X^\nu)^\star$ by Condition B3 of Definition 6.2. Hence $(\mathcal{S}, s \triangleright w, s' \triangleright w') \in (((X^\nu)^\star)^\nu)^\star = (X^\nu)^\star$ by Definition 4.6 and 6.1.

Subcase $C = (C_1 \stackrel{ptr}{=} C_2)$. Then $[\tilde{v}/\tilde{x}]C_1$ and $[\tilde{v}/\tilde{x}]C_2$ are locations $\ell_1^{\pi_1}$ and

$\ell_2^{\pi_2}$, respectively, so $s \triangleright [\tilde{v}/\tilde{x}]C \to s \triangleright b$, where b is $\mathtt{true}$ if $\ell_1^{\pi_1} = \ell_2^{\pi_2}$ and $\mathtt{false}$ otherwise. Since contexts are location-free, it must be that C_1 and C_2 are variables x_i and x_j, respectively, with $v_i = \ell_1^{\pi_1}$ and $v_j = \ell_2^{\pi_2}$. By Definition 4.4, we have $\pi_1 = \pi_2 = \bot$, and v_i' and v_j' are also public locations $\ell_1'^{\bot}$ and $\ell_2'^{\bot}$, respectively, with $(\ell_1^{\bot} = \ell_2^{\bot}) \iff (\ell_1'^{\bot} = \ell_2'^{\bot})$. Hence $s' \triangleright [\tilde{v}'/\tilde{x}]C \to s' \triangleright b$, with $(\mathcal{S}, s \triangleright b, s' \triangleright b) \in (X^\nu)^\star$ by Definition 4.6.

This concludes the proof of Condition i (of Definition 4.3).

To prove Condition ii, suppose $[\tilde{v}/\tilde{x}]C$ is a value w. Then $[\tilde{v}'/\tilde{x}]C$ is also a value w' (Lemma 4.8) and $(w, w') \in \mathcal{R}^\star$ by Definition 4.2. Hence $(\mathcal{S} \cup \{(w, w')\}, s, s') \in (X^\nu)^\star$ by Definition 4.6.

The proof of Condition iii is symmetric to those of Condition i and ii.

Case $(\mathcal{S}, s, s') \in (X^\nu)^\star$ with $(\mathcal{R}, s, s') \in P$ and $\mathcal{S} \subseteq \mathcal{R}_{val}^\star$. Definition 4.3 requires no condition for elements of this form.

This concludes the proof that $(X^\nu)^\star$ is reduction-closed.

Then, we prove Conditions 1 to 4 (of Definition 4.5) for $(X^\nu)^\star$. Suppose $(\mathcal{R}, s, s') \in (X^\nu)^\star$ and $(u, u') \in R$. By Definition 4.6 and 6.1, we have $\mathcal{R} \subseteq (\mathcal{R}_0)_{val}^\star$ and $(\mathcal{R}_0, s, s') \in \{(\mathcal{R}_1, s_0, s_0')\}^\nu$ for some $\mathcal{R}_0$ and $(\mathcal{R}_1, s_0, s_0') \in X$.

To prove Condition 1, suppose $(v, v') \in \mathcal{R}^\star$. Since $(u, u') \in \mathcal{R}$ and $(v, v') \in \mathcal{R}^\star$ with $(\mathcal{R}, s, s') \in (X^\nu)^\star$, we have $(\mathcal{R}, s \triangleright uv, s' \triangleright u'v') \in ((X^\nu)^\star)^\star = (X^\nu)^\star$ by Definition 4.6.

To prove Condition 2, suppose $u = \langle v_1, \ldots, v_i, \ldots, v_n \rangle$ and $u' = \langle v_1', \ldots, v_i', \ldots, v_n' \rangle$. Since $(u, u') \in \mathcal{R} \subseteq (\mathcal{R}_0)_{val}^\star$, the tuples u and u' are either introduced by $(\mathcal{R}_0)_{val}^\star$ or else taken from $\mathcal{R}_0$. In the former case, we have $(v_i, v_i') \in (R_0)_{val}^\star$ by Definition 4.2, so $(\mathcal{R} \cup \{(v_i, v_i')\}, s, s') \in (X^\nu)^\star$ by Definition 4.6. In the latter case, since $(u, u) \in \mathcal{R}_0$ and $(\mathcal{R}_0, s, s') \in \{(\mathcal{R}_1, s_0, s_0')\}^\nu$ and u and u' are not locations, we have $(u, u') \in \mathcal{R}_1$ by Definition 6.1, so $(\mathcal{R}_1 \cup \{(v_i, v_i')\}, s_0, s_0') \in (X^\nu)^\star$ by Condition B2 of Definition 6.2. Hence $(\mathcal{R} \cup \{(v_i, v_i')\}, s, s') \in (((X^\nu)^\star)^\nu)^\star = (X^\nu)^\star$ by Definition 6.1 and 4.6.

To prove Condition 3a, 3b and 3c, suppose $u = \ell^{\bot}$, $u' = \ell'^{\bot}$, $s = t \uplus \{\ell^{\bot} \mapsto v\}$ and $s' = t' \uplus \{\ell'^{\bot} \mapsto v'\}$. Since $(\ell^{\bot}, \ell'^{\bot}) \in \mathcal{R} \subseteq (\mathcal{R}_0)_{val}^\star$, we have $(\ell^{\bot}, \ell'^{\bot}) \in \mathcal{R}_0$ by Definition 4.2.

- Condition 3a. Since $(\mathcal{R}_0, s, s') \in X^\nu$ and $(\ell^{\bot}, \ell'^{\bot}) \in \mathcal{R}_0$, we have $(\mathcal{R}_0, t, t') \in (X^\nu)^\nu = X^\nu$ by Definition 6.1. Hence $(\mathcal{R}, t, t') \in (X^\nu)^\star$ by Definition 4.6.

- Condition 3b. Suppose $(w, w') \in \mathcal{R}^\star \subseteq ((\mathcal{R}_0)_{val}^\star)_{val}^\star = (\mathcal{R}_0)_{val}^\star$. Since $(\mathcal{R}_0, s, s') \in X^\nu$ and $(\ell^{\bot}, \ell'^{\bot}) \in \mathcal{R}_0$, we have $(\mathcal{R}_0, t \uplus \{\ell^{\bot} \mapsto w\}, t' \uplus \{\ell'^{\bot} \mapsto w'\}) \in (X^\nu)^\nu = X^\nu$ by Definition 6.1. Hence $(\mathcal{R}, t \uplus \{\ell^{\bot} \mapsto w\}, t' \uplus \{\ell'^{\bot} \mapsto w'\}) \in (X^\nu)^\star$ by Definition 4.6.

- Condition 3c. Since $(\ell^{\bot}, \ell'^{\bot}) \in \mathcal{R}_0$ and $(\mathcal{R}_0, s, s') \in \{(\mathcal{R}_1, s_0, s_0')\}^\nu$, the locations $\ell^{\bot}$ and $\ell'^{\bot}$ are either introduced by $\{(\mathcal{R}_1, s_0, s_0')\}^\nu$ or else taken from $\mathcal{R}_1$. In the former case, we have $(v, v') \in \mathcal{R}_0^\star$ by Definition 6.1, so $(\mathcal{R} \cup \{(v, v')\}, s, s') \in (X^\nu)^\star$ by Definition 4.6. In the later case, we have $(\mathcal{R}_1 \cup \{(v, v')\}, s_0, s_0') \in (X^\nu)^\star$ by Condition B3 of Definition 6.2. Hence $(\mathcal{R} \cup \{(v, v')\}, s, s') \in (((X^\nu)^\star)^\nu)^\star = (X^\nu)^\star$ by Definition 6.1 and 4.6.

Finally, to prove Condition 4, suppose $\ell^\perp \notin dom(s)$ and $(v, v') \in \mathcal{R}^\star \subseteq ((\mathcal{R}_0)^\star_{val})^\star_{val} = (\mathcal{R}_0)^\star_{val}$. Then we have $(\mathcal{R} \cup \{(\ell^\perp, \ell'^\perp)\}, s \uplus \{\ell^\perp \mapsto v\}, s' \uplus \{\ell'^\perp \mapsto v'\}) \in ((X^\nu)^\star)^\nu = (X^\nu)^\star$ for some $\ell'^\perp \notin dom(s')$ by Definition 6.1.　　□

7. Examples of P-bisimulations up-to

To (re-)prove the bisimulation and other results in Section 5, take

$$
\begin{aligned}
X \;=\; & \{(\emptyset, \emptyset \triangleright set, \emptyset \triangleright set')\} \\
\cup\; & \{(\mathcal{R}_{m,m'}, s \uplus \{m^\top \mapsto \ell^\top\}, s' \uplus \{m'^\top \mapsto \ell'^\top\}) \mid \\
& \quad Set(\ell^\top, S, s), Set'(\ell'^\top, S, s')\} \\
\cup\; & \{(\mathcal{S}, t \triangleright e, t' \triangleright e') \mid \\
& \quad Set(\ell^\top, S, s), Set'(\ell'^\top, S, s'), (d, d') \in \mathcal{R}_{m,m'}, \\
& \quad \mathcal{S} = \mathcal{R}_{m,m'} \cup \{(\tilde{\ell}^\perp, \tilde{\ell}'^\perp)\}, \\
& \quad (v, v'), (\tilde{w}, \tilde{w}') \in \mathcal{S}^\star, \\
& \quad (\tilde{m}^\perp, \tilde{m}'^\perp) \in \mathcal{S}, \\
& \quad s \uplus \{m^\top \mapsto \ell^\top\} \uplus \{\tilde{\ell}^\perp \mapsto \tilde{w}\} \setminus \tilde{m}^\perp \triangleright d(v) \twoheadrightarrow t \triangleright e, \\
& \quad s' \uplus \{m'^\top \mapsto \ell'^\top\} \uplus \{\tilde{\ell}'^\perp \mapsto \tilde{w}'\} \setminus \tilde{m}'^\perp \triangleright d'(v') \twoheadrightarrow t' \triangleright e'\}
\end{aligned}
$$

where

$$
\begin{aligned}
\mathcal{R}_{m,m'} \;=\; & \{([m^\top/z]add, [m'^\top/z]add'), \\
& \;\; ([m^\top/z]mem, [m'^\top/z]mem'), \\
& \;\; ([m^\top/z]del, [m'^\top/z]del')\}.
\end{aligned}
$$

Then $X \cap P$ is a P-bisimulation up-to, for any $P \in \{P^{obs}, P^{size}, P^{safe}\}$ (see Section 5 for their definitions). Note that $P^\nu \subseteq P$ holds for all of them (immediate from each of the definitions).

The first and second subsets (the first three lines) of X are the same as those in Section 5, representing the configurations with no running programs. The third subset (the other lines) corresponds to the third and fourth subsets of X in Section 5, and to the latter half of Y in Section 5, representing the configurations in the middle of reductions. The first half of Y in Section 5 is now omitted, thanks to the up-to allocation technique. We do not have to consider $Y^\star$, either, thanks to the up-to context technique.

It may also be possible to remove the reducts $t \triangleright e$ and $t' \triangleright e'$ from the X above by developing an "up-to deterministic reduction" technique. To be concrete, let $\rightharpoonup$ be the largest deterministic subset of $\rightarrow$. Then, we may replace the first two occurrences of $(X^\nu)^\star$ with $((X^\nu)^\star)^\rightharpoonup$ in Definition 6.2 (environmental P-simulation up-to), where $X^\rightharpoonup$ denotes (deterministic) reduction closure of an environmental bisimulation X. In our operational semantics, however, *all* allocations are non-deterministic per se: the reduction $s \triangleright (\texttt{new } x := v; e) \rightarrow (s \uplus \{\ell \mapsto v\}) \triangleright [\ell/x]e$ holds for *any* fresh (and non-null) location ℓ. Thus, in fact, we would need to develop an "up-to deterministic reduction and renaming" technique to allow the difference of fresh names. Confer Sumii (2009, Definition 7) for such a technique in a language that is deterministic up to renaming.

8. Unary environmental predicates

Suppose that we want to prove the local memory safety of the multiset implementation *set* by linked lists. In Section 5.4, we proved it in combination with contextual equivalence to another multiset implementation *set'* (by binary search trees). However, if we are interested only in the safety of *set*, there is no reason to care about *set'*. Instead, we can just consider the "bisimulation" between *set* and *set* itself! This idea generalizes to the following definitions.

Definition 8.1. *A* unary configuration *is a triple of the form* $(\mathcal{R}, s \triangleright e)$ *or a pair of the form* $(\mathcal{R}, s)$*, where* $\mathcal{R}$ *is a predicate on values.*

Definition 8.2 (environmental P-predicate). *Let P be a predicate on unary configurations. A set $X \subseteq P$ of unary configurations is called an* environmental P-predicate *if its* duplication $X^2 = \{(\mathcal{R}^2, s \triangleright e, s \triangleright e) \mid (\mathcal{R}, s \triangleright e) \in X\} \cup \{(\mathcal{R}^2, s, s) \mid (\mathcal{R}, s) \in X\}$ *is an environmental P^2-simulation, where $\mathcal{R}^2 = \{(v, v) \mid v \in \mathcal{R}\}$. To spell out all the conditions,*

1. *For every $(\mathcal{R}, s \triangleright d) \in X$,*
    (a) *If $s \triangleright d \to t \triangleright e$, then $(\mathcal{R}, t \triangleright e) \in X$.*
    (b) *If $d = v$, then $(\mathcal{R} \cup \{v\}, s) \in X$.*
2. *For every $(\mathcal{R}, s) \in X$ and $u \in \mathcal{R}$,*
    (a) *If $u = \lambda x. e$, then $(\mathcal{R}, s \triangleright uv) \in X$ for any $v \in \mathcal{R}^\star$.*
    (b) *If $u = \langle v_1, \ldots, v_i, \ldots, v_n \rangle$, then $(\mathcal{R} \cup \{v_i\}, s) \in X$.*
    (c) *If $u = \ell^\perp$ and $s = t \uplus \{\ell^\perp \mapsto v\}$, then $(\mathcal{R}, t) \in X$, $(\mathcal{R}, t \uplus \{\ell^\perp \mapsto w\}) \in X$ for any $w \in \mathcal{R}^\star$, and $(\mathcal{R} \cup \{v\}, s) \in X$.*
    (d) *$(\mathcal{R} \cup \{\ell^\perp\}, s \uplus \{\ell^\perp \mapsto v\}) \in X$ for any $\ell^\perp \notin dom(s)$ and $v \in \mathcal{R}^\star$.*

where the unary version of context closure is defined as $\mathcal{R}^\star = \{[\tilde{v}/\tilde{x}]C \mid \tilde{v} \in \mathcal{R}, \; fv(C) \subseteq \{\tilde{x}\}\}$.

All the results from binary environmental P-simulations apply to this unary version, because the latter is just a special case of the former. This includes soundness and the up-to technique. For pedagogy, we spell out the conditions of environmental P-predicate up-to context and allocation.

Definition 8.3 (allocation closure). *The (unary)* allocation closure *of X is defined as:*

$$
\begin{aligned}
X^\nu \;=\; & \{(\mathcal{R}, s \triangleright e) \mid (\mathcal{R}, s \triangleright e) \in X\} \\
\cup \; & \{(\mathcal{S}, s \uplus \{\tilde{\ell}^\perp \mapsto \tilde{w}\} \setminus \tilde{m}^\perp) \mid (\mathcal{R}, s) \in X, \; \mathcal{S} = \mathcal{R} \cup \{\tilde{\ell}^\perp\}, \; \tilde{w} \in \mathcal{S}^\star, \; \tilde{m}^\perp \in \mathcal{S}\}
\end{aligned}
$$

Definition 8.4 (environmental P-predicate up-to). *A set $X \subseteq P$ of unary configurations is called an* environmental P-predicate up-to context and allocation *(or just a "P-predicate up-to" in short) if*

A. *For every $(\mathcal{R}, s \triangleright d) \in X$,*
    (i) *If $s \triangleright d \to t \triangleright e$, then $(\mathcal{S}, t \triangleright e) \in (X^\nu)^\star$.*
    (ii) *If $d = v$, then $(\mathcal{R} \cup \{v\}, s) \in (X^\nu)^\star$.*

B. *For every $(\mathcal{R}, s) \in X$ and $u \in \mathcal{R}$,*

    (1) *If $u = \lambda x.\, e$, then for any $(\mathcal{S}, t) \in \{(\mathcal{R}, s)\}^\nu$ and $v \in \mathcal{S}^\star$, we have $(\mathcal{S}, t \rhd uv) \in X$.*

    (2) *If $u = \langle v_1, \ldots, v_i, \ldots, v_n \rangle$, then $(\mathcal{R} \cup \{v_i\}, s) \in (X^\nu)^\star$.*

    (3) *If $u = \ell^\perp$ and $\ell^\perp \in dom(s)$, then $(\mathcal{R} \cup \{s(\ell^\perp)\}, s) \in (X^\nu)^\star$.*

where $^\nu$ and * denote unary versions of allocation and context closures, respectively.

9. Example of environmental P-predicates up-to

The code in Figure 1 implements directed acyclic graphs (DAGs), with garbage collection by reference counting. For simplicity, we use *immutable* lists in this example (in addition to a mutable data structure for representing the DAGs themselves), and assume their basic operations such as *member*, *append*, and *remove1* (the function to remove the first instance of a given element from a given list).

Here, z is bound to the location of the last added node in the DAG. A node is either **null** or a quintuple $\langle i, b, n, p, \ell \rangle$, where i is an integer ID of the node, b a Boolean value meaning whether the node is "in the root set" (i.e., cannot be garbage collected), n the reference count of the node, p the (immutable) list of the integer IDs of child nodes, and ℓ the pointer to the second last added node. This pointer is different from child pointers, for which we use the list of integer IDs.

Function *addn* takes a pair of integer x and integer list p, and adds a node with ID x and children p. The code $x + 0$ and $map(\lambda y.\, y + 0)p$ ensures they are indeed an integer and an integer list (assuming that $\_ + 0$ is defined only for integers). An auxiliary function $incr_x$ is used to increment the reference counts of nodes in p, as well as to check if node x already exists (in which case it diverges). Note that the same node may appear more than once in p. Its reference count is increased by the number of appearance.

Function *deln* prepares to delete a node by (un)marking it as non-root. Function *gc* invokes the garbage collector *decr*, which takes a node pointer n and an integer list p. It decreases the reference counts of nodes in p, again according to the number of their appearances. If the reference count becomes 0, and if the root flag is not set, then the node is deleted, and its children are added to p so that their reference counts will be decreased recursively. In the end, *decr* returns the updated node pointer n.

We define the shape predicate for DAGs by induction.

- $DAG_S(\texttt{null}, \emptyset, \emptyset)$

- $DAG_S(\ell, [(i, b, S_0)] \,@\, L_0, s_0 \uplus \{\ell \mapsto \langle i, b, S(i), S_0, \ell_0 \rangle\})$
  if $\ell \neq \texttt{null}$, $DAG_{S+S_0}(\ell_0, L_0, s_0)$, and $i \neq i_0$ for any $(i_0, \_, \_) \in L_0$.

Here, the subscript S is a multiset of node IDs, representing the number of references to each node.

We also give a specification of our garbage collector as follows. It is more abstract than the implementation because it looks at only the positiveness of the reference count $S(i)$, not its concrete value (i.e., only whether the node is referred to, not how many times).

$$
\begin{aligned}
GC_S([\,]) &= [\,] \\
GC_S([(i, b, S_0)] @ L_1) &= [(i, b, S_0)] @ GC_{S+S_0}(L_1) \quad \text{if } b = \texttt{true} \text{ or } S(i) > 0 \\
GC_S([(i, \texttt{false}, S_0)] @ L_1) &= GC_S(L_1) \quad\quad\quad\quad\quad\quad\quad\quad \text{if } S(i) = 0
\end{aligned}
$$

GC_S takes a list of triples (i, b, S_0) that represent nodes, where i is the node ID, b the root flag, and S_0 the multiset of the IDs of the children. Here, the subscript S is the multiset of the IDs of nodes pointed to by "external" nodes, i.e., by nodes that are not in the list.

Now, the following lemma can be proved.

Lemma 9.1. *Suppose $DAG_S(\ell, L, s)$. Then, for any t and T, we have $s \uplus t \triangleright decr(\ell)T \twoheadrightarrow s_0 \uplus t \triangleright \ell_0$ with $DAG_{S-T}(\ell_0, GC_{S-T}(L), s_0)$. (Here, we are abusing notation and writing T for an integer list representing the integer multiset T.)*

PROOF. By lexical induction on the lengths of L and T. Trivial if $\ell = \texttt{null}$. Suppose

- $\ell \neq \texttt{null}$,

- $L = [(i, b, S_1)] @ L_1$,

- $s = s_1 \uplus \{\ell \mapsto \langle i, b, S(i), S_1, \ell_1 \rangle\}$,

- $DAG_{S+S_1}(\ell_1, L_1, s_1)$, and

- $i \neq i_1$ for any $(i_1, \_, \_) \in L_1$.

If $T(i) > 0$, then

$$
\begin{aligned}
&s \uplus t \triangleright decr(\ell)T \\
\twoheadrightarrow\quad &s_1 \uplus \{\ell \mapsto \langle i, b, S(i) - 1, S_1, \ell_1 \rangle\} \uplus t \triangleright decr(\ell)(T - \{i\}).
\end{aligned}
$$

Since $DAG_{S-\{i\}}(\ell, L, s_1 \uplus \{\ell \mapsto \langle i, b, S(i) - 1, S_1, \ell_1 \rangle\})$, we have

$$
\begin{aligned}
&s_1 \uplus \{\ell \mapsto \langle i, b, S(i) - 1, S_1, \ell_1 \rangle\} \uplus t \triangleright decr(\ell)(T - \{i\}) \\
\twoheadrightarrow\quad &s_0 \uplus t \triangleright \ell_0
\end{aligned}
$$

with $DAG_{S-T}(\ell_0, GC_{S-T}(L), s_0)$ by induction, which concludes the case. Suppose $T(i) = 0$, and $b = \texttt{true}$ or $S(i) > 0$. Since $DAG_{S+S_1}(\ell_1, L_1, s_1)$, we have

$$
s_1 \uplus t \triangleright decr(\ell_1)T \quad \twoheadrightarrow \quad s_2 \uplus t \triangleright \ell_2
$$

with $DAG_{S+S_1-T}(\ell_2, GC_{S+S_1-T}(L_1), s_2)$ by induction. Thus,

$$
s \uplus t \triangleright decr(\ell)T \quad \twoheadrightarrow \quad s_2 \uplus t \uplus \{\ell \mapsto \langle i, b, S(i), S_1, \ell_2 \rangle\} \triangleright \ell
$$

33

with $DAG_{S-T}(\ell, GC_{S-T}(L), s_2 \uplus \{\ell \mapsto \langle i, b, S(i), S_1, \ell_2 \rangle\})$, concluding the case. Last, suppose $T(i) = 0$, $b = \texttt{false}$, and $S(i) = 0$. Since $DAG_{S+S_1}(\ell_1, L_1, s_1)$, we have

$$s_1 \uplus t \triangleright decr(\ell_1)(S_1 + T) \quad \twoheadrightarrow \quad s_0 \uplus t \triangleright \ell_0$$

with $DAG_{S-T}(\ell_0, GC_{S-T}(L_1), s_0)$ by induction. Hence

$$s \uplus t \triangleright decr(\ell)T \quad \twoheadrightarrow \quad s_0 \uplus t \triangleright \ell_0.$$

$\square$

Given the lemma above, it is straightforward to give an environmental predicate for *dag* and prove it to be safe under arbitrary (public) contexts. In fact, we can prove more properties, e.g., that the number of local locations matches the number of nodes (and therefore the number of *live* nodes after a call to *gc*) plus one (for z). To be specific, take

$$
\begin{aligned}
X \quad = \quad & \{(\emptyset, \emptyset \triangleright dag)\} \\
\cup \quad & \{(\mathcal{R}_m, s \uplus \{m^\top \mapsto \ell^\top\}) \mid DAG_\emptyset(\ell^\top, L, s)\} \\
\cup \quad & \{(\mathcal{R}_m, t \triangleright e) \mid \\
& DAG_\emptyset(\ell^\top, L, s), \ d \in \mathcal{R}_m, \ \mathcal{S} = \mathcal{R}_m \cup \{\tilde{\ell}^\perp\}, \ v, \tilde{w} \in \mathcal{S}^\star, \ \tilde{m}^\perp \in \mathcal{S}, \\
& s \uplus \{m^\top \mapsto \ell^\top\} \uplus \{\tilde{\ell}^\perp \mapsto \tilde{w}\} \setminus \tilde{m}^\perp \triangleright d(v) \twoheadrightarrow t \triangleright e\}
\end{aligned}
$$

where

$$\mathcal{R}_m \quad = \quad \{[m^\top/z]\,addn, [m^\top/z]\,deln, [m^\top/z]\,gc\}.$$

Then, X is an environmental P-predicate up-to, where

$$
\begin{aligned}
P \quad = \quad & \{(\mathcal{R}, t \triangleright e) \mid safe(t \triangleright e)\} \\
\cup \quad & \{(\mathcal{R}, s \uplus \{m^\top \mapsto \ell^\top\} \uplus \{\tilde{\ell}^\perp \mapsto \tilde{w}\}) \mid DAG_\emptyset(\ell^\top, L, s)\}.
\end{aligned}
$$

10. Conclusions

As is often the case in programming language theories, our theory may seem trivial in hindsight. In particular, all the proofs are arguably straightforward (though sometimes just lengthy because of case analyses) once organized in the way presented here. The technical contributions of this work—besides the very *idea* of using "bisimulations" for non-equivalence properties—are the organization and definitions. Specifically, our technicalities included:

- The definition of local memory safety (Definition 5.16)—based on the distinction of public and local locations (Section 1.3.4)—that makes sense "under arbitrary contexts" in the sense that it does not restrict their observational power.

- The definition of consistency (Definition 4.4), separated from the definition of bisimulations. Without consistency, the conditions (Condition 2, for example) of bisimulations would have required a number of extra elements (e.g., like "if u is a tuple, then $(\mathcal{R}, s \triangleright \#_i(u), s' \triangleright \#_i(u')) \in X$ for any i").

34

In addition, not only the bisimulations X, but also the predicates P, were required to be consistent. This requirement simplified the completeness statements (we would otherwise have had to say "the largest consistent subset of P" everywhere in place of P) without sacrificing the applicability of our approach (recall that all the predicates in our examples were trivially consistent, because none of them referred to the environments $\mathcal{R}$ at all).

- The definition of reduction closure (Definition 4.3) for environmental relations (Condition ii, in particular), separated from the definitions of bisimulations (Definition 4.5). Thanks to this separation, soundness (Lemma 4.9) and completeness (Lemma 4.10) of the $P^\star$-bisimilarity were stated simply as its equality to $P^\star_{\star\to}$ (Theorem 4.11), the largest contextual and reduction-closed subset of $P^\star$.

- The definition of allocation closure (Definition 6.1) and the up-to allocation technique (Definition 6.2).

Our future work includes systematically deriving the conditions of environmental bisimulations from the operational semantics of a language (cf. Koutavas and Wand, 2006), so that the definitions and proofs do not have to be manually repeated for every language. Another direction is mechanization. Although complete automation is clearly impossible, ideas from model checking and type-based analyses may be useful for sound approximation. Weakening the contextuality to restrict the possible contexts—so that more programs can be proved correct—would also be useful in practice.

Acknowledgments

References

Ahmed, A., 2004. Semantics of types for mutable state. Ph.D. thesis, Princeton University.

Ahmed, A., Dreyer, D., Rossberg, A., 2009. State-dependent representation independence. In: Proceedings of the 36th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages. pp. 340–353.

Ahmed, A., Fluet, M., Morrisett, G., 2005. A step-indexed model of substructural state. In: Proceedings of the Tenth ACM SIGPLAN International Conference on Functional Programming. pp. 78–91.

Ahmed, A., Fluet, M., Morrisett, G., 2007. L3: A linear language with locations. Fundamenta Informaticae 77 (4), 397–449, extended abstract appeared in *Typed Lambda Calculi and Applications*, *Lecture Notes in Computer Science*, Springer-Verlag, vol. 3461, pp. 293–307, 2005.

Crary, K., Walker, D., Morrisett, G., 1999. Typed memory management in a calculus of capabilities. In: Proceedings of the 26th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages. pp. 262–275.

Gustavsson, J., Sands, D., 1999. A foundation for space-safe transformations of call-by-need programs. In: Higher Order Operational Techniques in Semantics. Vol. 26 of Electronic Notes in Theoretical Computer Science. Elsevier Science, pp. 69–86.

Koutavas, V., Wand, M., 2006. Small bisimulations for reasoning about higher-order imperative programs. In: Proceedings of the 33rd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages. pp. 141–152.

Meyer, A. R., Sieber, K., 1988. Towards fully abstract semantics for local variables: Preliminary report. In: Proceedings of the 15th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages. pp. 191–203.

Milner, R., 1999. Communicating and Mobile Systems: The π-Calculus. Cambridge University Press.

Mitchell, J. C., 1996. Foundations for Programming Languages. MIT Press.

Morris, Jr., J. H., 1968. Lambda-calculus models of programming languages. Ph.D. thesis, Massachusetts Institute of Technology.

Pierce, B. C., 2002. Types and Programming Languages. MIT Press.

Pitts, A. M., Stark, I., 1998. Operational reasoning for functions with local state. In: Higher Order Operational Techniques in Semantics. Cambridge University Press, pp. 227–273.

Sangiorgi, D., Kobayashi, N., Sumii, E., 2007. Environmental bisimulations for higher-order languages. In: Twenty-Second Annual IEEE Symposium on Logic in Computer Science. pp. 293–302.

Sangiorgi, D., Milner, R., 1992. The problem of "weak bisimulation up to". In: CONCUR '92. Vol. 630 of Lecture Notes in Computer Science. Springer-Verlag, pp. 32–46.

Sangiorgi, D., Walker, D., 2001. The Pi Calculus – A Theory of Mobile Processes. Cambridge University Press.

Sumii, E., 2009. A complete characterization of observational equivalence in polymorphic λ-calculus with general references. In: Computer Science Logic. Vol. 5771 of Lecture Notes in Computer Science. Springer-Verlag, pp. 455–469.

Sumii, E., Pierce, B. C., 2007a. A bisimulation for dynamic sealing. Theoretical Computer Science 375 (1–3), 169–192, extended abstract appeared in *Proceedings of the 31st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pp. 161–172, 2004.

Sumii, E., Pierce, B. C., 2007b. A bisimulation for type abstraction and recursion. Journal of the ACM 54 (5-26), 1–43, extended abstract appeared in *Proceedings of the 32nd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pp. 63–74, 2005.

Tofte, M., Talpin, J.-P., 1994. Implementation of the typed call-by-value λ-calculus using a stack of regions. In: Proceedings of the 21st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages. pp. 188–201.

Wadler, P., 1990. Linear types can change the world! In: Programming Concepts and Methods. North Holland.