Who is pointing when to whom: on model-checking pointer structures

Dino Distefano, Arend Rensink, Joost-Pieter Katoen
Department of Computer Science, University of Twente
P.O. Box 217, 7500 AE Enschede, The Netherlands
E-mail: {ddino, rensink, katoen}@cs.utwente.nl

Abstract

This paper introduces a new model to reason about systems composed by entities that can refer to each other via pointers, such as objects in an object-based system. The model, based on History-Dependent Automata, treats particular cases of unboundedness by a special layered mechanism of abstraction. As an application, in this paper the model is used to define the semantics of a simple language dealing with dynamic allocation and deallocation of entities and pointers. Furthermore, the paper presents a temporal logic that allows to express properties for such systems and that is particularly focussed on the way entities refer to each other. Finally, a sound (but not complete) model checking algorithm for the logic is presented.

Contents

1 Introduction

Pointers (references) are a powerful programming mechanism. They are very flexible but at the same time error prone due to the so-called *complexity of pointer swing* [8, 12] resulting from aliasing: apparently unrelated expressions may be altered by the assignment to an entity in memory referred to by more than one pointer. It is difficult to control and to reason about this phenomenon. Run-time safety violations (e.g., dereferencing null or disposed pointers) easily arise and their detection cannot be ensured by type systems. Although the advent of object-oriented languages has limited the use of pointers (at least at the programming level), the possibility for an object *to refer to* other objects is still present in any practical object-oriented model describing interactions among objects. For example, proper concepts of object-orientation such as *object composition* reduces to references between objects. Thus, for reasoning about object-oriented systems it is indispensable to be able to reason about the way objects refer to each other.

In order to capture essential information of systems dealing with references, we introduce a logic, called $\mathcal{N}all\ell$TL, whose primitives address dynamic allocation and deallocation of entities and their dynamic pointers. Typical properties expressible in $\mathcal{N}all\ell$TL are, for example:

- in object-oriented systems:
  - *every object reachable from a particular object will be eventually deallocated.*
  - *objects o_1 and o_2 belong to disjoint lists.*
- in security: *no untrusted agent will have a reference to a secret datum.*

Along the line of the model checking algorithm defined in [6], in this paper, we define another algorithm that checks whether a $\mathcal{N}all\ell$TL formula is *not satisfiable* in a given model. This algorithm is based on the construction of a tableau graph [9], however in this particular case due to a more sophisticated abstraction which allows to describe rather expressive models, the algorithm may produce false counterexamples.

The models we propose in this paper are an enhanced version of High-level Allocational Büchi Automata (HABA) defined in [6] that in turn are based on History Dependent Automata [11]. Besides the ability to describe the dynamic allocation and deallocation of entities typical of HABA (and HD-automata), the extension we discuss here deals with *references* (pointers) between entities that are alive in the same state. From state to state, references can be created, removed and modified: i.e., the model describes dynamic topological structures of alive entities.

In order to achieve finite-state models, we propose an abstraction that is able to deal with some kind of unbounded systems. In HABAs we use a special class of entities that model finite but *unbounded* chains of "concrete" entities. Our unbounded entities are somehow a specialisation (to chains) of what in the literature is known as *summary node* [13]. Unbounded entities provide HABAs with the capacity to describe, by a single state, an infinite number of topological structures that can be represented at different *level of abstraction*. A precise characterisation of the relation between the different levels will result in the notion of *morphism*.

We show that HABAs with references can be used as a mathematical model for the definition of the semantics of a simple language with basic object-based notions such as object creation, and navigation. For this language we define two semantics: a concrete one, that is infinite state and a symbolic one that is finite state. The relation between the two semantics is then studied.

This paper is organised as follows: in Section 2 the syntax and semantics of $\mathcal{N}all\ell$TL is defined; in Section 3 we introduce ABA and HABA with references as well as the notion of morphism and an enhanced version of reallocation. Then, in Section 4 we study how to relate ABA and HABA with references. The programming language is defined in Section 5 and its operational semantics in Section 6. The model checking algorithm is given in Section 7. Finally, we conclude the paper with an overview of other techniques used for the analysis of pointer structures (Section 8). Proofs are reported in the appendixes.

2 A logic for navigation

In this section, we will enhance $\mathcal{A}ll\ell$TL [6], such that properties about *navigation* can be expressed. Navigations are essentially possible when entities reference each other. In this paper, we restrict ourselves to the case where every entity e has precisely one outgoing reference. This reference is defined for entity e, if e has a pointer (denoted by $.a$), otherwise it is undefined. Let LVar be a (countable) set of logical variables.

Definition 2.1. Let $x \in$ LVar. The syntax of $\mathcal{N}all\ell$TL is defined by the following grammar:

$$\begin{array}{rcl} (\alpha \in) Nav & ::= & nil \mid x \mid \alpha.a \\ \phi & ::= & \alpha = \alpha \mid \alpha\ \mathsf{new} \mid \alpha \rightsquigarrow \alpha \mid \exists x{:}\phi \mid \neg\phi \mid \phi \vee \phi \mid \mathsf{X}\phi \mid \phi\,\mathsf{U}\,\phi. \end{array}$$

We refer to elements of the set *Nav* as *navigation expressions*. For instance, the expression $x.a$ denotes the entity referred to by the entity denoted by x (if any). Similarly, $x.a.a$ denotes the entity referred to by $x.a$. The suffix of a navigation expression can be arbitrarily long, therefore we can specify lists of unbounded length. We write $x.a^n$ ($n \in \mathbb{N}$) as a shorthand for x followed by n successive pointers, that is formally:

$$\begin{array}{rcl} x.a^0 & \equiv & x \\ x.a^{n+1} & \equiv & (x.a^n).a \end{array}$$

Formula $\alpha\,\mathsf{new}$ holds if the entity denoted by α is fresh in the current state, i.e., the entity denoted by α did not exist in the previous state of the computation. Formula $\alpha_1 = \alpha_2$ holds if expressions α_1 and α_2 are aliases (i.e., they denote the same entity) in the current state. The predicate $\alpha_1 \rightsquigarrow \alpha_2$ (read α_1 *reaches* α_2 or α_1 *leads-to* α_2) means that from the entity denoted by α_1 there exists a reference path reaching the entity denoted by α_2 (i.e., $\alpha_1.a^k = \alpha_2$ for some $k \geqslant 0$). As known in the literature, reachability is not a first-order graph property [4] therefore the operator $\rightsquigarrow$ provide $\mathcal{N}\!\mathit{all}\ell$TL with some second order capabilities.

Besides standard LTL abbreviations like G, F throughout this paper we write $\alpha\,\mathsf{dead}$ for $\alpha = \mathit{nil}$, $\alpha\,\mathsf{alive}$ for $\neg(\alpha\,\mathsf{dead})$, and $\alpha_1 \not\rightsquigarrow \alpha_2$ for $\neg(\alpha_1 \rightsquigarrow \alpha_2)$.

2.1 Semantics

Let *Ent* be a countable universe of entities ranged over by $e, e', e_1, \ldots$. $\mathcal{N}\!\mathit{all}\ell$TL formulae are interpreted over infinite sequences of sets of entities and mappings defining information about mutual references of entities.

Let $\perp$ be a special value such that $\perp \notin \mathit{Ent}$. $\perp$ is used to represent *undefined*. For a set E we write $E^\perp = E \cup \{\perp\}$.

Definition 2.2. An *allocation sequence* σ is an infinite sequence of pairs $(E_0, \mu_0)(E_1, \mu_1)(E_2, \mu_2)\cdots$ where for $i \in \mathbb{N}$:

- $E_i \subseteq \mathit{Ent}$
- $\mu_i : E_i^\perp \to E_i^\perp$ such that $\mu_i(\perp) = \perp$.

Function μ_i gives the reference of the entity pointed to by its argument. Each μ_i is strict.

Let $\theta : \textsc{LVar} \to \mathit{Ent}^\perp$ be a valuation of the logical variables. The semantics of the navigation expression α is given by $[\![\cdot]\!] : \mathit{Nav} \times (\mathit{Ent}^\perp \rightharpoonup \mathit{Ent}^\perp) \times (\textsc{LVar} \to \mathit{Ent}^\perp) \to \mathit{Ent}$ defined as:

$$\begin{array}{rcl} [\![\mathit{nil}]\!]_{\mu,\theta} & = & \perp \\ [\![x]\!]_{\mu,\theta} & = & \theta(x) \\ [\![\alpha.a]\!]_{\mu,\theta} & = & \mu([\![\alpha]\!]_{\mu,\theta}). \end{array}$$

Let $\sigma^i = (E_i, \mu_i)(E_{i+1}, \mu_{i+1})\cdots$. For a given allocation sequence σ, let E_i^σ and μ_i^σ denote the first and the second component in the i-th state of σ, respectively. The semantics of $\mathcal{N}\!\mathit{all}\ell$TL is defined in terms of the satisfaction relation $\sigma, N, \theta \models \phi$, where σ is an allocation sequence, $N \subseteq E_0^\sigma$ is the set of entities initially new and θ is a valuation of the free logical variables of the formula ϕ.

Let $N_i^\sigma \subseteq \mathit{Ent}$ (i.e., $\perp \notin N_i$) denote the set of new entities in state i, defined as $N_0^\sigma = N$ and $N_{i+1}^\sigma = E_{i+1}^\sigma \backslash E_i^\sigma$. Let $\theta_i^\sigma : \textsc{LVar} \to \mathit{Ent}^\perp$ be a valuation of the logical free variables in state i (of σ) where:

$$\theta_i^\sigma(x) = \begin{cases} \theta(x) & \text{if } \forall k \leqslant i : \theta(x) \in E_k^\sigma \\ \perp & \text{otherwise.} \end{cases}$$

Note that once a logical variable is mapped to an entity, then this association remains along σ unless the entity dies, i.e., is deallocated.

The satisfaction relation $\models$ for $\mathcal{N}all\ell$TL is defined as follows:

$$
\begin{array}{lll}
\sigma, N, \theta \models \alpha_1 = \alpha_2 & \text{iff} & [\![\alpha_1]\!]_{\mu_0^\sigma,\theta} = [\![\alpha_2]\!]_{\mu_0^\sigma,\theta} \\
\sigma, N, \theta \models \alpha \ \mathsf{new} & \text{iff} & [\![\alpha]\!]_{\mu_0^\sigma,\theta} \in N \\
\sigma, N, \theta \models \alpha_1 \rightsquigarrow \alpha_2 & \text{iff} & \exists k \geqslant 0 : (\mu_0^\sigma)^k([\![\alpha_1]\!]_{\mu_0^\sigma,\theta}) = [\![\alpha_2]\!]_{\mu_0^\sigma,\theta} \\
\sigma, N, \theta \models \exists x{:}\phi & \text{iff} & \exists e \in E_0^\sigma : \sigma, N, \theta\{e/x\} \models \phi \\
\sigma, N, \theta \models \neg\phi & \text{iff} & \sigma, N, \theta \not\models \phi \\
\sigma, N, \theta \models \phi \vee \psi & \text{iff} & \text{either } \sigma, N, \theta \models \phi \text{ or } \sigma, N, \theta \models \psi \\
\sigma, N, \theta \models \mathsf{X}\phi & \text{iff} & \sigma^1, N_1^\sigma, \theta_1^\sigma \models \phi \\
\sigma, N, \theta \models \phi \,\mathsf{U}\, \psi & \text{iff} & \exists i : (\sigma^i, N_i^\sigma, \theta_i^\sigma \models \psi \text{ and } \forall j < i : \sigma^j, N_j^\sigma, \theta_j^\sigma \models \phi).
\end{array}
$$

Note that the proposition $\alpha_1 \rightsquigarrow \alpha_2$. is satisfied in case $[\![\alpha_2]\!]_{\mu_0^\sigma,\theta} = \bot$ and $[\![\alpha_1]\!]_{\mu_0^\sigma,\theta}$ can reach an entity with an undefined pointer.

Example 2.3. In Table 1 are reported some example properties expressible in $\mathcal{N}all\ell$TL. □

The expressions $x.a.a$ and $y.a$ eventually will become aliases	$\mathsf{F}(x.a.a = y.a)$
If $x_1.a^2$ and $x_2.a$ are aliases, the entity associated to y is deallocated before $x_1.a^2$ and $x_2.a$ are not aliases anymore	$\mathsf{G}(x_1.a^2 = x_2.a \wedge y\ \mathsf{alive} \Rightarrow (x_1.a^2 = x_2.a \,\mathsf{U}\, y\ \mathsf{dead}))$
Eventually, v will point to an entity in a non-empty cycle	$\mathsf{F}(\exists x : x \neq v \wedge x \rightsquigarrow v \wedge v \rightsquigarrow x)$
Variables v and w always point to disjoint parts of the heap (*non-interference*)	$\mathsf{G}(\forall x : v \rightsquigarrow x \Rightarrow w \not\rightsquigarrow x)$
Every entity reachable from v will be eventually deallocated	$\forall x : (v \rightsquigarrow x \Rightarrow \mathsf{F}x\ \mathsf{dead})$
All and only the entities currently reachable from v will be eventually deallocated	$(\forall x : v \rightsquigarrow x \Rightarrow \mathsf{F}x\ \mathsf{dead}) \wedge$ $(\forall x : v \not\rightsquigarrow x \Rightarrow \mathsf{G}x\ \mathsf{alive})$
v's list will be (and remains to be) eventually reversed	$\forall x : \forall y : (v \rightsquigarrow x \wedge x.a = y) \Rightarrow \mathsf{FG}(y.a = x)$
A tautology	$x.a\ \mathsf{alive} \Rightarrow x\ \mathsf{alive}$

Table 1: Some example properties expressible in $\mathcal{N}all\ell$TL.

3 ABA and HABA with references

Throughout this paper, let $M \in \mathbb{N}$ be a global constant, $\mathbb{M} = \{1, \ldots, M\}$ and $\mathbb{M}^* = \mathbb{M} \cup \{*\}$ where $*$ is a special distinguished symbol.

We now introduce the concept of cardinality which is the base of the abstraction mechanism we will develop (and use) throughout this paper.

Definition 3.1. Let $E \subseteq Ent$. A function $\mathcal{C}_E : E \to \mathbb{M}^*$ is called *cardinality* function (on E).

A cardinality function $\mathcal{C}_E$ associates to every entity in E a number ($\leqslant M$) or the special symbol $*$. $\mathcal{C}_E(e) = k \leqslant M$ means that the cardinality of e is precisely k. $\mathcal{C}_E(e) = *$ means that the cardinality of e is some natural number larger than M. If the set E is clear from the context we simply write $\mathcal{C}$ for $\mathcal{C}_E$.

Definition 3.2. The *unitary* cardinality function on E is the cardinality function $\mathbf{1}_E : E \to \mathbb{M}^*$ such that $\mathbf{1}_E(e) = 1$ for all $e \in E$.

We need to define the sum on cardinalities:

$$
n \oplus m = \begin{cases} n + m & \text{if } n, m \in \{0, \ldots, M\} \text{ and } n + m \leqslant M \\ * & \text{otherwise.} \end{cases}
$$

We lift the notion of cardinality to sets of entities as follows: if $E \subseteq Ent$ then

$$\mathcal{C}(E) = \bigoplus_{e \in E} \mathcal{C}(e).$$

The notation $\bigoplus$ stands for a sum that uses the $\oplus$ operator defined above instead of the standard sum on $\mathbb{N}$. Moreover, for $n \in \mathbb{N}$ let

$$\lceil n \rceil_M = \begin{cases} n & \text{if } n \leqslant M \\ * & \text{otherwise.} \end{cases}$$

The global constant M is mostly fixed and therefore if no confusion arise we will not indicate it explicitly and we write $\lceil n \rceil$ instead of $\lceil n \rceil_M$. The cardinality function imposes a distinction among entities. For the sake of exposition, in this paper, we identify three different classes:

- *concrete entities*: those with cardinality one;
- *multiple entities*: those with cardinality neither 1 nor $*$;
- *unbounded entities*: those with cardinality $*$.

For a set of entities E, we write E^* for the subset of its unbounded entities, i.e., $E^* = \{e \in E \mid \mathcal{C}_E(e) = *\}$. *Bounded* entities are either concrete or multiple.

3.1 Morphisms

A *configuration* $\gamma = (E, \mu, \mathcal{C}_E)$ represents a weighted directed graph where $E \subseteq Ent$ is the set of nodes, μ defines the set of arcs, and $\mathcal{C}_E$ represents the weights associated to the nodes. Figure 1 depicts three configurations as weighted graphs: γ_1, γ_2 and γ_3. Circles represent concrete entities. Filled circles represents multiple/unbounded entities. Numbers (or stars) associated with entities denote cardinalities.

Multiple and unbounded entities provide us with the possibility to *abstract* from specific portions of the graph (configuration) thus obtaining a more compact representation of the original graph. In this chapter, we apply the following abstraction:

> multiple and unbounded entities represent *chains* of more concrete (i.e., with lower cardinality) entities.

The length of the chain represented by a multiple entity e must be consistent with the cardinality of e. Unbounded entities represent chains of arbitrary length strictly larger than M. For example, in configuration γ_1 depicted in Figure 1, entities e_3, e_5 and e_7 are not concrete. In γ_2, e_3 has been replaced by the chain of entities e'_3 and e''_3. Since $\mathcal{C}_{\gamma_1}(e) = 2$, this is the only allowed concretization. We say that γ_2 is a more concrete configuration than γ_1, or symmetrically, γ_1 is more abstract than γ_2. Moreover, the other difference between γ_1 and γ_2 is that e_7 in γ_1 is replaced by the chain composed by e'_7 and e''_7 in γ_2. If we think of e_7 as a chain of arbitrary length (larger than M), this process corresponds to splitting this chain into a chain of arbitrary length (i.e., e'_7) followed by a concrete entity (i.e., e''_7). This phenomenon is sometimes known in the literature as *materialisation* [13]. Note that we can split e_7 in infinitely many ways. Any of such splitting yields a graph with a different level of abstraction than the original one. The level of abstraction depends on the number of abstract/concrete entities used. In the figure, the three different abstraction levels are represented by planes of configurations. Thus, γ_2 is more concrete than γ_1. Configuration γ_3, where entity e_5 of γ_2 has been replaced by the chain consisting of e'_5, e''_5, e_5, and e'''_5, is more concrete than γ_2 (and therefore than γ_1).

There exists an intimate relation among γ_1, γ_2 and γ_3. Any of these configurations can be obtained by the other by replacing some unbounded/multiple entities by a chain of more concrete ones or vice-versa by replacing chains of entities by more abstract ones. These considerations emphasise that these configurations represent the *same pointer structure* but at *different levels of abstraction*.

In this section, we try to give a general characterisation for such a relation between configurations (i.e., same pointer structure but different abstraction level) that will result in a notion of graph abstraction that we call, adopting categorical jargon, *morphism*.

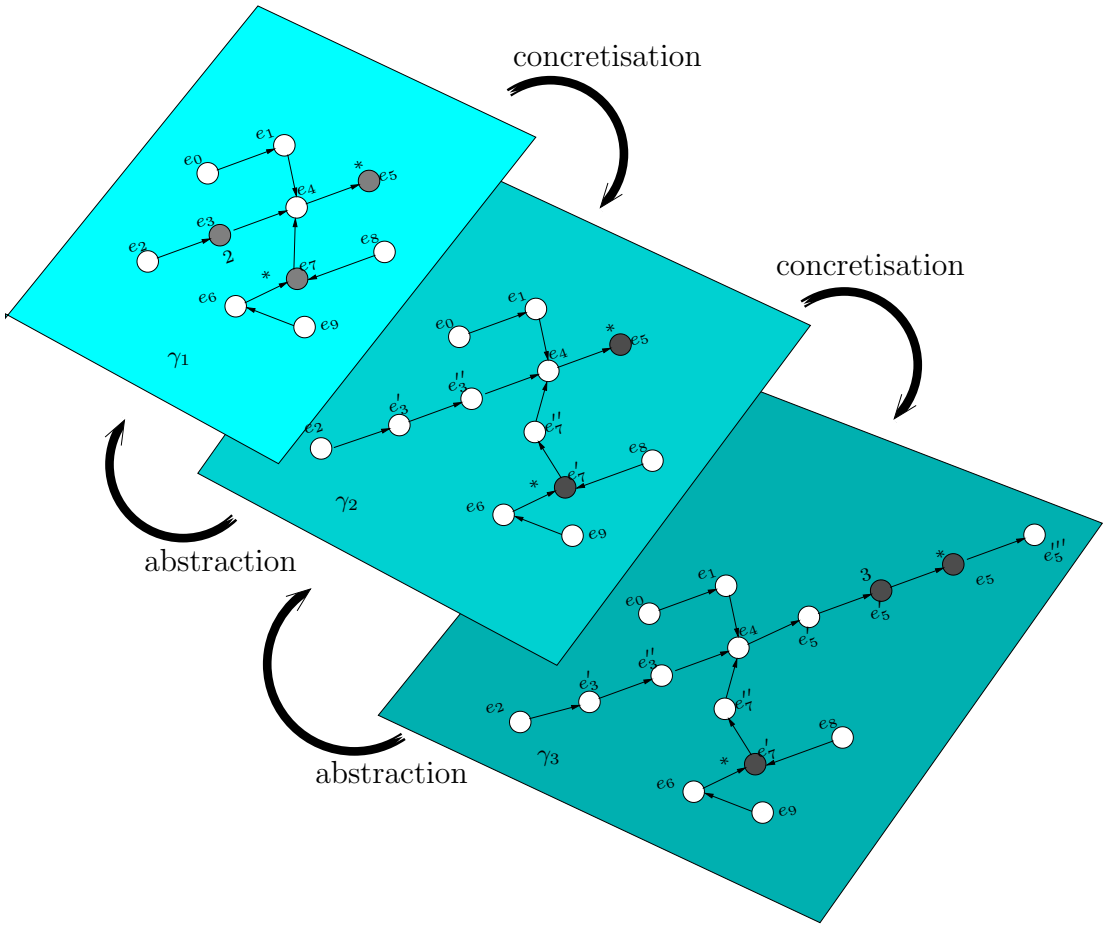

Figure 1: A pointer structure and some of its different levels of abstraction.

Preliminary notation. Throughout this paper, let

$$\textsc{Conf} = 2^{Ent} \times (Ent \rightharpoonup Ent) \times (Ent \rightharpoonup \mathbb{M}^*) \tag{1}$$

be the set of all configurations ranged over by γ. In the context of a configuration $(E, \mu, \mathcal{C})$[1], it is sometimes convenient to consider instead of μ, the relation $\prec \subseteq E \times E$ induced by μ. This is defined as

$$\prec = \{(e, \mu(e)) \mid e, \mu(e) \in Ent\}.$$

Note that $(\bot, \bot), (e, \bot), (\bot, e) \notin \prec$ for any $e \in Ent$. We write $e \prec e'$ as shorthand for $(e, e') \in \prec$. Furthermore we will freely interchange $(E, \mu, \mathcal{C})$ and $(E, \prec, \mathcal{C})$.

For $e \in E$, let $\mathit{indegree}(e) = |\{e' \mid (e', e) \in \prec\}|$. A sequence of E's elements $e_1, \ldots, e_j$ is a *chain* (of length j) if $e_i \prec e_{i+1}$ for $1 \leqslant i < j$. A set of nodes $E' \subseteq E$ with $|E'| = j \geqslant 1$ defines a chain of length j if there exists a bijection $f : \{1, \ldots, j\} \to E'$ such that $f(1), \ldots, f(j)$ is a chain. In this case, we define $\mathit{first}(E') = f(1)$ and $\mathit{last}(E') = f(j)$. E' defines a *pure* chain if $\forall e' \in \{f(2), \ldots, f(j)\} : \mathit{indegree}(e') = 1$ and f is unique. It follows from this definition that chains consisting of only one element are pure. Moreover, let E' be a chain such that $E' \subseteq E^{\bot}$. Then, by the definition of $\prec$, if $\bot \in E'$ then $E' = \{\bot\}$. In the following, let $\preceq$ be the reflexive closure of $\prec$, i.e, $\preceq = \prec \cup \{(e, e) \mid e \in E\}$.

The next definition defines our notion of graph transformation.

Definition 3.3. (morphism) Let $\gamma_1 = (E_1, \prec_1, \mathcal{C}_1)$, $\gamma_2 = (E_2, \prec_2, \mathcal{C}_2)$ be two configurations. A *morphism* h from γ_1 to γ_2 is a surjective function $h : E_1 \to E_2$ such that:

- 1m. $\forall e \in E_2 : h^{-1}(e)$ is a pure chain;
- 2m. $\forall e, e' \in E_2 : e \prec_2 e' \;\Rightarrow\; \mathit{last}(h^{-1}(e)) \prec_1 \mathit{first}(h^{-1}(e'))$;
- 3m. $\forall e, e' \in E_1 : e \prec_1 e' \Rightarrow h(e) \preceq_2 h(e')$;
- 4m. $\forall e \in E_2 : \mathcal{C}_2(e) = \mathcal{C}_1(h^{-1}(e))$;

We write $h : \gamma_1 \succ\!\!\longrightarrow \gamma_2$ or $\gamma_1 \overset{h}{\succ\!\!\longrightarrow} \gamma_2$ if h is a morphism from γ_1 to γ_2.

Condition 1m allows to abstract only pure chains of entities by a single entity. A single element is a chain of length 1. We require a chain to be pure in order to maintain the branching topology of γ_1. Conditions 2m and 3m preserve the dependency of the entities when going from γ_1 to γ_2. Condition 4m requires that the sum of the cardinalities of entities mapped onto the same element e must be equal to the cardinality of e.

The existence of a morphism between two configurations ensures the correspondence of the abstract shape of the graphs represented by the configurations. The correspondence is up to a certain degree of abstractness given by the morphism itself.

Example 3.4. For the configurations in Figure 1, there exists a morphism $h_{21} : \gamma_2 \succ\!\!\longrightarrow \gamma_1$ defined as: $h_{21} \restriction \{e_0, e_1, e_2, e_4, e_5, e_6, e_8, e_9\} = id$ and

$$\begin{array}{ll} h_{21}(e'_3) = e_3 & h_{21}(e''_3) = e_3 \\ h_{21}(e'_7) = e_7 & h_{21}(e''_7) = e_7. \end{array}$$

Moreover, there exists a morphism $h_{32} : \gamma_3 \succ\!\!\longrightarrow \gamma_2$ defined as:

$$h_{32} \restriction \{e_0, e_1, e_2, e'_3, e''_3, e_4, e_6, e'_7, e''_7, e_8, e_9\} = id$$

and

$$h_{32}(e_5) = e_5 \quad h_{32}(e'_5) = e_5 \quad h_{32}(e''_5) = e_5 \quad h_{32}(e'''_5) = e_5.$$

□

[1] From now on, for a triple $(E, \mu, \mathcal{C})$ it is clear that the domain of $\mathcal{C}$ is E.

Proposition 3.5. For a configuration γ there exists a morphism, called *identity* morphism, $id_\gamma : \gamma \rightarrowtail \gamma$ defined by $id_\gamma(e) = e$ for all $e \in E_\gamma$.

Proof. It is straightforward to verify that id_γ satisfies all the conditions of Def. 3.3. □

Definition 3.6. Given two morphisms $h : \gamma_1 \rightarrowtail \gamma_2$ and $h' : \gamma_2 \rightarrowtail \gamma_3$, the *composition* $h' \circ h : \gamma_1 \rightarrowtail \gamma_3$ is defined by $(h' \circ h)(e) = h'(h(e))$ for all $e \in E_{\gamma_1}$.

The composition of two morphisms is also a morphism as stated by the following:

Proposition 3.7. If $h : \gamma \rightarrowtail \gamma'$ and $h' : \gamma' \rightarrowtail \gamma''$ then $h' \circ h : \gamma \rightarrowtail \gamma''$.

Proof. It can be verified that $h' \circ h$ satisfies all the conditions of Def. 3.3. □

Propositions 3.5 and 3.7 imply that the set of configurations equipped with morphisms forms a category.

Example 3.8. We can define the morphism $h_{31} : \gamma_3 \rightarrowtail \gamma_1$ using the morphisms in Example 3.4 as $h_{31} = h_{21} \circ h_{32}$. □

Definition 3.9. Morphism $h : \gamma_1 \rightarrowtail \gamma_2$ is an *isomorphism* if and only if there exists a morphism $h' : \gamma_2 \rightarrowtail \gamma_1$ such that $h' \circ h = id_{\gamma_1}$. In this case, h' is the inverse of h and h is the inverse of h'.

If there exists an isomorphism between configurations γ_1 and γ_2 we say that they are isomorphic and write $\gamma_1 \cong \gamma_2$.

3.2 Allocational Büchi Automata

Allocational Büchi Automata are basically generalised Büchi automata where to each state a set of entities and a relation between entities are associated. These entities, in turn, serve as valuations of logical variables and the references as valuations for the navigation expressions.

Definition 3.10. An *Allocational Büchi Automaton* (ABA) $\mathcal{A}$ is a tuple $\langle X, Q, E, \rightarrow, I, \mathcal{F} \rangle$, with

- $X \subseteq \text{LVAR}$ a finite set of logical variables;
- Q a (possibly infinite) set of states;
- $E : Q \rightarrow \text{CONF}$ a function yielding for each state q a configuration $\gamma_q = (E_q, \mu_q, \mathbf{1}_{E_q})$.
- $\rightarrow \subseteq Q \times Q$ a transition relation;
- $I : Q \rightharpoonup 2^{Ent} \times (X \rightharpoonup Ent)$ a partial function yielding for every *initial state* $q \in \text{dom}(I)$ an *initial valuation* (N, θ), where $N \subseteq E_q$ is a finite set of entities, and $\theta : X \rightharpoonup E_q$ is a partial valuation of the variables in X;
- $\mathcal{F} \subseteq 2^Q$ a set of sets of accept states.

Notational conventions: we write $(q, N, \theta) \in I$ for $I(q) = (N, \theta)$ and $q \rightarrow q'$ for $(q, q') \in \rightarrow$ and γ_q for $E(q) = \gamma$. We adopt the generalised Büchi acceptance condition, i.e, $\rho = q_0 q_1 q_2 \cdots$ is a *run* of ABA $\mathcal{A}$ if $q_i \rightarrow q_{i+1}$ for all $i \in \mathbb{N}$ and $|\{i | q_i \in F\}| = \omega$ for all $F \in \mathcal{F}$. Let $\mathsf{runs}(\mathcal{A})$ denote the set of runs of $\mathcal{A}$. Run $\rho = q_0 q_1 q_2 \cdots$ is said to *accept* the (unfolded) allocation sequence $\sigma = (E_{q_0}, \mu_{q_0})(E_{q_1}, \mu_{q_1})(E_{q_2}, \mu_{q_2}) \cdots$. Let

$$\mathcal{L}(\mathcal{A}) = \{(\sigma, N, \theta) | \exists \rho = q_0 q_1 q_2 \cdots \in \mathsf{runs}(\mathcal{A}) : \rho \text{ accepts } \sigma \text{ and } I(q_0) = (N, \theta)\}.$$

Note that every configuration in an ABA has the unitary cardinality function. Thus, every entity in an ABA is concrete.

Notation on morphisms. The introduction of Q makes it convenient from now on — both for ABA as well as for HABA (cf, Definition 3.21) — to talk about morphisms on states. In that case, we intend to refer to morphisms actually defined on the configuration associated with those states. Therefore, in the rest of this paper we will freely interchange between states and configurations writing then $h : q \rightarrowtail\!\!\!\!\!\rightarrow q'$ for $h : \gamma_q \rightarrowtail\!\!\!\!\!\rightarrow \gamma_{q'}$ and $q \overset{h}{\rightarrowtail\!\!\!\!\!\rightarrow} q'$ for $\gamma_q \overset{h}{\rightarrowtail\!\!\!\!\!\rightarrow} \gamma_{q'}$.

3.3 Reallocations and HABAs

The HABAs that we will define in this section are intended to be used as semantical models for programming languages. The execution of statements is reflected in these automata by some modification on the structure of the graph defined by a (configuration of a) state. Therefore, from state to state, the abstract shape of the graph is only preserved for those parts not involved in the execution of the statement. In particular, modifications in the structure arise because:

i. entities may be allocated or deallocated (e.g., because of some creation/deletion mechanism of the language such as `new` in Java and `delete` in C++);

ii. from state to state some references (pointers) between entities may change (e.g., because of assignments);

iii. furthermore, multiple or unbounded entities may be disassembled into several entities, giving a more concrete structure.

The first two cases correspond to real changes in the structure of the configuration, whereas the third case corresponds to a change of representation of the configuration in terms of the level of abstraction.

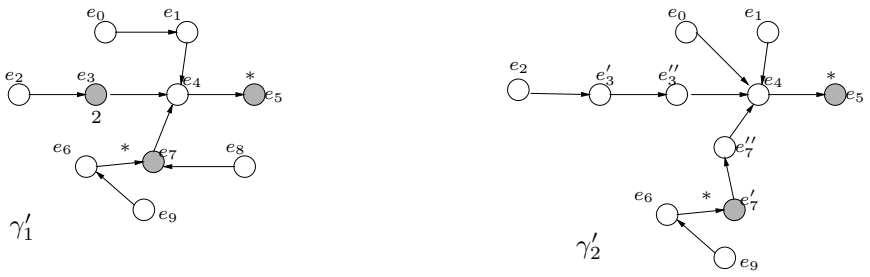

Figure 2: Two (configurations of) states not related by a morphism.

Example 3.11. Consider configurations γ'_1 and γ'_2 in Figure 2. They resemble configurations γ_1 and γ_2 of Figure 1: in particular, γ'_1 is the same as configuration γ_1, whereas γ'_2 is a slight modification of γ_2. Configuration γ'_2 is obtained from γ'_1 by the following alterations:

- replacing the reference between e_0 and e_1 by the reference between e_0 and e_4. This may reflect the execution of an assignment.
- Entity e_8 does not exist in γ'_2. This may reflect execution of a del statement.
- Entities e'_3 and e''_3 are materialised from e_3. This usually is not a direct consequence of the execution of a statement. However, it can be considered as a kind of rearrangement of the shape of the configuration that can be useful in particular situations (cf. the assignment rule of the symbolic semantics in Section 6.4).

Note that there does not exist a morphism between γ'_1 and γ'_2 because the two graphs do not represent the same pointer structure. □

These considerations show that, between states related by a transition in the automaton, we need a *weaker* notion of correspondence than the one defined by a morphism. Namely, the correspondence must be only partial, in the sense that we might have only correspondence of a subgraph but not of the complete graph. Moreover, in order to faithfully model modification *iii*), for every entity in a state we need to associate possibly more than one entity of the other state. This weaker notion of correspondence is captured by the definition of reallocation that is introduced below.

Preliminary notation. Given two sets A_1, A_2 and a relation $\mathcal{R} \subseteq A_1 \times A_2$, if $a \in A_1$ we indicate by $\mathcal{R}(a) = \{a' \in A_2 \mid (a, a') \in \mathcal{R}\}$ and if $a \in A_2$ we write $\mathcal{R}^{-1}(a) = \{a' \in A_1 \mid (a', a) \in \mathcal{R}\}$. For a subset $A \subseteq A_1$, we write $\mathcal{R}(A) = \bigcup_{a \in A} \mathcal{R}(a)$.

A *multiset* $\mathcal{M}$ of a given set A is a function $\mathcal{M} : A \to \mathbb{N}$. For $a \in A$, the image $\mathcal{M}(a)$ is called the *multiplicity* of a in $\mathcal{M}$. We write $a \in \mathcal{M}$ if $\mathcal{M}(a) \neq 0$ (and $a \notin \mathcal{M}$ if $\mathcal{M}(a) = 0$). The sum of two multisets $\mathcal{M}_1$ and $\mathcal{M}_2$ of A, denoted by $\mathcal{M}_1 + \mathcal{M}_2$, is defined as $(\mathcal{M}_1 + \mathcal{M}_2)(a) = \mathcal{M}_1(a) + \mathcal{M}_2(a)$ for all $a \in A$.

Moreover, given a configuration γ we extend to sets the relation $\prec_\gamma$ as follows: let $E, E' \subseteq E_\gamma^\perp$ then

$$E \prec_\gamma E' \quad \Leftrightarrow \quad (E = E' = \{\perp\}) \vee (\forall e \in E : \exists e' \in E' \backslash \{\perp\} : e \prec_\gamma e') \tag{2}$$

We now introduce the concept of reallocation that will be essential throughout this paper.

Definition 3.12. (reallocation) Let $(E_1, \prec_1, \mathcal{C}_1)$, $(E_2, \prec_2, \mathcal{C}_2)$ be two configurations. A *reallocation* λ is a function $\lambda : E_1^\perp \times E_2^\perp \to \mathbb{M}^*$ such that:

1. for all $e \in E_1 : \mathcal{C}_1(e) = \sum_{e' \in E_2^\perp} \lambda(e, e')$
2. for all $e \in E_2 : \mathcal{C}_2(e) = \sum_{e' \in E_1^\perp} \lambda(e', e)$
3. for all $e \in E_1 : |\{e' \in E_2 \mid \lambda(e, e') = *\}| \leqslant 1$ and $|\{e' \in E_2 \mid \lambda(\perp, e') = *\}| = 0$
4. for all $e \in E_1 : \{e' \in E_2^\perp \mid \lambda(e, e') \neq 0\}$ is a chain;
5. for all $e \in E_2 : \{e' \in E_1^\perp \mid \lambda(e', e) \neq 0\}$ is a chain.

We write $\lambda : (E_1, \prec_1, \mathcal{C}_1) \Rightarrow (E_2, \prec_2, \mathcal{C}_2)$ or $(E_1, \prec_1, \mathcal{C}_1) \stackrel{\lambda}{\Rightarrow} (E_2, \prec_2, \mathcal{C}_2)$ if there exists a reallocation λ from $(E_1, \prec_1, \mathcal{C}_1)$ to $(E_2, \prec_2, \mathcal{C}_2)$.

A reallocation is a multiset[2], with range $\mathbb{M}^*$. Accordingly, we write $(e, e') \in \lambda$ if $\lambda(e, e') \neq 0$ and $(e, e') \notin \lambda$ if $\lambda(e, e') = 0$. Furthermore, it is often convenient to treat λ as a relation over $Ent \times Ent \times \mathbb{M}^*$. In this case, we write $(e, e', n) \in \lambda$ if and only if $\lambda(e, e') = n$. Another notation we use (when convenient) is: $\lambda(e) = \{e' \mid (e, e') \in \lambda\}$ and $\lambda^{-1}(e) = \{e' \mid (e', e) \in \lambda\}$. Finally we write $\mathrm{dom}(\lambda) = \{e \neq \perp \mid \exists e' \neq \perp : (e, e') \in \lambda\}$ and symmetrically $\mathrm{cod}(\lambda) = \{e \neq \perp \mid \exists e' \neq \perp : (e', e) \in \lambda\}$.

The element $\perp$ is used by the reallocations in order to model birth and death. More precisely, the birth of an entity $e \in E_2$ is modelled by relating it to $\perp$, i.e. $\lambda(\perp, e) \neq 0$. Symmetrically, λ models the death of an $e \in E_1$ by relating it to $\perp$, i.e., $\lambda(e, \perp) \neq 0$.

A reallocation allows to change dependencies between entities, in fact if $e_1 \prec_1 e_2$ it is not required that entities in $\lambda(e_1)$ precede entities in $\lambda(e_2)$ according to $\prec_2$ (as is enforced by the definition of morphism, cf. Definition 3.3). The above definition requires the cardinality of entities in the source state to be consistent with the multiplicity of the outgoing arcs (condition 1). Symmetrically, condition 2 forces consistency between the multiplicity of outgoing arcs and the cardinality of entities in the target state. Condition 3 is meant to restrict the amount of nondeterminism derived by unbounded entities and to avoid the creation of unbounded entities. The first part of this condition can be rephrased by saying that the unknown cardinality of an unbounded entity e is reallocated only in another unbounded entity e' whereas for every other entity in $\lambda(e) \backslash \{e'\}$ the precise weight is known. By conditions 4 and 5, the image as well as the inverse image of entities under λ are chains.

[2]Strictly speaking it is slightly different since it may give multiplicity $*$ to a pair (e, e').

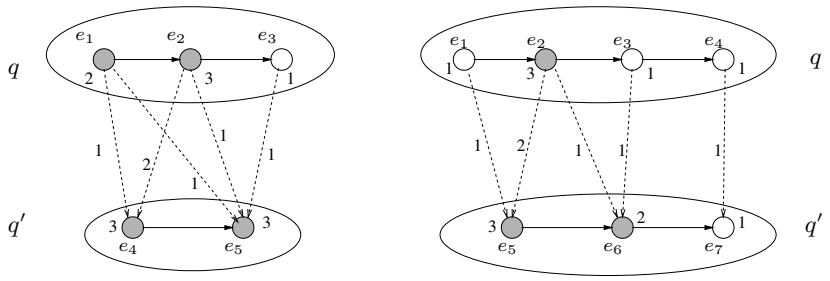

Figure 3: Reallocation examples.

Example 3.13. Figure 3 shows two possible reallocations between (configurations of) states. Note that Definition 3.12 does not exclude the crossing of relations between entities from one state to the other. For example, in the reallocation depicted on the left of the figure part of e_2 is reallocated to e_4 while part of e_1 (that precedes e_2) is reallocated to e_5 (that is preceded by e_4). It is also not difficult to see that the (configuration of) states in Figure 2 are related by a reallocation. □

Lemma 3.14 (impartiality). Let γ, γ' be two configurations. If $\gamma \stackrel{\lambda}{\Longrightarrow} \gamma'$ then:

- $\forall e \in E_{\gamma'} : (\lambda(\bot, e) \neq 0 \;\Leftrightarrow\; \forall e' \in E_\gamma : \lambda(e', e) = 0)$ (Common Birth)
- $\forall e \in E_\gamma : (\lambda(e, \bot) \neq 0 \;\Leftrightarrow\; (\forall e' \in E_{\gamma'} : \lambda(e, e') = 0)$. (Common Death)

Proof. By contradiction assume (Common Birth) does not hold, i.e., there exists $e \in E_{\gamma'}$ and $e \in E_\gamma$ such that $\bot, e' \in \lambda^{-1}(e)$. By condition 5 of the definition of reallocation, $\lambda^{-1}(e)$ is a chain, but this is impossible since $\bot$ cannot be related with entities to form a chain. Symmetrically, λ satisfies (Common Death) since for all $e \in E_\gamma$ by condition 4, $\lambda(e)$ is a chain and therefore cannot contain at the same $\bot$ and some entities of $E_{\gamma'}$. □

The previous lemma describe the impartiality property that a reallocation enjoys w.r.t. birth and death of entities (Common Birth) says that if an entity is born, it cannot be related with any entity except than $\bot$. Symmetrically, (Common Death) states that if an entity dies than it can be related only with $\bot$. The combination of condition 1 of the reallocation definition and (Common Death) (and symmetrically condition 2 and (Common Birth)) forces to transfer the complete cardinality of e by $\lambda(e, \bot)$ (and symmetrically $\lambda(\bot, e)$).

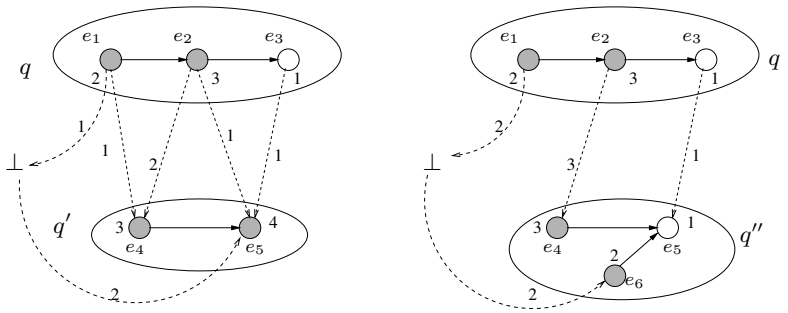

Figure 4: On the left a reallocation violating (Common Birth)/(Common Death); on the right a reallocation satisfying both these properties.

Example 3.15. The reallocation depicted on the left part of Figure 4 violates both (Common Birth) and (Common Death). In q, e_1 has two outgoing arcs. One would indicate that part of e_1

dies whereas the other arc reallocates part of e_1 onto e_4. Similarly, the two arcs starting from e_2 and e_3 and reaching e_5 denote that the latter is old . However, the incoming arc from $\perp$ indicates that e_5 represents also some new entities. (Common Birth) and (Common Death) state that, by conditions 4 and 5, a reallocation rules out explicitly these kinds of ambiguities. The reallocation on the right part of the same figure satisfies (Common Birth) and (Common Death). e_1 dies by mapping the complete cardinality onto $\perp$. Similarly e_6 is new since its cardinality is all provided by $\perp$. □

The relation between morphisms and reallocations is described by the following:

Proposition 3.16. Let γ_1 and γ_2 be two configurations. If $\gamma_1 \overset{h}{\rightarrowtail} \gamma_2$ then $\gamma_1 \overset{\lambda}{\Longrightarrow} \gamma_2$ where let $e \in E_{\gamma_1}$ and $e' \in E_{\gamma_2}$:

$$\lambda(e, e') = \begin{cases} \mathcal{C}_{\gamma_1}(e) & \text{if } e' = h(e) \\ 0 & \text{otherwise.} \end{cases}$$

Proof. See Appendix A. □

Definition 3.17. If $\gamma_1 \overset{\lambda}{\Longrightarrow} \gamma_2$ and $\gamma_1' \overset{\lambda'}{\Longrightarrow} \gamma_2'$ with $\mathcal{C}_{\gamma_1'} = \mathcal{C}_{\gamma_2'} = \mathbf{1}$, we say that λ' is a *concretion* of λ, denoted $\lambda' \rhd \lambda$, if and only if there exist h_1, h_2 such that:

1. $\gamma_1' \overset{h_1}{\rightarrowtail} \gamma_1$ and $\gamma_2' \overset{h_2}{\rightarrowtail} \gamma_2$
2. $\lambda : (e_1, e_2) = \sum_{(e_1,e_2)=(h_1(e_1'),h_2(e_2'))} \lambda'(e_1', e_2')$.
3. (No-Cross)

$$\forall e, e' \in E_{\gamma_1'} : h_1(e) = h_1(e') \vee h_2(\lambda'(e)) = h_2(\lambda'(e')) \;\Rightarrow\; (e \prec_{\gamma_1'} e' \;\Leftrightarrow\; \lambda'(e) \prec_{\gamma_2'} \lambda'(e')).$$

4. $\forall e \in E_{\gamma_2'} : (\mathcal{C}_{\gamma_2}(h_2(e)) = * \Rightarrow e \in \mathrm{cod}(\lambda'))$;

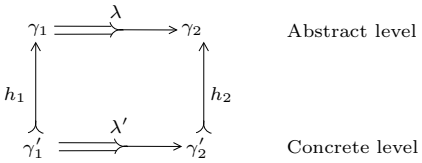

Figure 5: Commutative diagram for concretion of reallocations.

A concretion λ' of λ via h_1 and h_2 is a reallocation that makes the diagram depicted in Figure 5 commute. Informally, λ' can be seen as a reallocation that agrees with λ, but that is defined on a concrete version of γ_1 and γ_2, i.e., λ' is defined on configurations γ_1' and γ_2' related to γ_1 and γ_2 by morphisms. Condition 4 says that if an entity e is projected on an unbounded one then e must be old. This gives a correspondence between the number of new entities in γ_2 and γ_2'. A reallocation and its concretions have the same behaviour in terms of fresh entities (cf. Lemma 3.19). Note that this condition is necessary. In fact, consider the reallocation represented in Figure 6 assuming that $M = 2$ and $\mathcal{C}_{\gamma_1}(e_1) = \mathcal{C}_{\gamma_2}(e_2) = *$. We have $\lambda' \rhd \lambda$, and although $e_9 \in h_2^{-1}(\mathrm{cod}(\lambda))$, it is new. This circumstance is ruled out by condition 4. Condition (No-Cross) prevents the concretion to change the order of entities that are mapped onto a multiple/unbounded entity or that have as image the same multiple/unbounded entity. In fact, since a multiple or an unbounded entity corresponds to a chain we want that the order of the concrete entities corresponding to this chain is not modified after the reallocation. For example, in Figure 7 (left), λ' is a concretion of the reallocation in the right part of Figure 3, whereas λ'' (right) is not since it violates (No-Cross). In particular in Figure 7 (right) , entities e_5', e_5'', e_5''' are mapped on the same abstract entity e_5. However, their order does not correspond to the order of the (concrete) inverse image e_1', e_2', e_2''.

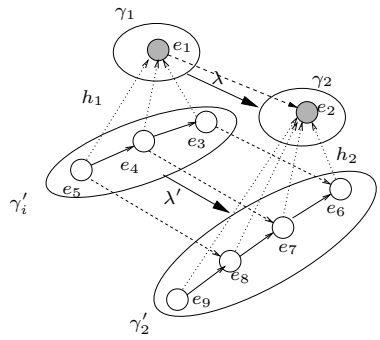

Figure 6: Incompatibility between new entities of the symbolic w.r.t. the concrete level.

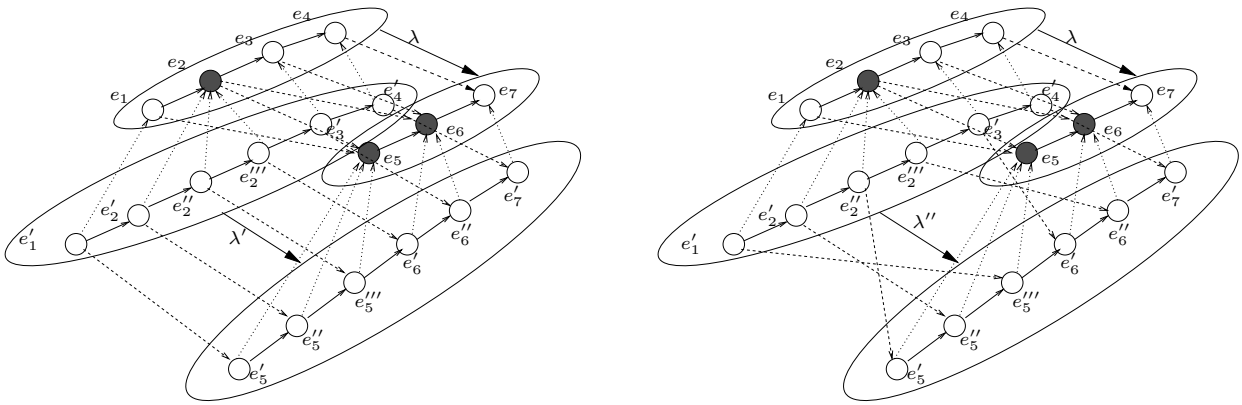

Figure 7: Concretions prevent reshuffling of entities in the concrete states.

Vice-versa, the order of the image of e'_2, e''_2 does not correspond to the order of their image e'''_5 and e'_5. The same phenomenon occurs for e'''_2, e'_3, e'_6 and e''_6.

A consequence of constraint (No-Cross) is that given an abstract entity $e \in E_{\gamma_1}$, its associated concrete entities — i.e., all the elements of the set $h_1^{-1}(e)$ — enjoy a *common fate*: either *all* of them survive the reallocation or *all* of them die. This fact is formalised in the following:

Lemma 3.18 (common fate). If $\gamma_1 \overset{\lambda}{\Longrightarrow} \gamma_2$ and $\gamma'_1 \overset{\lambda'}{\Longrightarrow} \gamma'_2$ and $\lambda' \rhd \lambda$, then

$$\forall e \in E_{\gamma_1} : \forall e', e'' \in h_1^{-1}(e) : (\lambda'(e') = \{\bot\} \Leftrightarrow \lambda'(e'') = \{\bot\}).$$

Proof. By contradiction. Assume there exist $e', e'' \in h_1^{-1}(e)$ such that $\lambda'(e') = \{\bot\}$ and $\lambda'(e'') = \{\tilde{e}\} \neq \{\bot\}$. We can choose e', e'' such that $e' \prec_{\gamma'_1} e''$ or $e'' \prec_{\gamma'_1} e'$. Assume $e' \prec_{\gamma'_1} e''$. By (No-Cross), since $h_1(e') = h_1(e'') = e$ and $e' \prec_{\gamma'_1} e''$ it follows $\lambda'(e') \prec_{\gamma'_2} \lambda'(e'')$ which — by (2) — implies that $\bot \prec_{\gamma'_2} \tilde{e}$ that is impossible. On the other hand, assume $e'' \prec_{\gamma'_1} e'$ that by (No-Cross) implies $\lambda'(e'') \prec_{\gamma'_2} \lambda'(e')$. But then by (2) there must exists a $e''' \in E_{\gamma'_2} \cap \lambda'(e')$ such that $\tilde{e} \prec_{\gamma'_2} e'''$. But again this is a contradiction because $E_{\gamma'_2} \cap \lambda'(e') = \varnothing$. □

Note that the property of common fate does not hold in absence of (No-Cross). The reason has to be sought among the large variety of consequences yielded by unbounded entities. Consider

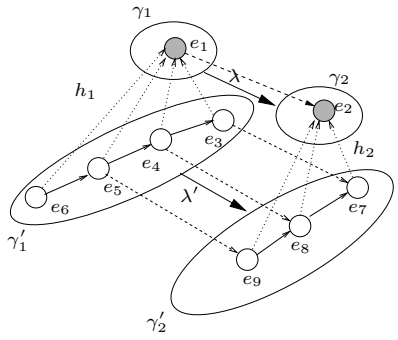

Figure 8: λ' violates the common fate property.

Figure 8, and assume $M = 2$. Since $\lambda(e_1, e_2) = *$, the second condition of $\rhd$ does not reveal that e_6 is deallocated because $\lambda(e_3, e_7) \oplus \lambda(e_4, e_8) \oplus \lambda(e_5, e_9) = *$. However, λ' does not satisfies (No-Cross) because $e_6 \prec_{\gamma'_1} e_5$ but $\lambda'(e_6) \not\prec_{\gamma'_2} \lambda'(e_5)$. Therefore λ and λ' do not have the same behaviour w.r.t. deallocation of entities related by morphisms.

Lemma 3.19. If $\gamma_1 \overset{\lambda}{\Longrightarrow} \gamma_2$ and $\gamma'_1 \overset{\lambda'}{\Longrightarrow} \gamma'_2$ and $\lambda' \rhd \lambda$ via h_1 and h_2 then:

a) $E_{\gamma'_2} \backslash \mathrm{cod}(\lambda') = h_2^{-1}(E_{\gamma_2} \backslash \mathrm{cod}(\lambda))$

b) $E^*_{\gamma_2} \subseteq \mathrm{cod}(\lambda)$.

Proof. See Appendix A. □

The relation $\rhd$ is neither reflexive nor symmetric, but, it is transitive:

Lemma 3.20 ($\rhd$ transitivity). Let $q_1 \overset{\lambda}{\Longrightarrow} q_2$, $q'_1 \overset{\lambda'}{\Longrightarrow} q'_2$ and $q''_1 \overset{\lambda''}{\Longrightarrow} q''_2$. Then:

$$(\lambda'' \rhd \lambda' \ \wedge \ \lambda' \rhd \lambda) \ \Rightarrow \ \lambda'' \rhd \lambda.$$

Proof. See Appendix A □

We are now in the position to define the enhancement of HABAs able to deal with pointer structures.

Definition 3.21. A *High-level ABA* (HABA) with references $\mathcal{H}$ is a tuple $\langle X, Q, E, \rightarrow, I, \mathcal{F}\rangle$ with X, Q, $\mathcal{F}$ as in Def. 3.10, and

- $E : Q \rightarrow \textsc{Conf}$ a function that associates to each state $q \in Q$ a configuration $\gamma_q = (E_q, \mu_q, \mathcal{C}_{E_q})$.
- $\rightarrow \subseteq Q \times (Ent \times Ent \rightharpoonup \mathbb{M}^*) \times Q$, such that if $q \rightarrow_\lambda q'$ then $\gamma_q \stackrel{\lambda}{\Longrightarrow} \gamma_{q'}$.
- $I : Q \rightharpoonup 2^{Ent} \times (X \rightharpoonup Ent)$ a partial function yielding for every *initial state* $q \in \mathrm{dom}(I)$ an *initial valuation* (N, θ), where $N \subseteq (E_q \backslash E_q^*)$ is a finite set of (bounded) entities, and $\theta : X \rightharpoonup E_q$ is a partial valuation of the variables in X;

Whereas in an ABA state we only have concrete entities, this is no longer true in a HABA: the corresponding enforcing condition on the cardinality function of configurations (of an ABA) is now relaxed. The condition $N \subseteq (E_q \backslash E_q^*)$ on the initial state is needed since we want to keep track of the number of new entities in every state.

Sometimes it is useful to specify, the precise range of the cardinalities used in a given HABA $\mathcal{H}$. We write $\mathcal{C}(\mathcal{H}) = M$ if M is the global upper bound on the cardinality of the entities, i.e., if for all $q \in Q_\mathcal{H}$, $\mathrm{cod}(\mathcal{C}_q) \subseteq \mathbb{M}^*$.

Example 3.22. Figure 9 shows an example HABA with references. It models a system where at every step either an entity is created or deallocated. Creation/deletion follows a LIFO policy (i.e., the system modelled is a stack). From state to state there are two transitions: one with a dashed reallocation that models creation (and insertion in the stack), and a transition with a dotted reallocation that models the deletion (and extraction from the stack). For example, in state q_3, entities that are created are accumulated in e_2 (following the dashed reallocations). The dotted reallocation extracts entities by splitting e_2 and remapping it on e_1 and e_2 respectively. Entity e_1 is not remapped and therefore deallocated. The system starts with only one entity in the initial state q_1. The accept state is q_2, therefore the automaton models a system in which every computation has infinitely many times a stack with two entities. □

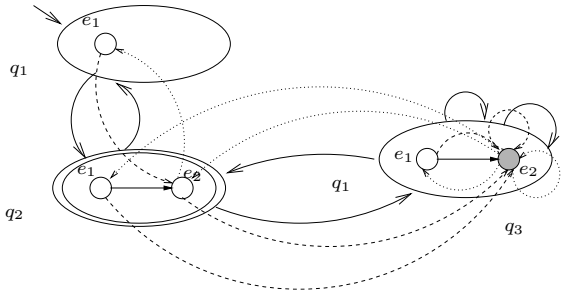

Figure 9: Example HABA with references modelling a stack.

HABA with references are used to generate models for $\mathcal{N}\!a\ell\ell$TL.

Definition 3.23. A *folded allocation sequence* is an infinite alternating sequence

$$(E_0, \mu_0, \mathbf{1}_{E_0})\lambda_0(E_1, \mu_1, \mathbf{1}_{E_1})\lambda_1 \cdots$$

where for $i \geqslant 0$, $(E_i, \mu_i, \mathbf{1}_{E_i}) \stackrel{\lambda_i}{\Longrightarrow} (E_{i+1}, \mu_{i+1}, \mathbf{1}_{E_{i+1}})$.

Note that because of the unitary cardinality functions, in the previous definition, λ_i $(i \geqslant 0)$ may associate at most one entity in E_{i+1} to an entity in E_i. Another consequence is that folded allocation sequences can be used together with to obtain an alternative semantics for $\mathcal{N}\!a\ell\ell$TLdenoted

by $\models_f$. This is done by giving a new definition for N_i^σ and θ_i^σ (defined in page 4) as follows. For an allocation triple (σ, N, θ) let:

$$N_i^\sigma = \begin{cases} N & \text{if } i = 0 \\ E_i^\sigma \backslash \mathrm{cod}(\lambda_{i-1}^\sigma) & \text{otherwise.} \end{cases} \tag{3}$$

$$\theta_i^\sigma = \begin{cases} \theta & \text{if } i = 0 \\ \lambda_{i-1}^\sigma \circ \theta_{i-1}^\sigma & \text{otherwise} \end{cases} \tag{4}$$

where the composition between a (concrete) reallocation $(E, \mu, \mathbf{1}_E) \stackrel{\lambda}{\Rightarrow\!\!\!\!\rightarrow} (E', \mu', \mathbf{1}_{E'})$ and a valuation of the logical variables $\hat{\theta}$ is given by:

$$\lambda \circ \hat{\theta}(x) = \begin{cases} e & \text{if } \hat{\theta}(x) \neq \bot \text{ and } \lambda(\hat{\theta}(x), e) = 1 \\ \bot & \text{otherwise.} \end{cases}$$

The semantics given by $\models_f$ is equivalent to $\models_u$, hence folded and unfolded allocation sequences are equivalent models for $\mathcal{N}\!\mathit{all}\ell\mathrm{TL}$ (the reader is referred to [5] for details).

Runs of a HABA generate folded allocation sequences. The correspondence between allocation triples and runs of HABA is given in terms of morphisms in the following definition.

Definition 3.24 (generator). A run $\rho = q_0\lambda_0 q_1 \lambda_1 \cdots$ of HABA $\mathcal{H} = \langle X, Q, E, \rightarrow, I, \mathcal{F}\rangle$ *generates* an allocation triple (σ, N, θ), where $\sigma = \gamma_0^\sigma \lambda_0^\sigma \gamma_1^\sigma \lambda_1^\sigma \cdots$ is a folded allocation sequence, if there is a *generator*, i.e., a family of morphisms h_i from γ_i^σ to γ_{q_i} satisfying for all $i \geqslant 0$:

1. $\lambda_i^\sigma \rhd \lambda_i$ via h_i and h_{i+1};
2. $I(q_0) = (N', h_0 \circ \theta)$ and $N = h_0^{-1}(N')$;

By condition 1, the morphism in the generator must preserve the reallocations in the run. In particular, λ_i^σ is a concretion of λ_i (for $i \geqslant 0$). Condition 2 imposes a correspondence between the set of initial entities and the interpretation of the logical variables in the state q_0 and the first state of the allocation sequence.

Runs of a HABA are defined in the same way as for ABA. Let $\mathsf{runs}(\mathcal{H})$ denote the set of runs of $\mathcal{H}$ and $\mathcal{L}(\mathcal{H}) = \{(\sigma, N, \theta) \mid \exists \rho \in \mathsf{runs}(\mathcal{H}) : \ \rho \text{ generates } (\sigma, N, \theta)\}$.

4 Relating HABA and ABA

In this section, we study how to relate ABAs and HABAs with references w.r.t. their generated languages and, therefore, the set of $\mathcal{N}\!\mathit{all}\ell\mathrm{TL}$-formulae that they satisfy. First of all, we introduce a simulation preorder (denoted $\sqsubseteq$) between a HABA and a concrete one (i.e., a HABA where every entity is concrete). The intuition behind $\sqsubseteq$ is that whenever $\mathcal{H} \sqsubseteq \mathcal{H}'$, then $\mathcal{H}'$ can simulate all behaviours of $\mathcal{H}$, but the converse does not necessarily hold. That is, $\mathcal{H}'$ may exhibit more behaviours than $\mathcal{H}$.

Definition 4.1. Let $\mathcal{H} = \langle X, Q, E, \rightarrow, I, \mathcal{F}\rangle$ such that $\mathcal{C}(\mathcal{H}) = 1$ and $\mathcal{H}' = \langle X, Q', E', \rightarrow', I', \mathcal{F}'\rangle$.

1. A binary relation $\sqsubseteq \subseteq Q \times (\mathit{Ent} \rightharpoonup \mathit{Ent}) \times Q'$ is a *simulation* if and only if for all $q_1 \sqsubseteq_{h_1} q_1'$ it holds:
   a) $q_1 \stackrel{h_1}{\rightarrowtail} q_1'$
   b) if $q_1 \rightarrow_\lambda q_2$ then $\exists \lambda', h_2$ such that
      i) $q_1' \rightarrow_{\lambda'} q_2' \ \wedge \ q_2 \sqsubseteq_{h_2} q_2'$
      ii) $\lambda \rhd \lambda'$ via h_1, h_2
2. $\mathcal{H}'$ *simulates* $\mathcal{H}$ (written $\mathcal{H} \sqsubseteq \mathcal{H}'$) if and only if there exists a simulation $\sqsubseteq \subseteq Q \times (\mathit{Ent} \rightharpoonup \mathit{Ent}) \times Q'$ such that:

a) for all $q \in I$ there exists $q' \in I'$ and h with $I'(q') = (N, h \circ \theta)$ such that:
   - $q \sqsubseteq_h q'$ and
   - $I(q) = (h^{-1}(N), \theta)$;

b) there exists a bijective $\psi : \mathcal{F} \to \mathcal{F}'$ such that
   $\forall F \in \mathcal{F} : (\forall q \in F : (\exists q' \in \psi(F), \exists h : q \sqsubseteq_h q'))$.

Hence, a state q' simulates q if they represent the same pointer structure (condition 1.a). Every transition of q can be simulated by q' with a proper transition whose reallocation is a concretion of the reallocation of the transition of q. The target states simulate each other and (because of $\triangleright$) have a corresponding set of new entities (condition 1.b).

The second part of the previous definition specifies when a HABA simulates another one. Condition 2.a) requires that every initial state in $\mathcal{H}$ has a corresponding (simulating) initial state in $\mathcal{H}'$ and there is correspondence in the initial valuation. Condition 2.b) says that every accept state in $q \in F \in \mathcal{F}$ is simulated by an accept state $q' \in F' \in \mathcal{F}'$. The bijective function ψ enforces that every $F' \in \mathcal{F}'$ is considered so that the generalised Büchi acceptance condition of $\mathcal{H}$ is ensured to exists in $\mathcal{H}'$.

Example 4.2. The HABA $\mathcal{H}$ depicted in Figure 10 (left) models a FIFO queue. In the initial state there is only one entity. At every step either a new entity is created and enqueued (transitions with dashed reallocations), or the system nondeterministically performs a step where the first entity of the queue is extracted and deallocated (transitions with dotted reallocations). $\mathcal{H}$ is infinite-state since the queue can grow unboundedly. The HABA $\mathcal{H}'$ (where $\mathcal{C}(\mathcal{H}') = 2$), depicted on the right side of the figure, simulates $\mathcal{H}$. □

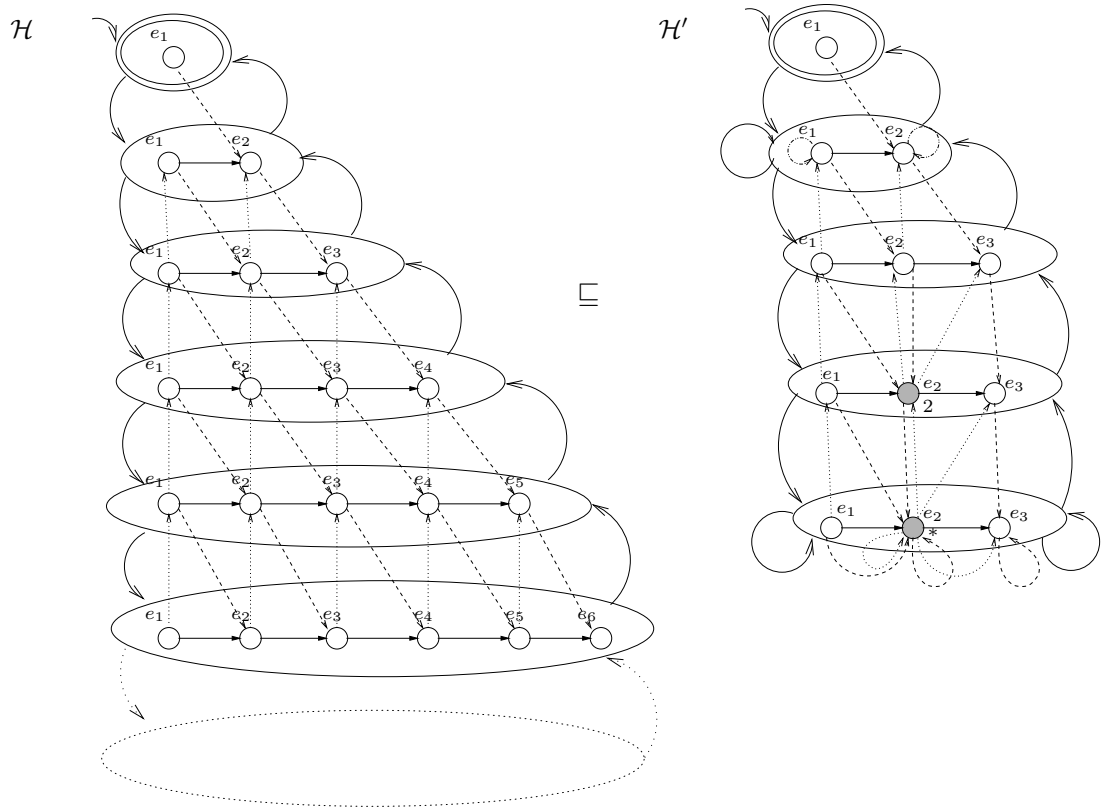

Figure 10: A HABA simulating the one in Figure 10

The comparison between HABAs and ABAs involves also a notion of isomorphic folded allocation sequences. The concept is introduced in the next definition.

Definition 4.3.

- Two folded allocation sequences σ_1, σ_2 are *isomorphic* (written $\sigma_1 \cong \sigma_2$) if there exists a family of isomorphisms $(h_i)_{i\in\mathbb{N}}$ such that for all $i \geqslant 0$,
  1. $h_i : \sigma_1[i] \rightarrowtail \sigma_2[i]$
  2. $\lambda_i^{\sigma_1} \rhd \lambda_i^{\sigma_2}$ via h_i, h_{i+1}.
- Two allocation triples (σ, N, θ) and (σ', N', θ') are *isomorphic* (written $(\sigma, N, \theta) \cong (\sigma', N', \theta')$) if $\sigma \cong \sigma'$, $\mathrm{dom}(\theta) = \mathrm{dom}(\theta')$, $N' = h_0(N)$ and $\theta' = h_0 \circ \theta$.

Isomorphic allocation triples are the same up to renaming of entities. It is therefore not surprising that they satisfy the same set of $\mathcal{N}all\ell$TL formulae, as stated in the next result.

Proposition 4.4. For $\mathcal{N}all\ell$TL formula ϕ and folded allocation sequence σ, σ':

$$(\sigma, N, \theta) \cong (\sigma', N', \theta') \;\Rightarrow\; (\sigma, N, \theta \models \phi \;\text{ iff }\; \sigma', N', \theta' \models \phi).$$

Proof. Straightforward by induction on the structure of ϕ. □

The next important proposition states the relation between the language of HABAs related by a simulation relation.

Theorem 4.5. If $\mathcal{H} \sqsubseteq \mathcal{H}'$ then $\mathcal{L}(\mathcal{H}) \subseteq \mathcal{L}(\mathcal{H}')$.

Proof. See Appendix A. □

$\mathcal{H}'$ describes more behaviours of $\mathcal{H}$, or equivalently we can say that, $\mathcal{H}'$ models spurious behaviours that are not actually present in the concrete system at hand. Nevertheless, the definition of simulation provides a means to compare different models w.r.t. the satisfiability of $\mathcal{N}all\ell$TL formulae.

Definition 4.6. Given a HABA $\mathcal{H}$ and a $\mathcal{N}all\ell$TL-formula ϕ we say that:

- ϕ is $\mathcal{H}$-*satisfiable* if there exists $(\sigma, N, \theta) \in \mathcal{L}(\mathcal{H})$ such that $\sigma, N, \theta \models \phi$;
- ϕ is $\mathcal{H}$-*valid* if for all $(\sigma, N, \theta) \in \mathcal{L}(\mathcal{H}) : \sigma, N, \theta \models \phi$.

The previous definition implies that ϕ is $\mathcal{H}$-valid ($\mathcal{A}$-valid) if and only if $\neg\phi$ is not $\mathcal{H}$-satisfiable ($\mathcal{A}$-satisfiable). For ABA $\mathcal{A}$ the definition for $\mathcal{A}$-*satisfiability* and $\mathcal{A}$-*validity* can be given in precisely the same way.

The relation between models related by a simulation relation is given by the following result.

Proposition 4.7. For HABAs $\mathcal{H}$ and $\mathcal{H}'$ such that $\mathcal{H} \sqsubseteq \mathcal{H}'$ and $\mathcal{N}all\ell$TL-formula ϕ:

$$\phi \text{ is } \mathcal{H}'\text{-valid} \Rightarrow \phi \text{ is } \mathcal{H}\text{-valid}.$$

Proof. Straightforward from Theorem 4.5. □

The converse does not hold as illustrated in the next example.

Example 4.8. Consider the following formula:

$$\phi \quad \equiv \quad \mathsf{G}[\mathsf{X}(\exists x : x\ \mathsf{new} \wedge x.a \neq nil) \vee (\exists y : y.a = nil \wedge \mathsf{X}y\ \mathsf{dead})] \tag{5}$$

expressing that in *every step* of the system either a new entity is created (which is enqueued) or an existing entity not pointing to any other one (i.e., the first of the queue) is deallocated. It is easy to see that ϕ holds in every allocation sequence generated by a run of $\mathcal{H}$ in Figure 10, i.e., ϕ is $\mathcal{H}$-valid. However, ϕ is *not* $\mathcal{H}'$-valid because, in the second state of $\mathcal{H}'$ there exists a self loop with a reallocation mapping every entity to itself. Note that $\mathcal{H}$ does not have such transition. By taking this self-loop no entities are created or destroyed. Hence, ϕ does not hold in every allocation sequence generated by $\mathcal{H}'$. □

Relating folded and unfolded allocation sequences. We can relate folded and unfolded allocation sequences with references in a straightforward manner. The next definition introduces a special kind of reallocation which is useful in order to establish this relation[3].

Definition 4.9. For two configurations γ and γ' such that $\mathcal{C} \restriction E_\gamma \cap E_{\gamma'} = \mathcal{C}' \restriction E_\gamma \cap E_{\gamma'}$, a reallocation $\gamma \overset{\lambda}{\Longrightarrow} \gamma'$ is called *identity* reallocation if for all $e \in E_\gamma$ and $e' \in E_{\gamma'}$:

$$id(e, e') = \begin{cases} \mathcal{C}_\gamma(e) & \text{if } e = e' \\ 0 & \text{otherwise.} \end{cases}$$

We indicate an identity reallocation by id.

Two configurations (and therefore states) related by the identity reallocation may differ only in the links between entities and because some entity has born or has died. The condition on E_γ and $E_{\gamma'}$ says that an entity not related to itself must be related to $\perp$ (in which case is born or dies).

For an unfolded allocation sequence $\sigma = (E_0, \mu_0, \mathbf{1})(E_1, \mu_1, \mathbf{1})(E_2, \mu_2, \mathbf{1}) \cdots$ let $id(\sigma)$ be the folded allocation sequence

$$id(\sigma) = (E_0, \mu_0, \mathbf{1}) id_0 (E_1, \mu_1, \mathbf{1}) id_1 (E_2, \mu_2, \mathbf{1}) id_2 \cdots$$

where $(E_i, \mu_i, \mathbf{1}) \overset{id_i}{\Longrightarrow} (E_{i+1}, \mu_{i+1}, \mathbf{1})$. The two sequences satisfy the same set of $\mathcal{N}all\ell$TL formulae: one on the unfolded semantics of $\mathcal{N}all\ell$TL $\models_u$ and the other on the folded version, i.e., $\models_f$:

Proposition 4.10. For any $\mathcal{N}all\ell$TL formula ϕ we have $\sigma, N, \theta \models_u \phi$ iff $id(\sigma), N, \theta \models_f \phi$.

Proposition 4.11. For every ABA $\mathcal{A}$ there exists a HABA $\mathcal{H}$ such that given a $\mathcal{N}all\ell$TL formula ϕ, we have ϕ is $\mathcal{A}$-valid if and only if ϕ is $\mathcal{H}$-valid.

Proof. Let $\mathcal{A} = \langle X, Q, E, \rightarrow, I, \mathcal{F} \rangle$, then we define $\mathcal{H} = \langle X, Q, E, \rightarrow', I, \mathcal{F} \rangle$ where $\rightarrow'$ is such that

$$q \rightarrow q' \;\Leftrightarrow\; q \rightarrow_{id} q'$$

where $\gamma_q \overset{id}{\Longrightarrow} \gamma_{q'}$. Hence $\mathcal{H}$ is defined as $\mathcal{A}$ with an identity reallocation on the transitions. It is clear that

$$\mathcal{L}(\mathcal{H}) = \{(\sigma', N', \theta') \mid (\sigma', N', \theta') \cong (id(\sigma), N, \theta) \text{ and } (\sigma, N, \theta) \in \mathcal{L}(\mathcal{A})\}.$$

Therefore from Propositions 4.10 and 4.4 it follows that every $\mathcal{A}$-valid formula ϕ is also $\mathcal{H}$-valid and vice-versa. □

We indicate the HABA $\mathcal{H}$ defined in the proof of Proposition 4.11 by $id(\mathcal{A})$. By Propositions 4.7 and 4.11, in order to verify a $\mathcal{N}all\ell$TL-formula ϕ on an $\mathcal{A}$ we can alternatively verify it on a HABA $\mathcal{H}$ that simulates $id(\mathcal{A})$. This is particularly interesting when $\mathcal{A}$ is an infinite model whereas $\mathcal{H}$ is finite. In fact provided we have an effective method to check the validity of $\mathcal{N}all\ell$TL-formulae on finite HABA this can be used to extend the methodology to infinite-state systems.

Example 4.12. The formula $\phi \equiv \mathsf{GF}(\forall x \forall y : x = y)$ stating that the queue will have infinitely often only one entity is $\mathcal{H}'$-valid because only $\mathcal{H}'$ accept state has only one entity, and in an accepting run it is visited infinitely often. It follows that ϕ is also $\mathcal{H}$-valid. □

5 A language for navigation

In this section we introduce a simple programming language, called $\mathcal{L}_n$, dealing with pointers. $\mathcal{L}_n$ gives some insight into the kind of systems that can be modelled by HABA with references. We will define the semantics of $\mathcal{L}_n$ in terms of ABA and HABA. The first semantics is very concrete and intuitive, but infinite-state, whereas the second is symbolic and finite. In Section 6.6, we will study the relation between the two semantics.

[3]It will also be useful for describing reallocations in the transitions of the symbolic operational semantics (cf. Section 6.4).

5.1 Syntax

Let *nil* be a special syntactic constant. For PVAR a set of program variables with $v, v_i \in \text{PVAR}$, and such that $\text{PVAR} \cap \text{LVAR} = \varnothing$, the set of statements of the language $\mathcal{L}_n$ is given by:

$$\begin{array}{lcl}
(p \in) \mathcal{L}_n & ::= & \mathsf{decl}\ v_1, \ldots, v_n : (s_1 \parallel \cdots \parallel s_k) \\
(s \in) \mathit{Stat} & ::= & \mathsf{new}(\alpha) \mid \mathsf{del}(\alpha) \mid \alpha := \alpha \mid \mathsf{skip} \mid s; s \\
 & & \mid \mathsf{if}\ b\ \mathsf{then}\ s\ \mathsf{else}\ s\ \mathsf{fi} \mid \mathsf{while}\ b\ \mathsf{do}\ s\ \mathsf{od} \\
(\alpha \in) \mathit{Nexp} & ::= & \mathit{nil} \mid v \mid \alpha.a \\
(b \in) \mathit{Bexp} & ::= & \alpha = \alpha \mid b \vee b \mid \neg b
\end{array}$$

A program p is thus a parallel composition of a finite number of statements preceded by the declaration of a finite number of global variables.

Informal semantics of $\mathcal{L}_n$. $\mathsf{new}(\alpha)$ creates (i.e., allocates) a new entity that will be referred to by the expression α. If α is the only way to refer to entity e, say, then after the execution of $\mathsf{new}(\alpha)$, e (being not reachable) is automatically garbage collected together with the entities reachable from e. $\mathsf{del}(\alpha)$ destroys (i.e., deallocates) the entity associated to α, and makes α and every other pointer pointing to it undefined. The assignment $\alpha_1 := \alpha_2$ passes the reference held by α_2 to α_1. Again, the entity α_1 was referring to might become unreferenced and therefore may be garbage collected. As assignments create aliases they introduce non-trivial side-effects. Sequential composition, while loop, skip, and conditional statement have the standard interpretation. Meaningless statements and expressions as $\mathsf{new}(\mathit{nil})$, $\mathsf{del}(\mathit{nil})$, $\mathit{nil}.a$ will be ruled out at the semantical level (cf. Section 6).

Example 5.1. The following $\mathcal{L}_n$ program is a classical example often reported in the literature (e.g., see [1, 12, 13]). It reverses a list originally pointed to by the variable v.

```
decl v, w, t :
w := nil;
while v ≠ nil do
      t := w;
      w := v;
      v := v.a;
      w.a := t;
od;
t := nil;
```

□

5.2 Adding program variables to $\mathcal{N}all\ell$TL

$\mathcal{N}all\ell$TL expresses properties about logical variables LVAR. Given a program, $p \equiv \mathsf{decl}\ v_1, \ldots, v_n : (s_1 \| \cdots \| s_k)$, in principle it is not possible to talk about v_i $(1 \leqslant i \leqslant n)$ in $\mathcal{N}all\ell$TL-formulae. For software verification this would represent a severe limitation. We would like to add this feature to $\mathcal{N}all\ell$TL so that properties involving the (declared) program variables $\mathit{DeclPVar} = \{v_1, \ldots, v_n\} \subseteq \text{PVAR}$ (with $v_i \neq v_i$ for $i \neq j$) of p can be naturally formulated. To model $\mathit{DeclPVar}$, we use a special set of entities $PV = \{e_{v_1}, \ldots, e_{v_n}\} \subseteq \mathit{Ent}$ that are alive in every state of the allocation sequence. We assume the existence of a set of free logical variables $\mathit{DeclLVar} = \{x_{v_1}, \ldots, x_{v_n}\}$ such that the interpretation function is:

$$\vartheta(x_{v_i}) = e_{v_i} \qquad \forall 1 \leq i \leq n. \tag{6}$$

Notice that since entities in PV are alive in every state of the allocation sequence, the interpretation $\vartheta \restriction PV$ will remain fixed.

The value of the logical variables x_{v_i} is not very interesting when writing properties about programs. Rather we are interested in the entity that a program variable v points to, that is, $\mu(\vartheta(x_{v_i}))$. Thus it is convenient to define

$$v_i \equiv x_{v_i}.a \qquad \forall v_i \in \mathit{DeclPVar} \tag{7}$$

as syntactic sugar in the logic. This suffices us to express properties about program variables.

Furthermore, we would like to exclude PV from the set of entities considered in the domain of quantifiers. Again this is done by adding syntactic sugar. Throughout Sections 5 and 6, we consider the formula $\exists x : \phi$ as shorthand for

$$\exists x : (x \neq x_{v_1} \wedge \cdots \wedge x \neq x_{v_n}) \Rightarrow \phi. \tag{8}$$

Example 5.2. The configurations (of the states) of the program in Example 5.1 — using entities $PV = \{e_t, e_v, e_w\}$ and the logical variables $\mathit{DeclLVar} = \{x_t, x_v, x_w\}$ to refer to them — are shown in Figure 11. It represents the execution of the loop manipulating a list with three elements.

The main property we want to verify for the program described in Example 5.1 is:

> v's list will be eventually reversed

that can be expressed in $\mathcal{N}\acute{a}\ell\ell$TL (as we have seen before) by:

$$\forall x : \forall y : (v \rightsquigarrow x \wedge x.a = y) \Rightarrow \mathsf{FG}(y.a = x).$$

Moreover, another plausible property could be:

> all the elements in v's list will be eventually contained in w's list

which is expressed by:

$$\forall x : (v \rightsquigarrow x) \Rightarrow \mathsf{FG}(w \rightsquigarrow x).$$

□

6 Operational semantics

In this section, we describe how ABAs and HABAs can be used to give a semantics to programs in our example programming language $\mathcal{L}_n$.

6.1 Preliminary terminology, assumptions and results

As discussed in Section 5.2, program variables can be modelled by a set of special entities that are alive in every state. Therefore, given a program $p \equiv \mathsf{decl}\ v_1, \ldots, v_n : (s_1 \| \cdots \| s_k)$ declaring the set of program variables $\mathit{DeclPVar} = \{v_1, \ldots, v_n\}$, from now let $\mathit{DeclLVar} = \{x_{v_1}, \ldots, x_{v_n}\}$ and $PV = \{e_{v_1}, \ldots, e_{v_n}\}$. Since p is globally given, also $\mathit{DeclLVar}$ and PV will be globally given. The introduction of PV entails some constraints on some definitions already given. In particular, we give the following:

Definition 6.1 (well-formed configuration). A configuration $\gamma = (E, \mu, \mathcal{C})$ with $PV \subseteq E$ is *PV-well-formed* if

$$\mathrm{cod}(\mu) \cap PV = \varnothing \tag{9}$$

$$\mathcal{C} \restriction PV = \mathbf{1}_{PV}. \tag{10}$$

Entities representing program variables cannot be referred to, and are concrete. In the rest of this section we consider only well-formed configurations.

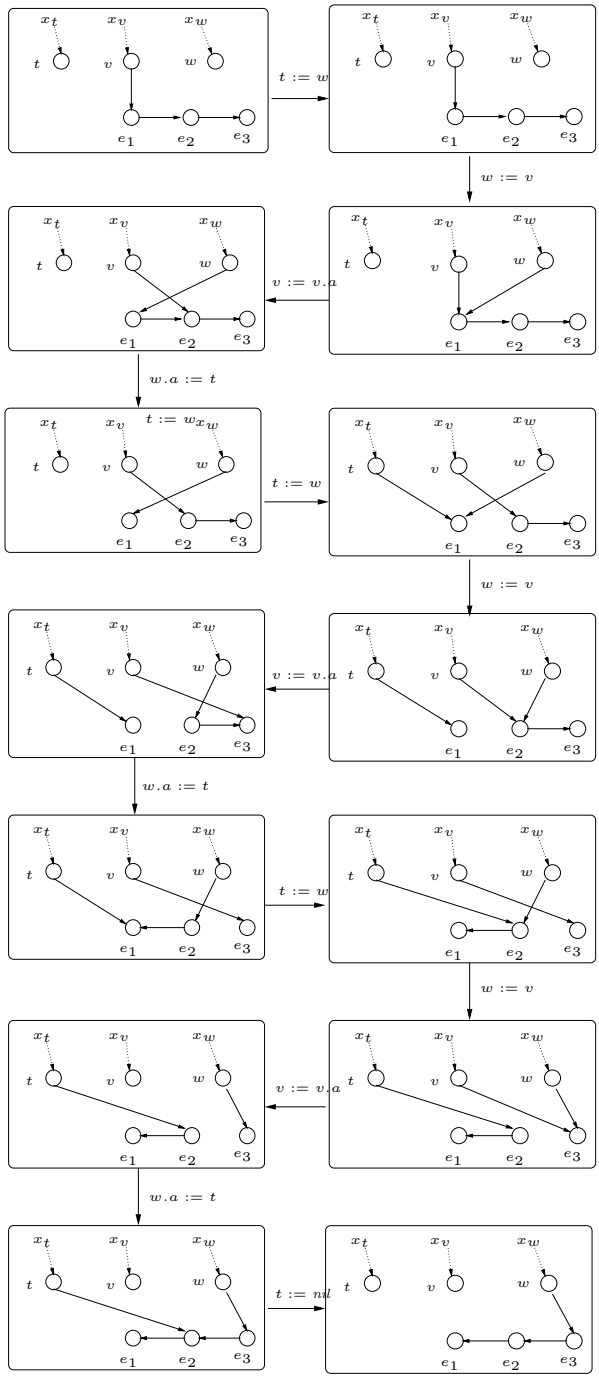

Figure 11: Execution of the program in Example 5.1

Assumptions. For two well-formed configurations γ_1 and γ_2 we consider throughout Section 6 only morphisms as well as reallocations that satisfy the following conditions:

$$\gamma_1 \overset{h}{\rightarrowtail} \gamma_2 \quad \Rightarrow \quad h \restriction PV = id_{PV} \tag{11}$$

$$\gamma_1 \overset{\lambda}{\Rightarrow} \gamma_2 \quad \Rightarrow \quad \forall e \in PV : \lambda(e, e) = 1 \tag{12}$$

Conditions (11) and (12) force the correspondence of the program variables in configurations related by morphisms or reallocations. These constraints seem to be rather natural because of the very special purpose assigned to PV (i.e., modelling p program variables). In fact, it is convenient — for example, from state to state of the computation — to have every program variable remapped on itself. Finally, we can see that (12) not only forces reallocations to map $e_v \in PV$ onto itself, but in combination with (10), ensures also that e_v is the *only* entity that can be reallocated to itself.

Preliminary notation. Let μ^* denote the reflexive and transitive closure of the relation induced by the function μ, i.e., of the relation $\{(e, \mu(e)) \mid e, \mu(e) \in Ent\}$. For configuration γ, and $e \in E_\gamma$, $\mu^*_\gamma(e)$ is the set of entities in the *remainder* of the chain starting with e. Note that $e \in \mu^*_\gamma(e)$. Moreover, let

$$\langle \gamma \rangle_{PV} = (\mu^*_\gamma(PV), \mu_\gamma \restriction \mu^*_\gamma(PV), \mathcal{C}_\gamma \restriction \mu^*_\gamma(PV))$$

be the configuration obtained from γ after garbage collection, i.e., having as set of entities those reachable via μ_γ by some program variable in PV. γ is called *PV-reachable* if $E_{\langle \gamma \rangle_{PV}} = E_\gamma$.

A special class of morphisms that will turn out to be useful later on are characterised by the following:

Definition 6.2.

- A morphism $h : \gamma \rightarrowtail \gamma'$ is *contractive*, denoted $h\!\downarrow$, if $|h^{-1}(e)| > 1$ for some $e \in E_{\gamma'}$;
- the *shrink factor* of a morphism $h : \gamma \rightarrowtail \gamma'$, is $\max\{|h^{-1}(e)| \mid e \in E_{\gamma'}\}$.

We write $h\!\downarrow^C$ if the shrink factor of h is at most C. Non-contractive morphisms have shrink factor 1. A contractive morphism abstracts a chain of entities into a multiple or an unbounded entity. Note that contractive morphisms correspond to non-injective morphisms. We introduce the term "contractive" because it closely resembles the idea that the morphism collapses several entities into one, thus providing a more compact view of the configuration. A straightforward fact is given by the following observation.

Proposition 6.3. Let $h : \gamma \rightarrowtail \gamma'$. If h is non-contractive then h is an isomorphism.

Proof. It is straightforward to see that h is bijective. In fact, every morphism is surjective by definition and h is also injective because it is not contractive. Then, let $h' : \gamma' \rightarrowtail \gamma$ such that $h'(e) = h^{-1}(e)$. h' is well-defined since h is bijective and it can be proved to be a morphism. Moreover, by construction we have $h' \circ h = id_\gamma$. Thus, by definition h is an isomorphism. □

Definition 6.4. For a configuration $(E, \mu, \mathcal{C})$ and entities $e, e' \in E$, if there exists $n \in \mathbb{N}$ such that $e' = \mu^n(e)$, the distance $d(e, e') = n$, otherwise $d(e, e') = \perp$.

Definition 6.5. (L-safety) Let $L > 0$. A configuration $(E, \mu, \mathcal{C})$ is *L-safe* if

$$\forall e \in PV : (\forall e' : d(e, e') \leqslant L \Rightarrow \mathcal{C}(e') = 1).$$

If a configuration is not L-safe it is called *L-unsafe*.

Example 6.6. The configuration depicted in Figure 12 is 2-safe, since every entity $e_{v_1}, \cdots, e_{v_6}$ (representing a program variable) has a distance of at least two from the unbounded entity e. The configuration is not 3-safe, since $d(e_{v_6}, e) = 3$. □

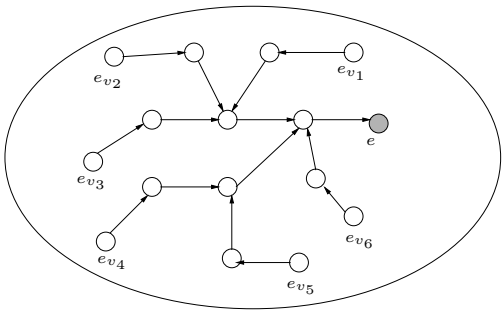

Figure 12: A 2-safe configuration.

The following result says that isomorphic configurations have the same safeness.

Proposition 6.7. For all $L > 0$: $\gamma_1 \cong \gamma_2 \Rightarrow (\gamma_1\ L\text{-safe} \ \Leftrightarrow\ \gamma_2\ L\text{-safe})$.

Proof. Straightforward since isomorphic configurations are equal up to renaming of entities. □

The concepts just introduced for configurations, like well-formedness, PV-reachability and L-safeness are transferred to states in a straightforward manner, saying that a state q is respectively well-formed, PV-reachable, and L-safe if its configuration γ_q is so.

In the following, for the concrete and symbolic semantics we will guarantee that every state q is L-safe, and PV-reachable as well as well-formed.

6.2 Concrete semantics

A concrete semantics of $\mathcal{L}_n$ is given in terms of ABA. Let Par denotes the set of compound statements, i.e., $r(\in Par) ::= s \mid r \parallel s$.

Definition 6.8 (Concrete automaton $\mathcal{A}_p$). The concrete semantics of $p = \mathsf{decl}\ v_1, \ldots, v_n : (s_1 \parallel \cdots \parallel s_k)$ is the ABA $\mathcal{A}_p = \langle X_p, Q, E, \rightarrow, I, \mathcal{F}\rangle$ where

- $X_p = \{x_{v_1}, \ldots, x_{v_n}\}$;
- $Q \subseteq (Par \times \textsc{Conf}) \cup \{\mathsf{error}\}$, where for state $(r, \gamma) \in Q$, r is the compound statement to be executed, and γ is a PV-well-formed and PV-reachable configuration.
- $E(r, \gamma) = \gamma$ and $E(\mathsf{error}) = (\varnothing, \varnothing, \varnothing)$;
- $\rightarrow \subseteq Q \times Q$ is the smallest relation satisfying the rules in Table 2;
- $\mathrm{dom}(I) = \{(s_1 \parallel \cdots \parallel s_k, PV, \varnothing, \mathbf{1}_{PV})\}$ and $I(s_1 \parallel \cdots \parallel s_k, PV, \varnothing, \mathbf{1}_{PV}) = (\varnothing, \vartheta)$ where $\vartheta(x_{v_i}) = e_{v_i}$ $(1 \leqslant i \leqslant n)$;
- let

$$\begin{aligned} \widehat{F}_i &= \{(s'_1 \parallel \cdots \parallel s'_k, \gamma) \in Q \mid s'_i = \mathsf{skip} \vee s'_i = \mathsf{while}\ b\ \mathsf{do}\ s\ \mathsf{od}; s''\} \\ \widetilde{F}_i &= \{(s'_1 \parallel \cdots \parallel s'_k, \gamma) \in Q \mid s'_i = \mathsf{skip} \vee s'_i = s; \mathsf{while}\ b\ \mathsf{do}\ s\ \mathsf{od}; s''\} \end{aligned}$$

  then $\mathcal{F} = \{\widehat{F}_i \cup \{\mathsf{error}\} \mid 0 < i \leqslant k\} \cup \{\widetilde{F}_i \cup \{\mathsf{error}\} \mid 0 < i \leqslant k\}$.

A few remarks are in order. Every configuration γ in a state has $\mathbf{1}_\gamma$ because $\mathcal{A}_p$ is an ABA (cf. Definition 3.10). There is a special error state resulting from an attempt to evaluate an expression $\alpha.a$, where α does not denote any entity (illegal statement). The set of logical variables only contains those that are used to encode the program variables declared in the program p. $\mathcal{A}_p$ has a single initial state $s_1 \parallel \cdots \parallel s_k$. The entities that are initially alive are those used for modelling program variables. ϑ gives the standard interpretation for variables in X_p according to our initial

assumptions (cf. Section 5.2). The set of accept states for the i-th sequential component consists of all states in which the component has either terminated ($s_i = \mathsf{skip}$) or is processing a loop (which could be infinite) or is the error state.

Using the interpretation of navigation expressions defined in Section 2.1 and ϑ given by the initial state, we define the semantics of the boolean expressions used in conditional statements.

Definition 6.9. The semantics of boolean expressions is the function $\mathcal{V} : (Bexp \times \textsc{Conf}) \to \mathbb{B}$ given by (by definition $(\bot = \bot) = \mathsf{tt}$)

$$\begin{aligned}
\mathcal{V}(\alpha_1 = \alpha_2, \gamma) &= \begin{cases} \mathsf{tt} & \text{if } [\![\alpha_1]\!]_{\mu_\gamma,\vartheta} = [\![\alpha_2]\!]_{\mu_\gamma,\vartheta} \\ \mathsf{ff} & \text{otherwise} \end{cases} \\
\mathcal{V}(b_1 \vee b_2, \gamma) &= \mathcal{V}(b_1, \gamma) \vee \mathcal{V}(b_2, \gamma) \\
\mathcal{V}(\neg b, \gamma) &= \neg \mathcal{V}(b, \gamma).
\end{aligned}$$

Manipulating configurations. For the definition of the concrete and symbolic operational rules we rely on three operations that are meant to perform an update on the configuration according to the statement (new, del, or assignment) that is executed by the rule.

The first operation, $add(\gamma, \alpha)$ adds to the configuration γ a fresh entity, and assigns the reference to the entity denoted by the expression α. We assume w.l.o.g. that the set Ent is totally ordered; this is convenient for selecting the fresh entity in a deterministic way, in fact we can take the first one not used in γ, i.e., $\min(Ent \backslash E_\gamma)$. The resulting configuration is composed only by PV-reachable entities, i.e., garbage collection is applied at this stage. Formally, the function $add : \textsc{Conf} \times \Pi \to \textsc{Conf}$ is given by:

$$add(\gamma, \alpha) = \langle E_\gamma \cup \{e\}, \mu_\gamma\{e/[\![\alpha]\!]_{\mu_\gamma,\vartheta}\}, \mathcal{C}_\gamma\{1/e\}\rangle_{PV} \qquad \text{where } e = \min(Ent \backslash E_\gamma). \tag{13}$$

The operation $cancel(\gamma, \alpha)$ deletes from the configuration γ the entity denoted by α. $cancel : \textsc{Conf} \times \Pi \to \textsc{Conf}$ is given by

$$cancel(\gamma, \alpha) = \langle E_\gamma \backslash \{[\![\alpha]\!]_{\mu_\gamma,\vartheta}\}, \mu_\gamma \circ \psi, \mathcal{C}_\gamma \restriction (E_\gamma \backslash \{[\![\alpha]\!]_{\mu_\gamma,\vartheta}\})\rangle_{PV} \tag{14}$$

where $\psi : E_\gamma \to E_\gamma^\bot$ is defined as

$$\psi(e) = \begin{cases} \bot & \text{if } e \in \mu_\gamma^{-1}([\![\alpha]\!]_{\mu_\gamma,\vartheta}) \cup [\![\alpha]\!]_{\mu_\gamma,\vartheta} \\ e & \text{otherwise} \end{cases}$$

In the resulting configuration, every pointer to $[\![\alpha]\!]_{\mu_\gamma,\vartheta}$ are set to $\bot$ by ψ and the domain of the cardinality function is restricted to the remaining set of entities.

Finally, the last operation $modify(\gamma, \alpha_1, \alpha_2)$ performs an update on γ's pointer structure so that the entity denoted by α_1 will point to $[\![\alpha_2]\!]_{\mu_\gamma,\vartheta}$. This will be used in the assignment rules. As usual, the resulting configuration contains only reachable entities. The function $modify : \textsc{Conf} \times \Pi \to \textsc{Conf}$ is given by

$$modify(\gamma, \alpha_1, \alpha_2) = \langle E_\gamma, \mu_\gamma\{[\![\alpha_2]\!]_{\mu_\gamma,\vartheta}/[\![\alpha_1]\!]_{\mu_\gamma,\vartheta}\}, \mathcal{C}_\gamma\rangle_{PV} \tag{15}$$

Concrete operational rules. Transitions for the basic statements follows the general pattern:

$$\overline{r, \gamma \to \mathsf{skip}, \gamma'}$$

The configuration of the target state of a transition has a transformation of the configuration of the source state, and has only reachable entities. The transformation is carried out by one of the operations (13), (14), and (15) defined above. If garbage is produced by a transition it is immediately collected and removed by the application of the operation itself. Furthermore, recall that a program variable v is syntactic sugar for $x_v.a$. Therefore every navigation expression occurring in a statement is of the form $\alpha.a$ (or nil). Finally, we write $[\![\alpha]\!]$ as a shorthand for $[\![\alpha]\!]_{\mu,\vartheta}$. We briefly comment on the rules contained in Table 2.

($\mathsf{NEW}_{\mathsf{error}}$-conc)	$\dfrac{[\![\alpha]\!] = \bot}{\mathsf{new}(\alpha.a), \gamma \rightarrow \mathsf{error}}$
(NEW-conc)	$\dfrac{[\![\alpha]\!] \neq \bot}{\mathsf{new}(\alpha.a), \gamma \rightarrow \mathsf{skip}, add(\gamma, \alpha)}$
($\mathsf{DEL}_{\mathsf{error}}$-conc)	$\dfrac{[\![\alpha]\!] = \bot}{\mathsf{del}(\alpha.a), \gamma \rightarrow \mathsf{error}}$
(DEL-conc)	$\dfrac{[\![\alpha]\!] \neq \bot}{\mathsf{del}(\alpha.a), \gamma \rightarrow \mathsf{skip}, cancel(\gamma, \alpha.a)}$
($\mathsf{ASGN}_{\mathsf{error}}$-conc)	$\dfrac{[\![\alpha_1]\!] = \bot}{\alpha_1.a := \alpha_2, \gamma \rightarrow \mathsf{error}}$
(ASGN-conc)	$\dfrac{[\![\alpha_1]\!] \neq \bot}{\alpha_1.a := \alpha_2, \gamma \rightarrow \mathsf{skip}, modify(\gamma, \alpha_1, \alpha_2)}$
(IF_1-conc)	$\dfrac{\mathcal{V}(b, \gamma)}{\mathsf{if}\ b\ \mathsf{then}\ s_1\ \mathsf{else}\ s_2\ \mathsf{fi}, \gamma \rightarrow s_1, \gamma}$
(IF_2-conc)	$\dfrac{\neg\mathcal{V}(b, \gamma)}{\mathsf{if}\ b\ \mathsf{then}\ s_1\ \mathsf{else}\ s_2\ \mathsf{fi}, \gamma \rightarrow s_2, \gamma}$
(WHILE-conc)	$\overline{\mathsf{while}\ b\ \mathsf{do}\ s\ \mathsf{od}, \gamma \rightarrow \mathsf{if}\ b\ \mathsf{then}\ s; \mathsf{while}\ b\ \mathsf{do}\ s\ \mathsf{od}\ \mathsf{else}\ \mathsf{skip}\ \mathsf{fi}, \gamma}$
($\mathsf{SEQ}_{\mathsf{error}}$-conc)	$\dfrac{s_1, \gamma \rightarrow \mathsf{error}}{s_1; s_2, \gamma \rightarrow \mathsf{error}}$
(SEQ_1-conc)	$\dfrac{s_1, \gamma \rightarrow s_1', \gamma' \ \wedge\ s_1' \neq \mathsf{error}}{s_1; s_2, \gamma \rightarrow s_1'; s_2, \gamma'}$
(SEQ_2-conc)	$\overline{\mathsf{skip}; s_2, \gamma \rightarrow s_2, \gamma}$
($\mathsf{PAR}_{\mathsf{error}}$-conc)	$\dfrac{1 \leqslant j \leqslant k \ \wedge\ s_j, \gamma \rightarrow \mathsf{error}}{s_1 \parallel \cdots \parallel s_j \parallel \cdots \parallel s_k, \gamma \rightarrow \mathsf{error}}$
(PAR_1-conc)	$\dfrac{1 \leqslant j \leqslant k \ \wedge\ s_j, \gamma \rightarrow s_j', \gamma' \ \wedge\ s_j' \neq \mathsf{error}}{s_1 \parallel \cdots \parallel s_j \parallel \cdots \parallel s_k, \gamma \rightarrow s_1 \parallel \cdots \parallel s_j' \parallel \cdots \parallel s_k, \gamma'}$
(PAR_2-conc)	$\overline{\mathsf{skip} \parallel \cdots \parallel \mathsf{skip}, \gamma \rightarrow \mathsf{skip} \parallel \cdots \parallel \mathsf{skip}, \gamma}$
(ERROR)	$\overline{\mathsf{error} \rightarrow \mathsf{error}}$

Table 2: Operational rules for the concrete semantics of $\mathcal{L}_n$.

- **Assignment.** Trying to perform an assignment $\alpha_1.a := \alpha_2$ results — by the application of rule ($\mathsf{ASGN}_{\mathsf{error}}$-conc) — in a run-time error if α_1 is a null pointer. Otherwise, rule (ASGN-conc) applies[4]. By the application of $modify(\gamma, \alpha_1, \alpha_2)$, the execution of the assignment corresponds to updating the outgoing reference $\mu([\![\alpha_1]\!])$ to the entity denoted by α_2 (e.g. see a transition resulting from an assignment in Figure 11). After the execution, the assignment statement is replaced by skip that is either consumed in the context of a sequential composition rule or is blocked. This general pattern is also followed by rules (NEW-conc) and (DEL-conc).

- **Creation.** Unless α is an illegal expression — in which case ($\mathsf{NEW}_{\mathsf{error}}$-conc) applies producing a run-time error — the reference of the first (fresh) entity e available from Ent according to the total order is assigned (by $add(\gamma, \alpha)$) to the outgoing reference of the entity denoted by α, cf. (13).

- **Deletion.** By (DEL-conc), the entity denoted by $\alpha.a$ is deallocated and every reference to this entity is cancelled according to the definition of $cancel(\gamma, \alpha.a)$. Rule ($\mathsf{DEL}_{\mathsf{error}}$-conc) only applies if α dereferences a null pointer.

- **Conditional and loop.** Rules (IF_1-conc)/(IF_2-conc)/(WHILE-conc) are straightforward[5].

- **Sequential composition.** By rules (SEQ_1-conc)/(SEQ_2-conc) when the first statement is reduced to a skip statement, it is consumed, if it is reduced to error, $s_1; s_2$ reduces to error.

- **Parallel composition.** By (PAR_1-conc), if one of the components of the compound statement performs a step, the whole compound statement can do so, unless an error is produced in which case the whole compound statement reduces to error. By (PAR_2-conc), a self-loop in an accept state with a terminated compound statement ensures that, in a run, it is visited infinitely many times.

- **Error.** A self-loop in an error state produces accepting runs that have terminated in an abnormal way because of run-time errors due to the dereferencing of null pointers.

Proposition 6.10. For any $L > 0$, if $q \in Q_{\mathcal{A}_p}$ then q is L-safe, PV-reachable and well-formed.

Proof. Straightforward. In fact, any $q \in Q_{\mathcal{A}_p}$ has the unitary cardinality function therefore it is safe by definition. Furthermore, states are PV-reachable by definition. Finally, q is well-formed; in fact, (10) is trivially satisfied and condition (9) it satisfied by the initial state. Moreover, modifications of pointers in the rules of Table 2 are performed only by (NEW-conc) and (ASGN-conc). But both rules assign entities which do not belong to PV by definition. In particular, for (ASGN-conc) note that $[\![\alpha_2.a]\!]$ cannot be in PV because of the syntactic structure of $\alpha_2.a$. □

6.3 Canonical form for HABA states

For the definition of the symbolic semantics it is convenient to define the concept of normal form which provides a standard representation (unique up to isomorphism) for a set of safe states that may be related by morphisms. To this end we first define a notion that it somehow complementary to safeness.

[4]Note that if α_2 is of the form $\alpha_3.a$ where $[\![\alpha_3]\!]_{\mu_\gamma,\vartheta} = \perp$ the assignment $\alpha_1.a := \alpha_2$ is treated by the current semantics as $\alpha_1.a := nil$. Another possible choice in the design of the semantics could be to treat such statement as "illegal", and therefore, by the operational rule give a run-time error by making a transition to the error state. The conversion of the current approach to the other is straightforward.

[5]As already observed for the assignment rule, note that, b may contain expressions like $\alpha_1.a = \alpha_2.a$ where either $[\![\alpha_1]\!]_{\mu_\gamma,\vartheta} = \perp$ or $[\![\alpha_2]\!]_{\mu_\gamma,\vartheta} = \perp$. The function $\mathcal{V}(b, \gamma)$ return a boolean value also in this case. As above, such expression are not treated as "illegal" but as nil. Again, it would not be problematic to adapt this semantics to the opposite approach.

Definition 6.11 (L-compactness). A configuration $(E, \mu, \mathcal{C})$ is *L-compact* if

$$\forall e \in E : (indegree(e) > 1 \ \vee \ \exists e' \in PV : d(e', e) \leqslant L + 1).$$

As usual we extend this notion to states in the standard way: a state q is L-compact if its configuration γ_q is so.

Thus, a compact state is a state that can have pure chains only if they are distant at most $L + 1$ entities from a program variable. Given two states q and q' one of which is L-compact, in general there might exist more than one morphism relating them. This is normally the case if the state is not L-safe.

Example 6.12. Let $L = 2$ and the global constant $M = 1$. Let (cf. Figure 13):

$$\begin{aligned} q &= (\{e_v, e_1, e_2\}, \{(e_v, e_1), (e_1, e_2)\}, \{(e_v, 1), (e_1, *), (e_2, *)\}) \\ q' &= (\{e_v\} \cup \{e_i \mid 1 \leqslant i \leqslant 6\}, \{(e_v, e_1)\} \cup \{(e_i, e_{i+1}) \mid 1 \leqslant i \leqslant 5\}, \{(e_v, 1)\} \cup \{(e_i, 1) \mid 1 \leqslant i \leqslant 6\}). \end{aligned}$$

State q is L-compact (but not L-safe), however, there exists more than one morphism between q' and q. In fact let $h_1 : q' \succ\!\!\longrightarrow q$ and $h_2 : q' \succ\!\!\longrightarrow q$ defined as:

$$\begin{aligned} h_1(e_i) &= \begin{cases} e_1 & \text{if } 1 \leqslant i \leqslant 2 \\ e_2 & \text{otherwise} \end{cases} \\ h_2(e_i) &= \begin{cases} e_1 & \text{if } 1 \leqslant i < 4 \\ e_2 & \text{otherwise.} \end{cases} \end{aligned}$$

Hence, h_1 and h_2 distribute entities of q' onto entities of q in different ways. Figure 13 gives a pictorial representation of this situation. h_1 is represented by dashed lines whereas h_2 by dotted ones. □

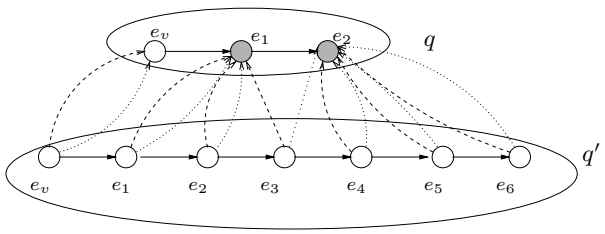

Figure 13: More than one morphism can relate a L-compact state to another.

We now define the notion of canonical form.

Definition 6.13 (canonical form). A configuration γ is *L-canonical* (or in L-normal form) if

- γ is L-safe;
- γ is L-compact.

A configuration in canonical form enjoys several properties.

Proposition 6.14. If a configuration γ is L-canonical then:

a) γ is PV-reachable;

b) for every configuration γ', if there exists a morphism $h : \gamma \succ\!\!\longrightarrow \gamma'$ then either $\gamma \cong \gamma'$ or γ' is L-unsafe.

Proof. See Appendix B. □

The previous proposition shows that if a configuration is in canonical form then it is in the most compact (safe) form up to isomorphism. In fact, if compacted further, it would be unsafe.

Proposition 6.15. Let γ_1 and γ_2 be a PV-reachable and a L-canonical configurations, respectively. If $h_1 : \gamma_1 \rightarrowtail \gamma_2$ and $h_2 : \gamma_1 \rightarrowtail \gamma_2$ then $h_1 = h_2$.

Proof. See Appendix B. □

Theorem 6.16 (Existence of the canonical form). For every L-safe and PV-reachable configuration γ there exists an L-canonical configuration γ' and a unique morphism $h : \gamma \rightarrowtail \gamma'$.

Proof. See Appendix B. □

Corollary 6.17. The canonical form of L-safe configuration γ is unique (up to isomorphism).

We call γ' (of the previous theorem) the canonical form of γ and indicate it by $\mathsf{cf}(\gamma)$. We write $h_{\mathsf{cf}}(\gamma)$ for the unique morphism h relating γ and $\mathsf{cf}(\gamma)$.

Safe expansions. Theorem 6.16 ensures the existence of a unique canonical form for safe configuration. However, in the definition of the assignment rule we need to deal with unsafe configurations. We have also seen that an unsafe configuration can be related to a safe one by more than one morphism. The following notion defines a finite set of pairs (γ', h) where, the first component γ' is an L-safe configuration representing the same topological structure of a possibly unsafe configuration γ; the second component h is the morphism relating γ' and γ.

Definition 6.18. The set of *safe expansions* of a configuration γ is

$$\mathsf{SExp}(\gamma) = \{(\gamma', h) \mid \gamma' \text{ is } L\text{-safe and } h\downarrow^{L+1} \colon \gamma' \rightarrowtail \gamma\}.$$

In $\mathsf{SExp}(\gamma)$ all configurations that are included are related to γ by a contractive morphism (cf. Definition 6.2) with shrink factor at most $L+1$. This bound ensures that $\mathsf{SExp}(\gamma)$ is finite (up to isomorphism). Moreover, we will see that this is enough to include in $\mathsf{SExp}(\gamma)$ all the necessary configurations needed for the definition of the assignment rule in Section 6.4.

Example 6.19. Assume the global constant $M = 2$. Then configuration γ_q depicted in Figure 14 (left) is not 4-safe. The $\mathsf{SExp}(q_\gamma)$ contains the configurations reported in the right part of figure enclosed in the dashed box. For each of them the shrink factor is indicated together with the corresponding cardinality of e_0 that the single (configuration of the) state represents. □

Combining reallocations and morphisms. Combining reallocations with morphisms in general does not result neither in a morphism nor in a reallocation. However, in some special cases the combination of an *id* reallocation followed by morphisms defines a reallocation. For the definition of the symbolic operational rules we are interested in some of these special cases.

The following proposition proves that it is possible to complete the diagram reported in Figure 15 by a reallocation λ.

Proposition 6.20. Let γ and γ''' be two L-canonical states. If $\gamma \overset{id}{\Longrightarrow} \gamma' \overset{h_1}{\prec\!\!\!\!\leftarrow} \gamma'' \overset{h_2}{\rightarrowtail} \gamma'''$ such that

(a) $\forall e \in E_{\gamma'''} : id^{-1}(h_1 \circ h_2^{-1}(e)) \subseteq E_\gamma^\perp$ is a chain and

(b) $\mathcal{C}_{\gamma'''}(e) = * \;\Rightarrow\; \perp \notin id^{-1}(h_1 \circ h_2^{-1}(e))$.

Then $\gamma \overset{\lambda}{\Longrightarrow} \gamma'''$ where:

$$\begin{aligned} \lambda \;=\; & \{(e_1, e_2, \mathcal{C}_{\gamma''}(h_1^{-1}(e_1) \cap h_2^{-1}(e_2))) \mid e_1 \in E_\gamma\} \cup \\ & \{(e, \perp, \mathcal{C}_\gamma(e)) \mid e \in E_\gamma \backslash E_{\gamma'}\} \cup \\ & \{(\perp, e, \mathcal{C}_{\gamma'''}(e)) \mid e \in E_{\gamma'''} \;\wedge\; h_1 \circ h_2^{-1}(e) \cap E_\gamma = \varnothing\}. \end{aligned}$$

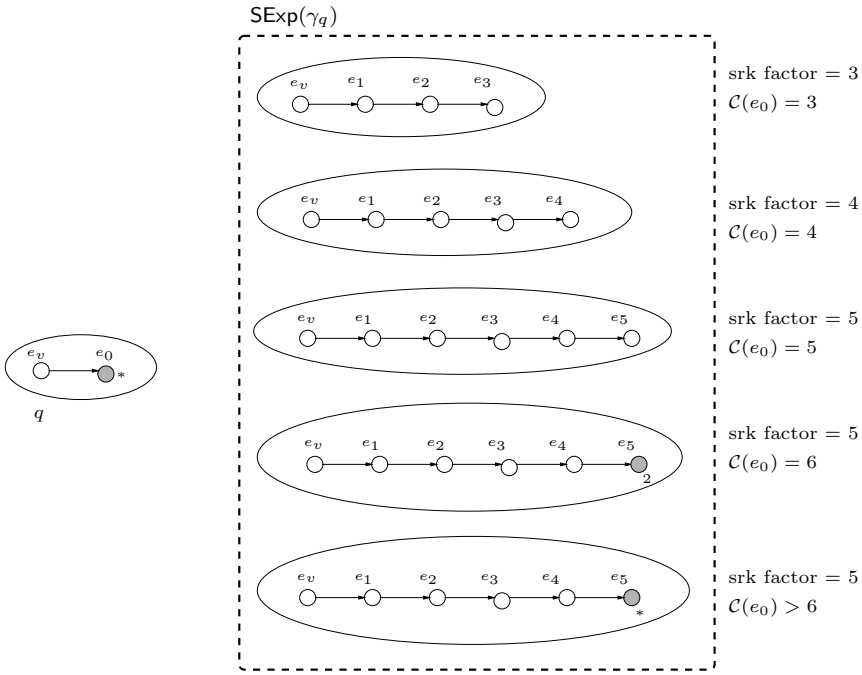

Figure 14: Example of safe expansions for an unsafe state (case $L = 4$, $M = 2$).

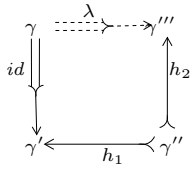

Figure 15: Diagram of Proposition 6.20

Proof. See Appendix B. □

We write $h_2 \circ h_1^{-1} \circ (\gamma \overset{id}{\Longrightarrow} \gamma')$ to indicate the reallocation λ defined in the previous proposition. In some cases we will use a simplified version of $h_2 \circ h_1^{-1} \circ (\gamma \overset{id}{\Longrightarrow} \gamma')$ that corresponds to the special case $\gamma \overset{id}{\Longrightarrow} \gamma' \overset{h}{\rightarrowtail} \gamma''$. In this case we define:

$$\begin{aligned} h \circ (\gamma \overset{id}{\Longrightarrow} \gamma') \quad = \quad & \{(e, h(e), id(e,e)) \mid e \in E_\gamma \cap E_{\gamma'}\} \cup \\ & \{(e, \bot, \mathcal{C}_\gamma(e)) \mid e \in E_\gamma \backslash E_{\gamma'}\} \cup \\ & \{(\bot, h(e), \mathcal{C}_{\gamma''}(e)) \mid e \in E_{\gamma'} \backslash E_\gamma\}. \end{aligned} \tag{16}$$

As $h_2 \circ h_1^{-1} \circ (\gamma \overset{id}{\Longrightarrow} \gamma')$ also $h \circ (\gamma \overset{id}{\Longrightarrow} \gamma')$ defines a reallocation provided that the same hypothesis of Proposition 6.20 are valid. This is stated in the next remark.

Corollary 6.21. Let γ, γ' be two L-canonical configurations. If $\gamma \overset{id}{\Longrightarrow} \gamma' \overset{h}{\rightarrowtail} \gamma''$ such that

(a) $\forall e \in E_{\gamma''} : id^{-1}(h^{-1}(e)) \subseteq E_\gamma^\bot$ is a chain and

(b) $\mathcal{C}_{\gamma''}(e) = * \ \Rightarrow \ \bot \notin id^{-1}(h^{-1}(e))$.

then $\gamma \overset{\lambda}{\Longrightarrow} \gamma''$ where $h \circ (\gamma \overset{id}{\Longrightarrow} \gamma')$.

Proof. Straightforward. In fact, by Proposition 6.20 we have $\gamma \overset{\lambda}{\Longrightarrow} \gamma''$ because $\gamma \overset{id}{\Longrightarrow} \gamma' \overset{id}{\rightarrowtail} \gamma' \overset{h}{\rightarrowtail} \gamma''$ where λ, defined in the proposition, is precisely $h \circ (\gamma \overset{id}{\Longrightarrow} \gamma')$. □

The composition of *id* reallocations with morphisms is interesting because the operation *add, cancel, modify* — that we will use for the definition of the symbolic rules — transform a configuration γ into a configuration γ' which is related to the former by an *id* reallocation.

Lemma 6.22. Let γ be a configuration, and α, α' be navigation expressions. Then:

1. $\gamma \overset{id}{\Longrightarrow} add(\gamma, \alpha)$
2. $\gamma \overset{id}{\Longrightarrow} cancel(\gamma, \alpha)$
3. $\gamma \overset{id}{\Longrightarrow} modify(\gamma, \alpha, \alpha')$.

Proof. Straightforward. In fact, for the three operation it is easy to define the corresponding identity reallocations.

1. For $e \in E_\gamma, e' \in E_{add(\gamma,\alpha)}$, let $\lambda : \gamma \Longrightarrow E_{add(\gamma,\alpha)}$ defined as:

$$\begin{aligned} \lambda(e, e') \quad &= \quad \begin{cases} \mathcal{C}_\gamma(e) & \text{if } e = e' \\ 0 & \text{otherwise.} \end{cases} \\ \lambda(\bot, e') \quad &= \quad \begin{cases} 1 & \text{if } e' = \min(Ent \backslash E_\gamma) \\ 0 & \text{otherwise.} \end{cases} \\ \lambda(e, \bot) \quad &= \quad \begin{cases} \mathcal{C}_\gamma(e) & \text{if } E_\gamma \backslash E_{add(\gamma,\alpha)} \\ 0 & \text{otherwise.} \end{cases} \end{aligned}$$

   It is possible to verify that λ defines a reallocation according to Definition 3.12 and moreover it satisfies the Definition 4.9, therefore it is an identity reallocation.

2. Similar to the previous case. For $e \in E_\gamma, e' \in E_{cancel(\gamma,\alpha)}$, let $\lambda' : \gamma \Longrightarrow E_{cancel(\gamma,\alpha)}$ defined as:

$$\begin{aligned} \lambda'(e, e') \quad &= \quad \begin{cases} \mathcal{C}_\gamma(e) & \text{if } e = e' \\ 0 & \text{otherwise.} \end{cases} \\ \lambda'(e, \bot) \quad &= \quad \begin{cases} \mathcal{C}_\gamma(e) & \text{if } E_\gamma \backslash E_{cancel(\gamma,\alpha)} \\ 0 & \text{otherwise.} \end{cases} \end{aligned}$$

   Like the *add* case, it can be verified that λ' is an identity reallocation.

3. Similar to the *cancel* case. For $e \in E_\gamma, e' \in E_{modify(\gamma,\alpha,\alpha')}$, let $\lambda'' : \gamma \Longrightarrow\!\!\!\!\!\rightarrow E_{modify(\gamma,\alpha,\alpha')}$ defined as:

$$\lambda''(e,e') = \begin{cases} \mathcal{C}_\gamma(e) & \text{if } e = e' \\ 0 & \text{otherwise.} \end{cases}$$

$$\lambda''(e,\perp) = \begin{cases} \mathcal{C}_\gamma(e) & \text{if } E_\gamma \backslash E_{modify(\gamma,\alpha,\alpha')} \\ 0 & \text{otherwise.} \end{cases}$$

Again, like the *add* case, it can be verified that λ'' is an identity reallocation.

□

6.4 Symbolic semantics

The concrete semantics of $\mathcal{L}_n$ defined in the previous section is rather intuitive, but results in an infinite state space. For example, this can happen, in our case, because during the computation the mechanism of entity creation is invoked infinitely often without equally many deletions. In this section we define a symbolic semantics of $\mathcal{L}_n$ in terms of HABA with references. We will exploit the notion of canonical form for states introduced in Section 6.3 and we will observe that this helps to achieve a finite-state semantics for every program of the language.

Assumptions. In the definition of the symbolic semantics, we assume to know the longest navigation expression occurring in the program p. This number, denoted by L_p, can be determined statically by a syntactic analysis over p. In any state of the program, L_p provides us with an upper bound (in terms of distance from a program variable) to the most distant entity accessed by a statement of p. For example, if $x.a^4$ is the longest occurring reference expression in p then $L_p = 5$. Thus we define:

$$L_p = \max\{n \mid v.a^n \text{ occurs in } p\} + 1. \tag{17}$$

Informal idea of the symbolic model. In the symbolic semantics, we exploit unbounded entities in order to keep the model finite. The price to pay for the resulting finitary treatment of the semantics is an increase in the complexity of the machinery needed for the definition of the transition system. In particular, the difficulties inherent to the employment of unbounded entities are two:

- It may be unclear which entities are involved in the execution of a statement.
- it may be unclear which is the state resulting after the execution of a statement.

The direct consequence is the unavoidable introduction of *non-determinism* in the model. We would like to exploit as much information as possible in order to minimise the amount of this nondeterminism. For this reason, the symbolic semantics applies the following strategy:

> whenever an unbounded entity appears in a state we make sure that this is preceded by a chain of length at least L_p of concrete entities.

In other words, any state of the symbolic semantics is enforced to be L_p-safe (cf. Definition 6.5). This implies, by assumption on L_p (see (17)), that in every state we can precisely determine the *concrete* entity denoted by any navigation expression occurring in new or del statements. The major benefit is that for these statements the operational rules are deterministic and easy to define. The only source of *nondeterminism* in the symbolic semantics may be the assignment statement. In fact, although the entities denoted by both the expressions on the left-hand side and on the right-hand side of the assignment are uniquely identified (by L_p-safeness assumption), the effect of an assignment may result in an unsafe state. In general, this happens when variables change their reference to some entities that are closer to an unbounded entity. For example, the state q depicted in Figure 16 is 3-safe. However, the assignment $w := w.a.a.a$ produces the 3-unsafe state

q'. Hence, since we admit only safe states, it is not possible to take as a result of an assignment q' that merely results from the manipulation of pointers dictated by the assignment. We need to consider "safe versions" of q', i.e., more concrete states that represent the same pointer structure. In terms of Definition 6.18, this means to consider states in $\mathsf{SExp}(q')$. In general there can be more than one possibility and therefore a nondeterministic step is unavoidable.

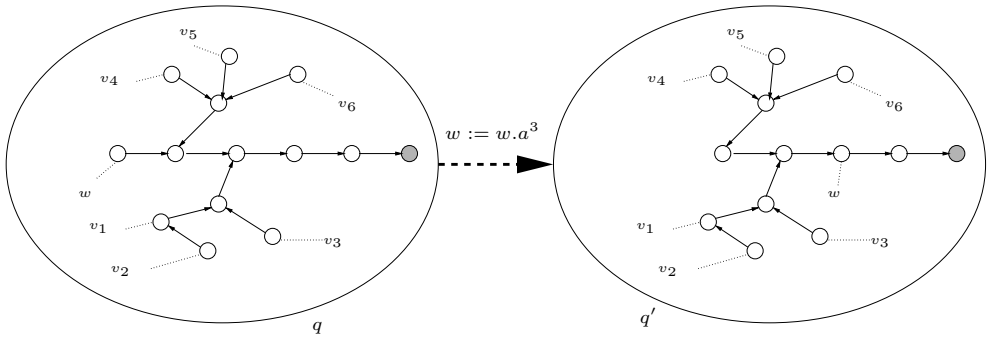

Figure 16: From the 3-safe state q the assignment $w := w.a^3$ produces the 3-unsafe state q'.

Definition 6.23 (Symbolic automaton $\mathcal{H}_p$). The symbolic semantics of $p = \mathsf{decl}\ v_1, \ldots, v_n : (s_1 \parallel \cdots \parallel s_k)$ is the HABA $\mathcal{H}_p = \langle X_p, Q, E, \rightarrow, I, \mathcal{F} \rangle$ where

- $Q \subseteq (Par \times \textsc{Conf}) \cup \{\mathsf{error}\}$, i.e., a state (r, γ) consists of a compound statement r, and a PV-well-formed and L_p-canonical configuration γ.
- $\rightarrow \subseteq Q \times (Ent \times Ent \rightharpoonup \mathbb{M}^*) \times Q$, is the smallest relation defined by the rules in Table 6.4; see Section 6.5 for concepts and notation.

and X_p, E, I and $\mathcal{F}$ are defined in the same way as Definition 6.8.

The definition of $\mathcal{H}_p$ resembles the one given for $\mathcal{A}_p$ in many aspects, in particular concerning the initial state and the accept state, and the reader is referred to Definition 6.8 for comments on these components.

6.5 Symbolic operational rules

The rules of the symbolic operational semantics are heavily based on the central notion of canonical form defined in Section 6.3. They follow the general pattern:

$$\overline{r, \gamma \rightarrow_\lambda \mathsf{skip}, \mathsf{cf}(\gamma')}$$

The idea is that the target state is the canonical form of a certain manipulation of the source state performed — as for the concrete semantics — by one of the operation (13), (14) or (15). The canonical form must be enforced since manipulating pointers in the source state (in particular, during an assignment) may indeed result in a target state that is not L_p-canonical (see above). Observe that since the target state is PV-reachable[6], garbage collection is applied at every transition. Hence, apart from the canonical form, any symbolic rule resembles the corresponding concrete rule. The only exception is (ASGN-sym) (that we comment below). There exists in general a relation between the reallocation λ and the morphism $h_{\mathsf{cf}}(\gamma')$, which we abbreviate with h_{cf}, as can be observed in the rules.

In the symbolic operational rules the evaluation of a navigation expression α is done by the definition of semantics $[\![\alpha]\!]$ given in Section 2.1 and already used for concrete semantics. This is possible because $\mathcal{H}_p$ has only L_p-canonical states (that are L_p-safe by definition) and therefore,

[6]Because of the application of the operations that manipulate the configurations.

(NEW$_{\text{error}}$-sym)	$\dfrac{[\![\alpha]\!] = \bot}{\mathsf{new}(\alpha.a), \gamma \rightarrow_{\varnothing} \mathsf{error}}$
(NEW-sym)	$\dfrac{[\![\alpha]\!] \neq \bot}{\mathsf{new}(\alpha.a), \gamma \rightarrow_{\lambda} \mathsf{skip}, \mathsf{cf}(add(\gamma, \alpha))} \quad \lambda = h_{\mathsf{cf}} \circ (\gamma \xRightarrow{id} add(\gamma, \alpha))$
(DEL$_{\text{error}}$-sym)	$\dfrac{[\![\alpha]\!] = \bot}{\mathsf{del}(\alpha.a), \gamma \rightarrow_{\varnothing} \mathsf{error}}$
(DEL-sym)	$\dfrac{[\![\alpha]\!] \neq \bot}{\mathsf{del}(\alpha.a), \gamma \rightarrow_{\lambda} \mathsf{skip}, \mathsf{cf}(cancel(\gamma, \alpha.a))} \quad \lambda = h_{\mathsf{cf}} \circ (\gamma \xRightarrow{id} cancel(\gamma, \alpha.a))$
(ASGN$_{\text{error}}$-sym)	$\dfrac{[\![\alpha_1]\!] = \bot}{\alpha_1.a := \alpha_2, \gamma \rightarrow_{\varnothing} \mathsf{error}}$
(ASGN-sym)	$\dfrac{[\![\alpha_1]\!] \neq \bot}{\alpha_1.a := \alpha_2, \gamma \rightarrow_{\lambda} \mathsf{skip}, \mathsf{cf}(\gamma'')} \quad (\gamma'', h) \in \mathsf{SExp}(modify(\gamma, \alpha_1, \alpha_2)) \quad \lambda = h_{\mathsf{cf}} \circ h^{-1} \circ (\gamma \xRightarrow{id} modify(\gamma, \alpha_1, \alpha_2))$
(IF$_1$-sym)	$\dfrac{\mathcal{V}(b, \gamma)}{\mathsf{if}\ b\ \mathsf{then}\ s_1\ \mathsf{else}\ s_2\ \mathsf{fi}, \gamma \rightarrow_{id} s_1, \gamma}$
(IF$_2$-sym)	$\dfrac{\neg\mathcal{V}(b, \gamma)}{\mathsf{if}\ b\ \mathsf{then}\ s_1\ \mathsf{else}\ s_2\ \mathsf{fi}, \gamma \rightarrow_{id} s_2, \gamma}$
(WHILE-sym)	$\dfrac{}{\mathsf{while}\ b\ \mathsf{do}\ s\ \mathsf{od}, \gamma \rightarrow_{id} \mathsf{if}\ b\ \mathsf{then}\ s; \mathsf{while}\ b\ \mathsf{do}\ s\ \mathsf{od}\ \mathsf{else}\ \mathsf{skip}\ \mathsf{fi}, \gamma}$
(SEQ$_{\text{error}}$-sym)	$\dfrac{s_1, \gamma \rightarrow \mathsf{error}}{s_1; s_2, \gamma \rightarrow_{\varnothing} \mathsf{error}}$
(SEQ$_1$-sym)	$\dfrac{s_1, \gamma \rightarrow_{\lambda} s_1', \gamma' \ \wedge\ s_1' \neq \mathsf{error}}{s_1; s_2, \gamma \rightarrow_{\lambda} s_1'; s_2, \gamma'}$
(SEQ$_2$-sym)	$\dfrac{}{\mathsf{skip}; s_2, \gamma \rightarrow_{id} s_2, \gamma}$
(PAR$_{\text{error}}$-sym)	$\dfrac{1 \leqslant j \leqslant k \ \wedge\ s_j, \gamma \rightarrow \mathsf{error}}{s_1 \parallel \cdots \parallel s_j \parallel \cdots \parallel s_k, \gamma \rightarrow_{\varnothing} \mathsf{error}}$
(PAR$_1$-sym)	$\dfrac{1 \leqslant j \leqslant k \ \wedge\ s_j, \gamma \rightarrow_{\lambda} s_j', \gamma' \ \wedge\ s_j' \neq \mathsf{error}}{s_1 \parallel \cdots \parallel s_j \parallel \cdots \parallel s_k, \gamma \rightarrow_{\lambda} s_1 \parallel \cdots \parallel s_j' \parallel \cdots \parallel s_k, \gamma'}$
(PAR$_2$-sym)	$\dfrac{}{\mathsf{skip} \parallel \cdots \parallel \mathsf{skip}, \gamma \rightarrow_{id} \mathsf{skip} \parallel \cdots \parallel \mathsf{skip}, \gamma}$
(ERROR)	$\dfrac{}{\mathsf{error} \rightarrow_{\varnothing} \mathsf{error}}$

Table 3: Operational rules for the symbolic semantics of $\mathcal{L}_n$.

the scope of any expression α is within the part of the configuration containing only concrete entities. Hence, for every state in a configuration γ_q of a state $q \in Q_{\mathcal{H}_p}$, the expression $[\![\alpha]\!]_{\mu_{\gamma_q},\vartheta}$ is well-defined.

- **Creation.** Performing $\mathsf{new}(\alpha.a)$ results in the canonical form of the configuration obtained by $add(\gamma, \alpha.)$. Corollary 6.21 and Lemma 6.22 ensure that this is a reallocation. Performing a $\mathsf{new}(\alpha.a)$ statement with α undefined results in a run-time error. As in the concrete semantics, this is modelled by the special state error (cf. ($\mathsf{NEW_{error}}$-sym)rule).
- **Deletion.** The rule (DEL-sym) deletes the entity denoted by $\alpha.a$ according to $cancel(\gamma, \alpha.a)$.
- **Assignment** As usual, if a null pointer is dereferenced, a run-time error is produced, cf. ($\mathsf{ASGN_{error}}$-sym) rule[7].

  Otherwise, (ASGN-sym) applies. Informally speaking, it employs the following strategy:

  1. First of all, the manipulation of pointers dictated by the assignment takes place according to $modify(\gamma, \alpha_1, \alpha_2)$.
  2. If the configuration $modify(\gamma, \alpha_1, \alpha_2)$ is unsafe, we consider its safe expansion.
  3. Every state having as a configuration the canonical form of a $\gamma'' \in \mathsf{SExp}(modify(\gamma, \alpha_1, \alpha_2))$ is a target state of the assignment rule.

  Safety can be lost only if in the remainder of the expression considered by the assignment there exists some unbounded entity. For example, consider the case where a variable w is assigned with some entity down in the same list pointed to by w (cf. Figure 16).

  If in the remainder of the expression we want to assign there are no unbounded (or multiple) entities, readjusting the pointers performed by $modify(\gamma, \alpha_1, \alpha_2)$ according to the assignment and taking the canonical form suffices, i.e., step 2 is not necessary.

  Rule (ASGN-sym) is *non-deterministic* if the set $\mathsf{SExp}(modify(\gamma, \alpha_1, \alpha_2))$ contains more than one configuration and they do not have the same canonical form. The existence of the canonical form $\mathsf{cf}(\gamma'')$ is guaranteed by Theorem 6.16. The definition of the reallocation λ can be understood by Figure 17. Configurations γ and $modify(\gamma, \alpha_1, \alpha_2)$ are related by an identity reallocation as stated by Lemma 6.22. λ reallocates entities $e \in E_\gamma$, $e' \in E_{\mathsf{cf}(\gamma'')}$ related by the morphisms h in $\mathsf{SExp}(modify(\gamma, \alpha_1, \alpha_2))$ and h_{cf}.

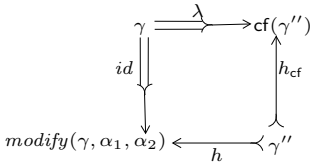

Figure 17: Correspondence between the source and the target state in the assignment rule.

Rules for conditional, while loop, sequential composition, parallel composition, and run-time error are similar to those in Table 2 and are listed here for completeness. For an explanation, we refer the reader to Section 6.2.

Example 6.24. Let $L = 3$ and $M = 2$. The actual transitions resulting from the assignment $w := w.a^3$ considered in Figure 16 are shown in Figure 18. $\gamma_{q_1}, \gamma_{q_2}, \gamma_{q_3}$ are the canonical form of the states in $\mathsf{SExp}(\gamma_{q'})$.

In Figure 19, we focus on the steps corresponding to the diagram in Figure 17 taken to obtain the transition $q \rightarrow_\lambda q_1$. Only the mapping of the unbounded entity is drawn. The mappings for the other entities are identities. State q'' is obtained by the morphism $h\downarrow^4$. The canonical form then reduces the chain of four elements in one with three elements. □

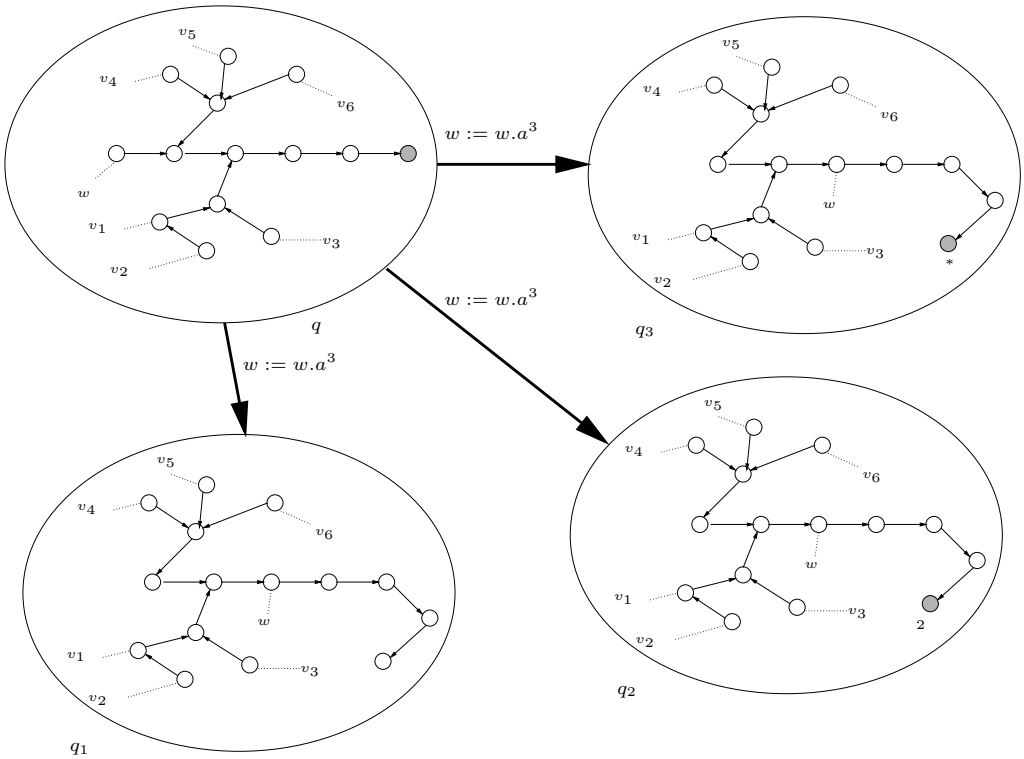

Figure 18: Non-deterministic step given by an unsafe assignment.

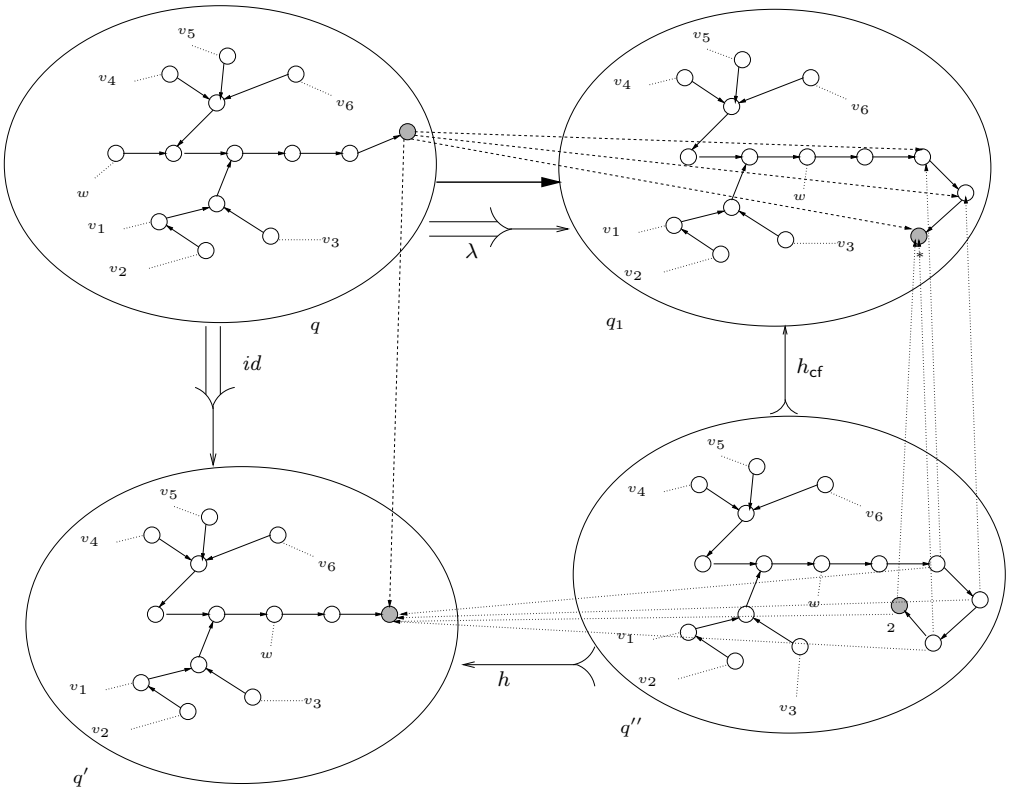

Figure 19: Zoom of the diagram in Fig. 17 applied to the assignment of Fig. 18.

The next lemma states a corollary of Proposition 6.20.

Lemma 6.25. λ defined in rule (ASGN-sym) is a reallocation.

Proof. See Appendix B. □

Proposition 6.26. If $q \in Q_{\mathcal{H}_p}$ then q is L-safe, PV-reachable and well-formed.

Proof. Straightforward by the definition of $Q_{\mathcal{H}_p}$. □

6.6 Relating the concrete and symbolic semantics

In this section, we study the relation between the two semantics we have defined for $\mathcal{L}_n$. The correspondence between $\mathcal{A}_p$ and $\mathcal{H}_p$ is stated by the following:

Theorem 6.27. For all programs $p : id(\mathcal{A}_p) \sqsubseteq \mathcal{H}_p$.

Proof. See Appendix B. □

The symbolic automaton simulates the concrete one, therefore the theory and results developed in Section 4 can be applied here to $\mathcal{A}_p$ and $\mathcal{H}_p$. In particular concerning the verification of $\mathcal{N}\!\mathit{all}$TL properties of the program p whose semantics is given by $\mathcal{A}_p$ and $\mathcal{H}_p$. In fact, as a straightforward consequence of the previous theorem we have:

Corollary 6.28. For every program p and every $\mathcal{N}\!\mathit{all}$TL-formula ϕ:

$$\phi \text{ is } \mathcal{H}_p\text{-valid} \;\Rightarrow\; \phi \text{ is } \mathcal{A}_p\text{-valid.}$$

Proof. Straightforward application of Theorem 6.27 and Propositions 4.7 and 4.11. □

Finally, the next result gives another important step towards the development of techniques for exhaustive state space verification for HABA.

Theorem 6.29. For all programs p, $\mathcal{H}_p$ is finite-state.

Proof. See Appendix B. □

On the refining of the model. The construction of the model $\mathcal{H}_p$ depends on the canonical form that in turn is parametric to the number L_p. The precision of the model may be increased by tuning opportunely L_p: in fact, by increasing L_p, the corresponding canonical form of the states will be more concrete. A second parameter that can be used to change the precision of the model is $\mathbb{M}$. The higher is $\mathcal{C}(\mathcal{H}_p)$ the more concrete is the model. Hence, a tool that extracts a model from p should be designed to provide the user with the capability to input L_p and M.

7 Model checking $\mathcal{N}\!\mathit{all}$TL

In this section, we define an algorithm for model-checking $\mathcal{N}\!\mathit{all}$TL formulae against a HABA with references. The algorithm is based on the one defined in [5, 6] that, in turn, extends the tableau-based method for LTL [9].

We evaluate $\mathcal{N}\!\mathit{all}$TL-formulae on states of a HABA by mapping the free variables of the formula to entities of the state. Again, these mappings are used to resolve all basic propositions like: the freshness predicate x new; the entity equation $x.a^n = y.a^m$; and the (new) leads-to propositions $x.a^n \rightsquigarrow y.a^m$. The same obstacles encountered for $\mathcal{A}\!\mathit{ll}$TL need to be addressed here, together with new difficulties proper of $\mathcal{N}\!\mathit{all}$TL. These can be summarised as follow.

[7] Concerning the possibility to detect a run-time error in case α_2 is an illegal expression, the same observation done for the concrete semantics holds also for the symbolic one (cf. foot note page 28).

- It is not always uniquely determined whether or not an entity is fresh in a state. Arriving in the states from different transitions an entity can be new or old depending whether or not it is in the codomain of the reallocation attached to that transition.

  As for $\mathcal{A}\ell\ell$TL, this obstacle is dealt with by the *duplication* of states defined in [5].

- For variables (of the formula we want to model-check) that are mapped onto unbounded entities, propositions like entity equation or leads-to cannot be decided since it is not clear in which instances of the unbounded entity the variables are interpreted.

  To deal with this difficulty, we define a notion of *distance* between the interpretation of the free variables of a formula. Since variables, say x and y, can be mapped onto the same unbounded entity e, the distance needs to be sensible to the level of the "instances" of e where x and y are precisely interpreted[8].

- For the formula ϕ we want to model-check, the HABA can be too abstract. In particular, the global constant M up-to which we have precise knowledge of the number of concrete entities an multiple entity represents can be too small with respect to the navigation expressions occurring in ϕ. When this is the case, it is not possible to decide equality propositions and the leads-to predicate.

  To overcome this problem typical of HABA with references and $\mathcal{N}\acute{a}\ell\ell$TL, we transform the model in an equivalent one where the global constant M is raised up to a suitable upper bound (dependent from ϕ) that provides correct information for the atomic propositions in ϕ. This transformation process is called *stretching*.

7.1 Stretching HABAs

In a HABA $\mathcal{H}$ an entity can have precise cardinality *only* up to the global constant M. In the process of model checking a formula ϕ, the abstraction imposed by M can result to be too strong. In fact, it is necessary to find suitable assignments for ϕ's variables in order to decide whether it cannot be satisfiable in $\mathcal{H}$. Thus there exists a dependency between the atomic propositions in ϕ and the constant M. In case M is too small, and therefore the representation of $\mathcal{H}$ is too abstract to support the definition of valuations (cf. Section 7.6), $\mathcal{H}$ must be unfolded, or as we say, stretched. The stretching process increases the precision of $\mathcal{H}$ states.

Example 7.1. Assume $M = 2$ and consider the state q depicted in Figure 20. Moreover, assume we want to decide if the formula $\phi \equiv x.a^5 = y$ holds in q. The truth value of ϕ changes depending on the number of entities actually represented by the unbounded entity e. Nevertheless, since $M = 2$, e can represents any number of instances strictly greater than 2. In particular, ϕ is true only in the case, e represents precisely 4 instances, and it is false in every other case. Hence, in q, ϕ can be both true or false. By stretching the model to a sufficient extent, we can avoid this kind of ambiguities (at the cost of nondeterminism). □

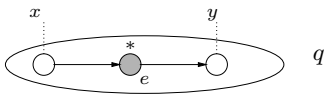

Figure 20: In q is the truth value of $x.a^5 = y$ ambiguous.

Stretching means augmenting the precision of the states by increasing the value of M. This yields for every state with an abstract configuration a set of states associated to more concrete configurations and represents the same original pointer structure (i.e., these configurations are related by morphisms). The first obvious question is, of course, how much concrete the resulting stretched HABA should be. Clearly, the risk we run is that the model must be magnified so much

[8] In this context, instances of e are synonym of the concrete entities that e abstractly represents.

to become infinite which, in the context of model checking, would mean to neutralise the complete system of abstraction built on unbounded entities. Fortunately, for a given formula ϕ we need only a bounded stretching, since after a certain point, more concrete HABAs would not provide any further useful information. For a formula ϕ this bound, written $K(\phi)$, is given by:

$$K(\phi) = \max(M+1, \sum_{x \in fbv(\phi)} \max\{n+1 \mid x.a^n \text{ occurs in } \phi\}). \tag{18}$$

where $fbv(\phi)$ is the set of free and bound variables of ϕ.

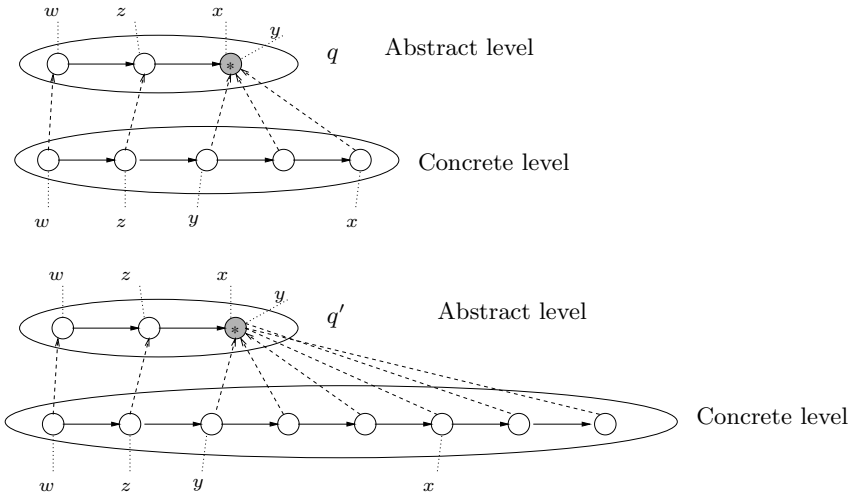

Figure 21: Example of suitable $K(\phi)$.

Example 7.2. Consider the formula $\phi_1 \equiv \exists x : \exists y : (z = x.a^3 \wedge w.a^2 = y)$, and the HABA state q depicted in Figure 21 (top) where a possible given interpretation of the variables in ϕ_1 is considered. In order to satisfy the formula, x and y must be interpreted as part of the unbounded entity. It is clear from the picture that we must consider the unbounded entity to represent *at least* 3 entities in order to find a suitable assignment for x and y that makes the formula true in the state. $K(\phi_1) = 9$ $(= 4+3+1+1)$ is obviously enough to decide the validity of ϕ_1.

Now consider $\phi_2 \equiv \exists x : \exists y : (z.a^5 = x.a^2 \wedge w = y.a^2)$ and state q' (cf. Figure 21, bottom part). In this case because of the given interpretation of the free variable z and the offset $.a^5$, it results that the unbounded entity must represent at least 6 entities in order to find a suitable assignment for the variable that makes ϕ_2 valid in the state. This explains why we need to sum up the expression related to free variables together with those related to bounded variables. Note that $K(\phi_2) = 13$ $(= 6+3+3+1)$. □

Definition 7.3 (HABA stretching). Let $\mathcal{H} = \langle X, Q, E, \rightarrow, I, \mathcal{F} \rangle$ be a HABA such that $\mathcal{C}(\mathcal{H}) = M$. The *stretching of* $\mathcal{H}$ *up to* $\hat{M}$ (with $M < \hat{M}$) is the HABA $\mathcal{H} \Uparrow \hat{M} = \langle X, Q', E', \rightarrow', I', \mathcal{F}' \rangle$ where for $q \in Q$ let $S_q = \{(q, E_q, \mu_q, \mathcal{C}) \mid \text{cod}(\mathcal{C}) \subseteq \hat{\mathbb{M}}^*, \mathcal{C}_q = \lceil \mathcal{C} \rceil_M\}$ then

- $Q' = \bigcup_{q \in Q} S_q$;
- $E'(q, \gamma) = \gamma$
- $\rightarrow'$ is the smallest relation such that:

$$\frac{q_1 \rightarrow_\lambda q_2 \ \wedge \ \gamma_1 \stackrel{\lambda_s}{\Longrightarrow} \gamma_2}{(q_1, \gamma_1) \rightarrow'_{\lambda_s} (q_2, \gamma_2)} \qquad \forall \lambda_s : \lambda = \lceil \lambda_s \rceil_M$$

- $I' = \bigcup_{q \in I} S_q$;
- $\mathcal{F}' = \{\bigcup_{q \in F} S_q \mid F \in \mathcal{F}\}$.

The automaton $\mathcal{H} \Uparrow \hat{M}$ includes for each state q of $\mathcal{H}$ the set S_q containing all the states obtained by assigning to entities with cardinality $*$, every cardinality in $\{M+1, \ldots, \hat{M}\} \cup \{*\}$. The transition relation $\rightarrow'$ is obtained from the original one by adding for every transition $q \rightarrow q'$, all possible transitions from states in S_q to state $S_{q'}$ taking care that there exists a reallocation λ_s between the configurations of the modified states. This constraint concerns the compatibility of the new cardinalities of their entities. The initial and accept states of $\mathcal{H} \Uparrow \hat{M}$ are those corresponding to initial and accept states of $\mathcal{H}$.

Example 7.4. Consider the HABA depicted in Figure 22, such that $\mathcal{C}(\mathcal{H}) = 2$. The stretching up to 4 is shown in Figure 23. In this example we have:

$$\begin{aligned} S_q &= \{q_1, q_2, q_3\} \\ S_{q'} &= \{q'_1, q'_2, q'_3\}. \end{aligned}$$

Between q_1 and q'_1 there are no transitions since $\lambda(e_2, e_4) = *$, therefore, every suitable $\lambda_s(e_2, e_4) \in \{3, 4, *\}$, but since $\mathcal{C}_{q_1}(e_2) = 3$ and $\lambda(e_2, e_5) \neq 0$ (i.e., $\mathcal{C}_{q_1}(e_2)$ is redistributed between e_4 and e_5) it is clear that there cannot be a reallocation $q_1 \overset{\lambda_s}{\Longrightarrow} q'_1$. On the contrary, q_3 and q'_3 have the configurations as q and q', respectively. However, between q_3 and q'_3 there exist two transitions corresponding to the reallocations that assign to the pair (e_2, e_4) the multiplicity 4 and $*$.

Consider now, the formula $\phi \equiv x.a^5 = y$ where x and y in q have the following interpretation: $\theta(x) = e_1$ and $\theta(y) = e_3$. In Example 7.2 we have seen that in q (for $M = 2$) ϕ can be either true of false. In $\mathcal{H} \Uparrow 4$, the ambiguity is resolved. ϕ holds only in q_2 and it is false in q_1 and q_3 since in the latter e_2 represents at least 5 entities. □

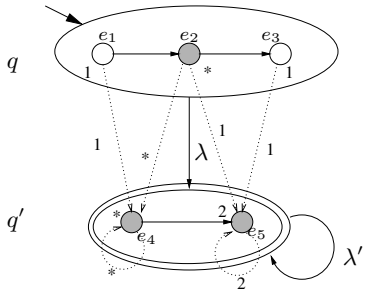

Figure 22: The HABA $\mathcal{H}$ with $\mathcal{C}(\mathcal{H}) = 2$.

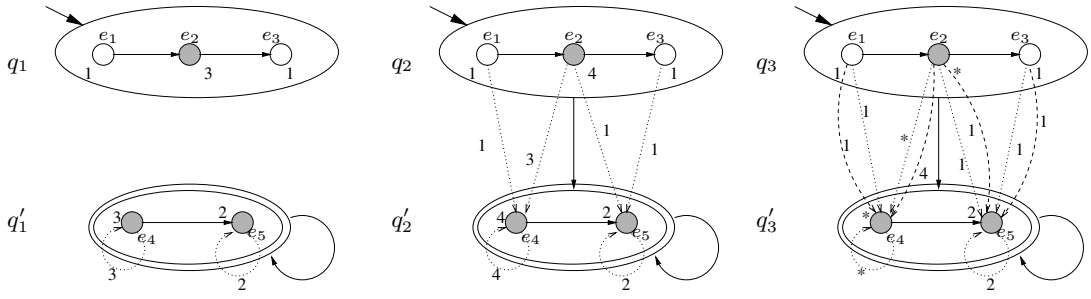

Figure 23: The HABA $\mathcal{H} \Uparrow 4$.

Theorem 7.5. For all HABA $\mathcal{H}$ such that $\mathcal{C}(\mathcal{H}) = M < \hat{M}$: $\mathcal{L}(\mathcal{H}) = \mathcal{L}(\mathcal{H} \Uparrow \hat{M})$.

Proof. See Appendix C. □

Assumptions. In the remainder of this section, we assume that the necessary duplication as well as stretching have been carried out already: that is, we will assume that a state $q \in Q$ has an extra component $N_q \subseteq E_q$ that contains the entities that are *new in* q, and that the global constant M equals $K(\phi) - 1$. Other assumptions needed below are that every quantified variable is different (i.e., the formula has been α-converted) and every quantified variable actually appears free in the sub-formula; that is, we only consider formulae $\exists x.\phi$ for which $x \in fv(\phi)$. Note that this imposes no real restriction, since $\exists x.\phi$ is equivalent to $\exists x.(x \;\mathsf{alive} \wedge \phi)$.

7.2 Valuations

In a given state, a valuation of a $\mathcal{N}all\ell$TL formula is an interpretation of its free logical variables as entities of the state. Such an interpretation is meant to establish the validity of the atomic propositions within the formula, which in $\mathcal{N}all\ell$TL have the form $\alpha_1 = \alpha_2$, $\alpha\;\mathsf{new}$, and $\alpha_1 \rightsquigarrow \alpha_2$. Because we allow unbounded and multiple entities as interpretations of logical variables, a simple mapping from variables to entities would not be enough to decide these atomic propositions. The interpretation of the valuations should be able to express not only whether two variables mapped onto an unbounded entity refer to *the same instance* or not. In fact, in the latter case, it should provide us with information about the *distance* between the two instances (within the unbounded/multiple entity) where the variables are supposed to be interpreted. Such information is necessary in order to decide equality propositions (e.g. $x.a^2 = y.a^5$) as well as leads-to propositions (e.g. $x.a^2 \rightsquigarrow y.a^5$).

For a set of entities E let $E^-, E^+ \subseteq \text{LVar}$ be two special sets of logical variables defined as:

$$\begin{aligned} E^- &= \{e^- \mid e \in E\} \\ E^+ &= \{e^+ \mid e \in E\}. \end{aligned}$$

In the following let $E^\pm = E^- \cup E^+$. For these special variables, we will have always a fixed interpretation: variable e^- is interpreted in the first instance of the entities represented by e, and e^+ is interpreted in the last instance of e.

Definition 7.6 (Valuations). Let γ be a configuration. A γ-*valuation* is a tuple (ψ, Θ, δ) where

- ψ is a $\mathcal{N}all\ell$TL-formula;
- Θ:$fv(\psi) \cup E^\pm_\gamma \rightharpoonup E_\gamma$ a partial function mapping every free variable to an entity such that $\forall e \in E^\pm_\gamma : \Theta(e^-) = \Theta(e^+) = e$.
- δ:$(fv(\psi) \cup E^\pm_\gamma) \times (fv(\psi) \cup E^\pm_\gamma) \rightharpoonup \mathbb{M}^*$, a partial function that satisfies the conditions in Table 4.

We denote by V_γ the set of *all* γ-valuations ranged over by v and we write V_q for V_{γ_q}.

(Delta Gamma 1)	$\forall e \in E_\gamma : \delta(e^-, e^+) \oplus 1 = \mathcal{C}_\gamma(e)$
(Delta Gamma 2)	$\forall e_1, e_2 \in E_\gamma : (e_1 \prec_\gamma e_2 \;\Leftrightarrow\; \delta(e_1^+, e_2^-) = 1)$
(Delta Theta 1)	$\forall x, y \in fv(\psi) : (x, y) \in \text{dom}(\delta) \;\Rightarrow\; x, y \in \text{dom}(\Theta) \wedge \Theta(x) \preceq^*_\gamma \Theta(y)$
(Delta Theta 2)	$\Theta(x) = e \;\Leftrightarrow\; \delta(e^-, x) \geqslant 0 \wedge \delta(x, e^+) \geqslant 0$
(Delta Met 1)	$(x, x) \in \text{dom}(\delta) \;\Rightarrow\; \delta(x, x) = 0$
(Delta Met 2)	$(x, y), (y, z) \in \text{dom}(\delta) \;\Rightarrow\; (x, z) \in \text{dom}(\delta)$
(Delta Met 3)	$(x, y), (x, z) \in \text{dom}(\delta) \Rightarrow (\delta(x, y) \oplus \delta(y, z) = \delta(x, z)) \vee (\delta(x, z) \oplus \delta(z, y) = \delta(x, y))$

Table 4: Conditions on a valuation (ψ, Θ, δ).

The function δ is the notion of distance that is used in a valuation together with the interpretation Θ to decide the equality and leads-to propositions. Some comments on the conditions

reported in Table 4 are now in order. Condition (Delta Gamma 1) enforces the consistency between δ and the cardinality function $\mathcal{C}_\gamma$ of the configuration. In particular it implies that $\{(e^-, e^+) \mid e \in E^{\pm}\} \subseteq \mathrm{dom}(\delta)$. Condition (Delta Gamma 2) gives consistency between δ and μ_γ. Condition (Delta Theta 1) and (Delta Theta 2) relate Θ and δ. (Delta Theta 1) can be rephrased saying that the distance between two variables is defined only if the variables are interpreted in reachable entities. The other direction is also a reasonable property to require. However, it is not necessary to impose it since it can be derived by the other properties as stated by the following result.

Proposition 7.7. For all γ-valuations (ψ, Θ, δ):

(a) $(\exists j > 0 : \mu^j \circ \Theta(x) = \Theta(y) \neq \bot) \;\Rightarrow\; (x, y) \in \mathrm{dom}(\delta)$

(b) $\Theta(x) = \Theta(y) \neq \bot \;\Rightarrow\; (x, y) \in \mathrm{dom}(\delta) \vee (y, x) \in \mathrm{dom}(\delta)$

Proof. See Appendix C. □

Corollary 7.8. For all γ-valuations $(\psi, \Theta, \delta) : x \in \mathrm{dom}(\Theta) \;\Rightarrow\; (x, x) \in \mathrm{dom}(\delta)$.

Proof. Straightforward from Proposition 7.7. □

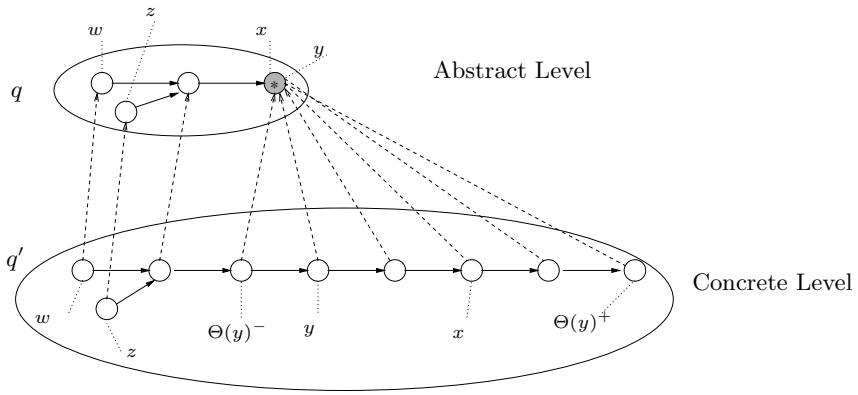

Figure 24: An interpretation of the variables corresponding to a valuation.

Example 7.9. Consider Figure 24. Assume that in a valuation, Θ and δ are defined as follows:

$$\begin{array}{ll} \delta(\Theta(y)^-, y) = 1 & \delta(\Theta(y)^-, x) = 3 \\ \delta(y, \Theta(y)^+) = * & \delta(x, \Theta(y)^+) = 2 \\ \delta(y, x) = 2 & \delta(x, y) = \bot \\ \delta(w, \Theta(y)^-) = 2 & \delta(z, \Theta(y)^-) = 2 \end{array}$$

And $\delta(w, x) = \delta(z, x) = \delta(w, \Theta(y)^+) = \delta(z, \Theta(y)^+) = *$. Note that, since $\Theta(w)$ and $\Theta(z)$ are unreachable we have $\delta(w, z) = \delta(z, w) = \bot$. On the concrete level, for a state where the unbounded entity is instantiated by six concrete entities, the interpretation given by the valuation corresponds to evaluate the free variables as depicted in the concrete state q'. □

Metrics versus δ. Since δ expresses our notion of distance, it is somehow natural as well as interesting to relate this concept to metrics in the context of metric spaces. For, let us deviate a bit from the exposition of the model checking algorithm and make a short comparison.

A *metric space* is a set S with an associated function $m : S \times S \to \mathbb{R}_{\geqslant 0}$ (called metric) satisfying the following properties:

1. $\forall x \in S : m(x, x) = 0$

2. $\forall x, y \in S : m(x, y) = 0 \;\Rightarrow\; x = y$

3. $\forall x, y \in S : m(x, y) = m(y, x)$

4. $\forall x, y, z \in S : m(x, z) \leqslant m(x, y) + m(y, z)$

By condition 2, the distance between two points is zero only if these two points are actually the same. Condition 3 is the symmetric law and 4 is the well-known triangle inequality. Different choices of metric on a given set give rise to different metric spaces. If condition 2 is dropped, a *pseudo metric* space is obtained, whereas without condition 3, we would obtain a so-called *quasi metric* space [15]. As defined in Definition 7.6, the partial function δ clearly satisfies postulate 1 of a metric by (Delta Met 1). Moreover, from condition (Delta Met 1) – (Delta Met 3) it is possible to derive postulate 4 as stated by:

Proposition 7.10 (triangle inequality). $(x, y), (y, z) \in \mathrm{dom}(\delta) \Rightarrow \delta(x, z) \leqslant \delta(x, y) \oplus \delta(y, z)$.

Proof. $(x, y), (y, z) \in \mathrm{dom}(\delta)$ implies by (Delta Met 2) $(x, z) \in \mathrm{dom}(\delta)$. Then, by (Delta Met 3) we have

$$\delta(x, y) \oplus \delta(y, z) = \delta(x, z) \quad \text{or} \tag{19}$$
$$\delta(x, z) \oplus \delta(z, y) = \delta(x, y) \tag{20}$$

If (19) holds then the statement of the proposition holds as well. If (20) holds then we have

$$\delta(x, y) \oplus \delta(y, z) = \delta(x, z) \oplus \delta(z, y) \oplus \delta(y, z) \geqslant \delta(x, z)$$

that is what we wanted to prove. □

Hence, δ, where defined, satisfies condition 1 and 4. By the consideration described above we could then classify δ as a "partial pseudo-quasi metric".

Abstract semantics and distance for navigation expressions. We can adapt the definition of $[\![\alpha]\!]$ given in Section 2 in order to exploit the information given by a valuation. First of all let us consider an example that shows some difficulties towards such definition.

Example 7.11. Consider the configuration γ represented in Figure 25 and a γ-valuation v were $\Theta_v(x) = e_1$ and $\Theta_v(y) = e_{10}$ and δ_v is given by:

$$\begin{array}{ll}
\delta_v(x, e_1^-) = 0 & \delta_v(x, e_1^+) = 0 \\
\delta_v(x, e_2^-) = 1 & \delta_v(x, e_2^+) = 1 \\
\delta_v(x, e_3^-) = 2 & \delta_v(x, e_3^+) = 4 \\
\delta_v(x, e_4^-) = 5 & \delta_v(x, e_4^+) = 6 \\
\delta_v(x, e_5^-) = 7 & \delta_v(x, e_5^+) = 7 \\
\delta_v(x, e_6^-) = 8 & \delta_v(x, e_6^+) = 9 \\
\delta_v(x, e_7^-) = 10 & \delta_v(x, e_7^+) = 10.
\end{array}$$

The natural way to define the entity corresponding to expression $x.a^n$ would be to exploit δ_v by taking the entity e such that $\delta_v(x, e^-) \leqslant n \leqslant \delta_v(x, e^+)$. Hence, for example, the interpretation of $x.a^6$ should be e_4 whereas $x.a^8$ should be interpreted onto e_6. However, this is too naive. In fact, consider the expressions $x.a^{14}$ or $x.a^{25}$, it is clear that the approach described before does not work since there are no entities with a distance more than 10 from x, and nevertheless looking at the configuration, both expressions must point to an existing entity. □

It is not difficult to see that the problem described in the previous example stems from the existence of a cycle. In order to deal with it we introduce the following notion. For a configuration $\gamma = (E, \mu, \mathcal{C})$ and a γ-valuation v let $\mathcal{C}_v : (\mathrm{LVar} \times \mathbb{N}) \rightharpoonup \mathbb{M}^*$ given by:

$$\mathcal{C}_v(x, k) = \delta_v(x, \Theta_v(x)^+) \oplus \textstyle\sum_{1 \leqslant i \leqslant k} \mathcal{C}(\mu^i(\Theta_v(x))).$$

Informally, $\mathcal{C}_v(x,k)$ returns the cumulative cardinality obtained performing k steps from the interpretation of x in v, i.e, $\Theta_v(x)$ up to $\mu^k(\Theta(x))$ and summing up all the cardinalities of the entities encountered in between. Note that if there exists a cycle some cardinality may be considered more than once.

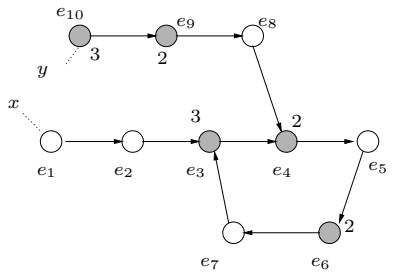

Figure 25: A cycle induces a difference between distance and cumulative cardinality.

Example 7.12. Consider again the configuration γ in Figure 25. We have:

$$\begin{array}{lll} \mathcal{C}_v(x,1)=1 & \mathcal{C}_v(x,2)=4 & \mathcal{C}_v(x,3)=6 \\ \mathcal{C}_v(x,4)=7 & \mathcal{C}_v(x,5)=9 & \mathcal{C}_v(x,6)=10 \\ \mathcal{C}_v(x,7)=13 & \mathcal{C}_v(x,8)=15 & \mathcal{C}_v(x,9)=16\ldots \end{array}$$

Looking at the previous values we can see that performing 7 steps from $\Theta(x)$, at the concrete level, we traverse 13 entities, whereas after 8 steps at the concrete level we visit 15 entities. Therefore, at this point it should be obvious that the interpretation of $x.a^{14}$, should be the entity $\mu^8(\Theta(x))$. □

Definition 7.13. Let $\gamma = (E, \mu, \mathcal{C})$ be a configuration and $v = (\phi, \Theta, \delta) \in V_\gamma$. The abstract semantics of the navigation expression is the function $[\![\cdot]\!]_{\gamma,v} : Nav \to (Ent^\perp \times (\mathbb{M}^* \cup \{0\}))$ given by:

$$[\![nil]\!]_{\gamma,v} = (\perp, 0)$$

$$[\![x.a^n]\!]_{\gamma,v} = \begin{cases} (\Theta(x), \delta(\Theta(x)^-, x) \oplus n) & \text{if } \delta(x, \Theta(x)^+) \geqslant n \\ (\mu^j \circ \Theta(x), n - \mathcal{C}_v(x, j-1) - 1) & \text{if } \delta(x, \Theta(x)^+) < n \wedge j = \min\{k > 0 \mid \mathcal{C}_v(x,k) \geqslant n\} \\ (\perp, 0) & \text{otherwise} \end{cases}$$

The abstract semantics of the navigation expression $x.a^n$ is defined as a pair (e,k) where e is either the (abstract) entity in which $x.a^n$ is actually interpreted or $\perp$ in case the $x.a^n$ dereferences a null pointer. The second component $k \in \{0, \ldots M, *\}$ represents the offset of the instance corresponding to $x.a^n$ with respect to the first instance in e. If $e = \perp$ then k is set to 0. Note that the value $n - \mathcal{C}(x, j-1) - 1$ can be computer since $n < K(\phi)$ and it is not $*$. As expected the semantics of *nil* does not denote any entity and therefore is $(\perp, 0)$. As a notation we write $[\![\alpha]\!]^1$ and $[\![\alpha]\!]^2$ for the first and the second component of $[\![\alpha]\!]$, respectively. Moreover, $[\![\alpha_1]\!] = [\![\alpha_2]\!]$ stands for $[\![\alpha_1]\!]^1 = [\![\alpha_2]\!]^1 \wedge [\![\alpha_1]\!]^2 = [\![\alpha_2]\!]^2$. Finally, we write $[\![\alpha]\!] = \perp$ and $[\![\alpha]\!] \neq \perp$ for $[\![\alpha]\!]^1 = \perp$ and $[\![\alpha]\!]^1 \neq \perp$ respectively.

Example 7.14. Using the cumulative cardinality computed in the previous example, we have $[\![x.a^3]\!]_{\gamma,v} = (\mu^2 \circ \Theta(x), 3 - \mathcal{C}_v(x,1) - 1) = (e_3, 1)$. □

In the definition of atomic valuations (see Definition 7.15) it is useful to refer to a notion of distance of two navigation expressions α_1 and α_2 in the context of some valuation. To be more specific we do not need the precise distance between α_1 and α_2, but it is enough to know whether the distance is undefined (i.e., α_2 is unreachable from α_1), zero (i.e., α_1 and α_2 denotes the same instance of the same entity), or positive (i.e., from α_1 it is possible to reach α_2). These values are precisely what is needed to decide atomic propositions such as equations like $\alpha_1 = \alpha_2$ and leads-to

propositions like $\alpha_1 \rightsquigarrow \alpha_2$. Thus, we define a three-valued function Δ that given two navigation expressions returns undefined ($\bot$), zero, or strictly positive ($\top$) depending on the distance between the two expressions.

For a configuration γ and a $v \in V_\gamma$, the function $\Delta_{\gamma,v} : Nav \times Nav \to \{\bot, 0, \top\}$ is given by:

$$\Delta_{\gamma,v}(nil, nil) = 0$$

$$\Delta_{\gamma,v}(x.a^n, nil) = \begin{cases} 0 & \text{if } [\![x.a^n]\!]^1_{\gamma,v} = \bot \\ \top & \text{if } \exists j > 0 : \mu^j([\![x.a^n]\!]^1_{\gamma,v}) = \bot \\ \bot & \text{otherwise} \end{cases}$$

$$\Delta_{\gamma,v}(nil, x.a^n) = \begin{cases} 0 & \text{if } [\![x.a^n]\!]^1_{\gamma,v} = \bot \\ \bot & \text{otherwise} \end{cases}$$

$$\Delta_{\gamma,v}(x.a^n, y.a^m) = \begin{cases} 0 & \text{if } [\![x.a^n]\!]^1_{\gamma,v} = [\![y.a^m]\!]^1_{\gamma,v} = \bot \\ 0 & \text{if } [\![x.a^n]\!]_{\gamma,v} = [\![y.a^m]\!]_{\gamma,v} \neq \bot \text{ and } [\![x.a^n]\!]^2_{\gamma,v} \neq * \\ 0 & \text{if } [\![x.a^n]\!]_{\gamma,v} = [\![y.a^m]\!]_{\gamma,v} \neq \bot \text{ and } [\![x.a^n]\!]^2_{\gamma,v} = * \text{ and} \\ & \quad (\delta(x,y) \oplus m = n \vee \delta(y,x) \oplus n = m)) \\ \top & \text{if } [\![x.a^n]\!]^1_{\gamma,v} \neq [\![y.a^m]\!]^1_{\gamma,v} \text{ and } \exists j > 0 : \mu^j([\![x.a^n]\!]^1_{\gamma,v}) = [\![y.a^m]\!]^1_{\gamma,v} \\ \top & \text{if } [\![x.a^n]\!]^1_{\gamma,v} = [\![y.a^m]\!]^1_{\gamma,v} \neq \bot \text{ and } [\![x.a^n]\!]^2_{\gamma,v} < [\![y.a^m]\!]^2_{\gamma,v} \\ \top & \text{if } [\![x.a^n]\!]^1_{\gamma,v} = [\![y.a^m]\!]^1_{\gamma,v} \neq \bot \text{ and } [\![x.a^n]\!]^2_{\gamma,v} = [\![y.a^m]\!]^2_{\gamma,v} = * \text{ and} \\ & \quad (\delta(x,y) \oplus m > n \ \vee \ \delta(y,x) \oplus n < m) \\ \bot & \text{otherwise} \end{cases}$$

The distance between *nil* and itself is 0 by definition. The distance between an expression $x.a^n$ and *nil* is 0 if $x.a^n$ refers to a null pointer and $\top$ if $x.a^n$ reaches a null pointer. The symmetric case $\Delta_{\gamma,v}(nil, x.a^n)$ is defined only if $x.a^n$ denotes a null pointer and undefined otherwise. This is because *nil* cannot lead to any special entity. The more complex case is $\Delta_{\gamma,v}(x.a^n, y.a^m)$. In particular by definition the distance between $x.a^n$ and $y.a^m$ is 0 if either both expressions dereference a null pointer, or they have the same semantics (defined in both components). However, in this last case, special attention must be devoted to the case $[\![x.a^n]\!]^2_{\gamma,v} = *$. In fact, this means that both $x.a^n$ and $y.a^m$ denote something beyond our range of precision, therefore it is not clear whether the expressions denote the same instance or not. By definition of $K(\phi)$, $[\![x.a^n]\!]^2_{\gamma,v} = *$ can only happen when x and y are interpreted in the same entity as $x.a^n$ and $y.a^m$ (recall that $n, m < K(\phi)$). Thus, to solve this ambiguity we exploit the distance $\delta(x, y)$. It is straightforward to see that $x.a^n$ and $y.a^m$ point to the same instance (of the entity) if $\delta(x, y) \oplus m = n$ or $\delta(y, x) \oplus n = m$ depending whether the interpretation of x precedes the interpretation of y or vice-versa. Finally, the distance $\Delta_{\gamma,v}(x.a^n, y.a^m)$ is positive, either when the entity where $x.a^n$ is interpreted reaches the interpretation of $y.a^m$ after $j > 0$ steps (provided $[\![x.a^n]\!]^1_{\gamma,v} \neq [\![y.a^m]\!]^1_{\gamma,v}$). Otherwise, when $x.a^n$ and $y.a^m$ are interpreted in the same entity but the offset (from the first instance of the interpretation) of $x.a^n$ is smaller than the offset of $y.a^m$. Again, as discussed above, if both offsets are $*$ then a case distinction is needed according whether (x, y) or (y, x) belongs to $\mathrm{dom}(\delta)$.

Having at our disposal the functions $[\![\cdot]\!]_{\gamma,v}$ and $\Delta_{\gamma,v}$, we can introduce the *atomic proposition valuations* of a state q that are those q-valuations of basic propositions of $\mathcal{N}\!a\ell\ell$TL (i.e., freshness predicates, entity equations and leads-to predicates) that make the corresponding propositions true.

Definition 7.15. Let $\mathcal{H}$ be a HABA and let $q \in Q_\mathcal{H}$. The *atomic proposition valuations* of q are defined by the set $AV_q \subseteq V_q$ of all $v = (\phi, \Theta, \delta)$ such that:

- $\phi = \mathsf{tt}$;

- $\phi = (\alpha\ \mathsf{new})$ and $[\![\alpha]\!]^1_{q_\gamma,v} \in N_q$;
- $\phi = (\alpha_1 = \alpha_2)$, and $\Delta_{q_\gamma,v}(\alpha_1,\alpha_2) = 0$.
- $\phi = \alpha_1 \rightsquigarrow \alpha_2$, and $(\Delta_{q_\gamma,v}(\alpha_1,\alpha_2) = 0$ or $\Delta_{q_\gamma,v}(\alpha_1,\alpha_2) = \top)$

An atomic valuation with $\phi = \alpha\ \mathsf{new}$ must interpret α among the set of new entities in the state, i.e., N_q. For the equality proposition $\alpha_1 = \alpha_2$, the distance between α_1 and α_2 must be 0. For the leads-to proposition $\alpha_1 \rightsquigarrow \alpha_2$ the distance between α_1 and α_2 must be either 0 or positive ($\top$).

Model-checking $\mathcal{L}_n$ models. When we want to model check the HABA $\mathcal{H}_p$ representing the semantics of a program $p \equiv \mathsf{decl}\ v_1,\ldots,v_n : (s_1 \parallel \cdots \parallel s_k) \in \mathcal{L}_n$, we should make sure that the interpretation Θ in the valuations maps the special (free) logical variables $\mathit{DeclLVar} = \{x_{v_1},\ldots,x_{v_n}\}$ onto the special entities representing program variables, i.e., $\mathit{DeclPVar} = \{e_{v_1},\ldots,e_{v_n}\}$ according to the strategy described in Section 5.2. That is:

$$\Theta \restriction \mathit{DeclLVar} = \vartheta$$

where ϑ is the fixed special interpretation that links $\mathit{DeclLVar}$ to $\mathit{DeclPVar}$ defined by (6).

The next is the standard definition of the closure of a formula ϕ.

Definition 7.16. Let ϕ be an $\mathcal{N}\!\mathit{all}$TL-formula. The *closure* of ϕ, $CL(\phi)$, is the smallest set of formulae (identifying $\neg\neg\psi$ with ψ) such that:

- $\phi, \mathsf{tt}, \mathsf{ff} \in CL(\phi)$;
- $\neg\psi \in CL(\phi)$ iff $\psi \in CL(\phi)$;
- if $\psi_1 \vee \psi_2 \in CL(\phi)$ then $\psi_1, \psi_2 \in CL(\phi)$;
- if $\exists x.\psi \in CL(\phi)$ then $\psi \in CL(\phi)$;
- if $\mathsf{X}\psi \in CL(\phi)$ then $\psi \in CL(\phi)$;
- if $\neg\mathsf{X}\psi \in CL(\phi)$ then $\mathsf{X}\neg\psi \in CL(\phi)$;
- if $\psi_1\,\mathsf{U}\,\psi_2 \in CL(\phi)$ then $\psi_1, \psi_2, \mathsf{X}(\psi_1\,\mathsf{U}\,\psi_2) \in CL(\phi)$.

7.3 Tableau-graph for $\mathcal{N}\!\mathit{all}$TL

In the rest of the paper, for $\psi \in CL(\phi)$ let

$$\begin{aligned} \Theta \restriction \psi &= \Theta \restriction \mathit{fv}(\psi) \\ \delta \restriction \psi &= \delta \restriction (\mathit{fv}(\psi) \cup E^{\pm} \times \mathit{fv}(\psi) \cup E^{\pm}). \end{aligned}$$

We now construct a graph that will be the basis of the model-checking algorithm. The nodes of this graph are called atoms and are built from states of a HABA and valuations of formulae from the closure.

Definition 7.17 (atom). Given a HABA $\mathcal{H}$ and an $\mathcal{N}\!\mathit{all}$TL-formula ϕ, an *atom* is a pair (q, D) where $q \in Q_{\mathcal{H}}$, $D \subseteq \{(\psi,\Theta,\delta) \in V_q \mid \psi \in CL(\phi)\}$ such that for all $v = (\psi,\Theta,\delta) \in V_q$ with $\psi \in CL(\phi)$:

- $AV_q \subseteq D$;
- if $\psi = \neg\psi'$, then $v \in D$ iff $(\psi',\Theta,\delta) \notin D$;
- if $\psi = \psi_1 \vee \psi_2$, then $v \in D$ iff $(\psi_i, \Theta \restriction \psi_i, \delta \restriction \psi_i) \in D$ for $i = 1$ or $i = 2$;
- if $\psi = \exists x.\psi'$, then $v \in D$ iff there exists a $(\psi',\Theta',\delta') \in D$ such that $\Theta = \Theta' \restriction \psi$, $\delta = \delta' \restriction \psi$, and $\Theta'(x) \neq \bot$;
- if $\psi = \neg\mathsf{X}\psi'$, then $v \in D$ iff $(\mathsf{X}\neg\psi', \Theta, \delta) \in D$;

- if $\psi = \psi_1 \mathsf{U} \psi_2$, then $v \in D$ iff either $(\psi_2, \Theta \restriction \psi_2, \delta \restriction \psi_2) \in D$, or both $(\psi_1, \Theta \restriction \psi_1, \delta \restriction \psi_1) \in D$ and $(\mathsf{X}\psi, \Theta, \delta) \in D$.

The set of all atoms for a given formula ϕ constructed for HABA $\mathcal{H}$ is denoted $A_{\mathcal{H}}(\phi)$, ranged over by A, B. We denote the components of an atom A by (q_A, D_A).

In order to define the transitions between atoms in $A_{\mathcal{H}}(\phi)$, we need to define a notion of correspondence between some components of valuations of the source atom and valuations of the target atom. This correspondence must be such that the resulting graph is *sound* with respect to the semantics of the formulae and the semantics of the underlying HABA. First we need some intermediate definitions.

Auxiliary notation. For sets S_1 and S_2 the *disjoint union* is $S_1 \uplus S_2 = (S_1 \times \{1\}) \cup (S_2 \times \{2\})$. A relation $\mathcal{R}$ on sets S_1 and S_2 can be seen as a directed graph $G_{\mathcal{R}} = (S_1 \uplus S_2, \mathcal{R})$ with $S_1 \uplus S_2$ as a set of nodes and $\mathcal{R}$ as a set of arcs. For such a graph there are two injective functions $\iota_1 : 2^{(S_1 \uplus S_2)} \to 2^{S_1}$ and $\iota_2 : 2^{(S_1 \uplus S_2)} \to 2^{S_2}$ that project a subset of nodes $S \subseteq S_1 \uplus S_2$, onto S_1 and S_2:

$$\begin{aligned} \iota_1(S) &= \{a \mid (a, 1) \in S\} \\ \iota_2(S) &= \{a \mid (a, 2) \in S\}. \end{aligned}$$

An undirected graph is called *connected* if each node is reachable from any other node. A directed graph is called *weakly connected* if neglecting the direction of the arcs (i.e., treating the graph as an undirected one) it is connected. A connected subgraph is called *maximal* if it is not a proper subgraph of any other connected subgraph.

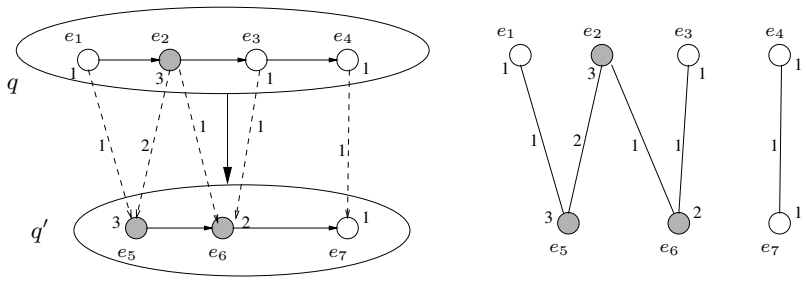

Figure 26: Weakly connected graphs defined by a reallocation.

Example 7.18. Consider the transitions $q \rightarrow_\lambda q'$ in Figure 26 (left). Consider the graph depicted in the right part of the same figure. It is obtained from the reallocation λ as relation between the sets of entities E_q and $E_{q'}$. In this case we have: sets $\{e_1, e_5\}$, $\{e_2, e_5\}$, $\{e_2, e_6\}$, $\{e_3, e_6\}$, $\{e_1, e_2, e_5\}$, $\{e_2, e_3, e_6\}$ are (weakly) connected. Sets $\{e_4, e_7\}$ and $\{e_1, e_2, e_3, e_5, e_6\}$ are (weakly) maximal connected. An example of non-connected set is $\{e_1, e_3, e_5, e_6\}$. □

Proposition 7.19. For $q_1 \stackrel{\lambda}{\Longrightarrow} q_2$, if E is a maximal weakly connected subgraph of $(E_{q_1} \uplus E_{q_2}, \lambda)$ such that $\perp \notin E$, then $\mathcal{C}_{q_1}(\iota_1(E)) = \mathcal{C}_{q_2}(\iota_2(E))$.

Proof. See Appendix C. □

This proposition states that in a reallocation λ, the consistency on cardinalities[9] is not limited to an entity e and its images (via λ): it naturally extends to the level of weakly maximal connected subgraphs derived by λ. For example, in Figure 26, there exists a correspondence between the global weight of $\{e_1, e_2, e_3\}$ and $\{e_5, e_6\}$ that are projections of a maximal weakly connected subgraph. The same consistency criterion holds for $\{e_4\}$ and $\{e_7\}$ since $\{e_4, e_7\}$ is maximal connected.

[9] $\perp$ is excluded since it is not a proper entity and it does not have a cardinality. Therefore it is meaningless to require any kind of consistency.

Atom reallocations. We need now to define the notion of arcs between atoms. They will be of the form:

$$(q, D) \rightarrow_{\lambda} (q', D'). \tag{21}$$

In defining this transition relation, however, few aspects deserve special attention. More specifically, there exist some dependencies between D and D' in case the former contains valuations with formulae involving a next operator, i.e., of the form $\mathsf{X}\psi$. In fact, such formulae express properties that are not confined to the current atom A but, on the contrary, refer to every atom connected to A by a transition. Therefore, if $(\mathsf{X}\psi, \Theta, \delta) \in D$ and $(\psi, \Theta', \delta') \in D'$ we have the following dependencies:

- the interpretation given by Θ' must agree with the interpretation Θ according to the reallocation λ;
- the distance given by δ' must agree to the distance δ at least for those entities that are "invariant" under the transition.

The correspondence between these components is not trivial. For example, it is clear that we would like to have $\Theta'(x) = \lambda \circ \Theta(x)$. However, for HABA with references, we can very well have that $|\lambda(\Theta(x))| > 1$ therefore, $\Theta'(x)$ can be interpreted onto an element of $\lambda(\Theta(x))$, say e'. Nevertheless, since $\Theta'(x) = e'$ imposes some restrictions on δ' (according to the definition of valuation) then it could be that these restrictions are incompatible with the dependencies that must exist between δ' and δ. The risk we run here is to set up a transition between atoms that would be unsound w.r.t. the semantics of $\mathcal{N}\!\mathit{all}\ell\mathrm{TL}$ and the behaviour of the HABA. These considerations lead us to define, given a valuation v which is the set of sound valuations w.r.t. v after a reallocation λ. We use this set in order to rule out "bad" valuations in the target atom.

Definition 7.20. For states q and q', the *valuation reallocator from q to q'* is the function $[- \circ -] : (E_q \times E_q \to \mathbb{M}) \times V_q \to 2^{V_{q'}}$ that, given a q-valuation v and a reallocation $\lambda : q \Longrightarrow q'$, returns a set of q'-valuations compatible w.r.t. v and λ. It is defined as:

$$[\lambda \circ (\psi, \Theta, \delta)] = \{(\psi, \Theta', \delta') \in V_{q'} \mid \text{condition 1 and 2 hold}\}$$

1. $\forall x \in \mathit{fv}(\psi) : \Theta'(x) \in \lambda \circ \Theta(x)$
2. for all weakly maximal connected subgraph E_{mcs} of $(E_q \uplus E_{q'}, \lambda)$ and for all $x \in \mathit{fv}(\psi)$ such that $\Theta(x) \in \iota_1(E_{mcs})$:
   (a) $\delta(\mathit{first}(\iota_1(E_{mcs}))^-, x) = \delta'(\mathit{first}(\iota_2(E_{mcs}))^-, x)$
   (b) $\delta(x, \mathit{last}(\iota_1(E_{mcs}))^+) = \delta'(x, \mathit{last}(\iota_2(E_{mcs}))^+)$
   (c) $\forall y \in \mathit{fv}(\psi) : \Theta(y) \in \iota_1(E_{mcs}) \;\Rightarrow\; \delta(x, y) = \delta'(x, y)$.

Condition 1 states that for free variable x, the interpretation $\Theta'(x)$ must be obtained by applying λ to $\Theta(x)$. Condition 2(c) enforces the correspondence between the distance of variables x, y interpreted on E_{mcs}. Moreover, the distance between a variable x w.r.t. the beginning and the end of the maximal connected graph (where x is interpreted), before and after the transition must be preserved (conditions 2(a) and 2(b)). The other distances between x and the entities within E_{mcs} can be derived from those distances imposed by condition 2. Note that since Θ' and δ' are in the context of a q'-valuation, they satisfy, by definition, the conditions of Table 4. The choice of constraints imposed in the previous definition derives from the compatibility we want to have between the graph $G_{\mathcal{H}}$ and the allocation sequences in the language of $\mathcal{H}$. It becomes clear in the next example.

Example 7.21. Consider the reallocation λ on top of Figure 27 where $K(\phi) = 4$ and:

$$\begin{array}{rr} \delta(e_1^-, y) = 0 & \delta(y, e_1^+) = 1 \\ \delta(y, e_2^-) = 2 & \delta(y, e_2^+) = * \\ \delta(y, x) = * & \\ \delta(e_1^-, x) = * & \delta(x, e_1^+) = * \\ \delta(x, e_2^-) = * & \delta(x, e_2^+) = * \end{array}$$

The maximal weakly connected subgraph we are concerned with in this example is given by $E = (\{e_1, e_2, e_3, e_4\}, \lambda)$. According to Definition 7.20, valuation (ψ, Θ', δ') in $[\lambda \circ (\psi, \Theta, \delta)]$ has at least $\Theta'(x) \in \{e_3, e_4\}$ because of condition 1. Furthermore, by condition 2 the following distances must be preserved:

$$\begin{array}{rcrcl} \delta'(e_3^-, y) & = & \delta(e_1^-, y) & = & 0 \\ \delta'(y, e_4^+) & = & \delta(y, e_2^+) & = & * \\ \delta'(y, x) & = & \delta(y, x) & = & * \\ \delta'(e_3^-, x) & = & \delta(e_1^-, x) & = & * \\ \delta'(x, e_4^+) & = & \delta(x, e_2^+) & = & * \end{array}$$

From these distances it is possible to derive the others. We start by computing the values for y. Since $\delta'(e_3^-, y) = 0$ and by condition (Delta Gamma 1) of Table 4 it follows $\delta'(e_3^-, e_3^+) \oplus 1 = *$, then we have:

$$\delta(y, e_3^+) \oplus 1 = *$$

which has as solution $\delta(y, e_3^+) \in \{3, *\}$. Moreover, from these values we can deduce $\delta(y, e_4^-) = \delta(y, e_3^+) \oplus 1 = *$.

Concerning x we can immediately exclude $\Theta'(x) = e_4$ otherwise $\delta'(x, e_4^+) \neq *$ contradicting condition 2 of Definition 7.20. Hence, $\Theta'(x) = e_3$. Because of condition (Delta Met 3) of Table 4,

$$* = \delta(e_3^-, e_3^+) \oplus 1 = \delta'(e_3^-, x) \oplus \delta'(x, e_3^+) \oplus 1$$

and since $\delta'(e_3^-, x) = *$ the solution of the previous equation is $\delta'(x, e_3^+) \in \{0, 1, 2, 3, *\}$. Moreover, since $\mathcal{C}(e_4) = 1$ then $\delta'(x, e_4^+) = \delta'(x, e_4^-) = 0$. Summarising we have the following possibilities for the other values of δ':

$$\begin{array}{rcl} \delta'(y, e_3^+) & \in & \{3, *\} \\ \delta'(y, e_4^-) & = & * \\ & & \\ \delta'(x, e_3^+) & \in & \{0, 1, 2, 3, *\} \\ \delta'(x, e_4^-) & = & * \end{array}$$

Some of these solution are impossible, and the informations we have to our disposal allow us to be more precise and by narrowing the set of possible δ'. We have not used so far that $\delta'(y, x) = *$. In fact this implies $\delta'(y, e_3^+) = *$ excluding therefore the value 3 in the solutions previously computed. Moreover, we can use that $\delta'(x, e_4^+) = *$ and $\delta(e_4^-, e_4^+) = 0$ to figure out that actually the range of solutions for $\delta'(x, e_3^+)$ is much smaller than $\{0, 1, 2, 3, *\}$. In fact,

$$* = \delta'(x, e_4^+) = \delta'(x, e_4^-) = \delta'(x, e_3^+) \oplus \delta'(e_3^+, e_4^-) = \delta'(x, e_3^+) \oplus 1.$$

The solution of the last equation is $\delta'(x, e_3^+) \in \{3, *\}$.

We may wonder why $[\lambda \circ (\psi, \Theta, \delta)]$ contains sensible options for the valuations (ψ, Θ', δ') that must be contained in an atom in order to yield an edge in the graph. In order to give some insights, let us consider how this abstract transition is reflected at the concrete level. In the bottom of the Figure 27, two possible concrete states are depicted. They are generated from q and q', and connected by a reallocation λ' that is a concretion of λ. This is the particular case where e_2 is instantiated with 11 concrete entities: $e_1^2, \ldots, e_{11}^2$ and e_3 is instantiated with 12 entities, namely $e_1^3, \ldots, e_{12}^3$. The shadow boxes depict the generator. The symbolic interpretation given in the valuation by Θ and δ, is projected at the concrete level to several choices for a concrete

interpretation θ. Indeed, for x we can have: either $\theta(x) = e_5^2$, or $\theta(x) = e_6^2$ or $\theta(x) = e_7^2$. Fixing one of these possibilities, by λ' we obtain $\theta'(x)$. Thus after the transition, we get the following possibilities:

$$\begin{array}{lllll} \theta(x) = e_5^2 & \Rightarrow & \Theta'(x) = e_3 & \delta'(e_3^-, x) = * & \delta'(x, e_3^+) = * \\ \theta(x) = e_6^2 & \Rightarrow & \Theta'(x) = e_3 & \delta'(e_3^-, x) = * & \delta'(x, e_3^+) = * \\ \theta(x) = e_7^2 & \Rightarrow & \Theta'(x) = e_3 & \delta'(e_3^-, x) = * & \delta'(x, e_3^+) = 3. \end{array}$$

Note that the first two options are actually the same and this is according to the solutions we have found before. □

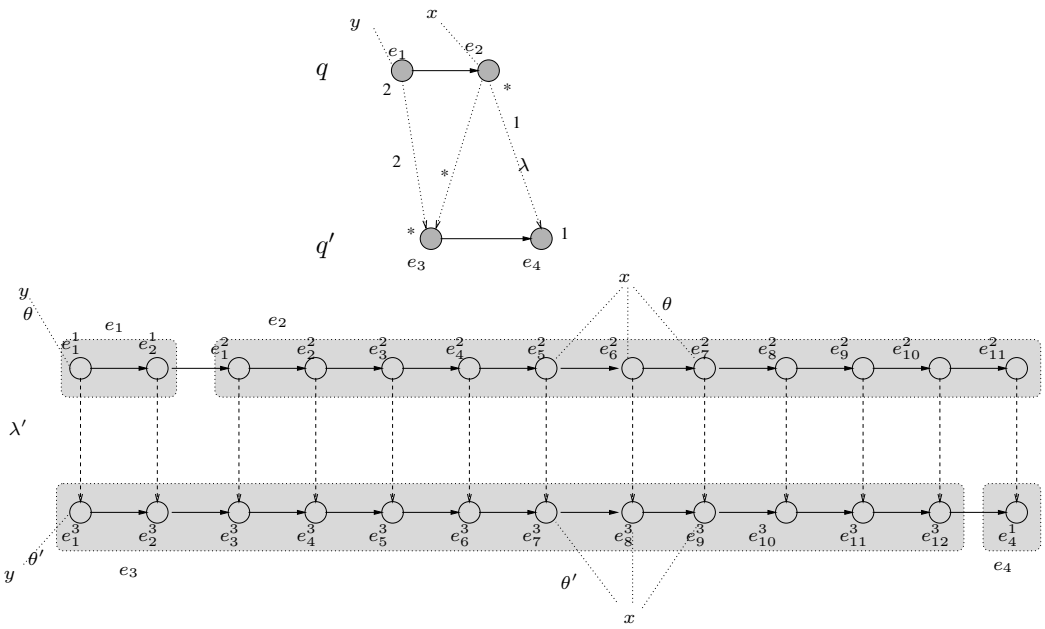

Figure 27: Dependencies (at the abstract and concrete level) between distances of free variables in a weakly maximal connected subgraph.

The definition of valuation reallocator is extended point-wise to a set $V \subseteq V_q$:

$$[\lambda \circ V] = \bigcup_{v \in V} [\lambda \circ v]$$

we can then define the composition for a chain of reallocations $\lambda_j, \cdots, \lambda_0$ (with $j \geqslant 0$) of q-valuation:

$$[\lambda_j \circ \cdots \lambda_0 \circ v] = [\lambda_j \circ [\lambda_{j-1} \circ \cdots \lambda_0 \circ v]]$$

Definition 7.22 (tableau graph). The *tableau graph* for a HABA $\mathcal{H}$ and an $\mathcal{N}all\ell$TL-formula ϕ, denoted $G_{\mathcal{H}}(\phi)$, consists of vertexes $A_{\mathcal{H}}(\phi)$ and edges $\rightarrow \subseteq A_{\mathcal{H}}(\phi) \times (Ent \times Ent \rightarrow \mathbb{M}^*) \times A_{\mathcal{H}}(\phi)$ such that $(q, D) \rightarrow_\lambda (q', D')$ iff

- $q \rightarrow_\lambda q'$
- for all $\mathsf{X}\psi \in CL(\phi)$: $(\mathsf{X}\psi, \Theta, \delta) \in D \Leftrightarrow \exists (\psi, \Theta', \delta') \in [\lambda \circ (\psi, \Theta, \delta)] \cap D'$.

The reallocation λ attached on the edge is the same as the reallocation of the corresponding transition in the HABA. Furthermore, in case of a formula $\mathsf{X}\psi$ in the source atom, there is a transition if we have a valuation on the target state that contains ψ where the components Θ', and δ' are compatible (in the sense of Definition 7.20) with the reallocation λ and the corresponding components of the source state. Since the set $[\lambda \circ (\Theta, \delta)]$ is not always a singleton, there can be more than one transition to different atoms, concerning the same $\mathsf{X}\psi$ (as we have seen in Example 7.21).

7.4 Paths

Definition 7.23. An *(allocation) path* in $G_{\mathcal{H}}(\phi)$ is an infinite sequence $\pi = (q_0, D_0)\ \lambda_0\ (q_1, D_1)\ \lambda_1\ \cdots$ such that:

1. $q_0\lambda_0 q_1\lambda_1 \cdots \in \mathsf{runs}(\mathcal{H})$;
2. for all $i \geqslant 0$, $(q_i, D_i) \rightarrow_{\lambda_i} (q_{i+1}, D_{i+1})$;
3. for all $i \geqslant 0$ and all $(\psi_1 \,\mathsf{U}\, \psi_2, \Theta, \delta) \in D_i$, there exists a $j \geqslant i$ such that $\exists(\psi_2, \Theta', \delta') \in D_j \cap [\lambda_{j-1} \circ \cdots \lambda_i \circ (\psi_2, \Theta \restriction \psi_2, \delta \restriction \psi_2)]$.

The next is the important concept of fulfilling path.

Definition 7.24 (fulfilling path). Given an allocation path π in $G_{\mathcal{H}}(\phi)$, we say that π *fulfils* ϕ if the underlying run generates an allocation triple (σ, N, θ) such that $\sigma, N, \theta \vDash \phi$.

As usual, if ϕ is clear from the context, we call π a *fulfilling path*. Furthermore, if there exists $(\sigma, N, \theta) \in \mathcal{L}(\mathcal{H})$ such that $\sigma, N, \theta \models \phi$ we say that ϕ is $\mathcal{H}$-satisfiable. In contrast with the definition of HABA given in [5, 6], a run ρ of the HABAs we consider in this paper does not always generates triples (σ, N, θ) in the sense of Definition 3.24. We write $Gen(\rho)$ for the set of allocation triples generated by ρ, and for a path π we write $Gen(\pi)$ for $Gen(\rho)$ where ρ is the underlying run of π.

There is a correspondence between the satisfiability of a formula in the HABA and the existence of a fulfilling path in the tableau graph.

Proposition 7.25. ϕ is $\mathcal{H}$-satisfiable if and only if there exists a path in $G_{\mathcal{H}}(\phi)$ that fulfils ϕ.

Proof. See Appendix C. □

Definition 7.26. A subgraph $G' \subseteq G_{\mathcal{H}}(\phi)$ is *self-fulfilling* if every node A in G' has at least an outgoing edge and for every $(\psi_1 \,\mathsf{U}\, \psi_2, \Theta, \delta) \in D_A$ there exists a node $B \in G'$ such that

- $A = A_0 \rightarrow_{\lambda_0} A_1 \rightarrow_{\lambda_1} \cdots \rightarrow_{\lambda_{i-2}} A_{i-1} \rightarrow_{\lambda_{i-1}} A_i = B$
- $\exists(\psi_2, \Theta_B, \delta_B) \in D_B \cap [\lambda_{i-1} \circ \cdots \lambda_0 \circ (\psi_2, \Theta \restriction \psi_2, \delta \restriction \psi_2)]$.

A *prefix* in $G_{\mathcal{H}}(\phi)$ is a sequence $A_0 \rightarrow_{\lambda_0} A_1 \rightarrow_{\lambda_1} \cdots \rightarrow_{\lambda_{i-2}} A_{i-1} \rightarrow_{\lambda_{i-1}} A_i$ such that A_0 is an initial atom (i.e., $q_{A_0} \in I_{\mathcal{H}}$) and A_i is in a self-fulfilling subgraph. Let $Inf(\pi)$ denote the set of nodes that appear infinitely often in the path π. $Inf(\pi)$ is a strongly connected subgraph (SCS). We can prove the following implication:

Proposition 7.27. π is a fulfilling path in $G_{\mathcal{H}}(\phi) \Rightarrow Inf(\pi)$ is a self-fulfilling SCS of $G_{\mathcal{H}}(\phi)$.

Proof. See Appendix C. □

Finally, we can collect all the previous results into the following theorem:

Theorem 7.28. For any HABA $\mathcal{H}$ and formula ϕ, it is possible to verify mechanically whether $\mathcal{H} \nvDash \phi$.

Proof. By Proposition 7.25, in order to prove that $\mathcal{H} \nvDash \phi$, it is sufficient to check that in the graph $G_{\mathcal{H}}(\phi)$ there does not exist a fulfilling path π. By Proposition 7.27, for a path π to be fulfilling it is necessary to have as a $Inf(\pi)$ a self-fulfilling SCS. Thus, in order to check that ϕ is not satisfiable in $\mathcal{H}$, it is sufficient to check that every self-fulfilling SCS is not the *Inf* of any path. That is, if Π is the set of *all* paths in $G_{\mathcal{H}}(\phi)$ and

$$\begin{aligned} \Pi_{ful} &= \{\pi \in \Pi \mid \pi \text{ is a fulfilling}\} \\ \Pi_1 &= \{\pi \in \Pi \mid Inf(\pi) \text{ is a self-fulfilling SCS}\} \end{aligned}$$

in order to prove $\Pi_{ful} = \varnothing$ it suffices to show that $\Pi_1 = \varnothing$ (since $\Pi_{ful} \subseteq \Pi_1$). But this is the case (see below) if the following set Π_{SCS} is empty.

$$\Pi_{SCS} = \{G' \subseteq G_{\mathcal{H}}(\phi) \mid G' \text{ is a self-fulfilling SCS such that a) and b) hold }\}$$

where

a) there exists a fulfilling prefix of G';

b) for all $F \in \mathcal{F}_{\mathcal{H}} : F \cap \{q_B | B \in G'\} \neq \varnothing$.

We prove that

$$\Pi_{SCS} = \varnothing \;\Rightarrow\; \Pi_1 = \varnothing.$$

By contradiction, assume $\Pi_1 \neq \varnothing$ (and $\Pi_{SCS} = \varnothing$). Take $\pi = A_0\lambda_0 A_1\lambda_1 \cdots \in \Pi_1$. Since π is a path then $\rho_\pi = q_{A_0}\lambda_0 q_{A_1}\lambda_1 \cdots$ is an accepting run of $\mathcal{H}$ (condition 1 of Definition 7.23). However, this implies that condition *b*) of the definition of Π_{SCS} is satisfied. Furthermore, since π is a path there exists a prefix for $Inf(\pi)$. Hence, it must be $Inf(\pi) \in \Pi_{SCS}$ which contradicts $\Pi_{SCS} = \varnothing$. Since SCS are finite and there are only a finite number of them, it is possible to verify the emptiness of Π_{SCS}. □

The proof of Theorem 7.28 suggests us a procedure described by Algorithm 1 that can be used to verify whether $\mathcal{H} \not\models \phi$.

Algorithm 1 Procedure for non-satisfiability of ϕ.

```
procedure NonSatisfiable(H, φ) do
  Construct G_H(φ);
  Construct the set of self-fulfilling SCS Π_SCS having a prefixes and satisfying the accept
  condition on F_H;
  if Π_SCS = ∅ then
    Output: "H does not satisfy φ";
  else
    return G' ∈ Π_SCS and its prefix as a (possible) counterexample;
  end if
end procedure
```

8 Related work

History-dependent automata. History-dependent (HD) automata [11] are the main inspiration for HABAs. An HD-automaton is an automaton where states, transitions and labels are equipped with a set of local names that can be created dynamically. HD-automata represent an adequate model for history-dependent formalisms such as the π-calculus [10]. Reallocation of entities in HABA resembles the reallocation of names in HD-automata. Several novelties have been introduced in the HABAs defined in this paper w.r.t. HD-automata. First of all, the ability of entities to refer to each other. This permits to describe, in a single state, rather complex structures. Another extension is given by the (local) cardinalities attached to entities. Cardinalities permit to represent a state at different level of abstraction. The relation between two levels representing the same pointer structure is characterised by the notion of morphism. By cardinalities, it is possible to treats some entities as black holes. This last feature allows us to deal with a possibly unbounded number of entities.

3-valued logic. In [13, 14], a framework for the generation of a family of shape analysis algorithms is presented. The framework allows to reason about pointer structures in case of destructive updates. This methodology can be instantiated in different ways to handle different kind of data structures at different levels of precision and efficiency. The 3-valued logic methodology and ours appear complementary in several aspects. The major differences with our approach can be summarised as follows. In [14], states are represented by predicates whereas our approach uses automata. Moreover, $\mathcal{N}all\ell$TL provides some second order capabilities given by the predicate leads-to. Both methodologies produce only safe approximations of the concrete system that is modelled, false negatives may be returned as result of the analysis. This phenomenon requires some means

to tune the abstraction, so that a more precise heap is obtained. To this end, 3-valued logic technology uses *instrumentation predicates*. Every predicate modifies the model (e.g., the operational semantics of the system) and imposes the duty to prove the correctness of the new system with respect to the original one. Our approach is instead parametric in the global constant M. For the programming language it also depends on the constant L given to define L-canonicity of the state. In order to increase the precision of the heap representation we only need to increase the two constants. The new model obtained corresponds automatically to the original one thanks to the machinery provided by morphisms and cardinalities. There is no need to establish the equivalence of the two models. States in [14] are very abstract because of the use of a single summary node and the approach is very general since no restriction on the type of graph is imposed. We take the opposite point of view. In our approach, we try to be more concrete by morphisms that exploit information on the cardinality for entities and by keeping explicit singularities of the heap. This should provide more precision in the analysis (i.e., less spurious counterexamples) since the amount of nondeterminism is reduced. The price we pay for such precision is a less general framework. It would be interesting to study an integration of the two strategies.

[16] presents a model and an algorithm to prove safety properties of Java objects and threads based on 3-valued logic. In this context, however, entities of different states cannot be related. This problem is circumvented in the recent paper [17] (again based on 3-valued logic) that uses reallocations similar to those employed by HD-automata and by HABA without references defined in [5, 6]. The paper proposes a first-order modal (temporal) logic for allocation and deallocation of objects and threads as well as for the specification of properties related to the evolution of the heap. The properties can be verified by an abstract-interpretation algorithm, sound but not complete, that is also defined in the paper.

Model-checking and logics for object-oriented systems. Bandera [3] is a model checker for Java that uses abstract interpretation and program slicing to yield compact state spaces. Another model checker for Java is Java PathFinder [7]. JPF employs garbage collection in order to obtain a finite state space. However, both approaches can only deal with a bounded number of objects.

Others. An intuitionistic extension of Hoare logic, called *Separation Logic* for reasoning about shared mutable data structure in presented by Reynolds in [12]. The logic introduces a special (conjunction) operator that allows to describe the separation of storage into disjoint parts. It is possible therefore to extend a local specification, involving only some variables and parts of the heap, to a global specification involving also other parts of the heap. The main strength of this approach is the capability to reason in a local fashion. The logic can address several kinds of data structures like lists and trees. [8] provides a classical model for the approach introduced by Reynolds, and the relation w.r.t. the intuitionistic is investigated.

In [1] a store-less formalism for describing properties of linked data structures is defined. Moreover the paper introduces a logic, called *Alias Logic*, for reasoning about destructive update performed on data structures. For it a Hoare logic-like proof system is defined.

A spatial logic for reasoning about directed graphs is studied in [2]. The logic is used for the analysis of the manipulation of such graphs that are described by constructs of process algebra. An interesting feature of this logic is the possibility to reason locally about disjoint subgraphs.

References

[1] M. Bozga, R. Iosif, and Y. Lakhnech. Storeless semantics and alias logic. In *Proc. ACM SIGPLAN 2003 Workshop on Partial Evaluation and Semantics Based Program Manipulation (PEPM)*. ACM, 2003.

[2] L. Cardelli, P. Gardner, and G. Ghelli. A spatial logic for querying graphs. In P. Widmayer, F.T. Ruiz, R. Morales, M. Hennessy, S. Eidenbenz, and R. Conejo, editors, *Automata, Lan-*

guages and Programming, 29th International Colloquium (ICALP), volume 2380 of *LNCS*, pages 597–610. Springer, 2002.

[3] J. Corbett, M. Dwyer, J. Hatcliff, C. Pasareanu, Robby, S. Laubach, and H. Zheng. Bandera: Extracting finite-state models from Java source code. In *22nd International Conference on Software Engineering*, pages 439–448. IEEE Computer Society, 2000.

[4] B. Courcelle. Graph rewriting: An algebraic and logic approach. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume Volume B: Formal Models and Sematics, pages 193–242. Elsevier and MIT Press, 1990.

[5] D. Distefano, A. Rensink, and J.-P. Katoen. Model checking dynamic allocation and deallocation. CTIT Technical Report TR–CTIT–01–40, Faculty of Informatics, University of Twente, December 2001.

[6] D. Distefano, A. Rensink, and J.-P. Katoen. Model checking birth and death. In R.A. Baeza-Yates, U. Montanari, and N. Santoro, editors, *Foundations of Information Technology in the Era of Networking and Mobile Computing. In 2nd IFIP International Conference on Theoretical Computer Science (TCS)*, pages 435–447. Kluwer, 2002.

[7] K. Havelund and T. Pressburger. Model checking Java programs using Java PathFinder. *International Journal on Software Tools for Technology Transfer*, 2(4):366–381, 2000.

[8] S. Ishtiaq and P. O'Hearn. BI as an assertion language for mutable data structures. In *Proceedings of the 28th ACM Symposium on Principles of Programming Languages (POPL)*, pages 14–26, 2001.

[9] O. Lichtenstein and A. Pnueli. Checking that finite state concurrent programs satisfy their linear specification. In *Proceedings of the Twelfth Annual ACM Symposium on Principles of Programming Languages (POPL'85)*, pages 97–107. ACM, 1985.

[10] R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes, I and II. *Information and Computation*, 100(1):1–40,41–77, 1992.

[11] U. Montanari and M. Pistore. History-dependent automata. Technical Report TR-98-11, Dipartimento di Informatica University of Pisa, October 1998.

[12] J.C. Reynolds. Separation logic: A logic for shared mutable data structures. In *Proceedings of the Seventeenth Annual IEEE Symposium on Logic in Computer Science (LICS)*, pages 55–74, 2002.

[13] M. Sagiv, T. Reps, and R. Wilhelm. Solving shape-analysis problems in languages with destructive updating. *TOPLAS*, 20(1):1–50, 1998.

[14] M. Sagiv, T.W. Reps, and R. Wilhelm. Parametric shape analysis via 3-valued logic. In *Proceedings of the 26th ACM Symposium on Principles of Programming Languages (POPL)*, pages 105–118, 1999.

[15] M. Smyth. Topology. In S. Abramsky, D. Gabbay, and T. Maibaum, editors, *Handbook of Logic in Computer Science*, volume 1, pages 641–761. Claredon Press, 1992.

[16] E. Yahav. Verifying safety properties of concurrent Java programs using 3-valued logic. In C. Norris and J.J.B. Fenwick, editors, *Proceedings of the 28th ACM Symposium on Principles of Programming Languages (POPL)*, pages 27–40. ACM Press, 2001.

[17] E. Yahav, T. Reps, M. Sagiv, and R. Wilhelm. Verifying temporal heap properties specified via evolution logic. In P. Degano, editor, *Programming Languages and Systems, 12th European Symposium on Programming (ESOP)*, volume 2618 of *LNCS*, pages 204–222. Springer, 2003.

A Proofs of Sections 3 and 4

Proposition 3.16 Let γ_1 and γ_2 be two configurations. If $\gamma_1 \overset{h}{\rightarrowtail} \gamma_2$ then $\gamma_1 \overset{\lambda}{\Longrightarrow} \gamma_2$ where let $e \in E_{\gamma_1}$ and $e' \in E_{\gamma_2}$:

$$\lambda(e, e') = \begin{cases} \mathcal{C}_{\gamma_1}(e) & \text{if } e' = h(e) \\ 0 & \text{otherwise.} \end{cases}$$

Proof. It is enough to prove that λ satisfies each of the conditions of Definition 3.12.

1. This condition is trivially satisfied since h is a morphism and therefore a function. Thus the complete cardinality of an entity e is transferred to $h(e)$.
2. Follows by condition 4m of Definition 3.3.
3. Straightforward since h is a function.
4. Straightforward since $\lambda(e)$ contains only one element.
5. Straightforward from 1m of Definition 3.3.

□

Lemma 3.19 If $\gamma_1 \overset{\lambda}{\Longrightarrow} \gamma_2$ and $\gamma_1' \overset{\lambda'}{\Longrightarrow} \gamma_2'$ and $\lambda' \rhd \lambda$ via h_1 and h_2 then:

a) $E_{\gamma_2'} \backslash \mathrm{cod}(\lambda') = h_2^{-1}(E_{\gamma_2} \backslash \mathrm{cod}(\lambda))$

b) $E^*_{\gamma_2} \subseteq \mathrm{cod}(\lambda)$.

Proof.

a) we prove part a) by showing the two set inclusions.

'$\subseteq$' Let $e \in E_{\gamma_2} \backslash \mathrm{cod}(\lambda')$, by condition 4 of Definition 3.17 we have $\mathcal{C}_{\gamma_2}(h_2(e)) \neq *$. By contradiction assume $h_2(e) \in \mathrm{cod}(\lambda)$, this implies that there exists $e_1, \ldots, e_k \in E_{\gamma_1}$ $(k > 0)$ such that

$$\lambda(e_1, h_2(e)) \oplus \cdots \oplus \lambda(e_k, h_2(e)) = \mathcal{C}_{\gamma_2}(h_2(e)) \neq *$$

By condition 2 of $\rhd$ we have:

$$\textstyle\sum_{e_1, h_2(e) = (h_1(e_1'), h_2(e_2'))} \lambda'(e_1', e_2') \oplus \cdots \oplus \sum_{e_k, h_2(e) = (h_1(e_k'), h_2(e_2'))} \lambda'(e_k', e_2') = \mathcal{C}_{\gamma_2}(h_2(e)) \neq *$$

But since $\mathcal{C}_{\gamma_2}(h_2(e)) \neq *$ there must be $e' \in h_1^{-1}(\{e_1, \ldots, e_k\})$ such that $\lambda(e', e) \neq 0$ otherwise the sum could not be precisely $\mathcal{C}_{\gamma_2}(h_2(e)) \neq *$. Therefore $e \in \mathrm{cod}(\lambda')$ that contradict our initial hypothesis.

'$\supseteq$' Let $e \in h_2^{-1}(E_{\gamma_2} \backslash \mathrm{cod}(\lambda))$ we prove that $e \in E_{\gamma_2} \backslash \mathrm{cod}(\lambda')$. To this end assume $e \in \mathrm{cod}(\lambda')$ then there exists $e' \neq \perp$ such that $\lambda(e', e) = 1$. This implies by condition 2 of definition $\rhd$ that $\lambda(h_1(e'), h_2(e)) \geqslant \lambda(e', e) = 1$. Hence $h_2(e) \in \mathrm{cod}(\lambda)$ that is impossible since we have assumed $e \in h_2^{-1}(E_{\gamma_2} \backslash \mathrm{cod}(\lambda))$.

b) Let $e \in E^*_{\gamma_2}$. By condition 4 of Definition 3.17 we have $h_2^{-1}(e) \subseteq \mathrm{cod}(\lambda')$. Hence, by part a) of the lemma we can conclude that $e \in \mathrm{cod}(\lambda)$.

□

Lemma 3.20 ($\rhd$ transitivity). Let $q_1 \overset{\lambda}{\Longrightarrow} q_2$, $q_1' \overset{\lambda'}{\Longrightarrow} q_2'$ and $q_1'' \overset{\lambda''}{\Longrightarrow} q_2''$. Then:

$$(\lambda'' \rhd \lambda' \ \wedge \ \lambda' \rhd \lambda) \ \Rightarrow \ \lambda'' \rhd \lambda.$$

Proof. By Definition 3.17 the following morphisms exist:

$$q_1' \succ\!\!\xrightarrow{h_1} q_1, \quad q_2' \succ\!\!\xrightarrow{h_2} q_2, \quad q_1'' \succ\!\!\xrightarrow{h_1'} q_1', \quad q_2'' \succ\!\!\xrightarrow{h_2'} q_2'.$$

Therefore by the properties of composition of morphisms we have $q_1'' \succ\!\!\xrightarrow{h_1''} q_1$ and $q_2'' \succ\!\!\xrightarrow{h_2''} q_2$ defined as

$$\begin{aligned} h_1'' &= h_1' \circ h_1 \\ h_2'' &= h_2' \circ h_2. \end{aligned}$$

Thus the first condition of Definition 3.17 for $\lambda'' \rhd \lambda$ is satisfied.

For the second condition, first of all observe that by definition, q_1', q_2' and q_1'' and q_2'' are concrete states since reallocations λ' and λ'' are concretions. This implies that h_1' and h_2' are bijective. Let $e_1 \in E_{q_1}$ and $e_2 \in E_{q_2}$, we have:

$$\begin{aligned} \lambda(e_1, e_2) : \ &\mapsto \ \textstyle\sum_{(e_1,e_2)=(h_1(e_1'),h_2(e_2'))} \lambda'(e_1', e_2') && \text{[by hp: } \lambda' \rhd \lambda\text{]} \\ &\mapsto \ \textstyle\sum_{(e_1,e_2)=(h_1(h_1'(e_1'')),h_2(h_2'(e_2')))} \lambda''(e_1'', e_2'') && \text{[by hp: } \lambda'' \rhd \lambda' \text{ and } h_1', h_2' \text{ bijective]} \\ &\mapsto \ \textstyle\sum_{(e_1,e_2)=(h_1''(e_1''),h_2''(e_2''))} \lambda''(e_1'', e_2'') && \text{[by def. } h_1'', h_2''\text{]}. \end{aligned}$$

Hence also the second condition of Definition 3.17 holds for $\lambda'' \rhd \lambda$.

For the condition (No-Cross), we first observe that:

$$\begin{aligned} & h_1''(e_1'') = h_1''(e_2'') \ \vee \ h_2''(\lambda''(e_1'')) = h_2''(\lambda''(e_2'')) && \text{[by definition of } h_1'', h_2''\text{]} \\ \Rightarrow \ & h_1(h_1'(e_1'')) = h_1(h_1'(e_2'')) \ \vee \ h_2(h_2'(\lambda''(e_1''))) = h_2(h_2'(\lambda''(e_2'))) && \text{[by (No-Cross) of } \lambda'\text{]} \\ \Rightarrow \ & (h_1'(e_1'') \prec_{q_1'} h_1'(e_2'') \iff \lambda'(h_1'(e_1'')) \prec_{q_2'} \lambda'(h_1'(e_2''))) && (22) \end{aligned}$$

Now, note that since $\lambda'' \rhd \lambda'$, for the second property of concretion and the bijectivity of h_1', h_2', we have $\lambda'(h_1'(e_1''), h_2'(\lambda''(e_1''))) = \lambda''(e_1'', \lambda''(e_1'')) = 1$. Therefore:

$$\lambda'(h_1'(e_1'')) = h_2'(\lambda''(e_1'')) \tag{23}$$

In Figure 28 this corresponds to say that the front-low part of the diagram commutes. Hence, assume $h_1''(e_1'') = h_1''(e_2'') \ \vee \ h_2''(\lambda''(e_1'')) = h_2''(\lambda''(e_2''))$, we have:

$$\begin{aligned} & e_1'' \prec_{q_2''} e_2'' && \text{[}q_1' \text{ concrete and Def. 3.3 3m]} \\ \Rightarrow \ & h_1'(e_1'') \prec_{q_1'} h_1'(e_2'') && \text{[by (22)]} \\ \Rightarrow \ & \lambda'(h_1'(e_1'')) \prec_{q_2'} \lambda'(h_1'(e_2'')) && \text{[by (23)]} \\ \Rightarrow \ & h_2'(\lambda''(e_1'')) \prec_{q_2'} h_2'(\lambda''(e_2'')) && \text{[by Def. 3.3 2m]} \\ \Rightarrow \ & \lambda''(e_1'') \prec_{q_2''} \lambda''(e_2'') \end{aligned}$$

Vice-versa:

$$\begin{aligned} & \lambda''(e_1'') \prec_{q_2''} \lambda''(e_2'') && \text{[}q_2' \text{ concrete and Def. 3.3 3m]} \\ \Rightarrow \ & h_2'(\lambda''(e_1'')) \prec_{q_2'} h_2'(\lambda''(e_2'')) && \text{[by (23)]} \\ \Rightarrow \ & \lambda'(h_1'(e_1'')) \prec_{q_2'} \lambda'(h_1'(e_2'')) && \text{[by (22)]} \\ \Rightarrow \ & h_1'(e_1'') \prec_{q_2'} h_1'(e_2'') \\ \Rightarrow \ & e_1'' \prec_{q_2''} e_2'' \end{aligned}$$

Therefore also condition (No-Cross) holds for $\lambda'' \rhd \lambda$. Finally, for condition 4, let $e \in E_{q_2''}$. If $\mathcal{C}_{q_2}(h_2(e)) = *$ then $\mathcal{C}_{q_2}(h_2(h_2'(e))) = *$ and because $\lambda' \rhd \lambda$ (using condition 4) if follows $h_2'(e) \in \mathrm{cod}(\lambda')$ that finally implies (by $\lambda'' \rhd \lambda'$ applying in particular condition 2) $e \in \mathrm{cod}(\lambda'')$. Since also condition 4 is satisfied we conclude that $\lambda'' \rhd \lambda$. ∎

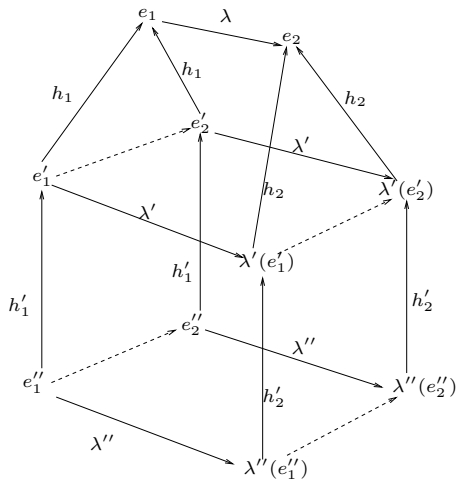

Figure 28: Transitivity of concretions: $\lambda'' \rhd \lambda' \rhd \lambda$.

Theorem 4.5 If $\mathcal{H} \sqsubseteq \mathcal{H}'$ then $\mathcal{L}(\mathcal{H}) \subseteq \mathcal{L}(\mathcal{H}')$.

Proof. Let $(\sigma, N, \theta) \in \mathcal{L}(\mathcal{H})$. Then there exists a run $\rho = q_0\lambda_0 q_1 \lambda_1 \cdots$ such that ρ generates (σ, N, θ) via a generator $(h_i)_{i\in\mathbb{N}}$. Since $\mathcal{H} \sqsubseteq \mathcal{H}'$ there exists $q_0' \in I_{\mathcal{H}'}$ such that $q_0 \sqsubseteq_{h_0'} q_0'$ and moreover for all $i \geqslant 0$ there exists $h_i' : q_i \rightarrowtail q_i'$ and λ_i' such that:

i) $q_i \rightarrow_{\lambda_i'} q_{i+1}$ and $q_{i+1} \sqsubseteq_{h_{i+1}} q_{i+1}'$

ii) $\lambda_i \rhd \lambda_i'$ via h_i', h_{i+1}';

Consider the sequence:

$$\rho' = q_0'\lambda_0' q_1' \lambda_1' \cdots$$

where if $q_i \in F \in \mathcal{F}$, then we take — among all states that simulate q_i — an accept state in $\mathcal{H}'$ that satisfies condition 2.b) of Definition 4.1, i.e., $q_i' \in \psi(F)$ (where ψ is the bijective function described in that condition). Since ρ is an accepting run of $\mathcal{H}$, and because of the choice of the accept states on ρ' described above, then also ρ' is an accepting run of $\mathcal{H}'$.

Hence, in order to prove that $(\sigma, N, \theta) \in \mathcal{L}(\mathcal{H}')$ we show that it is generated by ρ' by some generator $(h_i'')_{i\in\mathbb{N}}$. This corresponds to prove the existence of such generator. For all $i \in \mathbb{N}$, let

$$h_i'' = h_i' \circ h_i.$$

Since h_i'' is the composition of two morphisms it is a morphism by Proposition 3.7. We show that h_i'' satisfies condition (1)-(4) of Definition 3.24.

1. since $\lambda_i^\sigma \rhd \lambda_i \rhd \lambda_i'$ then by Lemma 3.20 it follows $\lambda_i^\sigma \rhd \lambda_i'$ via h_i'' and h_{i+1}''.

2. By definition of simulation we have $I(q_0') = (N', h_0 \circ \theta')$ where $I(q_0) = (h_0^{-1}(N'), \theta')$ and by definition of generator $N = (h_0')^{-1}(N)$. Therefore $N = (h_0'')^{-1}(N')$ and $I(q_0') = (N', h_0'' \circ \theta)$. But then h_0'' that satisfy condition 2 of the definition of generator.

Hence $(h_i'')_{i\in\mathbb{N}}$ is a generator from which it follows $(\sigma, N, \theta) \in \mathcal{L}(\mathcal{H}')$. □

B Proofs of Section 5

Proposition 6.14 If a state γ is L-canonical then:

a) γ is PV-reachable;

b) for every state γ', if there exists a morphism $h : \gamma \rightarrowtail \gamma'$ then either $\gamma \cong \gamma'$ or γ' is L-unsafe.

Proof.

a) Let us partition E_γ in two parts: the set E_γ^{PV} of PV-reachable entities and the set $E_\gamma^{\neg PV}$ given by all the entities not reachable by any program variable. It is clear that $E_\gamma^{\neg PV} \cup E_\gamma^{PV} = E_\gamma$ and $E_\gamma^{\neg PV} \cap E_\gamma^{PV} = \varnothing$. By contradiction assume γ is not PV-reachable, i.e., $E_\gamma^{\neg PV} \neq \varnothing$ and let $e \in E_\gamma^{\neg PV}$. By the properties of L-compactness (that must be satisfied by γ since it is L-canonical), since e is unreachable we have that $indegree(e) > 1$, i.e., there exists at least e', e'' such that $\mu_\gamma(e') = \mu_\gamma(e'') = e$. Furthermore, it must be $e', e'' \in E_\gamma^{\neg PV}$ otherwise we would have $e \in E_\gamma^{PV}$ that would contradict our original hypothesis. Hence, we can construct the following infinite series of sets:

$$\begin{aligned} A_0 &= \{e\} \\ A_{i+1} &= \{e' \mid \mu_\gamma(e') \in A_i\}. \end{aligned}$$

It can be proved by induction on i that

$$\forall i \in \mathbb{N} : A_i \subseteq E_\gamma^{\neg PV}. \tag{24}$$

This implies the following statement (by set theory)

$$\bigcup_{i \in \mathbb{N}} A_i \subseteq E_\gamma^{\neg PV}. \tag{25}$$

However, since every entities has only one outgoing reference, it can be also proved by induction, that for all $i \in \mathbb{N}$ we have:

$$\bigcup_{0 \leqslant j \leqslant i} A_j \subset \bigcup_{0 \leqslant j \leqslant i+1} A_j.$$

Therefore, $|\bigcup_{i \in \mathbb{N}} A_i| = \omega$ that together with (25) implies both $E_\gamma^{\neg PV}$ and E_γ are infinite. But this is impossible since in every state there is only a finite number of live entities.

b) Let γ' be a state such that $\gamma \overset{h}{\rightarrowtail} \gamma'$. We distinguish two cases:

- h is not contractive. Then h must be an isomorphism by Proposition 6.3.
- h is contractive. Therefore there exists $e' \in E_{\gamma'}$ such that $|h^{-1}(e')| > 1$, i.e., the pure chain $h^{-1}(e')$ that has more than one entity is contracted in e'. Since γ is L-compact, there exists $e_v \in PV$ such that $d(e_v, last(h^{-1}(e'))) \leqslant L + 1$, i.e., the last entity of the pure chain cannot be distant more than $L+1$ from some program variable (in this case e_v). But, this implies that $d(e_v, first(h^{-1}(e'))) \leqslant L$. We have that

  $$d(h(e_v), e') \leqslant L \tag{26}$$

  (the inequality is strict if h is also contracting for some other e'' preceding e' or if more than two entities are mapped onto e'). However, $|h^{-1}(e')| > 1$ implies $\mathcal{C}(e') > 1$ that together with (26) violates the L-safeness condition for e' since $h(e_v) \in PV$ (cf. condition (11)). Therefore γ' is L-unsafe.

□

Proposition 6.15 Let γ_1 and γ_2 be a PV-reachable and a L-canonical configurations, respectively. If $h_1 : \gamma_1 \rightarrowtail \gamma_2$ and $h_2 : \gamma_1 \rightarrowtail \gamma_2$ then $h_1 = h_2$.

Proof. Since γ_1 and γ_2 are PV-reachable, if $e \in E_{\gamma_1}$ then there exists $e_v \in PV$ such that $\mu^n_{\gamma_1}(e_v) = e$ for some $n \geq 0$. According to Definition 6.4, the number n is the distance between e_v and e and we denote it by $d(e_v, e)$. Therefore we can define the shortest distance from all the program variables: $d(PV, e) = \min\{d(e_v, e) \neq \bot \mid e_v \in PV\}$. Now, we prove:

$$\forall e \in E_{\gamma_1} : \; h_1(e) = h_2(e) \tag{27}$$

by induction on the distance $d(PV, e)$.

- **Base case** $d(PV, e) = 0$. That is $e \in PV$. By the global constraint (11) on page 24 we have $h_1(e) = h_2(e)$.
- **Inductive case** $d(PV, e) = n + 1$. By contradiction assume $h_1(e) \neq h_2(e)$. Since γ_1 is PV-reachable, there exists $e' \prec_1 e$. By condition 3m of Definition 3.3 this implies that $h_1(e') \preceq_2 h_1(e)$ and $h_2(e') \preceq_2 h_2(e)$. We have four different cases:

  1. $h_1(e') \prec_2 h_1(e)$ and $h_2(e') \prec_2 h_2(e)$
  2. $h_1(e') =_2 h_1(e)$ and $h_2(e') \prec_2 h_2(e)$
  3. $h_1(e') \prec_2 h_1(e)$ and $h_2(e') = h_2(e)$
  4. $h_1(e') = h_1(e)$ and $h_2(e') = h_2(e)$

  Case 1 implies $h_1(e') \neq h_2(e')$ because for every entity there exists only one outgoing reference. But this is a contradiction since, $d(PV, e') = n$, thus, by induction hypothesis, $h_1(e') = h_2(e')$.

  Case 4 implies by induction hypothesis, $h_1(e) = h_1(e') = h_2(e') = h_2(e)$ that again is a contradiction because we have assumed $h_1(e) \neq h_2(e)$.

  Finally, the more involved cases are 2 and 3. We prove 2 since 3 is symmetrical and can be shown precisely in the same way. Since $h_1(e') = h_1(e)$ and $h_2(e') \prec_2 h_2(e)$ and by induction hypothesis $h_1(e') = h_2(e')$, we have $\mathcal{C}_{\gamma_2}(h_1(e')) = *$ because, one morphism maps on it both e, e' whereas the other only one. Moreover $\{e', e\}$ must be a pure chain since h_1 maps this set in the same entity. In turn, this implies

  $$\{h_2(e'), h_2(e)\} \text{ is a pure chain} \tag{28}$$

  because being a morphism h_2 maps pure chains onto pure chains. Moreover, since γ_2 is L-canonical, $h_1(e')$ is at a distance at least $L + 1$ from every program variables otherwise γ_2 would be not L-safe. But since $h_1(e') = h_2(e')$ and $h_2(e') \prec_2 h_2(e)$ then $\{h_2(e'), h_2(e)\}$ is *not* a pure chain otherwise L-compactness would be violated. But this contradicts (28).

□

Theorem 6.16 (Existence of the canonical form). For every L-safe and PV-reachable configuration γ there exists an L-canonical configuration γ' and a unique morphism $h : \gamma \rightarrowtail \gamma'$.

Proof. If γ is L-compact, then it is L-canonical by definition, and we can take $h = id_\gamma$.

Assume then that γ in not L-compact, we construct a L-compact γ' out of γ. Since γ is not L-compact then by definition:

$$\exists e \in E_\gamma : (\mathit{indegree}(e) = 1 \; \wedge \; \forall e' \in PV : d(e', e) > L + 1). \tag{29}$$

In other words entities satisfying (29) form pure chains distant more than $L+1$ from every program variable. Let $C_1, \ldots, C_n \subseteq E_\gamma$ be *all* such chains. In the construction of γ', we need to exclude

them. Let:

$$E_{\gamma'} = (E_\gamma \backslash (C_1 \cup \cdots \cup C_n)) \cup \{\mathit{first}(C_1), \ldots, \mathit{first}(C_n)\}$$

$$\mu_{\gamma'} = (\mu_\gamma \upharpoonright E_\gamma \backslash (C_1 \cup \cdots \cup C_n)) \cup \{(\mathit{first}(C_i), \mu_\gamma(\mathit{last}(C_i))) \mid 1 \leqslant i \leqslant n\}$$

$$\mathcal{C}_{\gamma'}(e) = \begin{cases} \mathcal{C}_\gamma(e) & \text{if } e \in E_\gamma \backslash (C_1 \cup \cdots \cup C_n) \\ \sum_{e' \in C_i} \mathcal{C}_\gamma(e') & \text{if } e = \mathit{first}(C_i). \end{cases}$$

Hence, $\gamma' = (E_{\gamma'}, \mu_{\gamma'}, \mathcal{C}_{\gamma'})$ is L-compact by construction since every pure chain violating the L-compactness condition in γ has been collapsed in the first entity of the chain itself. Note that γ' is L-safe since γ is L-safe and the compacted chains are those in γ that are distant more than $L+1$ from the program variables, therefore, the multiple, unbounded entities that substitute these chain in γ' are still at a distance at least $L+1$. We conclude that γ' is indeed L-canonical.

We construct now, a morphism $h : \gamma \rightarrowtail \gamma'$. For $e \in E_\gamma$ and $1 \leqslant i \leqslant n$ let:

$$h(e) = \begin{cases} e & \text{if } e \in E_\gamma \backslash (C_1 \cup \cdots \cup C_n) \\ \mathit{first}(C_i) & \text{if } e \in C_i. \end{cases}$$

As expected, h corresponds to id on those entities that do not violate compactness, otherwise h collapses the pure chains $C_1, \ldots, C_n$ into their corresponding first elements. It can be shown that h is a morphism. In fact, by construction h satisfies condition 1m-4m of the Definition 3.3. Note that requiring γ to be PV-reachable is essential for the existence of h.

Finally, to complete the proof, we show that h is unique (up to isomorphism). To this end, consider a generic morphism $h' : \gamma \rightarrowtail \gamma'$. Since γ PV-reachable by hypothesis of this theorem, and γ' is L-canonical by construction, we can conclude by Proposition 6.15 that $h = h'$. Therefore, h is unique. □

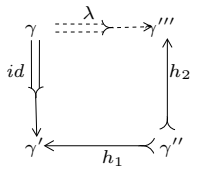

Figure 29: Diagram of Proposition 6.20.

In the following lemma we prove that we can complete the diagram reported in Figure 29 by a reallocation γ.

Proposition 6.20 Let γ and γ''' be two L-canonical states. If $\gamma \overset{id}{\Longrightarrow} \gamma' \overset{h_1}{\prec\!\!\!-} \gamma'' \overset{h_2}{\rightarrowtail} \gamma'''$ such that

(a) $\forall e \in E_{\gamma'''} : id^{-1}(h_1 \circ h_2^{-1}(e)) \subseteq E_\gamma^\perp$ is a chain and

(b) $\mathcal{C}_{\gamma'''}(e) = * \;\Rightarrow\; \perp \notin id^{-1}(h_1 \circ h_2^{-1}(e))$.

Then $\gamma \overset{\lambda}{\Longrightarrow} \gamma'''$ where:

$$\begin{aligned} \lambda = {} & \{(e_1, e_2, \mathcal{C}_{\gamma''}(h_1^{-1}(e_1) \cap h_2^{-1}(e_2))) \mid e_1 \in E_\gamma\} \cup \\ & \{(e, \perp, \mathcal{C}_\gamma(e)) \mid e \in E_\gamma \backslash E_{\gamma'}\} \cup \\ & \{(\perp, e, \mathcal{C}_{\gamma'''}(e)) \mid e \in E_{\gamma'''} \;\wedge\; h_1 \circ h_2^{-1}(e) \cap E_\gamma = \varnothing\}. \end{aligned}$$

Proof. We show that λ satisfies all the condition of Definition 3.12.

1. First of all note that $\gamma \xRightarrow{id} \gamma'$ and therefore $\mathcal{C}_\gamma \restriction E_\gamma \cap E_{\gamma'} = \mathcal{C}_{\gamma'} \restriction E_\gamma \cap E_{\gamma'}$. Let $e_1 \in E_\gamma \cap E_{\gamma'}$, we have $\mathcal{C}(e_1) = \mathcal{C}_{\gamma'}(e_1) = \mathcal{C}_{\gamma''}(h_1^{-1}(e_1))$ by definition of morphism. Every element of $h_1^{-1}(e_1)$ is then remapped by h_2 to the corresponding entities in $\lambda(e_1)$. Therefore

$$\mathcal{C}_\gamma(e_1) = \mathcal{C}_{\gamma''}(h^{-1}(e_1)) = \sum_{e_2 \in E_{\gamma'''}} \sum_{e_1 = h_1(e), e_2 = h_2(e)} \mathcal{C}_{\gamma''}(e) = \sum_{e_2 \in E_{\gamma'''}} \lambda(e_1, e_2)$$

   For $e_1 \in E_{\gamma'} \backslash E_\gamma$ we can show the correspondence of the cardinality by the same argument. For $e_1 \in E_\gamma \backslash E_{\gamma'}$ the correspondence of the cardinality is ensured since λ is defined as id that is a reallocation.

2. Similar to 1.

3. Let $e \in E_\gamma$ and $A = \{e' \mid \lambda(e, e') = *\}$. By contradiction, assume $|A| > 1$, then there exist $e_1, e_2 \in A$ with $\mathcal{C}_{\gamma'''}(e_1) = \mathcal{C}_{\gamma'''}(e_2) = *$. $|A| > 1$ implies that e is split in more than one entity during the transition. However, id precedes the application of morphism therefore, the splitting of e (that can only happens by h_1) must happen *after* any change in the topology of γ performed by id. Hence $h_2 \circ h_1^{-1}(e)$ must be a chain containing at least e_1 and e_2. In order to prove the statement it is enough to consider the case where $h_2 \circ h_1^{-1}(e)$ is only composed of e_1 and e_2 and we can either have $e_1 \prec_{\gamma'''} e_2$ or $e_2 \prec_{\gamma'''} e_1$. Without loss of generality, assume $e_1 \prec e_2$ (the other case is symmetric and can be proved in the same way). By hypothesis γ''' is compact, therefore by definition of L-safeness we have

$$\forall e_v \in PV : d(e_v, e_1) > L \ \wedge \ d(e_v, e_2) > L \tag{30}$$

   and by the hypothesis $e_1 \prec_{\gamma'''} e_2$ we have

$$d(e_v, e_2) > d(e_v, e_1) > L. \tag{31}$$

   Moreover, by definition of L-compactness we have

$$indegree(e_2) > 1 \vee \exists e_{v'} \in PV : d(e_{v'}, e_2) \leqslant L + 1. \tag{32}$$

   We distinguish the two cases:

   - if $\exists e_{v'} \in PV : d(e_{v'}, e_2) \leqslant L+1$ then by (31) we have $L < d(e_v, e_1) < d(e_v, e_2) \leqslant L+1$, which is a contradiction.
   - $indegree(e_2) > 1$ is also impossible since the rearrangement of links takes place before splitting, therefore $h_2 \circ h_1^{-1}(e)$ is a pure chain by definition of morphism, and $indegree(e_2) = 1$.

   The second part of this condition, i.e., $|\{e' \in E_{\gamma'''} \mid \lambda(\perp, e') = *\}| = 0$ follows in a straightforward manner from hypothesis (b).

4. If $\mathcal{C}_\gamma(e) = 1$, then e can be reallocated only on a single entity that is a chain by definition. Otherwise if $\mathcal{C}_\gamma(e) \neq 1$ then e can be reallocated in more than one entity. Nevertheless, e can be split in more than one entity only by the application of the inverse morphism h_1^{-1}, that is applied after the application of id. This means that when e is split, no other modifications in the link structure can happen. Therefore, since $h_1^{-1}(e)$ is a pure chain, it will be remapped by h_2 into another pure chain.

5. This condition follows in a straightforward manner from hypothesis (a).

□

Lemma B.1. Let γ_1 be a concrete configuration. Then:

$$\gamma_1 \succ\!\xrightarrow{h_1} \gamma \ \Rightarrow \ \exists (\gamma_2, h_2) \in \mathsf{SExp}(\gamma), \exists h_{12} : \gamma_1 \succ\!\xrightarrow{h_{12}} \gamma_2.$$

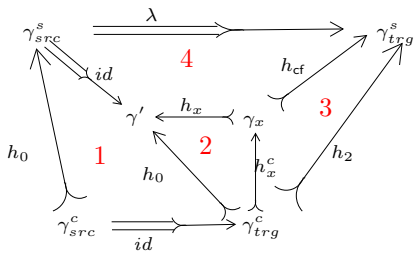

Figure 30: Interdependence among morphisms and reallocations in the assignment rule.

Proof. We show the existence of $(\gamma_2, h_2) \in \mathsf{SExp}(\gamma)$ and h_{12} by construction. First of all, if γ is L-safe then we trivially have $(\gamma, id_\gamma) \in \mathsf{SExp}(\gamma)$ and $h_{12} = h_1$. Therefore, let us assume that γ is not L-safe and let

$$E_\gamma^{<L} = \{e \in E_\gamma \mid \mathcal{C}(e) \neq 1 \ \wedge \ \exists e' \in PV : d(e', e) \leqslant L\}$$

that is $E_\gamma^{<L}$ is the subset of non concrete entities that — being closer than $L+1$ for some program variables — make γ non L-safe. Moreover, for $e \in E_\gamma$, with $h_1^{-1}(e) = e_1, \ldots, e_n$, given a $m > 0$ let $\mathit{Prefix}_m(e) = \{e_1, \ldots, e_m\}$, i.e., $\mathit{Prefix}_m(e)$ is the prefix of the pure chain $h_1^{-1}(e)$ containing the first m entities[10]. Now, let $\gamma_2 = (E_{\gamma_2}, \mu_{\gamma_2}, \mathcal{C}_{\gamma_2})$ defined as:

$$\begin{array}{rcl} E_{\gamma_2} & = & E_{\gamma_1} \backslash \{e \in h_1^{-1}(e') \mid e' \in E_\gamma^{<L}\} \cup \bigcup_{e \in E_\gamma^{<L}} \mathit{Prefix}_{L+1}(e) \\ \forall e \in E_{\gamma_2} : \mu_{\gamma_2}(e) & = & \begin{cases} \mu_{\gamma_1}(\mathit{last}(h_1^{-1}(h_1(e)))) & \text{if } e = \mathit{last}(\mathit{Prefix}_{L+1}(e')) \text{ and } e' \in E_\gamma^{<L} \\ \mu_{\gamma_1}(e) & \text{otherwise} \end{cases} \\ \forall e \in E_{\gamma_2} : \mathcal{C}_{\gamma_2}(e) & = & \begin{cases} \lceil h_1^{-1}(e) - L \rceil & \text{if } e = \mathit{last}(\mathit{Prefix}_{L+1}(e')) \text{ and } e' \in E_\gamma^{<L} \\ \mathcal{C}_{\gamma_1}(e) & \text{otherwise.} \end{cases} \end{array}$$

The previous definition has the following intuition: E_{γ_2} is a subset of entities in E_{γ_1}. Entities mapped by h_1 in $E_\gamma^{<L}$ are included only if they are among the first $L+1$ entities of pure chain described by h_1. μ_{γ_2} and $\mathcal{C}_{\gamma_2}$ are defined according to E_{γ_2} so that γ_2 corresponds to an abstract version of E_{γ_1}.

We now define the morphism h_2 and h_{12}. In particular for $e \in E_{\gamma_2}$ let:

$$h_2(e) = h_1(e) \tag{33}$$

and for $e' \in E_{\gamma_1}$ let $h_1^{-1}(h_1(e')) = e_1, \ldots, e_2$ $(n > 0)$:

$$h_{12}(e') = \begin{cases} e' & \text{if } e' = e_j \text{ and } 1 \leqslant j \leqslant L \\ e_{L+1} & \text{otherwise.} \end{cases} \tag{34}$$

Note that according to this definition:

$$h_1 = h_2 \circ h_{12}. \tag{35}$$

Since h_1 is a morphism (and by construction of γ_2) also h_2 is such. We prove that h_{12} is a morphism by showing that it fulfils condition 1m-4m of Definition 3.3.

1m. Let $e \in E_{\gamma_2}$. Let $h_1^{-1}(h_2(e)) = e_1, \ldots, e_n$. By definition of h_{12}, we have $e = e_j$ for some $1 \leqslant j \leqslant n$. We distinguish two cases:

1. if $j \leqslant L$ then $|h_{12}^{-1}(e)| = 1$ by construction, and therefore $h_{12}^{-1}(e)$ is a pure chain.

[10] In case $m \geqslant n$ then $\mathit{Prefix}_m(e) = h_1^{-1}(e)$.

2. otherwise (by construction of h_{12}) it is $j = L+1$, and $|h_{12}^{-1}(e)| = e_{L+1}, \ldots, e_n$ which is also a pure chain.

2m. Let $e, e' \in E_{\gamma_2}$ such that $e \prec_{\gamma_2} e'$. Since h_2 is a morphism, then either $h_2(e) \prec_{\gamma} h_2(e')$ or $h_2(e) = h_2(e')$. In the first case, we have that (also because h_1 is a morphism):

$$last(h_{12}^{-1}(e)) = last(h_1^{-1}(h_2(e))) \preceq_{\gamma_1} first(h_1^{-1}(h_2(e'))) = first(h_{12}^{-1}(e'))$$

If $h_2(e) = h_2(e')$ the the property is fulfilled observing that by construction $h_1^{-1}(h_2(e)) = h_1^{-1}(h_2(e'))$. Let $h_1^{-1}(h_2(e)) = e_1, \ldots, e_n$, then, if e, e' are both within the prefix of the first L entities of the pure chain $h_1^{-1}(h_2(e))$ then $last(h_{12}^{-1}(e)) = e \prec_{\gamma_1} e' = first(h_{12}^{-1}(e'))$. Otherwise, whereas e must be at position L and e' at position $L+1$. Thus, $last(h_{12}^{-1}(e)) = e_L \prec_{\gamma_1} first(h_{12}^{-1}(e')) = e_{L+1}$.

3m. Let $e, e' \in E_{\gamma_1}$ with $e \prec_{\gamma_1} e'$. Assume by contradiction that $h_{12}(e') \prec_{\gamma_2} h_{12}(e)$ then by definition of morphism $h_2(h_{12}(e')) \prec_{\gamma_2} h_2(h_{12}(e))$ that in turn implies, by (35), $h_1(e') \prec_{\gamma} h_1(e)$ that is a contradiction because h_1 is a morphism. Hence it must be $h_{12}(e) \preceq_{\gamma_2} h_{12}(e)$.

4m. For $e \in E_{\gamma_2}$, we prove that $\mathcal{C}_{\gamma_2}(e) = \mathcal{C}_{\gamma_1}(h_{12}^{-1}(e))$. According to the definition of γ_2, we have that the cardinality of entities are the same as in γ_1 except for every entity that corresponds to $e = last(Prefix_{L+1}(e'))$ for some $e' \in E_{\gamma}^{<L}$. By definition h_{12} maps onto e the subchain of (concrete) entities in the suffix $E_{L+1}, \ldots, e_{|h^{-1}(h_2(e))|}$ that corresponds to $C_{\gamma_2}(e)$.

Hence, h_{12} satisfies all the conditions 1m-4m, we conclude that it is a morphism. □

The next lemma proves a property enjoyed by configurations related by a morphism h after the application of operations *add*, *cancel* or *modify* on both configurations using the same parameters. The property is described visually in Figure 31: after the application of the corresponding operation the configuration are related by a morphism h' which, were defined, corresponds to h.

Lemma B.2. Let γ^s, γ^c be two $(0 <)$ L-safe configurations such that $\gamma^c \succ\!\!\xrightarrow{h} \gamma^s$ and $\alpha = x.a^n$, and $\alpha' = y.a^m$ with $n, m \leqslant L$. Then:

1. $add(\gamma^c, \alpha) \succ\!\!\xrightarrow{h'} add(\gamma^s, \alpha)$ where $h \restriction (E_{\gamma^c} \cap E_{add(\gamma^c,\alpha)}) = h'$;
2. $cancel(\gamma^c, \alpha) \succ\!\!\xrightarrow{h'} cancel(\gamma^s, \alpha)$ where $h \restriction E_{cancel(\gamma^c,\alpha)} = h'$;
3. $modify(\gamma^c, \alpha, \alpha') \succ\!\!\xrightarrow{h'} modify(\gamma^s, \alpha, \alpha')$ where $h \restriction E_{modify(\gamma^c,\alpha,\alpha')} = h'$.

Proof.

1. By definition we have:

$$\begin{aligned} add(\gamma^c, \alpha) &= \langle E_{\gamma^c} \cup \{e\}, \mu_{\gamma^c}\{e/[\![\alpha]\!]\}, \mathcal{C}_{\gamma^c}\{1/e\}\rangle_{PV} \\ add(\gamma^s, \alpha) &= \langle E_{\gamma^s} \cup \{e'\}, \mu_{\gamma^s}\{e'/[\![\alpha]\!]\}, \mathcal{C}_{\gamma^s}\{1/e'\}\rangle_{PV} \end{aligned}$$

where $e = \min(Ent \backslash E_{\gamma^c})$ and $e' = \min(Ent \backslash E_{\gamma^s})$. We define the function $h' : E_{add(\gamma^c,\alpha)} \to E_{add(\gamma^s,\alpha)}$ as follows:

$$\forall \tilde{e} \in E_{add(\gamma^c,\alpha)} : h'(\tilde{e}) = \begin{cases} e' & \text{if } \tilde{e} = e \\ h(\tilde{e}) & \text{otherwise.} \end{cases}$$

A consequence of the interpretation ϑ for program variables and of the hypothesis on the morphisms (11) of Section 6.1 (see page 24) is that the update on γ^s (being L-safe) corresponds to that in γ^c, i.e., $h([\![\alpha]\!]_{\mu_{\gamma^c},\vartheta}) = h([\![\alpha]\!]_{\mu_{\gamma^s},\vartheta})$. In particular, if unreachable entities are obtained by the changes on the topology, they are a subset of $\mu^*_{\gamma^c}([\![\alpha]\!])$ and $\mu^*_{\gamma^s}([\![\alpha]\!])$, respectively. This means, that in γ^c and γ^s entities garbage collected by the application of

$\langle\cdot\rangle_{PV}$ are related by the morphism h. Hence, since h is surjective, then it follows that h' is also surjective.

Moreover, because h is a morphism, it is possible to show that also h' satisfies condition 1m-4m of Definition 3.3 and thus h' is a morphism itself.

2. For the second case we have:

$$\begin{aligned} cancel(\gamma^c, \alpha) &= \langle E_{\gamma^c} \backslash \{[\![\alpha]\!]\}, \mu_{\gamma^c} \circ \psi^c, \mathcal{C}_{\gamma^c} \upharpoonright (E_{\gamma^c} \backslash \{[\![\alpha]\!]\}) \rangle_{PV} \\ cancel(\gamma^s, \alpha) &= \langle E_{\gamma^s} \backslash \{[\![\alpha]\!]\}, \mu_{\gamma^s} \circ \psi^s, \mathcal{C}_{\gamma^s} \upharpoonright (E_{\gamma^s} \backslash \{[\![\alpha]\!]\}) \rangle_{PV} \end{aligned}$$

where $\psi : E_{\gamma^c} \to E^{\perp}_{\gamma^c}$ and $\psi : E_{\gamma^s} \to E^{\perp}_{\gamma^s}$ are defined as

$$\psi^c(e) = \begin{cases} \perp & \text{if } e \in \mu^{-1}_{\gamma^c}([\![\alpha]\!]) \cup [\![\alpha]\!] \\ e & \text{otherwise} \end{cases} \qquad \psi^s(e) = \begin{cases} \perp & \text{if } e \in \mu^{-1}_{\gamma^s}([\![\alpha]\!]) \cup [\![\alpha]\!] \\ e & \text{otherwise} \end{cases}$$

In this case we define h' as restriction of h on the remaining entity after $[\![\alpha]\!]$ has been removed and garbage collection has taken place. That is

$$h' = h \upharpoonright E_{cancel(\gamma^c, \alpha)}.$$

By the same argument given for $add(\gamma^c, \alpha)$, it follows that h' is surjective and, by being a restriction of h, is a morphism.

3. Similar two the previous two cases.

□

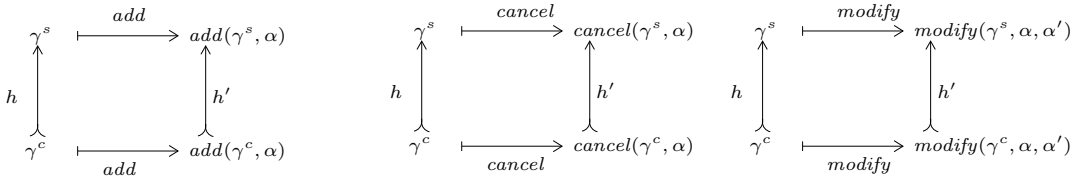

Figure 31: Applying the same operation on configurations related by a morphism results in configurations related by a morphism.

Lemma 6.25 λ defined in rule (ASGN-sym) is a reallocation.

Proof. By Proposition 6.20, it is enough to prove that it preconditions (a) and (b) are satisfied by λ. For condition (a) we have to show that there does not exist an e in the target state such that $h \circ h^{-1}_{\mathsf{cf}}(e)$ is not a chain in the source state. By contradiction let us assume such e does exist. Then, by definition of morphism, $h^{-1}_{\mathsf{cf}}(e)$ is a pure chain in $\gamma_{q''} \in \mathsf{SExp}(\gamma_{q'})$. This implies (in particular using 3m) that also $h \circ h^{-1}_{\mathsf{cf}}(e)$ is a pure chain in $\gamma_{q'}$. Hence, since the chain does not exist in the source state, then it must have been created by the rearrangement of pointers due to the execution of the assignment. In the context of Proposition 6.20 this corresponds to $\gamma_q \overset{id}{\Longrightarrow} \gamma_{q'}$. Since the chain is created by manipulation of a link by the assignment there must exist two entities $e_1, e_2 \in h \circ h^{-1}_{\mathsf{cf}}(e)$ such that $e_1 \not\prec_{\gamma_q} e_2$, before id and $e_1 \prec_{\gamma_{q'}} e_2$ after. However, γ_q is L-compact and therefore PV-reachable, then there exists $e_3 \neq e_1$ such that $e_3 \prec_{\gamma_q} e_2$ (e_3 could be either in PV or reachable from an entity in PV). But then in $\gamma_{q'}$ — after the link between e_1 and e_2 has been set — we have $indegree(e_2) > 1$, this implies that e_1, e_2 is not a pure chain contradicting our initial hypothesis. Therefore, precondition (a) is fulfilled.

The condition (b) is straightforward since in the assignment no new entities are created. □

Theorem 6.27 For all program $p : id(\mathcal{A}_p) \sqsubseteq \mathcal{H}_p$.

Proof. We prove the theorem by defining a relation between $id(\mathcal{A}_p)$ and $\mathcal{H}_p$ and proving that it is a simulation $id(\mathcal{A}_p) \sqsubseteq \mathcal{H}_p$. Let

$$\mathcal{R} = \{((r,\gamma^c),(r,\gamma^s),h) \in Q_{\mathcal{A}_p} \times Q_{\mathcal{H}_p} \times (Ent \rightharpoonup Ent) \mid r \in Par,\ \gamma^s = \mathsf{cf}(\gamma^c),\ h = h_{\mathsf{cf}}\}. \quad (36)$$

First of all we prove that $\mathcal{R}$ is a simulation, by showing that it satisfies conditions 1.a and 1.b of Definition 4.1. To this end, assume $((r,\gamma^c),(r,\gamma^s),h_0) \in \mathcal{R}$

1.a) Straightforward since by definition of $\mathcal{R}$ we have $\gamma^c \succ\!\xrightarrow{h_0}\! \gamma^s$.

1.b) We prove by induction on the structure of the statement r that the target state of every possible (r,γ^c) satisfies (i), (ii). We distinguish several cases.

Base of induction.

– **case** $r = \mathsf{new}(\alpha.a)$. By the rules of the concrete transition system the only possible transition is:

$$\mathsf{new}(\alpha.a), \gamma^c \rightarrow_{id} \mathsf{skip}, add(\gamma^c,\alpha).$$

On the symbolic level we have:

$$\mathsf{new}(\alpha.a), \gamma^s \rightarrow_{\lambda} \mathsf{skip}, \mathsf{cf}(add(\gamma^s,\alpha)).$$

with λ defined according to the operational rule. By Lemma B.2, $add(\gamma^c,\alpha) \succ\!\xrightarrow{h_1}\! add(\gamma^s,\alpha)$ where $h_1 \restriction (E_{\gamma^c} \cap E_{add(\gamma^c,\alpha)}) = h_0$. Therefore if $h_2 : add(\gamma^s,\alpha) \succ\!\longrightarrow \mathsf{cf}(add(\gamma^s,\alpha))$ then it follows that

$$h_3 = h_2 \circ h_1 : add(\gamma^c,\alpha) \succ\!\longrightarrow \mathsf{cf}(add(\gamma^s,\alpha))$$

because of composition of morphisms. moreover, since $\mathsf{cf}(add(\gamma^s,\alpha))$ is L-canonical by definition, then it is th canonical form of $add(\gamma^c,\alpha)$ via h_3 which is unique up to isomorphism (cf. Theorem 6.16). But then by definition of $\mathcal{R}$ we have:

$$(\mathsf{skip}, add(\gamma^c,\alpha)), (\mathsf{skip}, \mathsf{cf}(add(\gamma^s,\alpha))), h_3) \in \mathcal{R}$$

which proves (i).

For condition (ii) we must show that $id \rhd \lambda$ via h_1 and h_3. The first condition of Definition 3.17 has been already proved. For the second condition, let $e_1 \in E'$ then we have:

$$\begin{array}{rcll} \lambda(e_1, h_{\mathsf{cf}}(e_1)) & = & \mathcal{C}(e_1) & [\text{by def } \lambda \text{ in } (\mathsf{New\text{-}sym})] \\ & = & \lceil |h_0^{-1}(e_1)| \rceil & [\text{by morphism def.}] \\ & = & \sum_{\tilde{e} \in h_0^{-1}(e_1)} id(\tilde{e},\tilde{e}) & [\text{by def. } id \text{ and } \mathcal{C}(\tilde{e}) = 1] \\ & = & \sum_{(e_1, h_2(e_1)) = (h_0(\tilde{e}), h_3(\tilde{e}))} id(\tilde{e},\tilde{e}) & [h_0(\tilde{e}) = e_1 \Rightarrow h_3(\tilde{e}) = h_2(e_1)] \end{array}$$

The property (No-Cross) is straightforward since the execution of the statement boils down to the application of $add(\gamma^c,\alpha)$ that does not perform rearrangement of links between entities (cf. (13) page 26) and because id does not produce any permutation of the entities. Also condition 4 of $\rhd$ is straightforward. In fact the only new entity in the concrete level is mapped by the morphism of the canonical form onto the new entity of the symbolic level (that is concrete). To see this, let $E_{add(\gamma^c,\alpha)} \backslash \mathrm{cod}(id) = \{e\}$ and $(E_{add(\gamma^s,\alpha)} \backslash \mathrm{cod}(\lambda) = \{e'\}$. By Lemma B.2 it must be $(h_2 \circ h_1)^{-1}(e') = e$ otherwise $h_1 \restriction (E_{\gamma^c} \cap E_{add(\gamma^c,\alpha)}) = h_0$ would be contradicted (see also the proof of the lemma).

– **case** $r = \mathsf{del}(\alpha.a)$**.** In this case, on the concrete level the only possible transition is:

$$\mathsf{del}(\alpha.a), \gamma^c \rightarrow_{id} \mathsf{skip}, cancel(\gamma^c,\alpha.a)$$

On the symbolic level:

$$\mathsf{del}(\alpha.a), \gamma^s \rightarrow_\lambda \mathsf{skip}, \mathsf{cf}(\mathit{cancel}(\gamma^s, \alpha.a))$$

where λ is defined as the operational rule. Again by Lemma B.2 we have

$$\mathit{cancel}(\gamma^c, \alpha.a) \rightarrowtail^{h_1} \mathit{cancel}(\gamma^s, \alpha.a)$$

and if $\mathit{cancel}(\gamma^s, \alpha.a) \rightarrowtail^{h_2} \mathsf{cf}(\mathit{cancel}(\gamma^s, \alpha.a))$ then $\mathsf{cf}(\mathit{cancel}(\gamma^s, \alpha.a))$ is also the canonical form of $\mathit{cancel}(\gamma^c, \alpha.a)$ by the morphism $h_3 = h_2 \circ h_1$. This implies by definition of $\mathcal{R}$:

$$((\mathsf{skip}, \mathit{cancel}(\gamma^c, \alpha.a)), (\mathsf{skip}, \mathsf{cf}(\mathit{cancel}(\gamma^s, \alpha.a))), h_3) \in \mathcal{R}.$$

Moreover, λ defined in the del rule is the same as the one in the new rule, therefore, we have already shown in the case $\mathsf{new}(\alpha.a)$ that conditions 1, 2, 3, of $\rhd$ definition hold. For condition 4 it is enough to observe that it is trivially true since the set of new entities is empty both at the concrete as well as at the symbolic level. Hence we can conclude that also for the case of del we have $id \rhd \lambda$.

– **case** $r = \alpha_1.a := \alpha_2.a$**.** By the rules of the concrete transition system, the only possible transition is:

$$\alpha_1.a := \alpha_2.a, \gamma^c \rightarrow_{id} \mathsf{skip}, \mathit{modify}(\gamma^c, \alpha_1.a, \alpha_2.a)$$

Let us indicate by γ^c_{trg} the target state (of the previous concrete transition). On the symbolic level we have:

$$\alpha_1.a := \alpha_2.a, \gamma^s \rightarrow_{id} \mathsf{skip}, \mathsf{cf}(\gamma_x)$$

for $(\gamma_x, h_x) \in \mathsf{SExp}(\gamma')$ where $\gamma' = \mathit{modify}(\gamma^s, \alpha_1.a, \alpha_2.a)$ and $\lambda = h_{\mathsf{cf}} \circ h_x^{-1}(\gamma^s \overset{id}{\Longrightarrow} \gamma')$. As above, let us refer to the target configuration of the previous symbolic transition by γ^s_{trg}.

Since this rule is nondeterministic, i.e., there can be more than one $\mathsf{cf}(\gamma_x)$, we must show that there exists a morphism h_2 for *one* of the possible $\mathsf{cf}(\gamma_x)$ resulting as target state of the rule.

Lemma B.2 allows us to conclude that there exists a morphism from γ^c_{trg} and γ' and this is precisely h_0 (cf. Figure 30 left lower sub-diagram 1). Lemma B.1 then ensures the existence of a morphism h^c_x for one $\gamma_x \in \mathsf{SExp}(\gamma')$. Thus, we can define $h_2 : \gamma^c_{trg} \rightarrowtail \mathsf{cf}(\gamma_x)$ as composition of morphisms $h_2 = h_{\mathsf{cf}}(\gamma_x) \circ h^c_x$ that by Proposition 3.7 is indeed a morphism (cf. Figure 30, sub-diagram 3). But then since γ^s_{trg} is L-canonical, h_2 is the (unique) morphism that relate γ^c_{trg} to its normal form, that implies

$$((\mathsf{skip}, \gamma^c_{trg}), (\mathsf{skip}, \gamma^s_{trg}), h_2) \in \mathcal{R}.$$

Now, we show that $id \rhd \lambda$. As for the previous statement the first condition of $\rhd$ is straightforward. In order to prove the the second, first of all observe the interdependence of morphisms and reallocation depicted in Figure 30, and in particular, as we have already seen, by Lemma B.2, the morphism between γ^c_{trg} and γ' is in fact h_0. Hence, let $e_1 \in E_{\gamma'}$ and $e_2 \in E_{\gamma^s_{trg}}$, then we have:

$$\begin{aligned}
\lambda : (e_1, e_2) &= \sum\!\!\!\!\!\!\bigcirc_{\tilde{e} \in h_x^{-1}(e_1) \cap h^{-1}_{\mathsf{cf}(\gamma_x)}(e_1)} \mathcal{C}(\tilde{e}) && \text{[by def } \lambda \text{ in (ASGN-rule)]} \\
&= |(h^c_x)^{-1}(h_x^{-1}(e_1) \cap h^{-1}_{\mathsf{cf}(\gamma_x)}(e_2))| && \text{[by Figure 30]} \\
&= |h_0^{-1}(e_1) \cap h_2^{-1}(e_2)| && \text{[by Figure 30]} \\
&= \sum\!\!\!\!\!\!\bigcirc_{\tilde{e} \in h_0^{-1}(e_1) \cap h_2^{-1}(e_2)} id(\tilde{e}, \tilde{e}) \\
&= \sum\!\!\!\!\!\!\bigcirc_{(e_1, e_2) = (h_0(\tilde{e}), h_2(\tilde{e}))} id(\tilde{e}, \tilde{e}).
\end{aligned}$$

Finally, the reallocation id does not allow crossings, in particular because link updates occur only on entities with cardinality 1. Therefore the condition (No-Cross) is satisfied. Condition 4 of $\rhd$ is trivially fulfilled since no new entities are created in the concrete and symbolic transition. We conclude that $id \rhd \lambda$.

Inductive step.

We show now only one case related to inductive step, namely the sequential composition, since the others can be shown in the same way. Assume a concrete state

$$q^c \equiv s_1; s_2, \gamma_1^c$$

and a symbolic

$$q^s \equiv s_1; s_2, \gamma_1^s.$$

According to the concrete transition system, state q^c can only perform a single transition

$$s_1; s_2, \gamma_1^c \rightarrow_{id} s_1'; s_2, \gamma_2^c$$

corresponding to a transition

$$s_1, \gamma_1^c \rightarrow_{id} s_1', \gamma_2^c$$

for some statement s_1' completely determined by the structure of s_1. On the symbolic level to the transition performed by q^s corresponds to:

$$s_1; s_2, \gamma_1^s \rightarrow_{\lambda} s_1'; s_2, \gamma_2^s$$

for some λ determined by the transition

$$s_1, \gamma_1^s \rightarrow_{\lambda} s_1', \gamma_2^s.$$

By induction hypothesis we have $((s_1, \gamma_1^c), (s_1, \gamma_1^s), h_1) \in \mathcal{R}$. Therefore:

(a) there exists h_2 such that $((s_1', \gamma_2^c), (s_1', \gamma_2^s), h_2) \in \mathcal{R}$ and

(b) $id \rhd \lambda$.

But then by (a), (b) we have that:

* $h_1 : q^c \succ\!\!\longrightarrow q^s$
* $((s_1'; s_2, \gamma_2^c), (s_1'; s_2, \gamma_2^s), h_2) \in \mathcal{R}$
* $id \rhd \lambda$.

Then by definition, it is $(q^c, q^s, h_1) \in \mathcal{R}$.

We conclude that the relation $\mathcal{R}$ is a simulation.

Now in order to prove that $\mathcal{A}_p \sqsubseteq \mathcal{H}_p$ we show that condition 2 of Definition 4.1 is satisfied by the simulation $\mathcal{R}$.

2.a) Both automata $id(\mathcal{A}_p)$ and $\mathcal{H}_p$ have only one single initial state with the same configuration, namely $(PV, \varnothing, \mathbf{1}_{PV})$ that is also L-canonical. But the we have:

$$((r, (PV, \varnothing, \mathbf{1}_{PV})), (r, (PV, \varnothing, \mathbf{1}_{PV})), id_{(PV, \varnothing, \mathbf{1}_{PV})}) \mathsf{in}\ \mathcal{R}.$$

where r is the initial statement of the program p.

The second condition 2.a) that relates the set of new entities N and the initial valuation θ of the initial states is trivial since — as we have seen — both $id(\mathcal{A}_p)$ and $\mathcal{H}_p$ have the same single initial state.

2.b) According to the definition of accept state for $id(\mathcal{A}_p)$ and $\mathcal{H}_p$ have the following sets:

$$\begin{aligned}\mathcal{F}_{id(\mathcal{A}_p)} &= \{\hat{F}_1^c, \ldots, \hat{F}_k^c, \tilde{F}_1^c, \ldots, \tilde{F}_k^c\}\\ \mathcal{F}_{\mathcal{H}_p} &= \{\hat{F}_1^s, \ldots, \hat{F}_k^s, \tilde{F}_1^s, \ldots, \tilde{F}_k^s\}\end{aligned}$$

therefore as $\psi : \mathcal{F}_{id(\mathcal{A}_p)} \to \mathcal{F}_{\mathcal{H}_p}$ we naturally take:

$$\begin{aligned}\psi(\hat{F}_i^c) &= \hat{F}_i^s\\ \psi(\tilde{F}_i^c) &= \tilde{F}_i^s\end{aligned}$$

for $1 \leqslant i \leqslant k$. It is straightforward to see that ψ is bijective. Now, let $F \in \mathcal{F}_{id(\mathcal{A}_p)}$ and $q = (s, \gamma) \in F$, we must prove that there exists $q' \in \psi(F)$ and a morphism $h : q \rightarrowtail q'$ such that $(q, q', h) \in \mathcal{R}$. Let $q' = (s, \mathsf{cf}(\gamma))$, the existence of $\mathsf{cf}(\gamma)$ is ensured by Theorem 6.16 since q — being concrete — is L-safe by definition as well as PV-reachable by Proposition 6.10. Having a L-canonical and well-formed configuration implies that $q' \in Q_{\mathcal{H}_p}$. Moreover, $q' \in \psi(F)$ since q and q' have the same statement s. But then by definition of $\mathcal{R}$ we have $(q, q', h) \in \mathcal{R}$.

Since condition 2.a) and 2.b) of Definition 4.1 hold we finally conclude that $id(\mathcal{A}_p) \sqsubseteq \mathcal{H}_p$. □

Lemma B.3. Let $E_{\max} \geqslant 0$ be a fixed constant. If for all $q \in Q_{\mathcal{H}_p} : |E_q| \leqslant E_{\max}$ then $|Q_{\mathcal{H}_p}|$ is finite.

Proof. $\mathcal{H}_p$ states are of the form $q = (s, E, \mu, \mathcal{C})$. Let $s_{\max}$ be the longest sequential component in s and let m be the number of sequential components. The number of possibilities for s is bound by $|s_{\max}|^m$. Because of the use of reallocations we can consider states with entities up to renaming, i.e., we can always use entities within a finite set with $E_{\max}$ elements (by hypothesis $|E| \leqslant E_{\max}$)[11]. Furthermore, the component μ ranges over the set of all possible applications from E onto $E^{\perp}$. Therefore, fixing E we have $(|E|+1)^{|E|}$ different possible μ. Finally the component $\mathcal{C}$ is the set of all mappings from E to the set $\{1, \ldots, M\} \cup \{*\}$ therefore, we have $(M+1)^{|E_{\max}|}$ possibilities. Summarising we have:

$$\begin{aligned}|Q_{\mathcal{H}_p}| &\leqslant |s_{\max}|^m \cdot 2^{E_{\max}} \cdot \sum_{n=0}^{E_{\max}} (n+1)^n \cdot \sum_{n=0}^{E_{\max}} (M+1)^n\\ &\leqslant |s_{\max}|^m \cdot 2^{E_{\max}} \cdot \frac{(E_{\max}+1)^{E_{\max}} - 1}{E_{\max}} \cdot \frac{(M+1)^{E_{\max}} - 1}{M}\end{aligned}$$

Hence, $Q_{\mathcal{H}_p}$ is finite. □

Theorem 6.29 For all programs p, $\mathcal{H}_p$ is finite-state.

Proof. By contradiction, assume $\mathcal{H}_p$ is infinite-state. Therefore, by Lemma B.3 there does not exist a bound on the number of entities in the states, i.e., there does not exist a constant, say $E_{\max} \geqslant 0$ such that for all $q \in Q_{\mathcal{H}_p}$: $|E_q| \leqslant E_{\max}$. Since every state has a canonical and therefore PV-reachable (cf. Proposition 6.14) configurations, this can be the case only for two reasons:

- either the number of PV is infinite,
- or in the state there are unbounded chains of entities.

[11] As we have seen, this of course does not prevent to express unbounded number of entity creations.

The first possibility is impossible because, by definition, PV is finite. The second is also impossible. In fact, taking an unbounded chain, the number of entities with indegree greater than one is either unbounded or bounded. On the one hand, if we assume that the number of such entities is unbounded, then since the state is PV-reachable this again would imply that PV is infinite. On the other hand, if there is a bounded number of entities with indegree greater than one, then since the chain is unbound it must be that after a certain point we have an unbounded number of entities with indegree equal one, i.e., the chain becomes pure. Again this is not possible since states are in canonical form. □

C Proofs of Section 7

Theorem 7.5 For all HABA $\mathcal{H}$ such that $\mathcal{C}(\mathcal{H}) = M < \hat{M}$: $\mathcal{L}(\mathcal{H}) = \mathcal{L}(\mathcal{H} \Uparrow \hat{M})$.

Proof. $[\mathcal{L}(\mathcal{H}) \subseteq \mathcal{L}(\mathcal{H} \Uparrow \hat{M})]$ Let $(\sigma, N, \theta) \in \mathcal{L}(\mathcal{H})$, with $\sigma = E_0\lambda_0 E_1\lambda_1 \cdots$. Let $\rho = q_0\lambda_0 q_1\lambda_1 \cdots$ be the run generating (σ, N, θ) by some generator $(h_i)_{i\in\mathbb{N}}$. We define a run ρ' of $\mathcal{H} \Uparrow \hat{M}$ that generates (σ, N, θ) in the following way. Let $\rho' = q'_0\lambda_0 q'_1\lambda'_1 \cdots$ where for all $i \geqslant 0$, we take $q'_i \in S_{q_i}$ such that:

$$\forall \in E_{q'_i} : \lceil |h_i^{-1}(e)| \rceil_{\hat{M}} = \mathcal{C}_{q'_i}(e).$$

By construction of the states in ρ' and by the definition of $\mathcal{H} \Uparrow \hat{M}$ for all $i \geqslant 0$ there exists a transition $q'_i \rightarrow_{\lambda_i} q'_{i+1}$. As q'_0 is an initial state of $\mathcal{H} \Uparrow \hat{M}$ and for every accept state q_i visited infinitely often, also the corresponding accept state q'_i is visited infinitely often, we conclude that $\rho' \in \mathsf{runs}(\mathcal{H} \Uparrow \hat{M})$. Finally, it is possible to show that generator $(h_i)_{i\in\mathbb{N}}$ generates (σ, N, θ) also from run ρ'. Hence we conclude that $(\sigma, N, \theta) \in \mathcal{L}(\mathcal{H} \Uparrow \hat{M})$.

$[\mathcal{L}(\mathcal{H} \Uparrow \hat{M}) \subseteq \mathcal{L}(\mathcal{H})]$ Let $(\sigma, N, \theta) \in \mathcal{L}(\mathcal{H} \Uparrow \hat{M})$, with $\sigma = E_0\lambda_0 E_1\lambda_1 \cdots$, and let $\rho = q'_0\lambda_0 q'_1\lambda_1 \cdots$ be the run generating (σ, N, θ). We define $\rho = q_0\lambda_0 q_1\lambda_1 \cdots$ such that for all $i > 0$, where $q'_i \in S_{q_i}$. Since $\rho' \in \mathsf{runs}(\mathcal{H} \Uparrow \hat{M})$ implies $\rho \in \mathsf{runs}(\mathcal{H})$ and ρ' generates (σ, N, θ) by a generator $(h_i)_{i\in\mathbb{N}}$ implies of ρ generates (σ, N, θ) by the same $(h_i)_{i\in\mathbb{N}}$, we conclude that $(\sigma, N, \theta) \in \mathcal{L}(\mathcal{H})$. □

Proposition 7.7 For all γ-valuation (ψ, Θ, δ):

(a) $(\exists j > 0 : \mu^j \circ \Theta(x) = \Theta(y) \neq \bot) \Rightarrow (x, y) \in \mathrm{dom}(\delta)$

(b) $\Theta(x) = \Theta(y) \neq \bot \Rightarrow (x, y) \in \mathrm{dom}(\delta) \vee (y, x) \in \mathrm{dom}(\delta)$

Proof.

(a) If there exists $j > 0$ such that $\mu^j \circ \Theta(x) = \Theta(y)$ then by (Delta Gamma 2) it can be proved that for all $0 < i \leqslant j$ we have $\delta(\mu^{i-1} \circ \Theta(x)^+, \mu^i \circ \Theta(x)^-) = 1$ that in turn implies $(\mu^{i-1} \circ \Theta(x)^+, \mu^i \circ \Theta(x)^-) \in \mathrm{dom}(\delta)$. Similarly by (Delta Gamma 1), it follows that for all $0 < i \leqslant j$: $(\mu^i \circ \Theta(x)^-, \mu^i \circ \Theta(x)^+) \in \mathrm{dom}(\delta)$. Combining these facts, we have $(\Theta(x)^-, \mu^j \circ \Theta(x)^+) = (\Theta(x)^-, \Theta(y)^+) \in \mathrm{dom}(\delta)$ and moreover, $j > 0$ implies $(\Theta(x)^+, y) \in \mathrm{dom}(\delta)$. By (Delta Theta 2) $(x, \Theta(x)^+) \in \mathrm{dom}(\delta)$. Therefore, we can conclude using (Delta Met 2) that $(x, y) \in \mathrm{dom}(\delta)$.

(b) Since $\Theta(x) = \Theta(y) \neq \bot$ we have by (Delta Theta 2):

$$\delta(\Theta(x)^-, x) \geqslant 0 \ \wedge \delta(\Theta(x)^-, y) \geqslant 0$$
$$\delta(x, \Theta(x)^+) \geqslant 0 \ \wedge \delta(y, \Theta(x)^+) \geqslant 0$$

Therefore by (Delta Met 3) we have either $\delta(\Theta(x)^-, x) \oplus \delta(x, y) = \delta(\Theta(x)^-, y)$ or $\delta(\Theta(x)^-, y) \oplus \delta(y, x) = \delta(\Theta(x)^-, x)$. Hence we have either $(x, y) \in \mathrm{dom}(\delta)$ (corresponding to the first case) or $(y, x) \in \mathrm{dom}(\delta)$ (corresponding to the second case).

□

Proposition 7.19 For $q_1 \stackrel{\lambda}{\Longrightarrow} q_2$, if E is a maximal weakly connected subgraph of $(E_{q_1} \uplus E_{q_2}, \lambda)$ such that $\bot \notin E$ then $\mathcal{C}_{q_1}(\iota_1(E)) = \mathcal{C}_{q_2}(\iota_2(E))$.

Proof. Without loss of generality let $\iota_1(E) = \{e_1, \ldots, e_n\}$ and $\iota_2(E) = \{e'_1, \ldots, e'_k\}$ for $k, n \geqslant 1$. Since E is a maximal weakly connected subgraph, we have the property:

- $\forall e \in \iota_1(E) : \lambda(e) \in \iota_2(E)$
- $\forall e \in \iota_2(E) : \lambda^{-1}(e) \in \iota_1(E)$.

Thus we can construct a $n \times k$ matrix R where $R(r, c) = \lambda(e_r, e_c)$ (with $1 \leqslant r \leqslant n$ and $1 \leqslant c \leqslant k$). Hence we have:

$$
\begin{array}{lll}
& \mathcal{C}_1(\iota_1(E)) & \\
= & \mathcal{C}_1(e_1) + \cdots + \mathcal{C}_1(e_n) & \text{[by definition]} \\
= & \sum_{j=1}^{k} R(1, j) + \cdots + \sum_{j=1}^{k} R(n, j) & \text{[by def. of } R \text{ and Def 3.12 cond. 1]} \\
= & \sum_{j=1}^{n} R(j, 1) + \cdots + \sum_{j=1}^{n} R(j, k) & \text{[sorting by column, by def. of } R \text{ and Def 3.12 cond. 2]} \\
= & \mathcal{C}_2(e'_1) + \cdots + \mathcal{C}_2(e'_k) & \\
= & \mathcal{C}_2(\iota_2(E)). &
\end{array}
$$

□

We will now prove few lemmas that that will be used in the proof of Proposition 7.25. These proofs use the following notion. Let γ' and γ be a concrete and an abstract configuration, respectively. Let $\theta : fv(\psi) \rightharpoonup E_{\gamma'}$ be an interpretation for the logical variables over γ'. We extend θ to the set of special logical variables $E^{\pm}_{\gamma}$ as follows. Let $h : \gamma' \rightarrowtail \gamma$ and $e \in E_\gamma$, then:

$$
\theta^{\pm}(x) = \begin{cases} \theta(x) & \text{if } x \in fv(\psi) \\ \mathit{first}(h^{-1}(e)) & \text{if } x = e^- \\ \mathit{last}(h^{-1}(e)) & \text{if } x = e^+ \end{cases} \tag{37}
$$

The next definition provides us with a consistency notion between a concrete interpretation for logical variables θ (in γ') w.r.t. a symbolic interpretation given by Θ, δ in a γ-valuation when $\gamma' \stackrel{h}{\rightarrowtail} \gamma$.

Definition C.1. Let $h : \gamma' \rightarrowtail \gamma$ be a morphism where $\mathcal{C}_{\gamma'} = \mathbf{1}$, (ψ, Θ, δ) a γ-valuation and $\theta : \text{LVAR} \rightharpoonup E_{\gamma'}$. Then, (θ, h) *is consistent with* (Θ, δ) (written $(\theta, h) \fallingdotseq (\Theta, \delta)$) if

(a) $h \circ \theta = \Theta$

(b) $\forall x, y \in fv(\psi) \cup E^{\pm}_{\gamma} : \delta(x, y) = \min\{\lceil n \rceil \mid \mu^n_{\gamma'} \circ \theta^{\pm}(x) = \theta^{\pm}(y)\}$ where $\min \varnothing = \bot$.

We use the previous definition in order to keep consistency, from state to state, between the interpretation given by the valuations of a path π and the interpretation of an allocation sequence σ generated by π.

Lemma C.2. Let $\pi = (q_0, D_0)\lambda_0(q_1, D_1)\lambda_1 \cdots$ be a path and let σ be an allocation sequence generated by the underlying run of π with generator $(h_j)_{j \in \mathbb{N}}$ and $\psi \in CL(\phi)$. Then for all $i \geqslant 0$

$$
(\theta, h_i) \fallingdotseq (h_i \circ \theta, \delta) \quad \Rightarrow \quad (\psi, h_{i+1} \circ \lambda^\sigma_i \circ \theta, \delta') \in [\lambda_i \circ (\psi, h_i \circ \theta, \delta)]_.
$$

for any δ' such that $(\lambda^\sigma_i \circ \theta, h_{i+1}) \fallingdotseq (h_{i+1} \circ \lambda^\sigma_i \circ \theta, \delta')$.

Proof. First of all, we recall that a δ' with the property $(\lambda^\sigma_i \circ \theta, h_{i+1}) \fallingdotseq (h_{i+1} \circ \lambda^\sigma_i \circ \theta, \delta')$ is defined (according to Definition C.1) as

$$
\delta'(x, y) = \min\{\lceil n \rceil \mid \mu^n \circ (\lambda^\sigma_i \circ \theta)^{\pm}(x) = (\lambda^\sigma_i \circ \theta)^{\pm}(y)\}
$$

for $x, y \in fv(\psi) \cup E^{\pm}_{q_{i+1}}$. Now, for brevity, let $\Theta = h_i \circ \theta$ and $\Theta' = h_{i+1} \circ \lambda^{\sigma}_i \circ \theta$. By definition of generator (Def. 3.24) we have that $\lambda^{\sigma}_i \rhd \lambda_i$ and therefore

$$\lambda_i(\Theta(x), \Theta'(x)) \neq 0. \tag{38}$$

Hence, condition 1 of Definition 7.20 is fulfilled. We now prove condition 2.

Let E be a maximal (weakly) connected subgraph of $(E_{q_i} \uplus E_{q_{i+1}}, \lambda_i)$. Proposition 7.19 and the property (No-Cross) of $\rhd$ (Def. 3.17) imply:

$$|h_i^{-1}(\iota_1(E))| = |h_{i+1}^{-1}(\iota_2(E))|. \tag{39}$$

That is: the number of entities on the concrete level that correspond to the first and the second projection of the maximal connected subgraph is the same[12]. For brevity, let $C = |h_i^{-1}(\iota_1(E))|$ and let

$$\begin{aligned} h_i^{-1}(\iota_1(E)) &= e^i_1, \quad \ldots, \quad e^i_C \\ h_{i+1}^{-1}(\iota_2(E)) &= e^{i+1}_1, \quad \ldots, \quad e^{i+1}_C. \end{aligned}$$

Another consequence of (No-Cross) is that for all $1 \leqslant m \leqslant C$:

$$\lambda^{\sigma}_i(e^i_m, e^{i+1}_m) \neq 0 \tag{40}$$

that is, the entity at position m in the chain corresponding to $\iota_1(E)$ is reallocated in the entity at position m in the chain corresponding to $\iota_2(E)$. Therefore, it is straightforward to see that the interpretation of a variable x on $\iota_1(E)$ will have the same position in $\iota_2(E)$, in particular, the distance between $\mathit{first}(\iota_2(E))^-$ and $\mathit{last}(\iota_2(E))^+$ is unchanged w.r.t. $\mathit{first}(\iota_1(E))^-$ and $\mathit{last}(\iota_1(E))^+$. Hence, conditions 2(a)-(b) of Definition 7.20 hold.

For the other free variables y with interpretation both on $\iota_1(E)$ and $\iota_2(E)$ we have:

$$\mu^n_i(\theta(x)) = \theta(y) \;\Rightarrow\; \mu^n_{i+1}(\lambda^{\sigma}_i \circ \theta(x)) = \lambda^{\sigma}_i \circ \theta(y) \;\Rightarrow\; \delta'(x, y) = \lceil n \rceil \tag{41}$$

Moreover, since $(\theta, h_i) \leftrightarrows (h_i \circ \theta, \delta)$ we have:

$$\mu^n_i(\theta(x)) = \theta(y) \;\Rightarrow\; \delta(x, y) = \lceil n \rceil. \tag{42}$$

Thus, from (41) and (42) it follows that δ' satisfies condition 2(c) of Definition 7.20. We conclude that $(\psi, h_{i+1} \circ \lambda^{\sigma}_i \circ \theta, \delta') \in [\lambda_i \circ (\psi, h_i \circ \theta, \delta)]$. □

Lemma C.3. Let $\pi = (q_0, D_0)\lambda_0(q_1, D_1)\lambda_1 \cdots$ be a path and let σ be an allocation sequence generated by the underlying run of π with generator $(h_j)_{j \in \mathbb{N}}$. Then for all $j > i \geqslant 0$,

a) if $(\lambda^{\sigma}_{j-1} \circ \cdots \lambda^{\sigma}_i \circ \theta, h_j) \leftrightarrows (h_j \circ \lambda^{\sigma}_{j-1} \circ \cdots \lambda^{\sigma}_i \circ \theta, \delta)$ then

$$(\psi, h_{j+1} \circ \lambda^{\sigma}_j \circ \cdots \circ \lambda^{\sigma}_i \circ \theta, \delta') \in [\lambda_j \circ (h_j \circ \lambda^{\sigma}_{j-1} \circ \cdots \circ \lambda^{\sigma}_i \circ \theta, \delta)]$$

for any δ' such that $(\lambda^{\sigma}_j \circ \cdots \circ \lambda^{\sigma}_i \circ \theta, h_{j+1}) \leftrightarrows (h_{j+1} \circ \lambda^{\sigma}_j \circ \cdots \circ \lambda^{\sigma}_i \circ \theta, \delta')$.

b) if $(\theta, h_i) \leftrightarrows (h_i \circ \theta, \delta)$ then

$$(\psi, h_j \circ \lambda^{\sigma}_{j-1} \circ \cdots \circ \lambda^{\sigma}_i \circ \theta, \delta') \in [\lambda_{j-1} \circ \cdots \circ \lambda_i \circ (\psi, h_i \circ \theta, \delta)]$$

for any δ' such that $(\lambda^{\sigma}_{j-1} \circ \cdots \circ \lambda^{\sigma}_i \circ \theta, h_j) \leftrightarrows (h_j \circ \lambda^{\sigma}_{j-1} \circ \cdots \circ \lambda^{\sigma}_i \circ \theta, \delta')$.

Proof.

[12] Note that (No-Cross) is necessary since if the cardinalities on the abstract level correspond, in case of unbounded entities, this does not necessarily imply that also at the concrete level the corresponding cardinalities are the same. In fact, some entity may during the transition but this may not be revealed on the abstract level if the cardinality is $*$.

a) Straightforward. In fact, let $\theta' = \lambda^{\sigma}_{j-1} \circ \cdots \circ \lambda^{\sigma}_{i} \circ \theta$. Then by Lemma C.2 we have

$$(\psi, h_{j+1} \circ \lambda^{\sigma}_{j} \circ \theta', \delta') \in [\lambda_j \circ (\psi, h_j \circ \theta', \delta)]$$

and therefore: $(\psi, h_{j+1} \circ \lambda^{\sigma}_{j} \circ \cdots \circ \lambda^{\sigma}_{i} \circ \theta, \delta') \in [\lambda_j \circ (h_j \circ \lambda^{\sigma}_{j-1} \circ \cdots \circ \lambda^{\sigma}_{i} \circ \theta, \delta)]$ that in fact is what we wanted to prove.

b) We prove this part of the lemma by induction on $j \geqslant i$. The base step corresponds to Lemma C.2 therefore we show here the inductive step. For the sake of readability, let $\overline{\lambda_{j-1,i}} = \lambda_{j-1} \circ \cdots \circ \lambda_i$ and $\overline{\lambda^{\sigma}_{j-1,i}} = \lambda^{\sigma}_{j-1} \circ \cdots \circ \lambda^{\sigma}_{i}$. First of all we prove that

$$[\lambda_j \circ (\psi, h_j \circ \overline{\lambda^{\sigma}_{j-1,i}} \circ \theta, \delta)] \subseteq [\lambda_j \circ \overline{\lambda_{j-1,i}} \circ (\psi, h_i \circ \theta, \delta)]. \tag{43}$$

The set inclusion (43) can be proved using the induction hypothesis:

$$(\psi, h_j \circ \overline{\lambda^{\sigma}_{j-1,i}} \circ \theta, \delta') \in [\lambda_{j-1} \circ \overline{\lambda_{j-2,i}} \circ (\psi, h_i \circ \theta, \delta)]$$

as follows:

$$\begin{aligned}
& [\lambda_j \circ \overline{\lambda_{j-1,i}} \circ (\psi, h_i \circ \theta, \delta)] \\
= \;& \textstyle\bigcup_{v \in [\overline{\lambda_{j-1,i}} \circ (\psi, h_i \circ \theta, \delta)]} \lambda_j \circ v \\
= \;& \Big(\textstyle\bigcup_{v \in [\overline{\lambda_{j-1,i}} \circ (\psi, h_i \circ \theta, \delta)] \setminus \{(\psi, h_j \circ \overline{\lambda^{\sigma}_{j-1,i}} \circ \theta, \delta)\}} \lambda_j \circ v \Big) \cup \; [\lambda_j \circ (\psi, h_j \circ \overline{\lambda^{\sigma}_{j-1,i}} \circ \theta, \delta)]
\end{aligned}$$

Hence combining (43) and part a) of this lemma we conclude $(\psi, h_{j+1} \circ \lambda^{\sigma}_{j} \circ \cdots \lambda^{\sigma}_{i} \circ \theta, \delta) \in [\lambda_j \circ \cdots \circ \lambda_i \circ (\psi, h_i \circ \theta, \delta)]$.

□

Lemma C.4. Let $\gamma = (E, \mu, \mathbf{1})$ and γ' be two configurations such that $\gamma \overset{h}{\rightarrowtail} \gamma'$ and $v = (\psi, h \circ \theta, \delta)$ a γ'-valuation. Then

$$(h, \theta) \leftrightarrows (h \circ \theta, \delta) \;\Rightarrow\; h \circ \mu^n(\theta(x)) = [\![x.a^n]\!]^1_{\gamma,v}.$$

Proof. The intuition of this lemma is that the entity on the concrete level representing $x.a^n$ is mapped precisely on the entity on the abstract level that corresponds to the (first components of the) semantics of $x.a^n$.

We prove this lemma by induction on n.

- Base case $n = 0$. If $\theta(x) = \bot$ then $h \circ \theta(x) = \bot$ (is undefined) and by definition $[\![x]\!]^1_{\gamma,v} = \bot$. If $\theta(x) \neq \bot$ then we have $h \circ \mu^0(\theta(x)) = h \circ \theta(x) = [\![x.a^n]\!]^1_{\gamma,v}$ by Definition 7.13.
- Inductive step. Assume the statement holds for n. We prove it for $n+1$. If $\theta(x) = \bot$ then $\mu^n(\theta(x)) = \mu^{n+1}(\theta(x)) = \bot$, that implies $h \circ \mu^n(\theta(x))$ and $h \circ \mu^{n+1}(\theta(x))$ are undefined. Because of the hypothesis of consistency, we have immediately that also $h \circ \theta(x) = \bot$ therefore by definition $[\![x.a^n]\!]^1_{\gamma,v} = [\![x.a^{n+1}]\!]^1_{\gamma,v} = \bot$. Therefore, assume $\theta(x) \neq \bot$ and let $h^{-1}(h \circ \theta(x)) = e_1, \ldots, e_k$ and $\theta(x) = e_i$. We distinguish two cases.
  1. If $i + n + 1 \leqslant k$ then $h \circ \mu^n(\theta(x)) = h \circ \mu^{n+1}(\theta(x)) = h \circ \theta(x)$. Since $(h, \theta) \leftrightarrows (h \circ \theta, \delta)$, then $\delta(x, \Theta(x)^+) = \lceil k - i \rceil \geqslant n$ therefore by Definition 7.13 we have $[\![x.a^n]\!]^1_{\gamma,v} = h \circ \theta(x)$.
  2. If $i + n + 1 > k$ again we distinguish two cases:
     (a) Assume $h \circ \mu^n(\theta(x)) = h \circ \mu^{n+1}(\theta(x))$, that is $\mu^n(\theta(x))$ and $\mu^{n+1}(\theta(x))$ are mapped by the morphism h onto the same abstract entity and let $\Theta = h_i \circ \theta$. By the induction hypothesis we have that $h \circ \mu^n(\theta(x)) = [\![x.a^n]\!]^1_{\gamma',v}$. Moreover, if $[\![x.a^n]\!]^1_{\gamma',v} = \mu^j_{\gamma'} \circ \Theta(x)$ then
     $$\mathcal{C}_v(x, j) > n$$

where the inequality is strict since $x.a^n$ is not the last instance of $[\![x.a^n]\!]^1_{\gamma',v}$ otherwise $h \circ \mu^n(\theta(x))$ and $h \circ \mu^{n+1}(\theta(x))$ would not be the same contradicting the initial hypothesis. But then $\mathcal{C}_v(x,j) \geqslant n+1$ and by Definition 7.13 we have $[\![x.a^n]\!]^1_{\gamma,v} = [\![x.a^{n+1}]\!]^1_{\gamma,v}$. Summarising in one line:

$$h \circ \mu^{n+1}(\theta(x)) = h \circ \mu^n(\theta(x)) = [\![x.a^n]\!]^1_{\gamma,v} = [\![x.a^{n+1}]\!]^1_{\gamma,v}$$

that is precisely what we wanted to prove.

(b) As a second possibility, assume

$$h \circ \mu^{n+1}(\theta(x)) = \mu_{\gamma'}(h \circ \mu^n(\theta(x))) \tag{44}$$

that is $\mu^n(\theta(x))$ and $\mu^{n+1}(\theta(x))$ are mapped by the morphism h onto consecutive abstract entities. This initial hypothesis implies that $\mu^n(\theta(x))$ denotes precisely the *last* entity in the chain of concrete entities corresponding to $h^{-1}(h \circ \mu^n(\theta(x)))$. If this would not be the case then also $\mu^{n+1}(\theta(x))$ would be mapped by h on the same entity (since h is a morphism) therefore contradicting the initial hypothesis. Let $\Theta = h_i \circ \theta$ and assume that

$$[\![x.a^n]\!]^1_{\gamma',\Theta,\delta} = \mu^{j-1}_{\gamma'} \circ \Theta(x) \tag{45}$$

for some $j > 0$. Then, by Definition 7.13 we have:

$$n \leqslant \mathcal{C}_v(x, j-1) < n+1 \tag{46}$$

where the last inequality is true otherwise $[\![x.a^{n+1}]\!]^1_{\gamma',v} = \mu^{j-1}_{\gamma'} \circ \Theta(x)$ that would imply that $h \circ \mu^n \circ \theta(x)$ is not the last entity of $h^{-1}(h \circ \mu^n(\theta(x)))$.
Now, by (46) it is straightforward to see that $\mathcal{C}_v(x, j-1) = n$ and therefore $\mathcal{C}_v(x,j) \geqslant n+1$. By Definition 7.13 this implies $[\![x.a^{n+1}]\!]^1_{\gamma',v} = \mu^j_{\gamma'} \circ \Theta(x)$. Note that the entity $\mu^j_{\gamma'} \circ \Theta(x)$ does exist since it is the image (via h) of $\mu^{n+1} \circ \theta(x)$. Finally, we have:

$$\begin{array}{rcll} [\![x.a^{n+1}]\!]^1_{\gamma',v} & = & \mu^j_{\gamma'} \circ \Theta(x) & \text{[by definition]} \\ & = & \mu_{\gamma'} \circ \mu^{j-1}_{\gamma'} \circ \Theta(x) & \text{[by (45)]} \\ & = & \mu_{\gamma'}([\![x.a^n]\!]^1_{\gamma',v}) & \text{[by induction]} \\ & = & \mu_{\gamma'}(h \circ \mu^n \circ \theta(x)) & \text{[by (44)]} \\ & = & h \circ \mu^{n+1} \circ \theta(x). & \end{array}$$

□

Lemma C.5. Let $\rho = q_0\lambda_0 q_1 \lambda_1 \cdots$ be a run generating a triple (σ, N, θ) with generator $(h_i)_{i\in\mathbb{N}}$ such that $\sigma, N, \theta \models \phi$. Let

$$\begin{array}{rcl} D_i & = & \{(\psi, h_i \circ \theta, \delta_i) \mid \psi \in CL(\phi),\ \ \sigma^i, N^\sigma_i, \theta \models \psi\} \\ \forall x, y \in fv(\psi) \cup E^\pm_{q_i} : \delta_i(x,y) & = & \min\{\lceil n \rceil \mid \mu^n_i \circ \theta^\pm(x) = \theta^\pm(y)\} \end{array}$$

where $\theta^\pm$ is the extension of θ to the set $E^\pm_{q_i}$ as defined by (37) (cf. page 71). Then (q_i, D_i) is an atom for all $i \in \mathbb{N}$.

Proof. We show by induction on the structure of ψ that D_i satisfies the conditions of Definition 7.17.

- **Atomic propositions.** Give a valuation $v = (\psi, \Theta, \delta)$, where ψ is an atomic propositions, we have to prove the following implication:

$$v \in AV_{q_i} \ \Rightarrow\ v \in D_i. \tag{47}$$

To this end we need to prove the existence of a suitable (concrete) interpretation θ for $fv(\psi)$ and extend it according to (37) (cf. page 71). Suitable in this context means that the choice of $\theta(x)$ should be such that ψ is valid and — at the same time — $\Theta = h_i \circ \theta$ and $\delta = \delta_i$. We now single out the three possible cases occurring for the atomic proposition.

– case $\psi = x.a^n\ \mathsf{new}$. Let $v = (x.a^n\ \mathsf{new}, \Theta, \delta)$ and let $h_i^{-1}(\Theta(x)) = e_1, \ldots, e_k$ $(k \geqslant 1)$ and $\theta(x) = e_{i_x}$ where the index $1 \leqslant i_x \leqslant k$ is chosen according to the following rules:

$$i_x = \begin{cases} \delta(\Theta(x)^-, x) & \text{if } \delta(\Theta(x)^-, x) \neq * \ \wedge\ \delta(\Theta(x)^+, x) \neq * \\ k - \delta(x, \Theta(x)^+) & \text{if } \delta(\Theta(x)^-, x) = * \ \wedge\ \delta(\Theta(x)^+, x) \neq * \\ \delta(\Theta(x)^-, x) & \text{if } \delta(\Theta(x)^-, x) \neq * \ \wedge\ \delta(\Theta(x)^+, x) = * \end{cases} \tag{48}$$

Note furthermore, for the current case $x.a^n\ \mathsf{new}$, we cannot have $\delta(\Theta(x)^-, x) = \delta(x, \Theta(x)^+) = *$ otherwise this would imply $[\![x.a^n]\!]_{\gamma_{q_i}, v} = \Theta(x)$ and $\mathcal{C}(\Theta(x)) = *$ that is impossible since new entities cannot be unbounded. Hence the choice of i_x is deterministic. Given such θ, let as usual $\theta^{\pm}$ be its extension, by construction we have:

$$\begin{aligned} h_i \circ \theta^{\pm} &= \Theta \\ \delta(x, y) &= \delta_i(x, y) \end{aligned} \tag{49}$$

for all $x, y \in fv(\psi) \cup E_{q_i}^{\pm}$. Hence, we can conclude that:

$$v = (x.a^n\ \mathsf{new}, \Theta, \delta) = (x.a^n\ \mathsf{new}, h_i \circ \theta, \delta_i). \tag{50}$$

A consequence of (49) is that $(h_i, \theta) \fallingdotseq (\Theta, \delta)$ and by Lemma C.4 we have that $[\![x.a^n]\!]^1_{\gamma_{q_i}, v} = h_i \circ \mu^n(\theta(x))$. From $v \in AV_{q_i}$ it follows $[\![x.a^n]\!]^1_{\gamma_{q_i}, v} \in N_{q_i}$. Thus, by the properties of the generator (Def. 3.24) — in particular as stated by Lemma 3.19 — we have that new entities on the abstract level are related by the generator to the new entities on the concrete level, i.e., $\mu^n(\theta(x)) \in N_i^{\sigma}$ and therefore $\sigma^i, N_i^{\sigma}, \theta \models x.a^n\ \mathsf{new}$. Hence, by (50) and by definition of D_i, we conclude that $v \in D_i$ which proves (47).

– case $\psi = (x.a^n = y.a^m)$. Give a valuation $v = (x.a^n = y.a^m, \Theta, \delta) \in AV_{q_i}$, by Definition 7.15 we have $\Delta_{\gamma_{q_i}, v}(x.a^n, y.a^m) = 0$ that by definition implies

a) $[\![x.a^n]\!]^1_{\gamma,v} = [\![y, a^m]\!]^1_{\gamma,v} = \bot$
b) $[\![x.a^n]\!]^1_{\gamma,v} = [\![y, a^m]\!]^1_{\gamma,v} \neq \bot$ and $[\![x.a^n]\!]^2_{\gamma,v} \neq *$
c) $[\![x.a^n]\!]^1_{\gamma,v} = [\![y, a^m]\!]^1_{\gamma,v} \neq \bot$ and $[\![x.a^n]\!]^2_{\gamma,v} = *$ and $(\delta(x, y) \oplus m = n \vee \delta(y, x) \oplus n = m))$

Let us discuss here b) and c) only. We need to show the existence of a suitable θ. Assume:

$$\begin{aligned} h_i^{-1}(\Theta(x)) &= e_1, \ldots, e_{k_x} \\ h_i^{-1}(\Theta(y)) &= e_1, \ldots, e_{k_y} \end{aligned}$$

with $k_x, k_y \geqslant 1$. For the selection of $\theta(x)$ and $\theta(y)$ we apply the same strategy employed for $x.a^n\ \mathsf{new}$. Namely, let $\theta(x) = e_{i_x}$ and $\theta(y) = e_{i_y}$ where i_x, i_y are chosen according to (51) and (52), respectively.

$$i_x = \begin{cases} \delta(\Theta(x)^-, x) & \text{if } \delta(\Theta(x)^-, x) \neq * \ \wedge\ \delta(\Theta(x)^+, x) \neq * \\ k - \delta(x, \Theta(x)^+) & \text{if } \delta(\Theta(x)^-, x) = * \ \wedge\ \delta(\Theta(x)^+, x) \neq * \\ \delta(\Theta(x)^-, x) & \text{if } \delta(\Theta(x)^-, x) \neq * \ \wedge\ \delta(\Theta(x)^+, x) = * \\ M < j < k_x - M & \text{if } \delta(\Theta(x)^-, x) = * \ \wedge\ \delta(\Theta(x)^+, x) = * \end{cases} \tag{51}$$

$$i_y = \begin{cases} \delta(\Theta(y)^-, y) & \text{if } \delta(\Theta(y)^-, y) \neq * \ \wedge\ \delta(\Theta(y)^+, y) \neq * \\ k - \delta(y, \Theta(y)^+) & \text{if } \delta(\Theta(y)^-, y) = * \ \wedge\ \delta(\Theta(y)^+, y) \neq * \\ \delta(\Theta(y)^-, y) & \text{if } \delta(\Theta(y)^-, y) \neq * \ \wedge\ \delta(\Theta(y)^+, y) = * \\ M < j < k_y - M & \text{if } \delta(\Theta(y)^-, y) = * \ \wedge\ \delta(\Theta(y)^+, y) = * \end{cases} \tag{52}$$

Since when both $\delta(\Theta(y)^-, x) = \delta(\Theta(y)^+, x) = *$ and $\delta(\Theta(y)^-, y) = \delta(\Theta(y)^+, y) = *$ the indexes i_x, i_y are still not completely determined, then we select them such that the following conditions are satisfied:

$$\begin{aligned} i_x - i_y &= n - m \quad \text{if } \delta(y,x) \oplus m = n \\ i_y - i_x &= m - n \quad \text{if } \delta(y,x) \oplus n = m. \end{aligned}$$

By construction we have that $h_i \circ \theta^{\pm} = \Theta$ and $\delta = \delta_i$, or in other words, $(h_i, \theta) \leftrightarrows (\Theta, \delta)$. To prove that $v \in D_i$ it remains to show that by the previous choice of $\theta(x)$ and $\theta(y)$ we indeed have: $\sigma^i, N_i^\sigma, \theta \models x.a^n = y.a^m$.

Assume case b) and let $[\![x.a^n]\!]^2_{\gamma_{q_i},v} = l \neq *$ and $e = \mathit{first}(h_i^{-1}([\![x.a^n]\!]^1_{\gamma_{q_i},v}))$. Depending whether x is interpreted onto $[\![x.a^n]\!]_{\gamma_{q_i},v}$ or not, by definition we have two possibilities, namely:

(i) if $(x, [\![x.a^n]\!]^-_{\gamma_{q_i},v}) \in \mathrm{dom}(\delta)$ then by Definition 7.13 $\mathcal{C}_{\gamma_{q_i}}(x, j-1) = n - l - 1$ and therefore:

$$\mu_i^{n-l}(\theta(x)) = e \tag{53}$$

(ii) otherwise $([\![x.a^n]\!]^-_{\gamma_{q_i},v}, x) \in \mathit{dom}(\delta)$ in which case we have (again by Definition 7.13) $\delta([\![x.a^n]\!]^-_{\gamma_{q_i},v}, x) = l - n$ and by consistency of δ we have

$$\mu_i^{l-n}(e) = \theta(x). \tag{54}$$

Following the same lines, it is possible to show that depending on the interpretation of y w.r.t. $[\![y.a^m]\!]_{\gamma_{q_i},v}$ either

$$\mu_i^{m-l}(\theta(y)) = e \tag{55}$$

or

$$\mu_i^{l-m}(e) = \theta(y) \tag{56}$$

From the previous case distinction we can determine the precise interpretation on the concrete level for $\mu_i^n(\theta(x))$ and $\mu_i^m(\theta(y))$. In particular, from (53) and (54) we obtain:

$$\mu_i^n(\theta(x)) = \mu_i^l(\mu_i^{n-l}(\theta(x))) = \mu_i^l(e) \tag{57}$$
$$\mu_i^n(\theta(x)) = \mu_i^n(\mu_i^{l-n}(e)) = \mu_i^l(e) \tag{58}$$

For the variable y, from (55) and (56) we have

$$\mu_i^m(\theta(y)) = \mu_i^l(\mu_i^{m-l}(\theta(y))) = \mu_i^l(e) \tag{59}$$
$$\mu_i^m(\theta(y)) = \mu_i^m(\mu_i^{l-m}(e)) = \mu_i^l(e) \tag{60}$$

from which we can conclude that in case (b), in any possibility,

$$\mu_i^n(\theta(x)) = \mu_i^m(\theta(y)) = \mu_i^l(e).$$

Now, assume case (c). Let $\delta(x,y) = l$ (the case $\delta(y,x) = l$ is symmetrical). By hypothesis, we have $\delta(x,y) \oplus m = n$ that implies $l < K(\phi)$ (by assumption on stretching see page 7.1) and therefore, since δ is consistent we obtain $\mu_i^l(\theta(x)) = \theta(y)$. But then

$$\mu_i^n(\theta(x)) = \mu_i^{l+m}(\theta(x)) = \mu_i^m(\mu_i^l(\theta(x))) = \mu_i^m(\theta(y)).$$

Hence also in case (c) $\mu_i^n(\theta(x)) = \mu_i^m(\theta(y))$. Thus, by definition of the semantics of $\mathcal{N}\!a\ell\ell$TL for both cases (b) and (c) we conclude $\sigma^i, N_i^\sigma, \theta \models x.a^n = y.a^m$ and thus $v \in D_i$.

– case $x.a^n \rightsquigarrow y.a^m$.

Suppose $v = (x.a^n \rightsquigarrow y.a^m, \Theta, \delta) \in AV_{q_i}$. By definition of atomic valuation $\Delta(x.a^n, y.a^m) = \top$ or $\Delta(x.a^n, y.a^m) = 0$ and therefore be definition of Δ we have the following possibilities:

a) $[\![x.a^n]\!]^1_{\gamma,v} \neq [\![y.a^m]\!]^1_{\gamma,v}$ and $\exists j > 0 : \mu^j([\![x.a^n]\!]^1_{\gamma,v}) = [\![y.a^m]\!]^1_{\gamma,v}$

b) $[\![x.a^n]\!]^1_{\gamma,v} = [\![y.a^m]\!]^1_{\gamma,v} \neq \perp$ and $[\![x.a^n]\!]^2_{\gamma,v} < [\![y.a^m]\!]^2_{\gamma,v}$

c) $[\![x.a^n]\!]^1_{\gamma,v} = [\![y.a^m]\!]^1_{\gamma,v} \neq \perp$ and $[\![x.a^n]\!]^2_{\gamma,v} = [\![y.a^m]\!]^2_{\gamma,v} = *$ and $(\delta(x,y) \oplus m > n \ \vee \ \delta(y,x) \oplus n < m)$.

We define $\theta(x)$ and $\theta(y)$ according to (51) and (52) as we have done for the case of equality. Therefore we have by construction $\Theta = h_i \circ \theta$ and $\delta = \delta_i$. In order to have $v \in D_i$ it remains to show that $\sigma^i, N_i, \theta \models x.a^n \rightsquigarrow y.a^m$ for both a), b) as well as c).

a) It is straightforward that at the concrete level, there exists $j' \geqslant j$ such that $\mu_i^{n+j'}(\theta(x)) = \mu_i^m(\theta(y))$ and therefore by definition of the semantics of $\mathcal{N}\acute{a}\ell\ell$TL we can conclude $\sigma^i, N_i, \theta \models x.a^n \rightsquigarrow y.a^m$.

b) As we have done for case $x.a^n = y.a^m$, let $[\![x.a^n]\!]^2_{\gamma_{q_i} v} = l$ and $e = \mathit{first}(h_i^{-1}([\![x.a^n]\!]_{\gamma_{q_i},v}))$ then by (57) and (58) we have $\mu_i^n(\theta(x)) = \mu_i^l(e)$. By hypothesis there exists $l' \geqslant l$ such that $\mu_i^m(\theta(y)) = \mu_i^{l'}(e)$. But this implies $\mu_i^m(\theta(y)) = \mu_i^{l'-l}(\mu^n(\theta(x)))$ and by definition $\sigma^i, N_i, \theta \models x.a^n \rightsquigarrow y.a^m$.

c) if $\delta(x,y) \oplus m \geqslant n$ then $(x,y) \in \mathrm{dom}(\delta)$. We can have either $\delta(x,y) = *$, then on the concrete level since δ is consistent there must exist $l' > n$ such that $\mu_i^{l'}(\theta(x)) = \theta(y)$ that is enough to conclude that $x.a^n$ reaches $y.a^m$. Otherwise, $\delta(x,y) = c \neq *$ and since by hypothesis $c + m \geqslant n$ then, for the consistency of δ, at the concrete level $\mu_i^{c+m-n}(\mu^n(\theta(x))) = \theta(y)$. So we are done.
Finally if $\delta(y,x) \oplus n \leqslant m$ it is straightforward to verify that on the concrete level $x.a^n \rightsquigarrow y.a^m$.

Inductive step

- **case** $\psi = \neg\psi'$. According to the definition of atom we have to prove:

$$v = (\neg\psi', h_i \circ \theta, \delta) \in D_i \ \Leftrightarrow \ (\psi', h_i \circ \theta, \delta) \notin D_i.$$

Straightforward from the definition of D_i, and semantics of $\mathcal{N}\acute{a}\ell\ell$TL, in fact:

$$\begin{array}{lcl} v = (\neg\psi', h_i \circ \theta, \delta) \in D_i & \Leftrightarrow & \sigma^i, N_i^\sigma, \theta \models \neg\psi' \\ & \Leftrightarrow & \sigma^i, N_i^\sigma, \theta \not\models \psi' \\ & \Leftrightarrow & (\psi', h_i \circ \theta, \delta) \notin D_i \end{array}$$

- **case** $\psi = \psi_1 \vee \psi_2$. As the previous case, straightforward from the definition of D_i, and semantics of $\mathcal{N}\acute{a}\ell\ell$TL, in fact:

$$\begin{array}{lcl} (\psi_1 \vee \psi_2, h_i \circ \theta, \delta) \in D_i & \Leftrightarrow & \sigma^i, N_i^\sigma, \theta \models \psi_1 \vee \psi_2 \\ & \Leftrightarrow & \sigma^i, N_i^\sigma, \theta \restriction \psi_1 \models \psi_1 \text{ or } \sigma^i, N_i^\sigma, \theta \restriction \psi_2 \models \psi_2 \\ & \Leftrightarrow & (\psi_1, h_i \circ \theta \restriction \psi_1, \delta \restriction \psi_1) \in D_i \text{ or } (\psi_2, h_i \circ \theta \restriction \psi_2, \delta \restriction \psi_2) \in D_i. \end{array}$$

- **case** $\psi = \exists x.\psi'$.

[$\Rightarrow$] Suppose $(\exists x.\psi', h_i \circ \theta, \delta) \in D_i$ then $\sigma^i, N_i^\sigma, \theta \models \exists x.\psi'$. It follows that there exists an $e \in E_i^\sigma$ such that $\sigma^i, N_i^\sigma, \theta\{e/x\} \models \psi'$. By general assumption (see Page 42), $x \in \mathit{fv}(\psi')$ and hence $\mathrm{dom}(\theta\{e/x\}) \subseteq \mathit{fv}(\psi')$. Thus, $(\psi', h_i \circ \theta\{e/x\}, \delta_i') \in D_i$ where $\delta_i'(z,y) = \min\{\lceil n \rceil \mid \mu_i^n \circ \theta\{x/e\}^{\pm}(z) = \theta\{x/e\}^{\pm}(y)\}$.

[$\Leftarrow$] If $(\psi', h_i \circ \theta\{e/x\}, \delta_i') \in D_i$ where $\delta_i'(z,y) = \min\{\lceil n \rceil \mid \mu_i^n \circ \theta\{x/e\}^{\pm}(z) = \theta\{x/e\}^{\pm}(y)\}$ then $\sigma^i, N_i^\sigma, \theta\{e/x\} \models \psi'$. And therefore $\sigma^i, N_i^\sigma, \theta \models \exists x.\psi'$ and finally $(\exists x.\psi', h_i \circ \theta, \delta) \in D_i$ where $\delta = \delta_i' \restriction \psi$.

- **case** $\psi = \neg \mathsf{X}\psi'$. We have:

$$\begin{array}{lcl}(\neg \mathsf{X}\psi', h_i \circ \theta, \delta_i) \in D_i & \Leftrightarrow & \sigma^i, N_i^\sigma, \theta \models \neg \mathsf{X}\psi' \\ & \Leftrightarrow & \sigma^i, N_i^\sigma, \theta \not\models \mathsf{X}\psi' \\ & \Leftrightarrow & \sigma^{i+1}, N_{i+1}^\sigma, \lambda_i^\sigma \circ \theta \not\models \psi' \\ & \Leftrightarrow & \sigma^{i+1}, N_{i+1}^\sigma, \lambda_i^\sigma \circ \theta \models \neg\psi' \\ & \Leftrightarrow & \sigma^i, N_i^\sigma, \theta \models \mathsf{X}\neg\psi' \\ & \Leftrightarrow & (\mathsf{X}\neg\psi', h_i \circ \theta, \delta_i) \in D_i\end{array}$$

- **case** $\psi = \psi_1 \,\mathsf{U}\, \psi_2$. To prove this case we use the following known equivalence:

$$\sigma, N, \theta \models \psi_1 \,\mathsf{U}\, \psi_2 \;\Leftrightarrow\; \sigma, N, \theta \models \psi_2 \vee (\psi_1 \wedge \mathsf{X}(\psi_1 \,\mathsf{U}\, \psi_2))$$

for all allocation triple (σ, N, θ). Therefore we have:

$$\begin{array}{lcl}(\psi_1 \,\mathsf{U}\, \psi_2, h_i \circ \theta, \delta) \in D_i & \Leftrightarrow & \sigma^i, N_i^\sigma, \theta \models \psi_1 \,\mathsf{U}\, \psi_2 \\ & \Leftrightarrow & \sigma^i, N_i^\sigma, \theta \models \psi_2 \vee (\psi_1 \wedge \mathsf{X}(\psi_1 \,\mathsf{U}\, \psi_2)) \\ & \Leftrightarrow & \sigma^i, N_i^\sigma, \theta \restriction \psi_2 \models \psi_2 \text{ or} \\ & & (\sigma^i, N_i^\sigma, \theta \restriction \psi_1 \models \psi_1 \text{ and } \sigma^i, N_i^\sigma, \theta \models \mathsf{X}(\psi_1 \,\mathsf{U}\, \psi_2)) \\ & \Leftrightarrow & (\psi_2, h_i \circ \theta \restriction \psi_2, \delta \restriction \psi_2) \in D_i \text{ or} \\ & & (\psi_1, h_i \circ \theta \restriction \psi_1, \delta \restriction \psi_1), (\mathsf{X}(\psi_1 \,\mathsf{U}\, \psi_2), h_i \circ \theta, \delta) \in D_i\end{array}$$

□

Proposition 7.25. ϕ is $\mathcal{H}$-satisfiable if and only if there exists a path in $G_{\mathcal{H}}(\phi)$ that fulfils ϕ.

Proof.

[⇐] If there exists π in $G_{\mathcal{H}}(\phi)$ that fulfils ϕ, by definition this implies that ϕ is satisfied by an allocation triple (σ, N, θ) generated by the underlying run of π.

[⇒] Now assume that ϕ is $\mathcal{H}$-satisfiable, and let $\rho = q_0\lambda_0 q_1 \lambda_1 \cdots$ be a run generating a triple (σ, N, θ) with generator $(h_i)_{i\in\mathbb{N}}$ such that $\sigma, N, \theta \models \phi$. We construct a path $\pi = (q_0, D_0)\ \lambda_0\ (q_1, D_1)\ \lambda_1 \cdots$ that fulfils ϕ as follows:

$$D_i \;=\; \{(\psi, h_i \circ \theta, \delta_i) \mid \psi \in CL(\phi),\ \ \sigma^i, N_i^\sigma, \theta \models \psi\}$$

where let $\theta^\pm$ be the extension of θ to the set $E_{q_i}^\pm$ as defined by (37) (cf. page 71) then δ_i is defined for all $x, y \in fv(\psi) \cup E_{q_i}^\pm$ by:

$$\delta_i(x, y) = \min\{\lceil n \rceil \mid \mu_i^n \circ \theta^\pm(x) = \theta^\pm(y)\}.$$

First of all, by Lemma C.5 (q_i, D_i) is an atom for all $i \in \mathbb{N}$. Secondly, we show that π is indeed a path by proving that it satisfies the conditions of Definition 7.23.

1. By the construction of π.
2. By contradiction. Assume that there exists an $i \geqslant 0$ such that $(q_i, D_i) \rightarrow_{\lambda_i} (q_{i+1}, D_{i+1})$ is not a transition of G. Take the minimal such i. Then one of the following must hold:
   i) $q_i \rightarrow_{\lambda_i} q_{i+1}$ is not a transition in $\mathcal{H}$. But this contradicts the fact that ρ is a run of $\mathcal{H}$.
   ii) By contradiction assume that there exists $(\mathsf{X}\psi, h_i \circ \theta, \delta_i) \in D_i$ such that $(\psi, \Theta', \delta') \notin D_{i+1}$ for all $(\psi, \Theta', \delta') \in [\lambda_i \circ (\psi, h_i \circ \theta, \delta_i)]$. But then also $(\psi, h_{i+1} \circ \lambda_i^\sigma \circ \theta, \delta_{i+1}) \notin D_{i+1}$ since $(\psi, h_{i+1} \circ \lambda_i^\sigma \circ \theta, \delta_{i+1}) \in [\lambda_i \circ (\psi, h_i \circ \theta, \delta_i)]$ (see Lemma C.3); hence this would imply $\sigma^{i+1}, N_{i+1}^\sigma, \lambda_i^\sigma \circ \theta \not\models \psi$. But then also $\sigma^i, N_i^\sigma, \theta \not\models \mathsf{X}\psi$, contradicting the construction of D_i.

Assume $(\psi, \Theta', \delta') \in [\lambda_i \circ (\psi, \Theta, \delta)] \cap D_{i+1}$, but $(\mathsf{X}\psi, \Theta, \delta) \notin D_i$. By construction of D_{i+1}, the tuple (ψ, Θ', δ') must be of the form $(\psi, h_{i+1} \circ \theta, \delta_{i+1})$ and $\sigma^{i+1}, N^\sigma_{i+1}, \theta \models \psi$. Condition 1 of Definition 7.20 implies that $\mathrm{cod}(h_{i+1} \circ \theta) \subseteq \mathrm{cod}(\lambda_i)$. Then, since $\lambda^\sigma_i \rhd \lambda_i$, by Lemma 3.19 we have $\mathrm{cod}(\theta) \cap N^\sigma_{i+1} = \varnothing$. Therefore there exists $\theta' : fv(\psi) \rightharpoonup Ent$ such that

$$\lambda^\sigma_i \circ \theta' = \theta.$$

Now by the semantics of $\mathcal{M}all\mathrm{TL}$, $\sigma^{i+1}, N^\sigma_{i+1}, \lambda^\sigma_i \circ \theta' \models \psi$ implies $\sigma^i, N^\sigma_i, \theta' \models \mathsf{X}\psi$. Thus by definition of D_i we must have $(\mathsf{X}\psi, h_i \circ \theta', \delta_i) \in D_i$. Contradiction.

3. Assume $(\psi_1 \,\mathsf{U}\, \psi_2, h_i \circ \theta, \delta_i) \in D_i$. By the construction of D_i, $\sigma^i, N^\sigma_i, \theta \models \psi_1 \,\mathsf{U}\, \psi_2$. Therefore $\sigma^j, N^\sigma_j, \lambda^\sigma_{j-1} \circ \cdots \circ \lambda^\sigma_i \circ \theta \restriction fv(\psi_2) \models \psi_2$ for some $j \geqslant i$. Due to the construction of D_j, it follows that $(\psi_2, h_j \circ \lambda^\sigma_{j-1} \circ \cdots \circ \lambda^\sigma_i \circ \theta \restriction \psi_2, \delta_j \restriction \psi_2) \in D_j$ and Lemma C.3, $(\psi_2, h_j \circ \lambda^\sigma_{j-1} \circ \cdots \circ \lambda^\sigma_i \circ \theta \restriction \psi_2, \delta_j \restriction \psi_2) \in [\lambda_{j-1} \circ \cdots \circ \lambda_i \circ (\psi_2, h_i \circ \theta \restriction \psi_2, \delta_i \restriction \psi_2)]_{q_i, q_j}$.

Thus, π is a path and it is fulfilling since it satisfies ϕ by construction.

□

Proposition 7.27. If π is fulfilling path in $G_{\mathcal{H}}(\phi)$, then $Inf(\pi)$ is a self-fulfilling SCS of $G_{\mathcal{H}}(\phi)$.

Proof. Let $G' = Inf(\pi)$. G' is strongly connected. From the definition of infinite set it follows that there exists $i \geqslant 0$ such that the atoms in $\pi^i = (q_i, D_i)\lambda_i(q_{i+1}, D_{i+1})\lambda_{i+1} \cdots$ are precisely those in G'. Furthermore, if there is an atom $A \in G'$ such that $(\psi_1 \,\mathsf{U}\, \psi_2, \Theta, \delta) \in D_A$, then there exists $j \geqslant i : (q_j, D_j) = A$. By condition 3 of Def. 7.23 we have that there exists $n \geqslant j$ such that $(\psi_2, \Theta', \delta') \in D_n$ where $(\psi_2, \Theta', \delta') \in [\lambda_{n-1} \circ \cdots \circ \lambda_j \circ (\psi_2, \Theta \restriction \psi_2, \delta \restriction \psi_2)]$. But then $(q_n, D_n) \in G'$, hence G' is self-fulfilling. □