

Local Reasoning for Robust Observational Equivalence

Working Draft

Dan R. Ghica
University of Birmingham, UK

Koko Muroya
RIMS, Kyoto University
University of Birmingham, UK

Todd Waugh Ambridge
University of Birmingham, UK

Abstract

We propose a new core calculus for programming languages with effects, interpreted using a hypergraph-rewriting abstract machine inspired by the Geometry of Interaction. The intrinsic calculus syntax and semantics only deal with the basic structural aspects of programming languages: variable binding, name binding, and thunking. Everything else, including features which are commonly thought of as intrinsic, such as arithmetic or function abstraction and application, must be provided as extrinsic operations, with associated rewrite rules. The graph representation yields natural concepts of *locality* and *robustness* for equational properties and reduction rules, which enable a novel flexible and powerful reasoning methodology about (type-free) languages with effects. We illustrate and motivate the technique with challenging examples from the literature.

1 Introduction

1.1 Context and motivation

Program equivalence is an old central question in the study of programming languages. Establishing whether two program fragments are equivalent has immediate consequences regarding compiler optimisation and correctness, program verification and validation, and other key applications. The standard formal definition of the concept is that of *observational equivalence* [Morris Jr, 1969]. Two *executable* programs are observationally equivalent if they have the same input-output behaviour. On *terms* (program fragments) observational equivalence is the smallest congruence induced by using executable programs as contexts, a definition which makes direct reasoning difficult.

The mathematical challenge of observational equivalence is two-fold. On the one hand, quantifying over all contexts is unwieldy, which led to the development of indirect methods. As an extremal case, *denotational semantics* provides a model-theoretic route to proving program equivalence [Scott and Strachey, 1971], while the interplay of syntactic (operational) and denotational techniques led to the development of hybrid methods such as *Kripke logical relations* [Statman, 1985] or *trace semantics* [Jeffrey and Rathke, 2005]. On the other hand, program equivalences are notoriously fragile. Adding new semantic features to a language can break equivalences and, more significantly, can invalidate reasoning techniques wholesale. Increasing the expressiveness of a language goes hand in hand with increasing the discriminating power of contexts, and new techniques based on bisimulation have been devised to meet the challenge of increasingly realistic languages [Koutavas and Wand, 2006].

A methodological point that became increasingly clear is that the richness of features of modern programming languages demands a systematic study, ideally a classification, from the point of view of the consequences such features have on the equational properties of a language. The development of *game semantics* made it possible to give combinatorial, syntax-independent, orthogonal characterisations for entire classes of features such as state and control, e.g. the so-called *Abramsky* [1997] “cube”, or to replace the syntactic notion of context by an abstracted *adversary* [Ghica

and Tzevelekos, 2012]. A classification of language features and their impact on reasoning has also been undertaken using logical relations [Dreyer et al., 2012].

1.2 Overview and contribution

In this paper we present a new approach to defining effects and proving observational equivalence based on a novel operational machinery inspired by the Geometry of Interaction (GoI) semantics of languages with effects [Hoshino et al., 2014, Muroya et al., 2016]. In contrast with the usual definition of the semantic model as a token-passing abstract machine, we introduce a *token-guided rewriting* semantics of the underlying net [Muroya and Ghica, 2017]. Unlike the original GoI machine, the graph-rewriting abstract machine can be shown to be efficient (in the sense of Accattoli et al. [2014]) for common reduction strategies (by-value, by-need). In addition to the theoretical interest, the graph-rewriting abstract machine was also used to model exotic effects, inspired by machine learning applications, in which imperative computation can be transformed into pure computation by abstracting the state [Muroya et al., 2018].

We revise and generalise the formulation of the graph-rewriting machine from graphs to so-called *hypernets*, which are hierarchical hypergraphs, in the sense that entire hypergraphs can be used as edge labels. The hypernet-rewriting abstract machine is used to interpret SPARTAN, a core calculus of three basic constructs: name binding, variable binding, and thunking. Everything else, including function-abstraction and function application, must be provided as an extrinsic operation, which comes with its own associated rewrite rules. The flow of control through the program is represented by a distinguished edge called a *token* (or a *focus*), which traverses the hypernet triggering reductions.

The main conceptual contribution of the paper arises from the hypernet formulation of the language semantics: the underlying hypergraph for a program provides a natural notion of *locality*, i.e. those nodes and edges which belong to a neighbourhood around a point of interest such as the token or a redex we want to reason about. Locality in turn gives rise to a notion of *robustness* of sub-graphs, which states that rewrites occurring elsewhere cannot interfere with it. From a reasoning perspective, locality and robustness are useful concepts, since they can be established via elementary case analysis comparing reduction rules pairwise. The key technical result is a characterisation theorem (Thm. 6.14) which establishes, informally speaking, that robustness implies observational equivalence, a very handy conceptual tool¹.

We believe this is an important result because, for the first time, it goes beyond an enumerative approach to effects, to a syntax-independent characterisation of effects. There is no need to separate effects, for example, into state and control in order to reason about languages with effects. In fact there is no need to separate computations into pure and effectful. Only locality properties are important, in the sense that the formulation of a property involves a hypergraph neighbourhood rather than the whole graph, and that rewrite rules associated with various operations do not interfere with the property we aim to check. A large number of equivalences and effects encountered in programming languages will be seen to satisfy these conditions – but not all. Perhaps the simplest example of a reasonable non-local and non-robust effect is the function `stat : unit → stat` found in the OCaml GC module, providing memory usage information amounting to measuring the size of the term under execution. In the presence of `stat`, which can be defined as an operation in our framework, almost all basic equivalences fail, for example the beta law (since the term grows smaller).

The hypernet semantics is a genuine abstract machine in the sense that it provides syntax-independent execution with a well-defined and reasonably realistic cost model (time and space). The effects are not merely encoded into a base metalanguage, but can be expressed directly as arbitrary actions on the machine state. If the aim is a language that enjoys a nice equational theory, then these equivalences should be robust relative to desirable equivalences. But arbitrary, possibly ill-behaved exotic effects are also definable if so desired.

¹ Our concept of locality is not to be confused with memory locality, which concerns memory access during program execution. Our locality, together with robustness, captures impact that operations in a program make on other parts of the program during execution. Ours could be seen as a generalisation of memory locality, because memory access could be modelled as extrinsic features of SPARTAN.

Finally, another conceptual novelty introduced here is a refinement of the notion of contextual equivalence. The standard definition involves a quantification over all contexts, but in a language as flexible as SPARTAN contexts can be wild. Therefore it makes sense to restrict contexts so that they only have certain shapes, which correspond to the way contexts are formed in well-behaved languages. Moreover, equivalence is indexed by a preorder on natural numbers which allows the comparison of the number of evaluation steps. This means that if two terms are equivalent with respect to the greater-than-or-equal order then in-context substitution is guaranteed to reduce the number of evaluation steps. This can be used as a basis for qualitative and quantitative reasoning, as needed for example in compiler optimisation.

The structure of the paper is as follows. In Sec. 2 we present a simple calculus (SPARTAN) which intermediates syntactically between languages with effects and the focussed hypernet representation. Hypernet rewriting is overviewed in Sec. 3 with enough technical detail to understand the examples in Sec. 4. The technical details of focussed hypernet rewriting are given in Sec. 5, with the characterisation theorem and its proof presented in Sec. 6-7. Sec. 8 gives an application of the main characterisation theorem to proving observational equivalence. We conclude with a mention of related and future work.

An interpreter and visualiser of SPARTAN execution can be accessed [online](https://tnttodda.github.io/Spartan-Visualiser/).²

2 The Spartan calculus

The syntactic elements of SPARTAN are a set of *variables* \mathbb{V} , (ranged over by x, y), a set of *atoms* \mathbb{A} , (ranged over by a), and a set of *operations* \mathbb{O} (ranged over by ϕ). The set of operations is partitioned into $\mathbb{O} = \mathbb{O}_\vee \uplus \mathbb{O}_\&$; the partitions are called *passive* and *active* operations. We will usually denote a sequence of variables x_0, \dots, x_k by \vec{x} , a sequence of atoms a_0, \dots, a_k by \vec{a} , etc. These sequences may be empty, case in which we write $-$. We denote the length of a sequence by $|\vec{x}|$ and, if convenient we may treat the sequences as sets by writing $\vec{x} \cap \vec{y}$, $y \in \vec{x}$, etc.

Definition 2.1 (SPARTAN terms). The terms of SPARTAN, typically ranged over by s, t, u , are generated by the grammar:

$$t ::= x \mid a \mid \mathbf{new} \ a \multimap t \ \mathbf{in} \ t \mid \mathbf{bind} \ x \rightarrow t \ \mathbf{in} \ t \mid \vec{y}.t \mid \phi(\vec{t}; \vec{t}).$$

The term formers are variables, names, name binding, variable binding, thunking and operations. Note that the sequences above may be empty. In particular, thunking may be applied without binding any particular variable $(-).t$. In the formation of an operation term $\phi(\vec{s}; \vec{t})$ arguments \vec{s} are used eagerly and the arguments \vec{t} lazily (we also say they are *deferred*). The eager arguments are evaluated in the given order, whereas the evaluation of lazy arguments is deferred. The distinction between name-binding and variable-binding will be seen to play a crucial role in the management of sharing vs. copying during execution.

The calculus provides only the most basic infrastructure. All interesting computational content must be provided as particular operations. The following is a non-exhaustive list of such operations which may be added to the SPARTAN framework.

Some passive operations (constructors) that can be added to SPARTAN are numerical constants ($n \stackrel{\text{def}}{=} n(-; -)$), pairing ($\langle t, u \rangle \stackrel{\text{def}}{=} P(t, u; -)$), empty tuple ($\langle \rangle \stackrel{\text{def}}{=} \langle \rangle(-; -)$), injections ($\mathbf{inj}_i(t) \stackrel{\text{def}}{=} \mathbf{inj}_i(t; -)$, $i \in \{1, 2\}$), etc.

Some examples of active SPARTAN operations are successor ($\mathbf{succ}(t) \stackrel{\text{def}}{=} \mathbf{succ}(t; -)$), addition ($t + u \stackrel{\text{def}}{=} +(t, u; -)$), conjunction ($t \& u \stackrel{\text{def}}{=} \&(t, u; -)$), conditionals ($\mathbf{if} \ t \ \mathbf{then} \ u_1 \ \mathbf{else} \ u_2 \stackrel{\text{def}}{=} \mathbf{if}(t; u_1, u_2)$), recursion ($\mu x.t \stackrel{\text{def}}{=} \mu(-; x.t)$) etc.

Datatype destructors can also be added as active operations, for example projections ($\pi_i(t) \stackrel{\text{def}}{=} \pi_i(t; -)$, $i \in \{1, 2\}$), pattern matching ($\mathbf{match} \ t \ \mathbf{with} \ x_1 \mapsto u_1, x_2 \mapsto u_2 \stackrel{\text{def}}{=} \mathbf{match}(t; x_1.u_1, x_2.u_2)$), etc.

Operations with multiple arguments can come in different flavours depending on order of evaluations and eagerness. For example disjunction can be left-to-right $\&(t, u; -)$, right-to-left $\&(u, t; -)$

²<https://tnttodda.github.io/Spartan-Visualiser/>

or short-cut $\&(t; u)$. Pairs can be also evaluated left-to-right (as above), but also, right-to-left, or lazily.

Most strikingly, abstraction and application themselves can be presented as operations. Abstraction is simply a thunking of the function body: $\lambda x.t \stackrel{\text{def}}{=} \lambda(-; x.t)$, whereas application can be defined by-name or by-value, left-to-right or right-to-left: $t u \stackrel{\text{def}}{=} @ (t; u)(\text{call-by-name})$, $t u \stackrel{\text{def}}{=} \overrightarrow{@} (t, u; -)(\text{left-to-right call-by-value})$, or $t u \stackrel{\text{def}}{=} \overleftarrow{@} (u, t; -)(\text{right-to-left call-by-value})$.

Algebraic operations are operations of certain arity and equational properties, designed to characterise a class of computational effects [Plotkin and Power, 2003]. All examples of algebraic operations from *loc. cit.* can be presented as (active) SPARTAN operations. Non-deterministic choice $\sqcup(-; u_1, u_2)$ selects one of its deferred argument non-deterministically; equipping it with a probability $p \in [0, 1]$ yields a probabilistic choice $\sqcup_p(-; u_1, u_2)$. Interactive input $\text{read}(-; x.u)$ assigns an input value to bound variable x , whereas interactive output $\text{write}(t; u)$ produces the result of evaluating (eager) argument t as an output value. Both operations continue with the deferred argument u . State lookup and update take form $\text{lookup}(t_0; x.u)$ and, respectively, $\text{update}(t_0, t; u)$. The difference from interactive I/O is that the first eager argument t_0 is evaluated to yield an atom, which serves as a location in a store. In their generic form, state operations $\text{deref}(t_0; -)$ and $\text{assign}(t_0, t; -)$ return a stored value and the empty pair, respectively, instead of continuing with a deferred argument. Finally, creation of a local state, in the generic form, is modelled by $\text{ref}(t; -)$ that stores evaluation result of t to a fresh location (atom) and returns it.

Effect handlers are studied as a generalisation of exception handlers to algebraic effects in Plotkin and Pretnar [2013]. In our setting, a handler that targets operations ϕ_1, \dots, ϕ_m can be constructed by a passive operation $\text{handler}_{\phi_1, \dots, \phi_m}$ tagged with the targets. It is natural to assume that targeted operations are never passive (e.g. lambda-abstraction, pairing operation, injection, and handlers themselves). These handler constructors are a (rare) example in which a deferred argument may bind more than one variables. A handler can be then used with the handling operation $\text{with_handle}(t, u; -)$ to evaluate u with handler t .

Control operators are similarly dealt with. The undelimited control operator call/cc can be defined as:

$$\text{callcc}(t) \stackrel{\text{def}}{=} \text{callcc}(-; t), \quad \text{abort}(t) \stackrel{\text{def}}{=} \text{abort}(-; t)$$

The delimited control operator shift/reset of Danvy and Filinski [1990] can be defined as:

$$\text{shift}(t) \stackrel{\text{def}}{=} \text{shift}(-; t), \quad \text{reset}(t) \stackrel{\text{def}}{=} \text{reset}(t; -).$$

2.1 The Spartan type system

The type system is primarily used to distinguish thunks from eager terms and to ensure terms are well formed. However, this type system does not enforce any notion of runtime safety.

Eager terms have type \star and thunks have type $T^m(\star)$ for some $m \in \mathbb{N}$, representing the number of bound variables that accompany the thunk. Note that $T^0(\star)$ is a valid type and that $T^0(\star) \neq \star$. Types are typically ranged over by τ . A sequence τ_1, \dots, τ_n of types is denoted by an expression $\otimes_{i=1}^n \tau_i$. This is empty if $n = 0$, and equal to a single type if $n = 1$, i.e. $\otimes_{i=1}^1 \tau = \tau$. The empty sequence is written as ϵ . We use syntactic sugar $\tau^{\otimes n} \equiv \otimes_{i=1}^n \tau$, and $\tau^{\otimes n} \otimes \tau^{\otimes n'} = \tau^{\otimes (n+n')}$.

Each operation $\phi \in \mathbb{O}$ has an *arity*, given in the form of $\phi \vdash (m_e; \vec{n}) \in \mathbb{N} \times \mathbb{N}^{m_d}$, indicating it takes m_e eager arguments and m_d thunks.

For a type τ and a term t , judgements are written as $\vec{x} \mid \vec{a} \vdash t : \tau$, where all variables in \vec{x} and all atoms in \vec{a} are distinct. The rules are given in Fig. 1, where $|\vec{x}| = k$, $|\vec{y}| = n$, $|\vec{a}| = h$. If $- \mid - \vdash t : \star$ is derivable, term t is called a *program*.

3 Overview of focussed graph rewriting

3.1 Monoidal hypergraphs and hypernets

Given a set X we write by X^* the set of elements of the free monoid over X . Given a function $f : X \rightarrow Y$ we write $f^* : X^* \rightarrow Y^*$ for the pointwise application (map) of f to the elements of X^* .

$$\begin{array}{c}
\frac{}{\vec{x} \mid \vec{a} \vdash x_i : \star} \quad 1 \leq i \leq k \quad \frac{}{\vec{x} \mid \vec{a} \vdash a_j : \star} \quad 1 \leq j \leq h \quad \frac{\vec{x} \mid \vec{a} \vdash u : \star \quad \vec{x}, x \mid \vec{a} \vdash t : \star}{\vec{x} \mid \vec{a} \vdash \text{bind } x \rightarrow u \text{ in } t : \star} \quad x \notin \vec{x} \\
\\
\frac{\vec{x} \mid \vec{a} \vdash u : \star \quad \vec{x} \mid a, \vec{a} \vdash t : \star}{\vec{x} \mid \vec{a} \vdash \text{new } a \multimap u \text{ in } t : \star} \quad a \notin \vec{a} \quad \frac{\vec{y}, \vec{x} \mid \vec{a} \vdash t : \star}{\vec{x} \mid \vec{a} \vdash \vec{y}.t : T^n(\star)} \quad n \geq 0, \vec{x} \cap \vec{y} = \emptyset \\
\\
\frac{\phi \vdash (m; \vec{n}) \in \mathbb{N} \times \mathbb{N}^p \quad \{\vec{x} \mid \vec{a} \vdash t_i : \star\}_{i=1}^m \quad \{\vec{x} \mid \vec{a} \vdash s_j : T^{n_j}(\star)\}_{j=1}^p}{\vec{x} \mid \vec{a} \vdash \phi(\vec{t}; \vec{s}) : \star}
\end{array}$$

Figure 1: The SPARTAN type system

Definition 3.1. A *monoidal hypergraph* is a pair (V, E) of finite sets, *vertices* and (*hyper*)*edges* along with a pair of functions $S : E \rightarrow V^*$, $T : E \rightarrow V^*$ defining the *source list* and *target list*, respectively, of an edge.

Definition 3.2. A *labelled monoidal hypergraph* consists of a monoidal hypergraph, a set of vertex labels L , a set of edge labels M , and labelling functions $f_L : V \rightarrow L$, $f_M : E \rightarrow M$ such that:

- For any edge $e \in E$, its source list $S(e)$ consists of distinct vertices, and its target list $T(e)$ also consists of distinct vertices.
- For any vertex $v \in V$ there exists at most one edge $e \in E$ such that $v \in S(e)$ and at most one edge $e' \in E$ such that $v \in T(e')$.
- For any edges $e_1, e_2 \in E$ if $f_M(e_1) = f_M(e_2)$ then $f_L^*(S(e_1)) = f_L^*(S(e_2))$, and $f_L^*(T(e_1)) = f_L^*(T(e_2))$.

In words, the label of an edge is always consistent with the number and labelling of its endpoints. This makes it possible to use the vertex labels as *types* for edge labels. Given $m \in M$ we can write $m : x \Rightarrow x'$ for $x, x' \in L^*$ such that $x = f_L^*(S(e))$ and $x' = f_L^*(T(e))$ for any $e \in E$ such that $f_M(e) = m$.

If a vertex belongs to the target (resp. source) list of no edge we call it an *input* (resp. *output*).

Definition 3.3. A labelled monoidal hypergraph is *interfaced* if inputs and outputs are respectively ordered, and no vertex is both an input and an output.

We can type a hypergraph $G : f_L^*(I) \Rightarrow f_L^*(O)$ where I, O are the lists of inputs and outputs, respectively. In the sequel, when we say hypergraphs we always mean interfaced labelled monoidal hypergraphs.

Definition 3.4 (Interface permutation). The permutation of the interface of a hypergraph yields another hypergraph. Let G be a hypergraph with an input list i_1, \dots, i_n and an output list o_1, \dots, o_m . Given two bijections ρ and ρ' on sets $\{1, \dots, n\}$ and $\{1, \dots, m\}$, respectively, we write $\Pi_{\rho'}^{\rho}(G)$ to denote the hypergraph that is defined by the same data as G except for the input list $i_{\rho(1)}, \dots, i_{\rho(n)}$ and the output list $o_{\rho'(1)}, \dots, o_{\rho'(m)}$.

Definition 3.5 (Hypernet). Given a set of vertex labels L and edge labels M we write $\mathcal{H}(L, M)$ for the set of hypergraphs with these labels; we also call these *level-0 hypernets* $\mathcal{H}_0(L, M)$. We call *level-(k+1) hypernets* the set of hypergraphs

$$\mathcal{H}_{k+1}(L, M) = \mathcal{H}\left(L, M \cup \bigcup_{i \leq k} \mathcal{H}_i(L, M)\right).$$

We call (labelled monoidal) *hypernets* the set $\mathcal{H}_{\omega}(L, M) = \bigcup_{i \in \mathbb{N}} \mathcal{H}_i(L, M)$.

Informally, hypernets are nested hypergraphs, up to some finite depth, using the same sets of labels. An edge labelled with a hypergraph is called “box” edge, and a hypergraph labelling a

box edge is called “content”. Edges of a hypernet G are said to be “shallow”. Edges of nesting hypernets of G , i.e. edges of hypernets that recursively appear as edge labels, are said to be “deep” edges of G . Shallow edges and deep edges of a hypernet are altogether referred to as edges “at any depth”.

3.2 Graphical conventions

A monoidal hypergraph with vertices $V = \{v_1, v_2, v_3, v_4, v_5\}$ and edges $E = \{e_1, e_2\}$ such that

$$S(e_0) = \{v_0, v_1\}$$

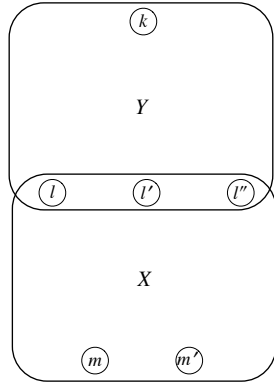
$$T(e_0) = S(e_1) = \{v_2, v_3, v_4\}$$

$$T(e_1) = \{v_5\}$$

$$f_L = \{v_0 \mapsto m, v_1 \mapsto m', v_2 \mapsto l, v_3 \mapsto l', v_4 \mapsto l'', v_k \mapsto k\}$$

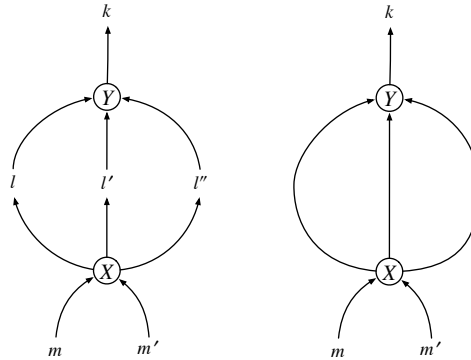
$$f_M = \{e_0 \mapsto X, e_1 \mapsto Y\}$$

is normally represented as



However, this style of representing hypergraphs is awkward for understanding their structure. We will often graphically represent hypergraphs as graphs, with both vertices and edges drawn as nodes marked by their label and connecting input vertices with edges, and edges with output vertices using arrows. To distinguish them, the edge labels are circled.

Since the node labels are often determined in our typed graphs by the edges we can omit them to avoid clutter, showing only the edges and the way they link. The graph above would be drawn like this:

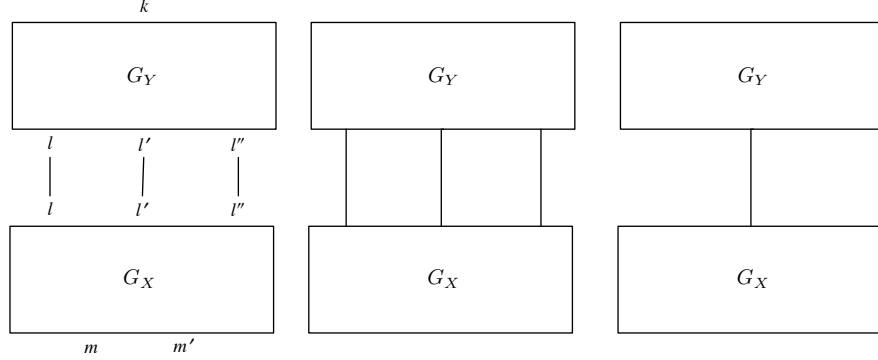


Graph showing the nodes

Graph hiding some nodes

Sometimes we will draw a hypergraph so that to identify sub-graphs of interest. In this case we may draw interface nodes twice and connect them with arrowless lines which indicate *identity*, i.e. the nodes at either ends are two graphical representations of the same node. If it is obvious from context we may just draw the identity lines between the sub-graphs, or just one line if the

entire input interface of a sub-graph is identified with the entire output interface of another sub-graph. For example, below we consider G_X, G_Y as the sub-graphs consisting just of edge X and Y , respectively:



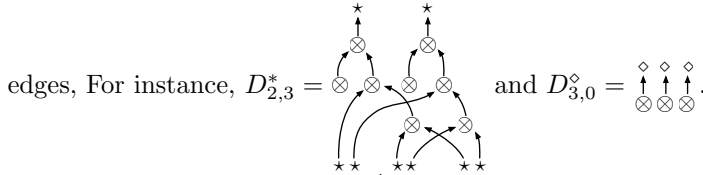
The final conventions are that a bunch of parallel arrows can be drawn as a single arrow with a dash across, and that a hypergraph-labelled edge is indicated with a dotted box, and decorated with its type.

3.3 From terms to hypernets

We represent terms in SPARTAN as hypernets with vertex labels either τ or \diamond , ranged over by ℓ . The same notational conventions apply, extended with the extra symbol \diamond . The edge labels have unique types, and they are: the operations ϕ (with the edge-label-type inherited from the SPARTAN type system), an *instance* label $! : \star \Rightarrow \diamond$, an *atom* label $\circ : \diamond \Rightarrow \star$, a family of *contraction* labels $\{\otimes_W^\ell : \epsilon \Rightarrow \ell, \otimes_C^\ell : \ell^{\otimes 2} \Rightarrow \ell \mid \ell \in \{\star, \diamond\}\}$, a family of *token* labels $?, \checkmark, \not\checkmark : \star \Rightarrow \star$. When a hypergraph is used as an edge label it must always have type $G : \star \Rightarrow \star^{\otimes n} \otimes \diamond^{\otimes h}$; the box edge is assigned type $T^{n-k}(\star) \Rightarrow \star^{\otimes k} \otimes \diamond^{\otimes h}$ for some $k \leq n$.

The hypernet is said to be *focussed* if there exist only one token-labelled edge and it is shallow, plus some other basic well-formedness conditions discussed later.

It will be helpful to define a family of subgraphs $D_{k,m}^\ell : \ell^{\otimes km} \Rightarrow \ell^{\otimes k}$ with $\ell \in \{\diamond, \star\}$ which we call *distributors* (Def. 5.6). Intuitively, they are the k -interleaving of bunches of m -contraction



edges, For instance, $D_{2,3}^* =$ [diagram] and $D_{3,0}^\diamond =$ [diagram]. The translation function $(-)^{\dagger}$ from SPARTAN terms to hypernets is given in Fig 2, where $\boxtimes_{i=1}^m G_i$ is a hypernet formed of a family of hypernets $(G_i : \star \Rightarrow \star^{\otimes k_i} \otimes \diamond^{\otimes h_i})$ placed side by side. We can easily see that the hypernet of a SPARTAN program always has type $(- \mid - \vdash t : \star)^{\dagger} : \star \Rightarrow \epsilon$, i.e. one input and no outputs. The hypernet of a program is focussed by adding a $?$ token in the input position.

3.4 Focussed rewriting

In this section we define an abstract machine for rewriting hypernets of SPARTAN terms, thus giving a notion of evaluation for the calculus. The machine is based on the *Dynamic Geometry of Interaction Machine* of Muroya and Ghica [2017], which it generalises for the modelling of effects. In this section we give a brief introduction to the machine, with the full technical details presented later, in Sec. 5.

In a conventional reduction semantics a subtle and often complex aspect of the reduction bureaucracy is the identification and isolation of a subterm which is a *redex* from the *context* [Felleisen and Hieb, 1992]. Similarly, in an abstract machine much of the work consists of moving information (representing the context) on and off the stack in order to reach redexes [Wright and Felleisen,

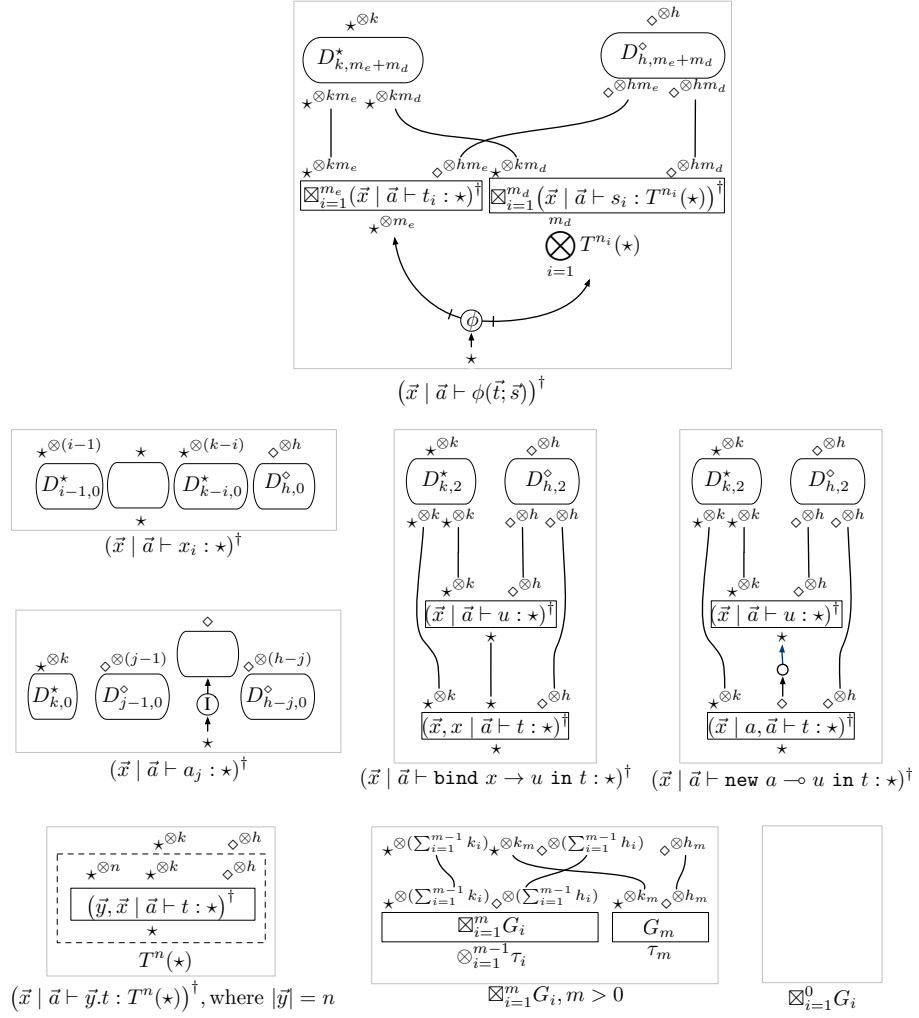


Figure 2: Hypernet semantics of SPARTAN

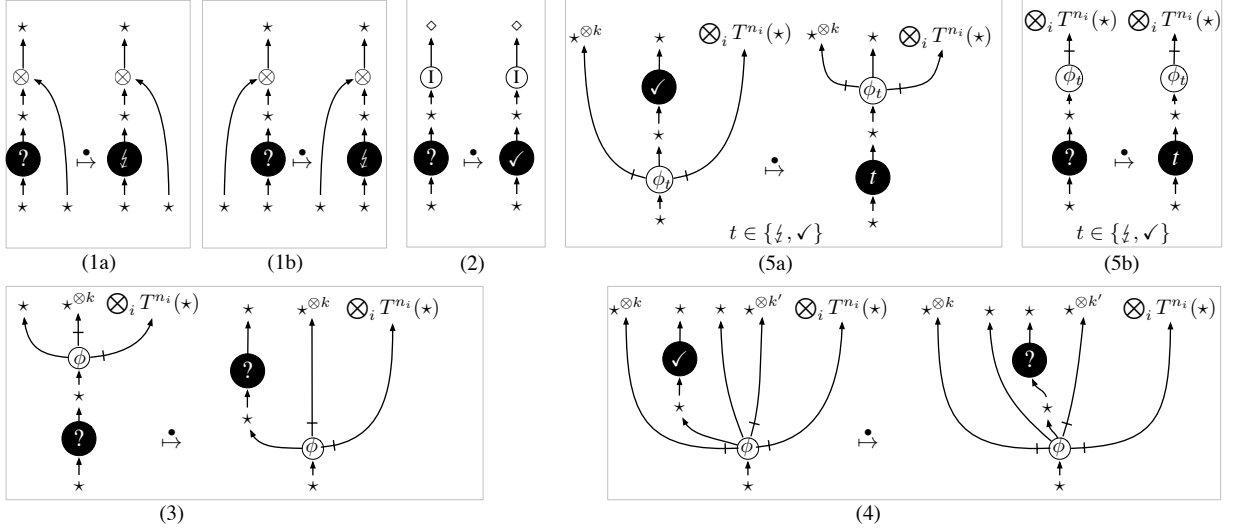


Figure 3: Interaction rules

1994]. In both cases the formal machinery identifies an implicit *focus* of action which either moves around the term or is acted upon via a substitution. In the case of focussed graph rewriting the focus is represented by the token edge. The redex search is defined by the local interaction between the token and the neighbouring edges, which can cause the token to navigate the graph. The same local interactions can determine a rewrite action.

The state of the token edge determines the possible actions. If the token is $?$ then the applicable rules (*interaction rules*) are search rules, which do not change the underlying graph; the $?$ token always travels in the direction of the edges and it is searching for a redex. In comparison to a conventional reduction semantics this is a narrowing of the term-in-context. If the token is \checkmark then the applicable rules are also search (interaction) rules but the token travels in the opposite direction of the edges. In comparison to a reduction semantics this is a widening of the term-in-context normally following the detection of a value. Finally, if the token is $\frac{1}{2}$ then a rewrite is about to be performed at the very next step. There are two kinds of rewrites, intrinsic to the SPARTAN calculus and extrinsic, defined by the operations. The intrinsic rewrites are only copying of subgraphs which are triggered when a $\frac{1}{2}$ token interacts with contraction \otimes .

Interaction rules are in Fig. 3. These rules capture the intuition behind redex search discussed earlier:

Rule 1. When encountering a contraction (\otimes) the search token ($?$) becomes a rewrite token ($\frac{1}{2}$) as a copying action will follow. This rule has two sub-rules depending on whether the focus is on the left or right branch.

Rule 2. When encountering an instance edge (I) the search token ($?$) changes to a value token (\checkmark) as atoms block both evaluation and copying.

Rule 3. When encountering an operation (ψ) with at least one eager argument the search token ($?$) will inspect the first argument.

Rule 4. After resolving eager argument $k + 1$ of an operation (ψ) the value token (\checkmark) changes to a search token ($?$) which proceeds to inspect the next eager argument, if it exists.

Rule 5a. After all eager arguments of an operation ψ_t ($t \in \{\checkmark, \frac{1}{2}\}$), the token will change to a t -token depending on the kind of operation ψ_t is, value or rewrite. (Rule 5b. is a degenerate version for operations with no eager arguments).

Contraction rules are given in Fig. 7 (Sec. 5). The extrinsic rules for various operations are discussed in the following section, while also reasoning about contextual equivalence.

4 Observational equivalence

The hypernet model supports a new style of reasoning centered around a graph-theoretic intuition of *locality*, as the incidence relation immediately identifies the relevant direct interactions arising in a term. A secondary, but equally important concept, is that of *robustness*, capturing the fact that a rewrite rule does not interfere with a particular subgraph of interest which we call a *template*. The key Theorem (Thm. 6.14 in Sec. 6) says, informally speaking, that robust templates are observational equivalences. As a property, robustness can be established by case analysis, comparing a template with all operations that may occur in the context, thus providing a relatively simple route to proving observational equivalence. Since robustness is only a sufficient criterion we test it with various challenging equivalences from the literature, involving in first instance (arbitrary untyped) state.

In the rest of the section we will formulate some such *desirable* equivalences. They hold in general in well-behaved languages, but the SPARTAN framework is broad enough to allow the definition of operations that would break these and most equivalences. This is why we would refer to them for now as *desiderata*. We will then show that they hold with respect to a specific operation set \mathbb{O}^{ex} described in Sec. 8.1. In Sec. 8 we give more details for one of the equivalences.

Recall that our framework is parametrised by the operation set \mathbb{O} . Our notion of equivalence is additionally parametrised by a set $B_{\mathbb{O}}$ (“behaviour”) of extrinsic rewrites associated with the operations \mathbb{O} . Given two derivable judgements $\vec{x} \mid \vec{a} \vdash t_1 : \star$ and $\vec{x} \mid \vec{a} \vdash t_2 : \star$, we write $B_{\mathbb{O}} \models (\vec{x} \mid \vec{a} \vdash t_1 \preceq^{\dagger_{\text{all}}} t_2 : \star)$ (resp. $B_{\mathbb{O}} \models (\vec{x} \mid \vec{a} \vdash t_1 \simeq^{\dagger_{\text{all}}} t_2 : \star)$) if the hypernet of t_1 is a refinement (resp. equivalence) of t_2 in any context, with the operation set being \mathbb{O} and extrinsic rewrites being $B_{\mathbb{O}}$. In particular, we write $B_{\mathbb{O}} \models (\vec{x} \mid \vec{a} \vdash t_1 \preceq^{\dagger_{\text{bf}}} t_2 : \star)$ (resp. $B_{\mathbb{O}} \models (\vec{x} \mid \vec{a} \vdash t_1 \simeq^{\dagger_{\text{bf}}} t_2 : \star)$), if the refinement (resp. equivalence) holds only in the contexts that do not bind a hole to a variable nor an atom. These concepts are formally defined in Sec. 8.3.

In the sequel, we simply write $t_1 \preceq^{\dagger_{\text{all}}} t_2$ etc., omitting the type \star and the sequences \vec{x} and \vec{a} , and making the parameter $B_{\mathbb{O}}$ implicit.

4.1 Structural laws

Before considering equivalences involving specific operations, it is useful to examine *structural* equivalences, which involve SPARTAN alone. Let $fv(t)$ and $fa(t)$ be the sets of free variables and free atoms of a term, respectively, defined as usual.

Desiderata 4.1 (Coherences).

$$\text{bind } x \rightarrow t \text{ in } \text{bind } y \rightarrow u \text{ in } s \simeq^{\dagger_{\text{all}}} \text{bind } y \rightarrow u \text{ in } \text{bind } x \rightarrow t \text{ in } s, \quad (1)$$

$$(\text{whenever } x \notin fv(u), y \notin fv(t))$$

$$\text{new } a \multimap t \text{ in } \text{new } b \multimap u \text{ in } s \simeq^{\dagger_{\text{all}}} \text{new } b \multimap u \text{ in } \text{new } a \multimap t \text{ in } s, \quad (2)$$

$$(\text{whenever } a \notin fa(u), b \notin fa(t))$$

$$\text{bind } x \rightarrow (\text{bind } y \rightarrow u \text{ in } t) \text{ in } s \simeq^{\dagger_{\text{all}}} \text{bind } y \rightarrow u \text{ in } \text{bind } x \rightarrow t \text{ in } s \quad (3)$$

$$(\text{whenever } y \notin fv(s))$$

$$\text{new } a \multimap (\text{new } b \multimap u \text{ in } t) \text{ in } s \simeq^{\dagger_{\text{all}}} \text{new } b \multimap u \text{ in } \text{new } a \multimap t \text{ in } s \quad (4)$$

$$(\text{whenever } b \notin fa(s))$$

The second batch of laws concern terms \bar{u} with no variable binders (‘new’) except inside thunks. Such terms are *referentially transparent*, in the sense that they can be safely replicated or substituted in other terms:

Desiderata 4.2 (Referential transparency).

$$\text{bind } x \rightarrow \bar{u} \text{ in } \text{bind } y \rightarrow x \text{ in } \text{bind } z \rightarrow x \text{ in } t \simeq^{\dagger_{\text{all}}} \text{bind } y \rightarrow \bar{u} \text{ in } \text{bind } z \rightarrow \bar{u} \text{ in } t, \quad (5)$$

$$(\text{whenever } x \notin fv(t))$$

$$\text{bind } x \rightarrow \bar{u} \text{ in } t \simeq^{\dagger_{\text{all}}} t[\bar{u}/x] \quad (6)$$

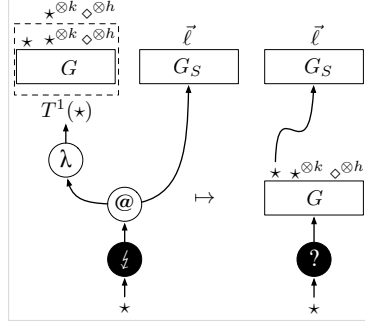


Figure 4: Beta rewrite rule (G, G_S are hypernets)

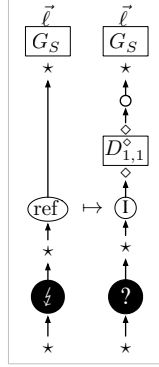


Figure 5: Reference creation (G_S is a hypernet)

To be clear, the referential transparency laws hold in contexts with most “reasonable” operations, including all mentioned in the section below. However, even though referential transparency laws are very robust, one can think of operations which may violate them such as `Gc.stat` mentioned earlier.

4.2 Operations

First we consider lambda-abstraction (a passive operation) and function application (active), in particular left-to-right call-by-value. The rewrite rule when the token reaches the application hyperedge is given in Fig. 4. The usual observational equivalences (laws) induced by beta rewrite are:

Desiderata 4.3 (Beta laws). *If v is a value then*

$$(\lambda x.t) @ v \xrightarrow{\rightarrow} \simeq^{\dagger_{\text{bf}}} \text{bind } x \rightarrow v \text{ in } t \quad (\text{Micro beta})$$

Moreover, if \bar{v} is referentially transparent then

$$(\lambda x.t) @ \bar{v} \xrightarrow{\rightarrow} \simeq^{\dagger_{\text{bf}}} t[\bar{v}/x] \quad (\text{Beta})$$

The beta law is also robust relative to the operations described in the sequel, but reasonable operations that violate it do exist. For example, we mentioned the `Gc.stat()` function of OCaml which returns the size of the term violates this law.

We further add to the language *state* (assignment and dereferencing). Since SPARTAN is (essentially) untyped by default we consider state which can store any terms (or rather values), including lambda-abstractions. The rewrite rule for reference creation (active operation `ref`) is given in Fig. 5, while assignment and dereferencing are given in Fig. 6. In the same figure we give the two rules for equality testing on references. Also note that in the absence of type restrictions it is

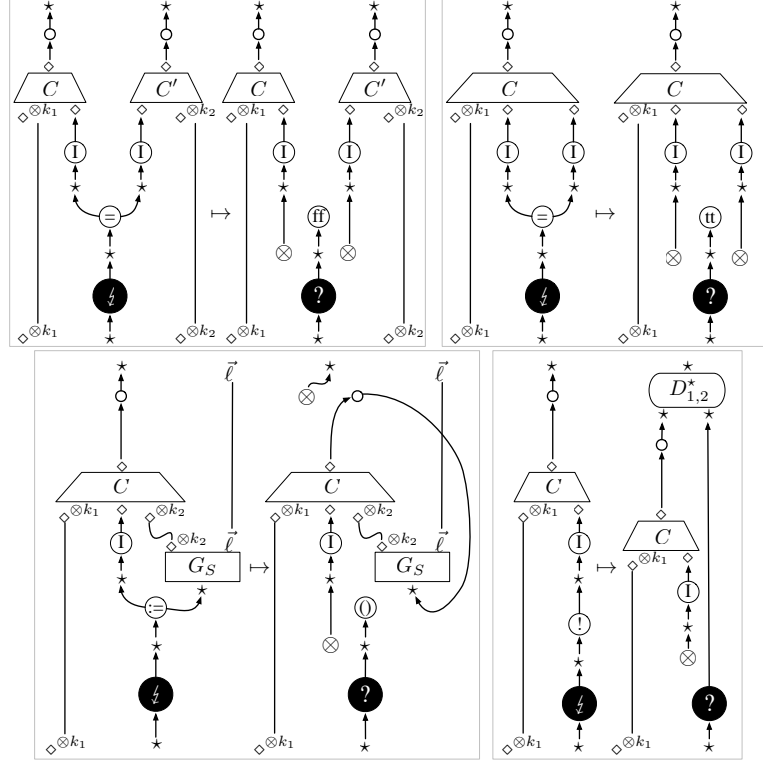


Figure 6: Equality, assignment, and dereferencing rewrite rules (C and C' are contraction trees, and G_S is a hypernet)

straightforward to tie recursive knots in the store. For example `let $x = \text{ref } ()$ in $x := x$` stores a self-reference into x , so that any dereferencing `!x`, `!!x`, `!!!x`, etc., will always produce the same result. For convenience we use the following syntactic sugar:

$$\begin{aligned}
 t \ u &\stackrel{\text{def}}{=} t @ u \\
 \nu x.t &\stackrel{\text{def}}{=} (\lambda x.t) (\text{ref } ()) \\
 \text{let } x = u \text{ in } t &\stackrel{\text{def}}{=} (\lambda x.t) u \\
 u; t &\stackrel{\text{def}}{=} (\lambda_.t) u
 \end{aligned}$$

Because we are focusing on the call-by-value function application, the let-binding as a sugar has different behaviour from the intrinsic variable-binding.

Some desirable equivalences which hold in contexts containing stateful operations are:

Desiderata 4.4 (Stateful laws).

$$\begin{aligned}
 \nu z. \lambda x. x = z &\preceq^{\dagger_{\text{bf}}} \lambda x. \text{false} && \text{(Freshness)} \\
 \nu x. f &\simeq^{\dagger_{\text{bf}}} f && \text{(Locality)} \\
 \text{let } x = \text{ref } 0 \text{ in } (f (\lambda_. x := !x + 1)) (\lambda_. !x) &\simeq^{\dagger_{\text{bf}}} \text{let } x = \text{ref } 0 \text{ in } (f (\lambda_. x := !x - 1)) (\lambda_. -!x) && \text{(Parametricity 1)} \\
 \text{let } x = \text{ref } 1 \text{ in } \lambda f. ((f ()); !x) &\simeq^{\dagger_{\text{bf}}} \lambda f. ((f ()); 1) && \text{(Parametricity 2)}
 \end{aligned}$$

The first two equivalences originate in the ν -calculus of Pitts and Stark [1993], a (CBV) lambda-calculus extended with construct $\nu x.t$ which creates a new *name* and binds it to x in t , plus equality testing on names. “Names” are virtually identical to unit references, therefore for economy of presentation we will consider them as such. The proof techniques in *loc. cit.* rely on logical

relations, which make essential use of the typing structure of the language. In contrast, we can show that these equivalences hold even in untyped state, with the caveat that one becomes a refinement, rather than an equivalence, since equality checking can get stuck if not applied to names.

The other equivalences originate in the Idealised Algol literature [O’Hearn and Tennent, 2013], a call-by-name language with ground-type *local* variables. The proofs again use logical relations, but formulated in a denotational semantics of functor categories. Escalating the proofs to call-by-value and higher-order state requires more complex techniques such as step-indexed logical relations [Ahmed et al., 2009]. Our approach will be seen to be rather different, involving a case analysis of the operations involved as they interact in all possible pairwise combinations, in a type-free setting.

5 Focussed graph rewriting (technical details)

In this section we give a definition of the abstract machine introduced informally in Sec. 3.4. We call it a *universal abstract machine* because it can be seen as a universal algebra (the operations) combined with a mechanism for sharing or copying resources and scheduling evaluation. The machine, and hence the definitions below, are all globally parametrised by the operation set \mathbb{O} and its behaviour $B_{\mathbb{O}}$.

5.1 Auxiliary definitions

We use the terms *incoming* and *outgoing* to characterise the incidence relation between neighbouring edges. Conventionally incidence is defined relative to nodes, but we find it helpful to extend this notion to edges.

Definition 5.1 (Incoming and outgoing edges). An *incoming* edge of an edge e has a target that is a source of the edge e . An *outgoing* edge of the edge e has a source that is a target of the edge e .

Definition 5.2 (Path). A *path* in a hypergraph is given by a non-empty sequence of edges, where an edge e is followed by an edge e' if the edge e is an incoming edge of the edge e' .

Note that, in general, the first edge (resp. the last edge) of a path may have no source (resp. target). A path is said to be *from* a vertex v , if v is a source of the first edge of the path. Similarly, a path is said to be *to* a vertex v' , if v' is a target of the last edge of the path. A hypergraph G is itself said to be a path, if all edges of G comprise a path from an input (if any) and an output (if any) and every vertex is an endpoint of an edge.

Definition 5.3 (Reachability). A vertex v' is *reachable* from a vertex v if $v = v'$ holds, or there exists a path from the vertex v to the vertex v' .

To represent SPARTAN terms, we fix the vertex label set L and the edge label set $M_{\mathbb{O}}$ as described in Sec. 3.3, using the given operation set \mathbb{O} .

$$\begin{aligned} L &:= \{\star, \diamond\} \cup \{T^n(\star) \mid n \in \mathbb{N}\} \\ M_{\mathbb{O}} &:= \{! : \star \Rightarrow \diamond, ? : \star \Rightarrow \star, \checkmark : \star \Rightarrow \star, \downarrow : \star \Rightarrow \star\} \\ &\quad \cup \{\phi : \star \Rightarrow \star^{\otimes m_e} \otimes \otimes_{i=1}^{m_d} T^{n_i}(\star) \mid (\phi \vdash (m_e; n_1, \dots, n_{m_d})) \in \mathbb{O}\} \\ &\quad \cup \{\circ : \diamond \Rightarrow \star, \otimes_W^\ell : \epsilon \Rightarrow \ell, \otimes_C^\ell : \ell^{\otimes 2} \Rightarrow \ell \mid \ell \in \{\star, \diamond\}\} \end{aligned}$$

Definition 5.4 (Operation path). A path whose edges are all labelled with operations is called *operation path*.

Definition 5.5 (Contraction tree). For each $\ell \in \{\star, \diamond\}$, a *contraction tree* is a hypernet $(C : \ell^{\otimes k} \Rightarrow \ell) \in \mathcal{H}(\{\ell\}, \{\otimes_W^\ell, \otimes_C^\ell\})$, such that the unique output is reachable from each vertex.

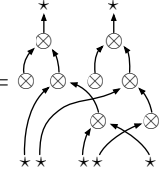

It can be observed that, for any contraction tree, an input (if any) is not an output but a source of a contraction edge.

Definition 5.6 (Distributor). We define a family $\{D_{k,m}^\ell : \ell^{\otimes km} \Rightarrow \ell^{\otimes k}\}_{k \in \mathbb{N}}$, with $\ell \in \{\star, \diamond\}$, of hypernets which we call *distributors*, inductively as follows:

$$\begin{aligned}
D_{0,m}^\ell &= \emptyset \\
D_{1,0}^\ell &= \uparrow \otimes \\
D_{1,1}^\ell &= \begin{array}{c} \uparrow \ell \\ \otimes \\ \swarrow \quad \searrow \\ \otimes \quad \ell \end{array} \\
D_{1,m+2}^\ell &= \begin{array}{c} \ell \\ \boxed{D_{1,m+1}^\ell} \\ \ell^{\otimes m} \end{array} \uparrow \otimes \begin{array}{c} \ell \\ \swarrow \quad \searrow \\ \ell \quad \ell \end{array} \\
D_{k+1,m}^\ell &= \Pi_\rho^{\text{id}} \left(\begin{array}{c} \ell^{\otimes k} \quad \ell \\ \boxed{D_{k,m}^\ell \quad D_{1,m}^\ell} \\ \ell^{\otimes km} \quad \ell^{\otimes m} \end{array} \right),
\end{aligned}$$

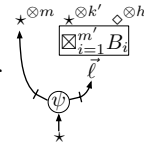
where \emptyset denotes the empty hypernet, id is the identity map, and ρ is a bijection such that, for each $j \in \{1, \dots, k\}$ and $i \in \{1, \dots, m\}$, $\rho(j + (k+1)(i-1)) = j + k(i-1)$ and $\rho((k+1)i) = km + i$.

When $k = 1$, a distributor $D_{1,m}^\ell$ is a contraction tree that includes one weakening edge. It is indeed a normal form with respect to the laws on contraction trees that we introduce later. For

instance, $D_{2,3}^\star =$  and $D_{3,0}^\diamond =$ .

Definition 5.7 (Box/stable hypernets). If a hypernet is a path of only one box edge, it is called *box* hypernet. A *stable* hypernet is a hypernet $(G : \star \Rightarrow \otimes_{i=1}^m \ell_i) \in \mathcal{H}(L, \{1\} \cup \mathbb{O}_\vee)$, such that $\otimes_{i=1}^m \ell_i \in (\{\diamond\} \cup \{T^n(\star) \mid n \in \mathbb{N}\})^m$ and each vertex is reachable from the unique input.

Definition 5.8 (Copyable hypernets). A hypernet $H : \star \Rightarrow \star^{\otimes k} \otimes \diamond^{\otimes h}$ is called *copyable* if it is .

or , where $\phi \in \mathbb{O}$, and each B_i is a box hypernet.

Definition 5.9 (One-way hypernets). A hypernet H is *one-way* if, for any pair (v_i, v_o) of an input and an output of H such that v_i and v_o both have type \star , any path from v_i to v_o is not an operation path.

Remark 5.10 (Distributors). The reader familiar with diagrammatic languages based on monoidal categories equipped with “sharing” (co)monoid operators, such as the ZX-calculus of [Coecke and Duncan \[2011\]](#), the distributor nets may seem an awkward alternative to quotienting the nets by the equational properties of the (co)monoid operator. Indeed a formulation of SPARTAN semantics in which distributor nets are collapsed into n -ary contractions would be quite accessible. However, the structural laws of SPARTAN including equational properties of contraction, mentioned in Sec. 4.1, can be invalidated by certain ill-behaved but definable operations. Forcing these properties into the framework does not seem to be practically possible, as it leads to intractable interactions between such complex n -ary contractions and operations in the context as required by the key notion of robustness which will be introduced in Sec. 6.3.

5.2 Focussed hypernets

Definition 5.11. A token edge in a hypergraph is said to be *exposed* if its source is an input and its target is an output, and *self-acyclic* if its source and its target are different vertices.

Definition 5.12 (Focussed hypernets). *Focussed* hypernets (typically ranged over by $\dot{G}, \dot{H}, \dot{N}$) are those that contain only one token edge and the edge is shallow, self-acyclic and not exposed.

Focus-free hypernets are given by $\mathcal{H}_\omega(L, M_\mathbb{O} \setminus \{?, \checkmark, \downarrow\})$, i.e. hypernets without token edges. A focussed hypernet \dot{G} can be turned into an *underlying* focus-free hypernet $|\dot{G}|$ with the same type, by removing its unique token edge and identifying the source and the target of the edge. When a focussed hypernet \dot{G} has a t-token, then changing the token label t to another one t' yields a focussed hypernet denoted by $\langle \dot{G} \rangle_{t'/t}$. The source (resp. target) of a token is called “token source” (resp. “token target”) in short.

Given a focus-free hypernet G , a focussed hypernet $t_i G$ with the same type can be yielded by connecting a t-token to the i -th input of G if the input has type \star . Similarly, a focussed hypernet $G_i t$ with the same type can be yielded by connecting a t-token to the i -th output of G if the output has type \star . If it is not ambiguous, we omit the index i in the notation $_i$.

5.3 Contexts

The set of *holed* hypernets (typically ranged over by \mathcal{C}) is given by $\mathcal{H}_\omega(L, M_\mathbb{O} \cup \mathbb{M})$, where the edge label set $M_\mathbb{O}$ is extended by a set \mathbb{M} of *hole* labels. Hole labels are typed, and typically ranged over by $\chi : \vec{\ell} \Rightarrow \vec{\ell}'$.

Definition 5.13 (Contexts). A holed hypernet \mathcal{C} is said to be *context* if each hole label appears at most once (at any depth) in \mathcal{C} .

Definition 5.14. A *simple* context is a context that contains a single hole, which is shallow.

When $\vec{\chi}$ gives a list of all and only hole labels that appear in a context \mathcal{C} , the context can be also written as $\mathcal{C}[\vec{\chi}]$; a hypernet in $\mathcal{H}_\omega(L, M_\mathbb{O})$ can be seen as a “context without a hole”, $\mathcal{C}[]$.

Let $\mathcal{C}[\vec{\chi}^1, \chi, \vec{\chi}^2]$ and $\mathcal{C}'[\vec{\chi}^3]$ be contexts, such that the hole χ and the latter context \mathcal{C}' have the same type and $\vec{\chi}^1 \cap \vec{\chi}^2 \cap \vec{\chi}^3 = \emptyset$. A new context $\mathcal{C}[\vec{\chi}^1, \mathcal{C}', \vec{\chi}^2] \in \mathcal{H}_\omega(L, M_\mathbb{O} \cup \vec{\chi}^1 \cup \vec{\chi}^3 \cup \vec{\chi}^2)$ can be obtained by *plugging* \mathcal{C}' into \mathcal{C} : namely, by replacing the (possibly deep) hole edge of \mathcal{C} that has label χ with the context \mathcal{C}' , and by identifying each input (resp. output) of \mathcal{C}' with its corresponding source (resp. target) of the hole edge (Def. A.5). Each edge of the new context $\mathcal{C}[\vec{\chi}^1, \mathcal{C}', \vec{\chi}^2]$ is inherited from either \mathcal{C} or \mathcal{C}' , keeping the type; this implies that the new context is indeed a context with hole labels $\vec{\chi}^1, \vec{\chi}^3, \vec{\chi}^2$. Inputs and outputs of the new context coincide with those of the original context \mathcal{C} , and hence these two contexts have the same type. The plugging is associative in two senses: plugging two contexts into two holes of a context yields the same result regardless of the order, i.e. $\mathcal{C}[\vec{\chi}^1, \mathcal{C}', \vec{\chi}^2, \mathcal{C}'', \vec{\chi}^3]$ is well-defined; and nested plugging yields the same result regardless of the order, i.e. $\mathcal{C}[\vec{\chi}^1, \mathcal{C}'[\vec{\chi}^3, \mathcal{C}'', \vec{\chi}^4], \vec{\chi}^2] = (\mathcal{C}[\vec{\chi}^1, \mathcal{C}', \vec{\chi}^2])[\vec{\chi}^3, \mathcal{C}'', \vec{\chi}^4, \vec{\chi}^2]$.

The notions of focussed and focus-free hypernets can be naturally extended to contexts. In a focussed context $\dot{\mathcal{C}}[\vec{\chi}]$, the token is said to be *entering* if it is an incoming edge of a hole, and *exiting* if it is an outgoing edge of a hole. The token may be both entering and exiting.

5.4 States and transitions

Given the two parameters \mathbb{O} and $B_\mathbb{O}$, the *universal abstract machine* $\mathcal{U}(\mathbb{O}, B_\mathbb{O})$ is defined as a state transition system. It is namely given by data $(S_\mathbb{O}, T \uplus B_\mathbb{O})$ as follows, each of which we will describe in the sequel.

- $S_\mathbb{O} \subseteq \mathcal{H}_\omega(L, M_\mathbb{O})$ is a set of *states*,
- $T \subseteq S_\mathbb{O} \times S_\mathbb{O}$ is a set of *intrinsic transitions*, and
- $B_\mathbb{O} \subseteq S_\mathbb{O} \times S_\mathbb{O}$ is a set of *extrinsic transitions*.

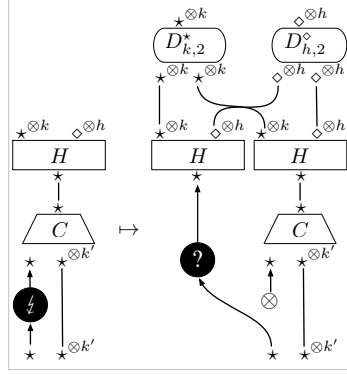


Figure 7: Contraction rules, with C a contraction tree, and H a copyable hypernet

A focussed hypernet of type $\star \Rightarrow \epsilon$ in $\mathcal{H}_\omega(L, M_\mathbb{O})$ is said to be a *state*. A state \dot{G} is called *initial* if $\dot{G} = ?; |\dot{G}|$, and *final* if $\dot{G} = \checkmark; |\dot{G}|$. A state is said to be *stuck* if it is not final and cannot be followed by any transition. An *execution* on a focus-free hypernet $G : \star \Rightarrow \epsilon$ is a sequence of transitions starting from an initial state $?; G$. The following will be apparent once transitions are defined: initial states are indeed initial in the sense that no search transition results in an initial state; and final states are indeed final in the sense that no transition is possible from a final state.

The interaction rules in Fig. 3 specify the first class of intrinsic transitions, *search* transitions, and the contraction rules in Fig. 7 specify the second class of intrinsic transitions, *copy* transitions. These intrinsic transitions are defined as follows: for each interaction rule $\dot{G} \mapsto^{\otimes} \dot{G}'$ (or resp. contraction rule $\dot{G} \mapsto \dot{G}'$), if there exists a focus-free simple context $\mathcal{C}[\chi] : \star \Rightarrow \epsilon$ such that $\mathcal{C}[\dot{G}]$ and $\mathcal{C}[\dot{G}']$ are states, $\mathcal{C}[\dot{G}] \rightarrow \mathcal{C}[\dot{G}']$ is a search transition (or resp. copy transition).

Search transitions are deterministic, because at most one interaction rule can be applied at any state. Although two different contraction rules may be possible at a state, copy transitions are still deterministic. Namely, if two different contraction rules $\dot{G} \mapsto \dot{G}'$ and $\dot{H} \mapsto \dot{H}'$ can be applied to the same state, i.e. there exist focus-free simple contexts \mathcal{C}_G and \mathcal{C}_H such that $\mathcal{C}_G[\dot{G}] = \mathcal{C}_H[\dot{H}]$, then these two rules yield the same transition, by satisfying $\mathcal{C}_G[\dot{G}'] = \mathcal{C}_H[\dot{H}']$. Informally, in Fig. 7, H is determined uniquely and the choice of C does not affect the result.

Intrinsic transitions are therefore all deterministic, and moreover, search transitions are reversible because the inverse of the interaction rules is again deterministic. When a sequence $\dot{G} \rightarrow^* \dot{G}'$ of transitions consists of search transitions only, it is annotated by the symbol \bullet as $\dot{G} \rightarrow^* \dot{G}'$.

An execution on any stable net, or on representation of any value, terminates successfully at a final state with only search transitions (by Lem. A.16, Lem. A.18 and Lem. A.20(1)).

The behaviour $B_\mathbb{O}$, which is a parameter of the machine, specifies a set of extrinsic transitions. Extrinsic transitions are also dubbed “compute” transitions, and each of them must target an active operation. Namely, a transition $\dot{G} \rightarrow \dot{G}'$ is a compute transition if: the first state \dot{G} has a rewrite token ($\frac{1}{2}$) that is an incoming edge of an active operation edge; and the second state \dot{G}' has a search token (?). Copy transitions or compute transitions are possible if and only if a state has a rewrite token ($\frac{1}{2}$), and they always change the token to a search token (?). We refer to copy transitions and compute transitions altogether as “rewrite” transitions.

Compute transitions may be specified locally, by *rewrite rules*, in the same manner as the intrinsic transitions. We leave it entirely open what the actual rewrite associated to some operation is, by having the behaviour $B_\mathbb{O}$ as parameter as well as the operation set \mathbb{O} . This is part of the semantic flexibility of our framework. We do not specify a meta-language for encoding effects as particular transitions. Any *algorithmic* state transformation is acceptable.

Remark 5.15. This abstract machine is “universal” in a sense of the word similar to the way it is used in “universal algebra” rather than in “universal Turing machine”. It is a general abstract framework in which a very wide range of concrete abstract machines can be instantiated by providing operations and their rewrite rules.

6 A characterisation theorem

Given the universal abstract machine $\mathcal{U}(\mathbb{O}, B_{\mathbb{O}})$ with parameters \mathbb{O} and $B_{\mathbb{O}}$, we can now define the notions of contextual refinement and equivalence, and formalise a proof method for them. All the technical development in this section is with respect to the machine $\mathcal{U}(\mathbb{O}, B_{\mathbb{O}})$, and hence globally parametrised by \mathbb{O} and $B_{\mathbb{O}}$.

Firstly in Sec. 6.1, we state the condition of the machine, to which the proof method applies. Sec. 6.2 proposes our generalised notions of contextual refinement and equivalence. Sec. 6.3 formalises the proof method as a *characterisation theorem* (Thm. 6.14), introducing the key concept of *robustness*. Finally, in Sec. 6.4, we list some useful lemmas that can be used in robustness check.

6.1 Determinism and refocusing

We here focus on the situation where the universal abstract machine is both *deterministic* as a state transition system, and *refocusing* in the following sense.

Definition 6.1 (Rooted states and refocusing machine).

- A state \dot{G} is *rooted* if $?\dot{;} |\dot{G}| \xrightarrow{*} \dot{G}$.
- The universal abstract machine $\mathcal{U}(\mathbb{O}, B_{\mathbb{O}})$ is *refocusing* if every transition preserves the rooted property.

Because intrinsic transitions are all deterministic, the machine becomes deterministic if the behaviour $B_{\mathbb{O}}$ specifies deterministic compute transitions.

Any initial state is trivially rooted, and search transitions preserve the rooted property. The *stationary* property below gives a sufficient condition for a rewrite transition to preserve the rooted property (Lem. A.13).

Definition 6.2 (Stationary rewrite transitions). A rewrite transition $\dot{G} \rightarrow \dot{G}'$ is *stationary* if there exist a focus-free simple context \mathcal{C} , focus-free hypernets H and H' , and a number $i \in \mathbb{N}$, such that H is one-way, $\dot{G} = \mathcal{C}[\dot{;}_i H]$ and $\dot{G}' = \mathcal{C}[\dot{;}_i H']$, and the following holds. For any $j \in \mathbb{N} \setminus \{i\}$, such that $\mathcal{C}[\dot{;}_j H]$ is a state, there exists a state \dot{N} with a rewrite token, such that $\mathcal{C}[\dot{;}_j H] \xrightarrow{*} \dot{N}$.

Copy transitions are stationary, and hence they preserve the rooted property, because each input of the contraction tree C in Fig. 7 is a source of a contraction edge. Therefore, the machine becomes refocusing if the behaviour $B_{\mathbb{O}}$ specifies compute transitions that preserve the rooted property.

Remark 6.3 (Refocusing). When a rewrite transition results in a rooted state \dot{N}' , starting the search process (i.e. search transitions) from the state \dot{N}' can be seen as resuming the search process $?\dot{;} |\dot{N}'| \xrightarrow{*} \dot{N}'$, from an initial state, on the underlying hypernet $|\dot{N}'|$. Resuming redex search after a rewrite, rather than starting from scratch, is an important aspect of abstract machines. In the case of the lambda-calculus, enabling the resumption is identified as one of the key steps (called “refocusing”) to synthesise abstract machines from reduction semantics by Danvy et al. [2012]. In our setting, it is preservation of the rooted property that justifies the resumption.

The stationary property, as a sufficient condition of preservation of the rooted property, characterises many operations with local behaviour. Compute transitions of operations that involve non-local change of a token position, like label jumping, could preserve the rooted property without being stationary.

6.2 Contextual refinement and equivalence

We propose notions of contextual refinement and equivalence that check for successful termination of execution, taking the number of transitions into account. These notions are with respect to the universal abstract machine $\mathcal{U}(\mathbb{O}, B_{\mathbb{O}})$, and parametrised by the following: a set $\mathbb{C} \subseteq \mathcal{H}_{\omega}(L, M_{\mathbb{O}} \cup \mathbb{M})$ of focus-free contexts that is closed under plugging (i.e. for any contexts $\mathcal{C}[\chi^1, \chi, \chi^2], \mathcal{C}' \in \mathbb{C}$ such that $\mathcal{C}[\chi^1, \mathcal{C}', \chi^2]$ is defined, $\mathcal{C}[\chi^1, \mathcal{C}', \chi^2] \in \mathbb{C}$); and a preorder Q on natural numbers.

Definition 6.4 (State refinement and equivalence). Let Q be a preorder on \mathbb{N} , and \dot{G}_1 and \dot{G}_2 be two states.

- \dot{G}_1 is said to *refine* \dot{G}_2 up to Q , written as $B_0 \models (\dot{G}_1 \dot{\preceq}_Q \dot{G}_2)$, if for any number $k_1 \in \mathbb{N}$ and any final state \dot{N}_1 such that $\dot{G}_1 \rightarrow^{k_1} \dot{N}_1$, there exist a number $k_2 \in \mathbb{N}$ and a final state \dot{N}_2 such that $k_1 Q k_2$ and $\dot{G}_2 \rightarrow^{k_2} \dot{N}_2$.
- \dot{G}_1 and \dot{G}_2 are said to be *equivalent* up to Q , written as $B_0 \models (\dot{G}_1 \dot{\simeq}_Q \dot{G}_2)$, if $B_0 \models (\dot{G}_1 \dot{\preceq}_Q \dot{G}_2)$ and $B_0 \models (\dot{G}_2 \dot{\preceq}_Q \dot{G}_1)$.

Definition 6.5 (Contextual refinement and equivalence). Let \mathbb{C} be a set of contexts that is closed under plugging, Q be a preorder on \mathbb{N} , and H_1 and H_2 be focus-free hypernets of the same type.

- H_1 is said to *contextually refine* H_2 in \mathbb{C} up to Q , written as $B_0 \models (H_1 \preceq_Q^{\mathbb{C}} H_2)$, if any focus-free context $\mathcal{C}[\chi] \in \mathbb{C}$, such that $?\mathcal{C}[H_1]$ and $?\mathcal{C}[H_2]$ are states, yields refinement $B_0 \models (?\mathcal{C}[H_1] \dot{\preceq}_Q ?\mathcal{C}[H_2])$.
- H_1 and H_2 are said to be *contextually equivalent* in \mathbb{C} up to Q , written as $B_0 \models (H_1 \simeq_Q^{\mathbb{C}} H_2)$, if $B_0 \models (H_1 \preceq_Q^{\mathbb{C}} H_2)$ and $B_0 \models (H_2 \preceq_Q^{\mathbb{C}} H_1)$.

In the sequel, we simply write $\dot{G}_1 \dot{\preceq}_Q \dot{G}_2$ etc., making the parameter B_0 implicit.

Because Q is a preorder, $\dot{\preceq}_Q$ and $\dot{\preceq}_Q^{\mathbb{C}}$ are indeed preorders, and accordingly, equivalences $\dot{\simeq}_Q$ and $\dot{\simeq}_Q^{\mathbb{C}}$ are indeed equivalences (Lem A.22). Examples of preorder Q include: the universal relation $\mathbb{N} \times \mathbb{N}$, the “greater-than-or-equal” order $\geq_{\mathbb{N}}$, and the equality $=_{\mathbb{N}}$.

When the relation Q is the universal relation $\mathbb{N} \times \mathbb{N}$, the notions concern successful termination, and the number of transitions is irrelevant. If all compute transitions are deterministic, contextual equivalences $\dot{\simeq}_{\geq_{\mathbb{N}}}^{\mathbb{C}}$ and $\dot{\simeq}_{=_{\mathbb{N}}}^{\mathbb{C}}$ coincide for any \mathbb{C} (as a consequence of Lem. A.23).

Because \mathbb{C} is closed under plugging, the contextual notions $\dot{\preceq}_Q^{\mathbb{C}}$ and $\dot{\simeq}_Q^{\mathbb{C}}$ indeed become congruences. Namely, for any $H_1 \sqsubseteq^{\mathbb{C}} H_2$ and $\mathcal{C} \in \mathbb{C}$ such that $\mathcal{C}[H_1]$ and $\mathcal{C}[H_2]$ are defined, $\mathcal{C}[H_1] \sqsubseteq^{\mathbb{C}} \mathcal{C}[H_2]$, where $\sqsubseteq \in \{\dot{\preceq}_Q, \dot{\simeq}_Q\}$.

As the parameter \mathbb{C} , we will particularly use the set $\mathbb{C}_0 \subseteq \mathcal{H}_w(L, M_0 \cup \mathbb{M})$ of any focus-free contexts, and its subset $\mathbb{C}_{0\text{-bf}}$ of *binding-free* contexts.

Definition 6.6 (Binding-free contexts). A focus-free context \mathcal{C} is said to be *binding-free* if there exists no path, at any depth, from a source of a contraction, atom, box or hole edge, to a source of a hole edge.

The set \mathbb{C}_0 is closed under plugging, and so is the set $\mathbb{C}_{0\text{-bf}}$ (Lem. A.21). Restriction to binding-free contexts are necessary to focus on call-by-value languages, because only values are bound in these languages. The restriction reflects syntactical restriction on contexts as $C ::= [] \mid \phi(\vec{t}, C, \vec{t}'; \vec{s}) \mid \vec{y}.C \mid \text{bind } x \rightarrow t \text{ in } C \mid \text{new } a \multimap t \text{ in } C$, where the hole ‘ $[]$ ’ does not appear in the bound positions.

6.3 Templates and robustness

Reasoning with graphs is based on the concept of *template*. Informally speaking, templates are families of graphs which will be used in (in)equational reasoning. The rewrite rules of the operations are natural candidates for templates. For example the families of graphs interpreting SPARTAN terms for $m + n$ and p (where $p = m + n$) could be used as templates. But any families of graphs, subject to the conditions below, can be used as templates.

Throughout this section, let \mathbb{C} be a set of contexts, and Q, Q' and Q'' be binary relations on \mathbb{N} .

Definition 6.7 (Pre-template). A *pre-template* is a union $\triangleleft := \cup_{I \in \mathcal{I}} \triangleleft_I$ of a family of binary relations on focus-free hypernets $\mathcal{H}_w(L, M_0 \setminus \{?, \checkmark, \dagger\})$ indexed by a set \mathcal{I} of types, such that for any $G_1 \triangleleft_I G_2$ where $I \in \mathcal{I}$, G_1 and G_2 are focus-free hypernets with type $G_1 : I$ and $G_2 : I$.

Obviously, if \triangleleft is a pre-template, its converse \triangleleft^{-1} is also a pre-template.

Definition 6.8 ((Quasi-)specimens). Let \triangleleft be a pre-template, and R and R' be binary relations on states.

1. A triple $(\dot{C}[\vec{\chi}]; \vec{H}^1; \vec{H}^2)$ is a \mathbb{C} -specimen of \triangleleft if the following hold:
 - (A) $|\dot{C}[\vec{\chi}]| \in \mathbb{C}$, and $|\vec{\chi}| = |\vec{H}^1| = |\vec{H}^2|$.
 - (B) $H_i^1 \triangleleft H_i^2$ for each $i \in \{1, \dots, |\vec{\chi}|\}$.
 - (C) $\dot{C}[\vec{H}^p]$ is a state for each $p \in \{1, 2\}$.
2. A pair (\dot{N}_1, \dot{N}_2) of states is a *quasi- \mathbb{C} -specimen* of \triangleleft up to (R, R') , if there exists a \mathbb{C} -specimen $(\dot{C}; \vec{H}^1; \vec{H}^2)$ of \triangleleft such that the following hold:
 - (A) The tokens of \dot{C} , \dot{N}_1 and \dot{N}_2 all have the same label.
 - (B) If \dot{N}_1 and \dot{N}_2 are rooted, then $\dot{C}[\vec{H}^1]$ and $\dot{C}[\vec{H}^2]$ are also rooted, $\dot{N}_1 R \dot{C}[\vec{H}^1]$, and $\dot{C}[\vec{H}^2] R' \dot{N}_2$.

We can refer to *the* token label of a \mathbb{C} -specimen and a quasi- \mathbb{C} -specimen. Any \mathbb{C} -specimen $(\dot{C}; \vec{H}^1; \vec{H}^2)$ gives a quasi- \mathbb{C} -specimen $(\dot{C}[\vec{H}^1], \dot{C}[\vec{H}^2])$ up to $(=, =)$. We say a \mathbb{C} -specimen $(\dot{C}[\vec{\chi}]; \vec{H}^1; \vec{H}^2)$ is *single* if $|\vec{\chi}| = 1$, i.e. the context \dot{C} has exactly one hole edge (at any depth).

Definition 6.9 (Input-safety). A pre-template \triangleleft is (\mathbb{C}, Q, Q') -input-safe if, for any \mathbb{C} -specimen $(\dot{C}; \vec{H}^1; \vec{H}^2)$ of \triangleleft such that \dot{C} has an entering search token, one of the following holds.

- (I) There exist two stuck states \dot{N}_1 and \dot{N}_2 such that $\dot{C}[\vec{H}^p] \rightarrow^* \dot{N}_p$ for each $p \in \{1, 2\}$.
- (II) There exist a \mathbb{C} -specimen $(\dot{C}'; \vec{H}^1; \vec{H}^2)$ of \triangleleft and two numbers $k_1, k_2 \in \mathbb{N}$, such that the token of \dot{C}' is not a rewrite token and not entering, $(1 + k_1) Q k_2, \dot{C}[\vec{H}^1] \rightarrow^{1+k_1} \dot{C}'[\vec{H}^1]$, and $\dot{C}[\vec{H}^2] \rightarrow^{k_2} \dot{C}'[\vec{H}^2]$.
- (III) There exist a quasi- \mathbb{C} -specimen (\dot{N}_1, \dot{N}_2) of \triangleleft up to $(\simeq_{Q'}, \simeq_{Q'})$, whose token is not a rewrite token, and two numbers $k_1, k_2 \in \mathbb{N}$, such that $(1+k_1) Q (1+k_2), \dot{C}[\vec{H}^1] \rightarrow^{1+k_1} \dot{N}_1$, and $\dot{C}[\vec{H}^2] \rightarrow^{1+k_2} \dot{N}_2$.

Definition 6.10 (Output-closure). A pre-template \triangleleft is *output-closed* if, for any hypernets $H_1 \triangleleft H_2$, either H_1 or H_2 is one-way.

Definition 6.11 (Templates). A pre-template \triangleleft is a (\mathbb{C}, Q, Q') -template, if it is (\mathbb{C}, Q, Q') -input-safe and also output-closed.

Definition 6.12 (Robustness). A pre-template \triangleleft is (\mathbb{C}, Q, Q', Q'') -robust relative to a rewrite transition $\dot{N} \rightarrow \dot{N}'$ if, for any \mathbb{C} -specimen $(\dot{C}; \vec{H}^1; \vec{H}^2)$ of \triangleleft , such that $\dot{C}[\vec{H}^1] = \dot{N}$ and the token of \dot{C} is a rewrite token and not entering, one of the following holds.

- (I) $\dot{C}[\vec{H}^1]$ or $\dot{C}[\vec{H}^2]$ is not rooted.
- (II) There exists a stuck state \dot{N}'' such that $\dot{N}' \rightarrow^* \dot{N}''$.
- (III) There exist a quasi- \mathbb{C} -specimen $(\dot{N}_1'', \dot{N}_2'')$ of \triangleleft up to $(\preceq_{Q'}, \preceq_{Q''})$, whose token is not a rewrite token, and two numbers $k_1, k_2 \in \mathbb{N}$, such that $(1 + k_1) Q k_2, \dot{N}' \rightarrow^{k_1} \dot{N}_1''$, and $\dot{C}[\vec{H}^2] \rightarrow^{k_2} \dot{N}_2''$.

Finally, the characterisation theorem (Thm. 6.14 below) enables us to prove contextual refinement $\preceq_Q^{\mathbb{C}}$ using state refinements $\preceq_{Q'}$ and $\preceq_{Q''}$, under the assumption that the triple (Q, Q', Q'') is “reasonable” in the following sense.

Definition 6.13 (Reasonable triples). A triple (Q, Q', Q'') of preorders on \mathbb{N} is *reasonable* if the following hold:

- (A) Q is closed under addition, i.e. any $k_1 Q k_2$ and $k'_1 Q k'_2$ satisfy $(k_1 + k'_1) Q (k_2 + k'_2)$.
- (B) $Q' \subseteq \geq_{\mathbb{N}}$, and $Q' \subseteq Q''$.
- (C) $Q' \circ Q \circ Q'' \subseteq Q$, where \circ denotes composition of binary relations.

Examples of a reasonable triple (Q, Q', Q'') include: $(\mathbb{N} \times \mathbb{N}, \geq_{\mathbb{N}}, \mathbb{N} \times \mathbb{N})$, $(\mathbb{N} \times \mathbb{N}, \geq_{\mathbb{N}}, \geq_{\mathbb{N}})$, $(\mathbb{N} \times \mathbb{N}, =_{\mathbb{N}}, =_{\mathbb{N}})$, $(\geq_{\mathbb{N}}, \geq_{\mathbb{N}}, \geq_{\mathbb{N}})$, $(\geq_{\mathbb{N}}, =_{\mathbb{N}}, \geq_{\mathbb{N}})$, $(\leq_{\mathbb{N}}, =_{\mathbb{N}}, \leq_{\mathbb{N}})$, $(\geq_{\mathbb{N}}, =_{\mathbb{N}}, =_{\mathbb{N}})$, $(\leq_{\mathbb{N}}, =_{\mathbb{N}}, =_{\mathbb{N}})$, $(=_{\mathbb{N}}, =_{\mathbb{N}}, =_{\mathbb{N}})$.

Theorem 6.14. *If an universal abstract machine $\mathcal{U}(\mathbb{O}, B_{\mathbb{O}})$ is deterministic and refocusing, it satisfies the following property. For any set $\mathbb{C} \subseteq \mathcal{H}_{\omega}(L, M_{\mathbb{O}} \cup \mathbb{M})$ of contexts that is closed under plugging, any reasonable triple (Q, Q', Q'') , and any pre-template \triangleleft on focus-free hypernets $\mathcal{H}_{\omega}(L, M_{\mathbb{O}} \setminus \{?, \checkmark, \frac{1}{2}\})$:*

1. *If \triangleleft is a (\mathbb{C}, Q, Q') -template and (\mathbb{C}, Q, Q', Q'') -robust relative to all rewrite transitions, then \triangleleft implies contextual refinement in \mathbb{C} up to Q , i.e. any $G_1 \triangleleft G_2$ implies $G_1 \preceq_Q^{\mathbb{C}} G_2$.*
2. *If \triangleleft is a (\mathbb{C}, Q^{-1}, Q') -template and the converse \triangleleft^{-1} is (\mathbb{C}, Q, Q', Q'') -robust relative to all rewrite transitions, then \triangleleft^{-1} implies contextual refinement in \mathbb{C} up to Q , i.e. any $G_1 \triangleleft G_2$ implies $G_2 \preceq_Q^{\mathbb{C}} G_1$.*

Proof. This is a consequence of Prop. 7.6, Prop. 7.2 and Prop. 7.4 in Sec. 7. \square

Remark 6.15 (Monotonicity). *Contextual/state refinement and equivalence are monotonic with respect to Q , in the sense that $Q_1 \subseteq Q_2$ implies $\square_{Q_1} \subseteq \square_{Q_2}$ for each $\square \in \{\preceq, \simeq, \preceq^{\mathbb{C}}, \simeq^{\mathbb{C}}\}$. Contextual refinement and equivalence are anti-monotonic with respect to \mathbb{C} , in the sense that $\mathbb{C}_1 \subseteq \mathbb{C}_2$ implies $\square^{\mathbb{C}_2} \subseteq \square^{\mathbb{C}_1}$ for each $\square \in \{\preceq_Q, \simeq_Q\}$. This means, in particular, $\simeq_Q^{\mathbb{C}_0} \subseteq \simeq_Q^{\mathbb{C}_0 \text{-bf}}$.*

Given that $\mathbb{C}_1 \subseteq \mathbb{C}_2$, $Q_1 \subseteq Q_2$, $Q'_1 \subseteq Q'_2$, $Q''_1 \subseteq Q''_2$, $R_1 \subseteq R_2$ and $R'_1 \subseteq R'_2$, the following holds. Any \mathbb{C}_1 -specimen is a \mathbb{C}_2 -specimen, and any quasi- \mathbb{C}_1 -specimen up to (R_1, R'_1) is a quasi- \mathbb{C}_2 -specimen up to (R_2, R'_2) . Any (\mathbb{C}, Q_1, Q'_1) -template is a (\mathbb{C}, Q_2, Q'_2) -template. If \triangleleft is $(\mathbb{C}, Q_1, Q'_1, Q''_1)$ -robust relative to a rewrite transition, then it is also $(\mathbb{C}, Q_2, Q'_2, Q''_2)$ -robust relative to the same transition. Note that the notions of template and robustness are not monotonic nor anti-monotonic with respect to \mathbb{C} .

To prove that a pre-template \triangleleft induces contextual equivalence, one can use Thm. 6.14(1) twice with respect to \triangleleft and \triangleleft^{-1} . One can alternatively use Thm. 6.14(1) and Thm. 6.14(2), both with respect to \triangleleft . This alternative approach is often more economical. The reason is that the approach involves proving input-safety of \triangleleft with respect to two parameters (\mathbb{C}, Q, Q') and (\mathbb{C}, Q^1, Q') , which typically boils down to a proof for one parameter, thanks to the monotonicity.

6.4 Sufficient conditions for robustness

A proof of robustness becomes trivial for a specimen with a rewrite token that gives a non-rooted state. Thanks to the lemma below, we can show that a state is not rooted, by checking paths from the token target.

Definition 6.16 (Accessible paths).

- A path of a hypernet is said to be *accessible* if it consists of edges whose all sources have type \star .
- An accessible path is called *stable* if the labels of its edges are included in $\{I\} \cup \mathbb{O}_{\checkmark}$.
- An accessible path is called *active* if it starts with one active operation edge and possibly followed by a stable path.

Note that box edges and atom edges never appear in an accessible path.

Lemma 6.17. *If a state has a rewrite token that is not an incoming edge of a contraction edge, then the state satisfies the following property: If there exists an accessible, but not active, path from the token target, then the state is not rooted.*

Proof. This is a contraposition of a consequence of Lem. A.10 and Lem. A.20(2). \square

Checking the condition (III) of robustness (see Def. 6.12) involves finding a quasi- \mathbb{C} -specimen of \triangleleft up to $(\preceq_{Q'}, \preceq_{Q''})$, namely checking the condition (B) of Def. 6.8(2). The following lemma enables us to use contextual refinement $\preceq_Q^{\mathbb{C}}$ to yield state refinement $\preceq_{Q'}$, via single \mathbb{C} -specimens of a certain pre-template \triangleleft .

Definition 6.18. A pre-template \triangleleft is a *trigger* if it satisfies the following:

(A) For any single \mathbb{C} -specimen $(\dot{\mathcal{C}}[\chi]; H^1; H^2)$ of \triangleleft , such that $\dot{\mathcal{C}}$ has an entering search token, $\dot{\mathcal{C}}[H^p] \rightarrow \langle \dot{\mathcal{C}}[H^p] \rangle_{\sharp/?}$ for each $p \in \{1, 2\}$.

(B) For any hypernets $H^1 \triangleleft H^2$, both H_1 and H_2 are one-way.

Lemma 6.19. Let \mathbb{C} be a set of contexts, and Q' be a binary relation on \mathbb{N} such that, for any $k_0, k_1, k_2 \in \mathbb{N}$, $(k_0 + k_1) Q' (k_0 + k_2)$ implies $k_1 Q' k_2$. Let \triangleleft be a pre-template that is a trigger and implies contextual refinement $\preceq_{Q'}^{\mathbb{C}}$. For any single \mathbb{C} -specimen $(\dot{\mathcal{C}}[\chi]; H^1; H^2)$ of \triangleleft , if compute transitions are all deterministic, and one of states $\dot{\mathcal{C}}[H^1]$ and $\dot{\mathcal{C}}[H^2]$ is rooted, then the other state is also rooted, and moreover, $\dot{\mathcal{C}}[H^1] \preceq_{Q'} \dot{\mathcal{C}}[H^2]$.

Proof. This is a corollary of Lem. A.24. □

Remark 6.20. The notion of contextual refinement concerns initial states, and therefore, only enables us to safely replace a part of a hypernet before execution. Because any initial state is rooted, if all transitions preserve the rooted property, we can safely assume that any state that arises in an execution is rooted. If all transitions are also deterministic, Lem. 6.19 enables us to use some contextual refinement and safely replace a part of a hypernet during execution. This can validate run-time garbage collection, for example.

7 Proof of the characterisation theorem

This section details the proof of Thm. 6.14, with respect to the machine $\mathcal{U}(\mathbb{O}, B_{\mathbb{O}})$ parametrised by \mathbb{O} and $B_{\mathbb{O}}$.

At the core of the proof is step-wise reasoning, or transition-wise reasoning, using a lax variation of simulation. Providing a simulation boils down to case analysis on transitions, namely on possible interactions between the token and parts of states contributed by a pre-template. While output-closure helps us disprove some cases, input-safety and robustness give the cases that are specific to a pre-template and an operation set.

We also employ the so-called *up-to* technique in the use of quasi-specimens. Our variation of simulation is up to state refinements, with a quantitative restriction implemented by the notion of reasonable triple. This restriction is essential to make this particular up-to technique work, in combination with our lax variation of simulation. A similar form of up-to technique is studied categorically by Bonchi et al. [2017], but for the ordinary notion of (weak) simulation, without this quantitative restriction.

The lax variation of simulation we use is namely (Q, Q', Q'') -simulation, parametrised by a triple (Q, Q', Q'') . This provides a sound approach to prove state refinement \preceq_Q , using $\preceq_{Q'}$ and $\preceq_{Q''}$, given that all transitions are deterministic and (Q, Q', Q'') forms a reasonable triple.

Definition 7.1 ((Q, Q', Q'') -(bi)simulations). Let R be a binary relation on states, and (Q, Q', Q'') be a triple of preorders on \mathbb{N} . The binary relation R is a (Q, Q', Q'') -simulation if, for any two related states $\dot{G}_1 R \dot{G}_2$, the following (A) and (B) hold:

(A) If \dot{G}_1 is final, \dot{G}_2 is also final.

(B) If there exists a state \dot{G}'_1 such that $\dot{G}_1 \rightarrow \dot{G}'_1$, one of the following (I) and (II) holds:

(I) There exists a stuck state \dot{G}''_1 such that $\dot{G}'_1 \rightarrow^* \dot{G}''_1$.

(II) There exist two states \dot{H}_1 and \dot{H}_2 , and numbers $k_1, k_2 \in \mathbb{N}$, such that $\dot{H}_1 (\preceq_{Q'} \circ R \circ \preceq_{Q''}) \dot{H}_2$, $(1 + k_1) Q k_2$, $\dot{G}'_1 \rightarrow^{k_1} \dot{H}_1$, and $\dot{G}_2 \rightarrow^{k_2} \dot{H}_2$.

Proposition 7.2. When the universal abstract machine $\mathcal{U}(\mathbb{O}, B_{\mathbb{O}})$ is deterministic, it satisfies the following.

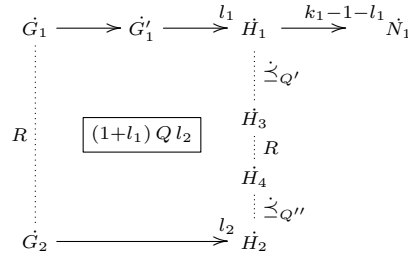
For any binary relation R on states, and any reasonable triple (Q, Q', Q'') , if R is a (Q, Q', Q'') -simulation, then R implies refinement up to Q , i.e. any $\dot{G}_1 R \dot{G}_2$ implies $\dot{G}_1 \preceq_Q \dot{G}_2$.

Proof. Our goal is to show the following: for any states $\dot{G}_1 R \dot{G}_2$, any number $k_1 \in \mathbb{N}$ and any final state \dot{N}_1 , such that $\dot{G}_1 \rightarrow^{k_1} \dot{N}_1$, there exist a number $k_2 \in \mathbb{N}$ and a final state \dot{N}_2 such that $k_1 Q k_2$ and $\dot{G}_2 \rightarrow^{k_2} \dot{N}_2$. The proof is by induction on $k_1 \in \mathbb{N}$.

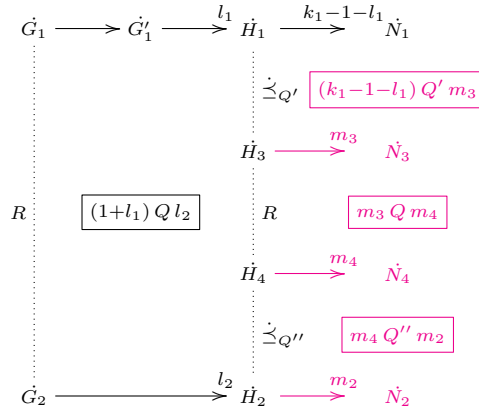
In the base case, when $k_1 = 0$, the state \dot{G}_1 is itself final because $\dot{G}_1 = \dot{N}_1$. Because R is a (Q, Q', Q'') -simulation, \dot{G}_2 is also a final state, which means we can take 0 as k_2 and \dot{G}_2 itself as \dot{N}_2 . Because (Q, Q', Q'') is a reasonable triple, Q is a preorder and $0 Q 0$ holds.

In the inductive case, when $k_1 > 0$, we assume the induction hypothesis on any $h \in \mathbb{N}$ such that $h < k_1$. Now that $k_1 > 0$, there exists a state \dot{G}'_1 such that $\dot{G}_1 \rightarrow \dot{G}'_1 \rightarrow^{k-1} \dot{N}_1$. Because all intrinsic transitions are deterministic, the assumption that compute transitions are all deterministic implies that states and transitions comprise a deterministic abstract rewriting system, in which final states and stuck states are normal forms. By Lem. A.6, we can conclude that there exists no stuck state \dot{G}''_1 such that $\dot{G}'_1 \rightarrow^* \dot{G}''_1$.

Therefore, by R being a (Q, Q', Q'') -simulation, there exist two states \dot{H}_1 and \dot{H}_2 , and numbers $l_1, l_2 \in \mathbb{N}$, such that $\dot{H}_1 (\preceq_{Q'} \circ R \circ \preceq_{Q''}) \dot{H}_2$, $(1 + l_1) Q l_2$, $\dot{G}'_1 \rightarrow^{l_1} \dot{H}_1$, and $\dot{G}_2 \rightarrow^{l_2} \dot{H}_2$. By the determinism, $1 + l_1 \leq k_1$ must hold; if \dot{H}_1 is a final state, $\dot{G}'_1 \rightarrow^{l_1} \dot{H}_1$ must coincide with $\dot{G}'_1 \rightarrow^{k-1} \dot{N}_1$; otherwise, $\dot{G}'_1 \rightarrow^{l_1} \dot{H}_1$ must be a suffix of $\dot{G}'_1 \rightarrow^{k-1} \dot{N}_1$. There exist two states \dot{H}_3 and \dot{H}_4 , and we have the following situation, where the relations R , $\preceq_{Q'}$ and $\preceq_{Q''}$ are represented by vertical dotted lines from top to bottom.



We expand the above diagram as below (indicated by magenta), in three steps.



Firstly, by definition of state refinement, there exist a number $m_3 \in \mathbb{N}$ and a final state \dot{N}_3 such that $(k_1 - 1 - l_1) Q' m_3$ and $\dot{H}_3 \rightarrow^{m_3} \dot{N}_3$. Because (Q, Q', Q'') is a reasonable triple, $Q' \subseteq \geq_{\mathbb{N}}$, and hence $k_1 > k_1 - 1 - l_1 \geq m_3$. Therefore, secondly, by induction hypothesis on m_3 , there exist a number $m_4 \in \mathbb{N}$ and a final state \dot{N}_4 such that $m_3 Q m_4$ and $\dot{H}_4 \rightarrow^{m_4} \dot{N}_4$. Thirdly, by definition of state refinement, there exist a number $m_2 \in \mathbb{N}$ and a final state \dot{N}_2 such that $m_4 Q'' m_2$ and $\dot{H}_2 \rightarrow^{m_2} \dot{N}_2$.

Now we have $(k_1 - 1 - l_1) Q' m_3$, $m_3 Q m_4$ and $m_4 Q'' m_2$, which means $(k_1 - 1 - l_1) (Q' \circ Q \circ Q'') m_2$. Because (Q, Q', Q'') is a reasonable triple, this implies $(k_1 - 1 - l_1) Q m_2$, and moreover, $k_1 Q (l_2 + m_2)$. We can take $l_2 + m_2$ as k_2 . \square

The token in a focussed context \dot{C} is said to be *remote*, if it is a search token, a value token, or not entering. The procedure of *contextual lifting* reduces a proof of contextual refinement down to that of state refinement.

Definition 7.3 (Contextual lifting). Let $\mathbb{C} \subseteq \mathcal{H}_\omega(L, M_\mathbb{O} \cup \mathbb{M})$ be a set of contexts. Given a pre-template \triangleleft on focus-free hypernets $\mathcal{H}_\omega(L, M_\mathbb{O} \setminus \{?, \checkmark, \dagger\})$, its \mathbb{C} -contextual lifting $\triangleleft^\mathbb{C}$ is a binary relation on states defined by: $\dot{G}_1 \triangleleft^\mathbb{C} \dot{G}_2$ if there exists a \mathbb{C} -specimen $(\dot{C}; \vec{H}^1; \vec{H}^2)$ of \triangleleft , such that the token of \dot{C} is remote, $\dot{G}_p = \dot{C}[\vec{H}^p]$, and $\dot{C}[\vec{H}^p]$ is rooted, for each $p \in \{1, 2\}$.

The contextual lifting $\triangleleft^\mathbb{C}$ is by definition a binary relation on rooted states.

Proposition 7.4. For any set $\mathbb{C} \subseteq \mathcal{H}_\omega(L, M_\mathbb{O} \cup \mathbb{M})$ of contexts that is closed under plugging, any preorder Q on \mathbb{N} , and any pre-template \triangleleft on focus-free hypernets $\mathcal{H}_\omega(L, M_\mathbb{O} \setminus \{?, \checkmark, \dagger\})$, if the \mathbb{C} -contextual lifting $\triangleleft^\mathbb{C}$ implies refinement $\dot{\preceq}_Q$ (resp. equivalence $\dot{\simeq}_Q$), then \triangleleft implies contextual refinement $\dot{\preceq}_Q^\mathbb{C}$ (resp. contextual equivalence $\dot{\simeq}_Q^\mathbb{C}$).

Proof of refinement case. Our goal is to show that, for any $H_1 \triangleleft H_2$ and any focus-free context $\mathcal{C}[\chi] \in \mathbb{C}$ such that $?; \mathcal{C}[H_1]$ and $?; \mathcal{C}[H_2]$ are states, we have refinement $?; \mathcal{C}[H_1] \dot{\preceq}_Q ?; \mathcal{C}[H_2]$.

Because $?; \mathcal{C}[H_p] = ?; (\mathcal{C}[H_p]) = (?; \mathcal{C})[H_p]$ for $p \in \{1, 2\}$, and $|?; \mathcal{C}| = \mathcal{C} \in \mathbb{C}$, the triple $((?; \mathcal{C}); H_1; H_2)$ is a \mathbb{C} -specimen of \triangleleft with a search token. Moreover the states $?; \mathcal{C}[H_1]$ and $?; \mathcal{C}[H_2]$ are trivially rooted. Therefore, $?; \mathcal{C}[H_1] \triangleleft^\mathbb{C} ?; \mathcal{C}[H_2]$, and by the assumption, $?; \mathcal{C}[H_1] \dot{\preceq}_Q ?; \mathcal{C}[H_2]$. \square

Proof of equivalence case. It suffices to show that, for any $H_1 \triangleleft H_2$ and any focus-free context $\mathcal{C}[\chi] \in \mathbb{C}$ such that $?; \mathcal{C}[H_1]$ and $?; \mathcal{C}[H_2]$ are states, we have refinements $?; \mathcal{C}[H_1] \dot{\preceq}_Q ?; \mathcal{C}[H_2]$ and $?; \mathcal{C}[H_2] \dot{\preceq}_Q ?; \mathcal{C}[H_1]$, i.e. equivalence $?; \mathcal{C}[H_1] \dot{\simeq}_Q ?; \mathcal{C}[H_2]$.

Because $?; \mathcal{C}[H_p] = ?; (\mathcal{C}[H_p]) = (?; \mathcal{C})[H_p]$ for $p \in \{1, 2\}$, and $|?; \mathcal{C}| = \mathcal{C} \in \mathbb{C}$, the triple $((?; \mathcal{C}); H_1; H_2)$ is a \mathbb{C} -specimen of \triangleleft with a search token. Moreover the states $?; \mathcal{C}[H_1]$ and $?; \mathcal{C}[H_2]$ are trivially rooted. Therefore, $?; \mathcal{C}[H_1] \triangleleft^\mathbb{C} ?; \mathcal{C}[H_2]$, and by the assumption, $?; \mathcal{C}[H_1] \dot{\simeq}_Q ?; \mathcal{C}[H_2]$. \square

Lemma 7.5. For any set $\mathbb{C} \subseteq \mathcal{H}_\omega(L, M_\mathbb{O} \cup \mathbb{M})$ of contexts that is closed under plugging, any pre-template \triangleleft on focus-free hypernets $\mathcal{H}_\omega(L, M_\mathbb{O} \setminus \{?, \checkmark, \dagger\})$, and any \mathbb{C} -specimen $(\dot{C}[\vec{\chi}]; \vec{H}^1; \vec{H}^2)$ of \triangleleft , the following holds.

1. The state $\dot{C}[\vec{H}^1]$ is final (resp. initial) if and only if the state $\dot{C}[\vec{H}^2]$ is final (resp. initial).
2. If \triangleleft is output-closed, and $\dot{C}[\vec{H}^1]$ and $\dot{C}[\vec{H}^2]$ are both rooted states, then the token of \dot{C} is not exiting.
3. If \triangleleft is output-closed, $\dot{C}[\vec{H}^1]$ and $\dot{C}[\vec{H}^2]$ are both rooted states, the token of \dot{C} is a value token or a non-entering search token, and a transition is possible from $\dot{C}[\vec{H}^1]$ or $\dot{C}[\vec{H}^2]$, then there exists a focussed context \dot{C}' with a remote token such that $|\dot{C}'| = |\dot{C}|$ and $\dot{C}[\vec{H}^p] \rightarrow \dot{C}'[\vec{H}^p]$ for each $p \in \{1, 2\}$.

Proof of point (1). Let (p, q) be an arbitrary element of a set $\{(1, 2), (2, 1)\}$. If $\dot{C}[\vec{H}^p]$ is final (resp. initial), the token source is an input in $\dot{C}[\vec{H}^p]$. Because input lists of $\dot{C}[\vec{H}^p]$, \dot{C} and $\dot{C}[\vec{H}^q]$ all coincide, the token source must be an input in \dot{C} , and in $\dot{C}[\vec{H}^q]$ too. This means $\dot{C}[\vec{H}^q]$ is also a final (resp. initial) state. \square

Proof of point (2). This is a consequence of the contraposition of Lem. A.12(3). \square

Proof of the point (3). The transition possible from $\dot{C}[\vec{H}^1]$ or $\dot{C}[\vec{H}^2]$ is necessarily a search transition. By case analysis on the token of \dot{C} , we can confirm that the search transition applies an interaction rule to the token and an edge from \dot{C} .

- When the token of \dot{C} is a value token, the transition can only change the token and its incoming operation edge. Because \triangleleft is output-closed, by the point (2), the token of \dot{C} is not exiting. This implies that the incoming operation edge of the token is from \dot{C} in both states $\dot{C}[\vec{H}^1]$ and $\dot{C}[\vec{H}^2]$.

- When the token of $\dot{\mathcal{C}}$ is a non-entering search token, the transition can only change the token and its outgoing edge. Because the token is not entering in $\dot{\mathcal{C}}$, the outgoing edge is from $\dot{\mathcal{C}}$ in both states $\dot{\mathcal{C}}[\vec{H}^1]$ and $\dot{\mathcal{C}}[\vec{H}^2]$.

Therefore, there exist a focus-free simple context $\mathcal{C}_0[\chi, \vec{\chi}]$ and an interaction rule $\dot{N}_0 \mapsto \dot{N}'_0$, such that $\dot{\mathcal{C}} = \mathcal{C}_0[\dot{N}_0, \vec{\chi}]$, and $\mathcal{C}_0[\dot{N}'_0, \vec{\chi}]$ is a focussed context.

Examining interaction rules confirms $|\dot{N}_0| = |\dot{N}'_0|$, and hence $|\dot{\mathcal{C}}| = |\mathcal{C}_0[\dot{N}_0, \vec{\chi}]| = |\mathcal{C}_0[\dot{N}'_0, \vec{\chi}]|$. By definition of search transitions, we have:

$$\dot{\mathcal{C}}[\vec{H}^p] = \mathcal{C}_0[\dot{N}_0, \vec{H}^p] \rightarrow \mathcal{C}_0[\dot{N}'_0, \vec{H}^p]$$

for each $p \in \{1, 2\}$.

The rest of the proof is to check that $\mathcal{C}_0[\dot{N}'_0, \vec{\chi}]$ has a remote token, namely that, if its token is a rewrite token, the token is not entering. This is done by inspecting interaction rules.

- When the interaction rule $\dot{N}_0 \mapsto \dot{N}'_0$ changes a value token to a rewrite token, this must be the interaction rule (5a), which means \dot{N}'_0 consists of the rewrite token and its outgoing operation edge. The operation edge remains to be a (unique) outgoing edge of the token in $\mathcal{C}_0[\dot{N}'_0, \vec{\chi}]$, and hence the token is not entering in $\mathcal{C}_0[\dot{N}'_0, \vec{\chi}]$.
- When the interaction rule $\dot{N}_0 \mapsto \dot{N}'_0$ changes a search token to a rewrite token, this must be the interaction rule (1a), (1b) or (5b), which means $\dot{N}'_0 = \langle \dot{N}_0 \rangle_{\sharp/?}$. Because the token is not entering in $\mathcal{C}_0[\dot{N}_0, \vec{\chi}] = \dot{\mathcal{C}}$, the token is also not entering in $\mathcal{C}_0[\dot{N}'_0, \vec{\chi}] = \langle \mathcal{C}_0[\dot{N}_0, \vec{\chi}] \rangle_{\sharp/?}$.

□

Proposition 7.6. *When the universal abstract machine $\mathcal{U}(\mathbb{O}, B_{\mathbb{O}})$ is deterministic and refocusing, it satisfies the following, for any set $\mathbb{C} \subseteq \mathcal{H}_{\omega}(L, M_{\mathbb{O}} \cup \mathbb{M})$ of contexts that is closed under plugging, any reasonable triple (Q, Q', Q'') , and any pre-template \triangleleft on focus-free hypernets $\mathcal{H}_{\omega}(L, M_{\mathbb{O}} \setminus \{?, \checkmark, \sharp\})$.*

1. *If \triangleleft is a (\mathbb{C}, Q, Q') -template and (\mathbb{C}, Q, Q', Q'') -robust relative to all rewrite transitions, then the \mathbb{C} -contextual lifting $\overleftarrow{\triangleleft}^{\mathbb{C}}$ is a (Q, Q', Q'') -simulation.*
2. *If \triangleleft is a (\mathbb{C}, Q^{-1}, Q') -template and the converse \triangleleft^{-1} is (\mathbb{C}, Q, Q', Q'') -robust relative to all rewrite transitions, then the \mathbb{C} -contextual lifting $\overleftarrow{\triangleleft^{-1}}^{\mathbb{C}}$ of the converse is a (Q, Q', Q'') -simulation.*

Proof prelude. Let $(\dot{\mathcal{C}}; \vec{H}^1; \vec{H}^2)$ be an arbitrary \mathbb{C} -specimen of \triangleleft , such that the token of $\dot{\mathcal{C}}$ is remote, and $\dot{G}_p := \dot{\mathcal{C}}[\vec{H}^p]$ is a rooted state for each $p \in \{1, 2\}$. By definition of contextual lifting, $\dot{G}_1 \overleftarrow{\triangleleft}^{\mathbb{C}} \dot{G}_2$, and equivalently, $\dot{G}_2 (\overleftarrow{\triangleleft}^{\mathbb{C}})^{-1} \dot{G}_1$. Note that $\overleftarrow{\triangleleft^{-1}}^{\mathbb{C}} = (\overleftarrow{\triangleleft}^{\mathbb{C}})^{-1}$.

Because \triangleleft is output-closed, by Lem. 7.5(2), the token is not exiting in $\dot{\mathcal{C}}$. This implies that, if the token has an incoming edge in \dot{G}_1 or \dot{G}_2 , the incoming edge must be from $\dot{\mathcal{C}}$.

Because the machine is deterministic and refocusing, rooted states and transitions comprise a deterministic abstract rewriting system, in which final states and stuck states are normal forms. By Lem. A.6, from any state, a sequence of transitions that result in a final state or a stuck state is unique, if any.

Because (Q, Q', Q'') is a reasonable triple, Q' and Q'' are reflexive. By Lem. A.22, this implies that $\dot{\prec}_{Q'}$ and $\dot{\prec}_{Q''}$ are reflexive, and hence $\overleftarrow{\triangleleft}^{\mathbb{C}} \subseteq \dot{\prec}_{Q'} \circ \overleftarrow{\triangleleft}^{\mathbb{C}} \circ \dot{\prec}_{Q''}$, and $(\overleftarrow{\triangleleft}^{\mathbb{C}})^{-1} \subseteq \dot{\prec}_{Q'} \circ (\overleftarrow{\triangleleft}^{\mathbb{C}})^{-1} \circ \dot{\prec}_{Q''}$. □

Proof of the point (1). Our goal is to check conditions (A) and (B) of Def. 7.1 for the states $\dot{G}_1 \overleftarrow{\triangleleft}^{\mathbb{C}} \dot{G}_2$.

If \dot{G}_1 is final, by Lem. 7.5(1), \dot{G}_2 is also final. The condition (A) of Def. 7.1 is fulfilled.

If there exists a state \dot{G}'_1 such that $\dot{G}_1 \rightarrow \dot{G}'_1$, we show that one of the conditions (I) and (II) of Def. 7.1 is fulfilled, by case analysis of the token in $\dot{\mathcal{C}}$.

- When the token is a value token, or a search token that is not entering, by Lem. 7.5(3), there exists a focussed context $\dot{\mathcal{C}}'$ with a remote token, such that $|\dot{\mathcal{C}}'| = |\dot{\mathcal{C}}|$ and $\dot{G}_p = \dot{\mathcal{C}}[\vec{H}^p] \rightarrow \dot{\mathcal{C}}'[\vec{H}^p]$ for each $p \in \{1, 2\}$. We have the following situation, namely the black part of the diagram below. Showing the magenta part confirms that the condition (II) of Def. 7.1 is fulfilled.

$$\begin{array}{ccc} \dot{G}_1 = \dot{\mathcal{C}}[\vec{H}^1] & \longrightarrow & \dot{\mathcal{C}}'[\vec{H}^1] = \dot{G}'_1 \\ \Downarrow^{\mathcal{C}} & \boxed{1 \ Q \ 1} & \Downarrow^{\mathcal{C}} \\ \dot{G}_2 = \dot{\mathcal{C}}[\vec{H}^2] & \longrightarrow & \dot{\mathcal{C}}'[\vec{H}^2] \end{array}$$

By the determinism, $\dot{\mathcal{C}}'[\vec{H}^1] = \dot{G}'_1$. Because (Q, Q', Q'') is a reasonable triple, Q is a preorder and $1 \ Q \ 1$. The context $\dot{\mathcal{C}}'$ satisfies $|\dot{\mathcal{C}}'| = |\dot{\mathcal{C}}| \in \mathbb{C}$, so $(\dot{\mathcal{C}}'; \vec{H}^1; \vec{H}^2)$ is a \mathbb{C} -specimen of \triangleleft . The context $\dot{\mathcal{C}}'$ has a remote token, and the states $\dot{\mathcal{C}}'[\vec{H}^1]$ and $\dot{\mathcal{C}}'[\vec{H}^2]$ are both rooted. Therefore, we have $\dot{\mathcal{C}}'[\vec{H}^1] \triangleleft^{\mathcal{C}} \dot{\mathcal{C}}'[\vec{H}^2]$.

- When the token is a search token that is entering in $\dot{\mathcal{C}}$, because \triangleleft is (\mathbb{C}, Q, Q') -input-safe, we have one of the following three situations corresponding to (I), (II) and (III) of Def. 6.9.
 - There exist two stuck states \dot{N}_1 and \dot{N}_2 such that $\dot{G}_p \rightarrow^* \dot{N}_p$ for each $p \in \{1, 2\}$. By the determinism of transitions, we have $\dot{G}_1 \rightarrow \dot{G}'_1 \rightarrow^* \dot{N}_1$, which means the condition (I) of Def. 7.1 is satisfied.
 - There exist a \mathbb{C} -specimen $(\dot{\mathcal{C}}'; \vec{H}^1; \vec{H}^2)$ of \triangleleft and two numbers $k_1, k_2 \in \mathbb{N}$, such that the token of $\dot{\mathcal{C}}'$ is not a rewrite token and not entering, $(1 + k_1) \ Q \ k_2$, $\dot{\mathcal{C}}[\vec{H}^1] \rightarrow^{1+k_1} \dot{\mathcal{C}}'[\vec{H}^1]$, and $\dot{\mathcal{C}}[\vec{H}^2] \rightarrow^{k_2} \dot{\mathcal{C}}'[\vec{H}^2]$. By the determinism of transitions, we have the following situation, namely the black part of the diagram below. Showing the magenta part confirms that the condition (II) of Def. 7.1 is fulfilled.

$$\begin{array}{ccc} \dot{G}_1 = \dot{\mathcal{C}}[\vec{H}^1] & \longrightarrow & \dot{G}'_1 \xrightarrow{k_1} \dot{\mathcal{C}}'[\vec{H}^1] \\ \Downarrow^{\mathcal{C}} & \boxed{(1+k_1) \ Q \ k_2} & \Downarrow^{\mathcal{C}} \\ \dot{G}_2 = \dot{\mathcal{C}}[\vec{H}^2] & \longrightarrow & \dot{\mathcal{C}}'[\vec{H}^2] \end{array}$$

The context $\dot{\mathcal{C}}'$ has a remote token, and states $\dot{\mathcal{C}}'[\vec{H}^1]$ and $\dot{\mathcal{C}}'[\vec{H}^2]$ are rooted. Therefore, $\dot{\mathcal{C}}'[\vec{H}^1] \triangleleft^{\mathcal{C}} \dot{\mathcal{C}}'[\vec{H}^2]$.

- There exist a quasi- \mathbb{C} -specimen (\dot{N}_1, \dot{N}_2) of \triangleleft up to $(\dot{\simeq}_{Q'}, \dot{\simeq}_{Q'})$, whose token is not a rewrite token, and two numbers $k_1, k_2 \in \mathbb{N}$, such that $(1 + k_1) \ Q \ (1 + k_2)$, $\dot{\mathcal{C}}[\vec{H}^1] \rightarrow^{1+k_1} \dot{N}_1$, and $\dot{\mathcal{C}}[\vec{H}^2] \rightarrow^{1+k_2} \dot{N}_2$. By the determinism of transitions, we have the following situation, namely the black part of the diagram below. Showing the magenta part confirms that the condition (II) of Def. 7.1 is fulfilled.

$$\begin{array}{ccc} \dot{G}_1 = \dot{\mathcal{C}}[\vec{H}^1] & \longrightarrow & \dot{G}'_1 \xrightarrow{k_1} \dot{N}_1 \\ \Downarrow^{\mathcal{C}} & \boxed{(1+k_1) \ Q \ (1+k_2)} & \Downarrow^{\mathcal{C}} \\ \dot{G}_2 = \dot{\mathcal{C}}[\vec{H}^2] & \longrightarrow & \dot{N}_2 \end{array} \quad \begin{array}{c} \dot{\simeq}_{Q'} \circ \triangleleft^{\mathcal{C}} \circ \dot{\simeq}_{Q''} \end{array}$$

Because (\dot{N}_1, \dot{N}_2) is a quasi- \mathbb{C} -specimen of \triangleleft up to $(\dot{\simeq}_{Q'}, \dot{\simeq}_{Q'})$, and states \dot{N}_1 and \dot{N}_2 are rooted, there exists a \mathbb{C} -specimen $(\dot{\mathcal{C}}'; \vec{H}^1; \vec{H}^2)$ of \triangleleft with a non-rewrite token, such that $\dot{\mathcal{C}}'[\vec{H}^1]$ and $\dot{\mathcal{C}}'[\vec{H}^2]$ are also rooted, $\dot{N}_1 \dot{\simeq}_{Q'} \dot{\mathcal{C}}'[\vec{H}^1]$, and $\dot{\mathcal{C}}'[\vec{H}^2] \dot{\simeq}_{Q'} \dot{N}_2$. Because (Q, Q', Q'') is a reasonable triple, $Q' \subseteq Q''$, and hence $\dot{\simeq}_{Q'} \subseteq \dot{\simeq}_{Q''}$. Therefore, we have:

$$\dot{N}_1 \dot{\simeq}_{Q'} \dot{\mathcal{C}}'[\vec{H}^1] \triangleleft^{\mathcal{C}} \dot{\mathcal{C}}'[\vec{H}^2] \dot{\simeq}_{Q''} \dot{N}_2.$$

- When the token is a rewrite token, $\dot{G}_1 \rightarrow \dot{G}'_1$ is a rewrite transition, and by definition of contextual lifting, the token is not entering in $\dot{\mathcal{C}}$. Because \triangleleft is (\mathbb{C}, Q, Q', Q'') -robust relative to all rewrite transitions, and \dot{G}_1 and \dot{G}_2 are rooted, we have one of the following two situations corresponding to (II) and (III) of Def. 6.12.

- There exists a stuck state \dot{N} such that $\dot{G}'_1 \rightarrow^* \dot{N}$. The condition (I) of Def. 7.1 is satisfied.
- There exist a quasi- \mathbb{C} -specimen (\dot{N}_1, \dot{N}_2) of \triangleleft up to $(\dot{\preceq}_{Q'}, \dot{\preceq}_{Q''})$, whose token is not a rewrite token, and two numbers $k_1, k_2 \in \mathbb{N}$, such that $(1 + k_1) Q k_2$, $\dot{G}'_1 \rightarrow^{k_1} \dot{N}_1$, and $\dot{G}_2 \rightarrow^{k_2} \dot{N}_2$. We have the following situation, namely the black part of the diagram below. Showing the magenta part confirms that the condition (II) of Def. 7.1 is fulfilled.

$$\begin{array}{ccc}
\dot{G}_1 = \dot{C}[\vec{H}^1] & \xrightarrow{\quad} & \dot{G}'_1 \xrightarrow{k_1} \dot{N}_1 \\
\downarrow \overleftarrow{\Delta}^C & \boxed{(1+k_1) Q k_2} & \downarrow \dot{\preceq}_{Q'} \circ \overleftarrow{\Delta}^C \circ \dot{\preceq}_{Q''} \\
\dot{G}_2 = \dot{C}[\vec{H}^2] & \xrightarrow{\quad} & \dot{N}_2
\end{array}$$

Because (\dot{N}_1, \dot{N}_2) is a quasi- \mathbb{C} -specimen of \triangleleft up to $(\dot{\preceq}_{Q'}, \dot{\preceq}_{Q''})$, and states \dot{N}_1 and \dot{N}_2 are rooted, there exists a \mathbb{C} -specimen $(\dot{C}'; \vec{H}^1; \vec{H}^2)$ of \triangleleft with a non-rewrite token, such that $\dot{C}'[\vec{H}^1]$ and $\dot{C}'[\vec{H}^2]$ are also rooted, $\dot{N}_1 \dot{\preceq}_{Q'} \dot{C}'[\vec{H}^1]$, and $\dot{C}'[\vec{H}^2] \dot{\preceq}_{Q''} \dot{N}_2$. This means $\dot{C}'[\vec{H}^1] \overleftarrow{\Delta}^C \dot{C}'[\vec{H}^2]$, and hence:

$$\dot{N}_1 \dot{\preceq}_{Q'} \dot{C}'[\vec{H}^1] \overleftarrow{\Delta}^C \dot{C}'[\vec{H}^2] \dot{\preceq}_{Q''} \dot{N}_2.$$

□

Proof of the point (2). It suffices to check the “reverse” of conditions (A) and (B) of Def. 7.1 for the states $\dot{G}_2 (\overleftarrow{\Delta}^C)^{-1} \dot{G}'_1$, namely the following conditions (A') and (B').

(A') If \dot{G}_2 is final, \dot{G}'_1 is also final.

(B') If there exists a state \dot{G}'_2 such that $\dot{G}_2 \rightarrow \dot{G}'_2$, one of the following (I') and (II') holds.

(I') There exists a stuck state \dot{G}''_2 such that $\dot{G}'_2 \rightarrow^* \dot{G}''_2$.

(II') There exist two states \dot{N}_2 and \dot{N}_1 , and numbers $k_2, k_1 \in \mathbb{N}$, such that $\dot{N}_2 (\dot{\preceq}_{Q'} \circ (\overleftarrow{\Delta}^C)^{-1} \circ \dot{\preceq}_{Q'}) \dot{N}_1$, $(1 + k_2) Q k_1$, $\dot{G}'_2 \rightarrow^{k_2} \dot{N}_2$, and $\dot{G}'_1 \rightarrow^{k_1} \dot{N}_1$.

The proof is mostly symmetric to the point (1). Note that there is a one-to-one correspondence between \mathbb{C} -specimens of \triangleleft and \mathbb{C} -specimens of \triangleleft^{-1} ; any \mathbb{C} -specimen $(\dot{C}_0; \vec{H}^{\vec{0}1}; \vec{H}^{\vec{0}2})$ of \triangleleft gives a \mathbb{C} -specimen $(\dot{C}_0; \vec{H}^{\vec{0}2}; \vec{H}^{\vec{0}1})$ of \triangleleft^{-1} . Because \triangleleft is output-closed, its converse \triangleleft^{-1} is also output-closed.

If \dot{G}_2 is final, by Lem. 7.5(1), \dot{G}'_1 is also final. The condition (A') is fulfilled.

If there exists a state \dot{G}'_2 such that $\dot{G}_2 \rightarrow \dot{G}'_2$, we show that one of the conditions (I') and (II') above is fulfilled, by case analysis of the token in \dot{C} .

- When the token is a value token, or a search token that is not entering, by Lem. 7.5(3), there exists a focussed context \dot{C}' with a remote token, such that $|\dot{C}'| = |\dot{C}|$ and $\dot{G}_p = \dot{C}'[\vec{H}^p] \rightarrow \dot{C}'[\vec{H}^p]$ for each $p \in \{1, 2\}$. We have the following situation, namely the black part of the diagram below. Showing the magenta part confirms that the condition (II') is fulfilled.

$$\begin{array}{ccc}
\dot{G}_2 = \dot{C}[\vec{H}^2] & \xrightarrow{\quad} & \dot{C}'[\vec{H}^2] = \dot{G}'_2 \\
(\overleftarrow{\Delta}^C)^{-1} \downarrow & \boxed{1 Q 1} & \downarrow (\overleftarrow{\Delta}^C)^{-1} \\
\dot{G}'_1 = \dot{C}[\vec{H}^1] & \xrightarrow{\quad} & \dot{C}'[\vec{H}^1]
\end{array}$$

By the determinism, $\dot{C}'[\vec{H}^2] = \dot{G}'_2$. Because (Q, Q', Q'') is a reasonable triple, Q is a preorder and $1 Q 1$. The context \dot{C}' satisfies $|\dot{C}'| = |\dot{C}| \in \mathbb{C}$, so $(\dot{C}'; \vec{H}^2; \vec{H}^1)$ is a \mathbb{C} -specimen of \triangleleft^{-1} . The context \dot{C}' has a remote token, and the states $\dot{C}'[\vec{H}^1]$ and $\dot{C}'[\vec{H}^2]$ are both rooted. Therefore, we have $\dot{C}'[\vec{H}^2] (\overleftarrow{\Delta}^C)^{-1} \dot{C}'[\vec{H}^1]$.

- When the token is a search token that is entering in \dot{C} , because \triangleleft is (\mathbb{C}, Q^{-1}, Q') -input-safe, we have one of the following three situations corresponding to (I), (II) and (III) of Def. 6.9.

- There exist two stuck states \dot{N}_1 and \dot{N}_2 such that $\dot{G}_p \rightarrow^* \dot{N}_p$ for each $p \in \{1, 2\}$. By the determinism of transitions, we have $\dot{G}_2 \rightarrow \dot{G}'_2 \rightarrow^* \dot{N}_2$, which means the condition (I') is satisfied.

- There exist a \mathbb{C} -specimen $(\dot{\mathcal{C}}'; \vec{H}^1; \vec{H}^2)$ of \triangleleft and two numbers $k_1, k_2 \in \mathbb{N}$, such that the token of $\dot{\mathcal{C}}'$ is not a rewrite token and not entering, $(1+k_1) Q^{-1} k_2, \dot{\mathcal{C}}[\vec{H}^1] \rightarrow^{1+k_1} \dot{\mathcal{C}}'[\vec{H}^1]$, and $\dot{\mathcal{C}}[\vec{H}^2] \rightarrow^{k_2} \dot{\mathcal{C}}'[\vec{H}^2]$. We have the following situation, namely the black part of the diagram below.

$$\begin{array}{ccc} \dot{G}_2 = \dot{\mathcal{C}}[\vec{H}^2] & \xrightarrow{k_2} & \dot{\mathcal{C}}'[\vec{H}^2] \\ (\overleftarrow{\mathcal{C}})^{-1} \vdots & \boxed{k_2 Q (1+k_1)} & \vdots (\overleftarrow{\mathcal{C}})^{-1} \\ \dot{G}_1 = \dot{\mathcal{C}}[\vec{H}^1] & \xrightarrow{1+k_1} & \dot{\mathcal{C}}'[\vec{H}^1] \end{array}$$

The magenta part holds, because the token of $\dot{\mathcal{C}}'$ is not a rewrite token and not entering, and because states $\dot{\mathcal{C}}'[\vec{H}^1]$ and $\dot{\mathcal{C}}'[\vec{H}^2]$ are rooted. We check the condition (II') by case analysis on the number k_2 .

- * When $k_2 > 0$, by the determinism of transitions, we have the following diagram, which means the condition (II') is fulfilled.

$$\begin{array}{ccc} \dot{G}_2 = \dot{\mathcal{C}}[\vec{H}^2] & \xrightarrow{k_2-1} & \dot{\mathcal{C}}'[\vec{H}^2] \\ (\overleftarrow{\mathcal{C}})^{-1} \vdots & \boxed{k_2 Q (1+k_1)} & \vdots (\overleftarrow{\mathcal{C}})^{-1} \\ \dot{G}_1 = \dot{\mathcal{C}}[\vec{H}^1] & \xrightarrow{1+k_1} & \dot{\mathcal{C}}'[\vec{H}^1] \end{array}$$

- * When $k_2 = 0$, $\dot{G}_2 = \dot{\mathcal{C}}[\vec{H}^2] = \dot{\mathcal{C}}'[\vec{H}^2]$, and we have the following situation, namely the black part of the diagram below.

$$\begin{array}{ccc} \dot{G}_2 = \dot{\mathcal{C}}[\vec{H}^2] & \xrightarrow{0} & \dot{G}_2 = \dot{\mathcal{C}}'[\vec{H}^2] \xrightarrow{\text{magenta}} \dot{G}_2' = \dot{\mathcal{C}}''[\vec{H}^1] \\ (\overleftarrow{\mathcal{C}})^{-1} \vdots & \boxed{0 Q (1+k_1)} & \vdots (\overleftarrow{\mathcal{C}})^{-1} \boxed{1 Q 1} \vdots (\overleftarrow{\mathcal{C}})^{-1} \\ \dot{G}_1 = \dot{\mathcal{C}}[\vec{H}^1] & \xrightarrow{1+k_1} & \dot{\mathcal{C}}'[\vec{H}^1] \xrightarrow{\text{magenta}} \dot{\mathcal{C}}''[\vec{H}^1] \end{array}$$

Because $\dot{G}_2 \rightarrow \dot{G}_2'$, and the token of $\dot{\mathcal{C}}'$ is a value token, or a non-entering search token, by Lem. 7.5(3), there exists a focussed context $\dot{\mathcal{C}}''$ with a remote token, such that $|\dot{\mathcal{C}}''| = |\dot{\mathcal{C}}'|$ and $\dot{\mathcal{C}}'[\vec{H}^p] \rightarrow \dot{\mathcal{C}}''[\vec{H}^p]$ for each $p \in \{1, 2\}$. By the determinism of transitions, $\dot{G}_2' = \dot{\mathcal{C}}''[\vec{H}^1]$. Because (Q, Q', Q'') is a reasonable triple, Q is a preorder and $1 Q 1$. The context $\dot{\mathcal{C}}''$ satisfies $|\dot{\mathcal{C}}''| = |\dot{\mathcal{C}}'| \in \mathbb{C}$, so $(\dot{\mathcal{C}}''; \vec{H}^2; \vec{H}^1)$ is a \mathbb{C} -specimen of \triangleleft^{-1} . The context $\dot{\mathcal{C}}''$ has a remote token, and the states $\dot{\mathcal{C}}''[\vec{H}^1]$ and $\dot{\mathcal{C}}''[\vec{H}^2]$ are both rooted. Therefore, we have $\dot{\mathcal{C}}''[\vec{H}^2] (\overleftarrow{\mathcal{C}})^{-1} \dot{\mathcal{C}}''[\vec{H}^1]$. Finally, because (Q, Q', Q'') is a reasonable triple, Q is closed under addition, and hence $1 Q (2+k_1)$. The condition (II') is fulfilled.

- There exist a quasi- \mathbb{C} -specimen (\dot{N}_1, \dot{N}_2) of \triangleleft up to $(\dot{\simeq}_{Q'}, \dot{\simeq}_{Q'})$, whose token is not a rewrite token, and two numbers $k_1, k_2 \in \mathbb{N}$, such that $(1+k_1) Q^{-1} (1+k_2), \dot{\mathcal{C}}[\vec{H}^1] \rightarrow^{1+k_1} \dot{N}_1$, and $\dot{\mathcal{C}}[\vec{H}^2] \rightarrow^{1+k_2} \dot{N}_2$. By the determinism of transitions, we have the following situation, namely the black part of the diagram below. Showing the magenta part confirms that the condition (II') is fulfilled.

$$\begin{array}{ccc} \dot{G}_2 = \dot{\mathcal{C}}[\vec{H}^2] & \xrightarrow{\quad} & \dot{G}_2' \xrightarrow{k_2} \dot{N}_2 \\ (\overleftarrow{\mathcal{C}})^{-1} \vdots & \boxed{(1+k_2) Q (1+k_1)} & \vdots \dot{\simeq}_{Q'} \circ (\overleftarrow{\mathcal{C}})^{-1} \dot{\simeq}_{Q''} \\ \dot{G}_1 = \dot{\mathcal{C}}[\vec{H}^1] & \xrightarrow{1+k_1} & \dot{N}_1 \end{array}$$

Because (\dot{N}_1, \dot{N}_2) is a quasi- \mathbb{C} -specimen of \triangleleft up to $(\dot{\simeq}_{Q'}, \dot{\simeq}_{Q'})$, and states \dot{N}_1 and \dot{N}_2 are rooted, there exists a \mathbb{C} -specimen $(\dot{\mathcal{C}}'; \vec{H}^1; \vec{H}^2)$ of \triangleleft with a non-rewrite token, such that $\dot{\mathcal{C}}'[\vec{H}^1]$ and $\dot{\mathcal{C}}'[\vec{H}^2]$ are also rooted, $\dot{N}_1 \dot{\simeq}_{Q'} \dot{\mathcal{C}}'[\vec{H}^1]$, and $\dot{\mathcal{C}}'[\vec{H}^2] \dot{\simeq}_{Q'} \dot{N}_2$. Because (Q, Q', Q'') is a reasonable triple, $Q' \subseteq Q''$, and hence $\dot{\simeq}_{Q'} \subseteq \dot{\simeq}_{Q''}$. Therefore, we have:

$$\dot{N}_2 \dot{\simeq}_{Q'} \dot{\mathcal{C}}'[\vec{H}^2] (\overleftarrow{\mathcal{C}})^{-1} \dot{\mathcal{C}}'[\vec{H}^1] \dot{\simeq}_{Q''} \dot{N}_1.$$

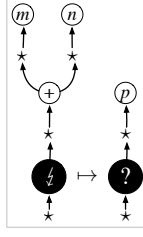


Figure 8: Arithmetic rewrite rule (selected), where $m, n, p \in \mathbb{N}$ and $p = m + n$

- When the token is a rewrite token, $\dot{G}_2 \rightarrow \dot{G}'_2$ is a rewrite transition, and by definition of contextual lifting, the token is not entering in $\dot{\mathcal{C}}$. Because \triangleleft^{-1} is (\mathbb{C}, Q, Q', Q'') -robust relative to all rewrite transitions, and \dot{G}_1 and \dot{G}_2 are rooted, we have one of the following two situations corresponding to (II) and (III) of Def. 6.12.
 - There exists a stuck state \dot{N} such that $\dot{G}'_2 \rightarrow^* \dot{N}$. The condition (I') is satisfied.
 - There exist a quasi- \mathbb{C} -specimen (\dot{N}_2, \dot{N}_1) of \triangleleft^{-1} up to $(\dot{\prec}_{Q'}, \dot{\prec}_{Q''})$, whose token is not a rewrite token, and two numbers $k_2, k_1 \in \mathbb{N}$, such that $(1 + k_2) Q k_1$, $\dot{G}'_2 \rightarrow^{k_2} \dot{N}_2$, and $\dot{G}_1 \rightarrow^{k_1} \dot{N}_1$. We have the following situation, namely the black part of the diagram below. Showing the magenta part confirms that the condition (II') is fulfilled.

$$\begin{array}{ccc}
\dot{G}_2 = \dot{\mathcal{C}}[\vec{H}^2] & \xrightarrow{\quad} & \dot{G}'_2 \xrightarrow{k_2} \dot{N}_2 \\
(\overleftarrow{\mathcal{C}})^{-1} \vdots & \boxed{(1+k_2) Q k_1} & \vdots \dot{\prec}_{Q'} \circ (\overleftarrow{\mathcal{C}})^{-1} \circ \dot{\prec}_{Q''} \\
\dot{G}_1 = \dot{\mathcal{C}}[\vec{H}^1] & \xrightarrow{\quad} & \dot{N}_1
\end{array}$$

Because (\dot{N}_2, \dot{N}_1) is a quasi- \mathbb{C} -specimen of \triangleleft^{-1} up to $(\dot{\prec}_{Q'}, \dot{\prec}_{Q''})$, and states \dot{N}_2 and \dot{N}_1 are rooted, there exists a \mathbb{C} -specimen $(\dot{\mathcal{C}}'; \vec{H}^2; \vec{H}^1)$ of \triangleleft^{-1} with a non-rewrite token, such that $\dot{\mathcal{C}}'[\vec{H}^2]$ and $\dot{\mathcal{C}}'[\vec{H}^1]$ are also rooted, $\dot{N}_2 \dot{\prec}_{Q'} \dot{\mathcal{C}}'[\vec{H}^2]$, and $\dot{\mathcal{C}}'[\vec{H}^1] \dot{\prec}_{Q''} \dot{N}_1$. This means $\dot{\mathcal{C}}'[\vec{H}^2] \overleftarrow{\triangleleft^{-1}}^{\mathbb{C}} \dot{\mathcal{C}}'[\vec{H}^1]$, and hence:

$$\dot{N}_2 \dot{\prec}_{Q'} \dot{\mathcal{C}}'[\vec{H}^2] (\overleftarrow{\mathcal{C}})^{-1} \dot{\mathcal{C}}'[\vec{H}^1] \dot{\prec}_{Q''} \dot{N}_1.$$

□

8 Applications of the characterisation theorem

This section shows applications of Thm. 6.14, with respect to an instantiation $\mathcal{U}(\mathbb{O}^{\text{ex}}, B_{\mathbb{O}^{\text{ex}}})$ of the universal abstract machine. We start by defining the specific operation set \mathbb{O}^{ex} and its behaviour $B_{\mathbb{O}^{\text{ex}}}$ in Sec. 8.1, which are informally introduced in Sec. 4.2, and proceed to illustrate proofs of the observational equivalences listed in Sec. 4. Necessary templates and their robustness are discussed in Sec. 8.2. Sec. 8.3 formally defines the notion of observational equivalence on terms, and describes how the templates can be combined to prove the observational equivalences. Finally, Sec. 8.4 and Sec. 8.5 give further details of the reasoning about templates and their robustness.

8.1 Properties of compute transitions

The operation set $\mathbb{O}^{\text{ex}} = \mathbb{O}_{\mathcal{J}}^{\text{ex}} \uplus \mathbb{O}_{\mathcal{I}}^{\text{ex}}$ we use here is given by passive operations $\mathbb{O}_{\mathcal{J}}^{\text{ex}} = \mathbb{N} \cup \{\lambda, \text{tt}, \text{ff}\}$ and active operations $\mathbb{O}_{\mathcal{I}}^{\text{ex}} = \{\text{@}, \text{ref}, =, :=, !, +, -, -_1\}$. The last three active operations '+', '-', and '-₁' are addition, subtraction and negation of natural numbers, respectively, and the other active operations are as discussed in Sec. 4.2.

The behaviour $B_{\mathbb{O}^{\text{ex}}}$, namely compute transitions for the active operations $\mathbb{O}_{\mathcal{I}}^{\text{ex}}$, are all defined locally via rewrite rules; for function application in Fig. 4, reference manipulation in Fig. 5 and

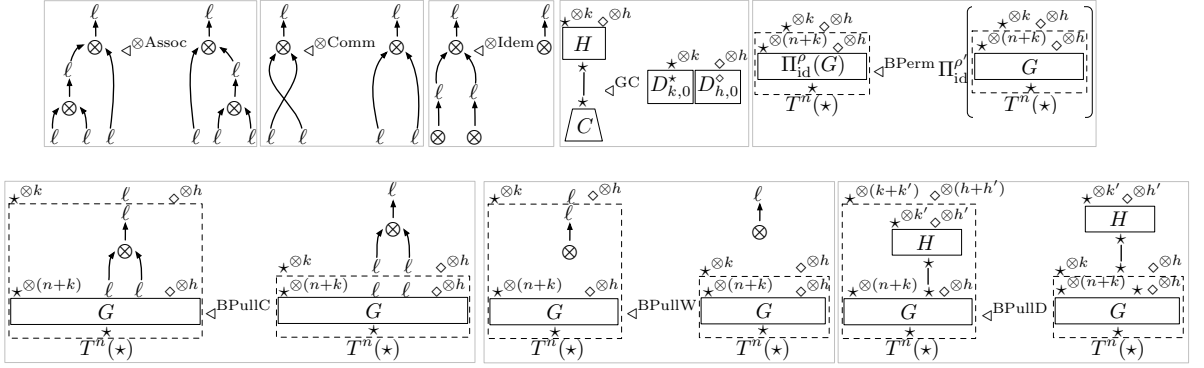


Figure 9: Structural pre-templates ($C : \epsilon \Rightarrow \star$ is a contraction tree, H is a copyable hypernet, G is a hypernet, (ρ, ρ') is a box-permutation pair)

Fig. 6, and arithmetic in Fig. 8 (showing just addition, but other operators can be similarly added). In these rules, the hypernet G_S is additionally required to be stable.

Determinism and refocusing of the particular machine $\mathcal{U}(\mathbb{O}^{\text{ex}}, B_{\mathbb{O}^{\text{ex}}})$ boils down to determinism and preservation of the rooted property of compute transitions.

Compute transitions of operations $\{\text{@}, \text{ref}, +, -, -_1\}$ are deterministic, because at most one rewrite rule can be applied to each state. In particular, the stable hypernet G_S in the figures is uniquely determined (by Lem. A.15(3)).

As discussed in Sec. 5, copy transitions are all deterministic, because different contraction rules applied to a single state result in the same state. Compute transitions of name-accessing operations $\{=, :=, !\}$ are deterministic for the same reason.

Compute transitions of all the operations \mathbb{O}_i^{ex} are stationary, and hence they preserve the rooted property. The stationary property can be checked using local rewrite rules. Namely, in each rewrite rule $\dot{H} \mapsto \dot{H}'$ of the operations, only one input of $|\dot{H}|$ has type \star , and $\dot{H} = \zeta; |\dot{H}|$ and $\dot{H}' = ?; |\dot{H}'|$. Moreover, any output of $|\dot{H}|$ with type \star is a target of an atom edge or a box edge (by definition of stable hypernets), which implies $|\dot{H}|$ is one-way.

Because any initial state is rooted, given that all transitions preserve the rooted property, we can safely assume that any state that arises in an execution is rooted. This means that the additional specification on stable hypernets in local rewrite rules is in fact guaranteed to be satisfied in any execution (by Lem. A.10, Lem. A.20 and Lem. A.18).

8.2 Example templates

Now we enumerate pre-templates that we will use to prove the equivalences listed in Sec. 4. We use the following auxiliary definition to specify one of these pre-templates (namely $\triangleleft^{\text{BPerm}}$ in Fig. 9).

Definition 8.1 (Box-permutation pair). For any $n, k, h \in \mathbb{N}$, let ρ and ρ' be bijections on sets $\{1, \dots, n + k + h\}$ and $\{1, \dots, k + h\}$, respectively. These bijections form a *box-permutation* pair (ρ, ρ') if, for each $i \in \{1, \dots, n + k + h\}$, the following holds:

- (A) $\rho(i) = i$ if $1 \leq i \leq n$,
- (B) $\rho(i) = \rho'(i - n)$ if $n < i \leq n + k + h$,
- (C) $1 \leq \rho'(i - n) \leq k$ if $n < i \leq n + k$,
- (D) $k < \rho'(i - n) \leq k + h$ if $n + k < i \leq n + k + h$.

The pre-templates we use are classified into three: *structural* pre-templates, *operational* pre-templates, and *name-exhaustive* pre-templates. While the structural laws in Sec. 4.1 can be proved using only structural pre-templates, the beta laws and stateful laws in Sec. 4.2 require operational and name-exhaustive pre-templates as well as structural pre-templates.

Fig. 9 shows all the structural pre-templates but the one derived from contraction rules: namely, $|\dot{G}_1| \triangleleft^\otimes |\dot{G}_2|$ whenever $\dot{G}_1 \mapsto \dot{G}_2$ is a contraction rule. The structural pre-templates primarily concern contraction edges, weakening edges and box edges. Contextual equivalences implied by $\triangleleft^{\otimes \text{Assoc}}$, $\triangleleft^{\otimes \text{Comm}}$ and $\triangleleft^{\otimes \text{Idem}}$ enable the so-called idempotent completion (aka. Karoubi envelope or Cauchy completion) on contractions and weakenings. This means that contraction trees with the same type can be identified, so long as they contain at least one weakening edge.

We use two operational pre-templates, which are directly derived from some local rewrite rules of active operations. Namely, $|\dot{G}_1| \triangleleft^{\bar{\otimes}} |\dot{G}_2|$ if $\dot{G}_1 \mapsto \dot{G}_2$ is a beta rewrite rule (Fig. 4); and $|\dot{G}_1| \triangleleft^{\text{ref}} |\dot{G}_2|$ if $\dot{G}_1 \mapsto \dot{G}_2$ is a reference-creation rewrite rule (Fig. 5). Note that we keep the additional specification that G_S in these figures are stable hypernets.

Fig. 10 shows the last class of pre-templates, namely four name-exhaustive pre-templates. They are specific to the four stateful laws in Conj. 4.4, and they analyse possible usages of a single name of interest.

Output-closure of all the pre-templates can be easily checked, typically by spotting that an input or an output, of type \star , is a source or a target of a contraction, atom or box edge.

Table. 1 outlines the way we will use Thm. 6.14 on all the pre-templates. For example, \triangleleft^\otimes is a $(\mathbb{C}_{\text{Oex}}, \geq_{\mathbb{N}}, =_{\mathbb{N}})$ -template, as shown in the “template” column, and both itself and its converse are $(\mathbb{C}_{\text{Oex}}, =_{\mathbb{N}}, =_{\mathbb{N}}, =_{\mathbb{N}})$ -robust relative to all rewrite transitions, as shown in the “robustness” columns. Thanks to the monotonicity (Remark 6.15), we can use Thm. 6.14(1) with a reasonable triple $(\geq_{\mathbb{N}}, =_{\mathbb{N}}, =_{\mathbb{N}})$, and Thm. 6.14(2) with a reasonable triple $(\leq_{\mathbb{N}}, =_{\mathbb{N}}, =_{\mathbb{N}})$. Consequently, $H_1 \triangleleft^\otimes H_2$ implies $H_1 \preceq_{\geq_{\mathbb{N}}}^{\mathbb{C}_{\text{Oex}}} H_2$ and $H_2 \preceq_{\leq_{\mathbb{N}}}^{\mathbb{C}_{\text{Oex}}} H_1$, which is shown in the “implication of $H_1 \triangleleft H_2$ ” column.

Pre-templates that relate hypernets with no input of type \star are trivially a (\mathbb{C}, Q, Q') -template for any \mathbb{C} , Q and Q' . The table uses ‘ $\square, \square, \square$ ’ to represent this situation.

For many pre-templates, a reasonable triple can be found by selecting “bigger” parameters from those of input-safety and robustness, thanks to the monotonicity. However, pre-templates $\triangleleft^{\text{BPullD}}$, $\triangleleft^{\text{NE3}}$ and $\triangleleft^{\text{NE4}}$ require non-trivial use of the monotonicity. For each of these pre-templates, an upper row shows a parameter $(\mathbb{C}, Q_1, Q'_1, Q''_1)$ that makes it (or its converse) robust, and a lower row shows a parameter $(\mathbb{C}, Q_2, Q'_2, Q''_2)$ to which Thm. 6.14 can be applied.

In the table, cyan symbols indicate where a proof of input-safety or robustness relies on contextual refinement. The “dependency” column indicates which pre-templates can be used to prove the necessary contextual refinement, given that these pre-templates imply contextual refinement as shown elsewhere in the table. This reliance specifically happens in finding a quasi-specimen, using contextual refinements/equivalences via Lem. 6.19. In the case of \triangleleft^\otimes , its input-safety and robustness are proved under the assumption that $\triangleleft^{\otimes \text{Assoc}}$ and $\triangleleft^{\otimes \text{Comm}}$ imply contextual equivalence $\simeq_{=_{\mathbb{N}}}^{\mathbb{C}_{\text{Oex}}}$.

The restriction to binding-free contexts plays a crucial role only in robustness regarding the operational pre-templates $\triangleleft^{\bar{\otimes}}$ and $\triangleleft^{\text{ref}}$. In fact, these pre-templates are input-safe with respect to both \mathbb{C}_{Oex} and $\mathbb{C}_{\text{Oex-bf}}$. This gap reflects duplication behaviour on atom edges, which is only encountered in a proof of robustness. A shallow atom edge is never duplicated, whereas a deep one can be duplicated as a part of a box (which represents a thunk).

Finally, the pre-template $\triangleleft^{\text{NE1}}$ is the only example whose converse is not robust. This is because the equality operation ‘ $=$ ’ is only defined on names, whereas it is possible in SPARTAN to give values, other than names, as arguments of the equality operation.

The key part of proving input-safety or robustness of a pre-template is to analyse how a rewrite transition involves edges (at any depth) of a state that are contributed by the pre-template. Given that all rewrite transitions (including copy transitions) are specified by local rewrite rules, the analysis boils down to check possible overlaps between a local rule and the pre-template. A typical situation is where a local rule simply preserves or duplicates edges contributed by a pre-template, without breaking them. Lem. 8.3 in Sec. 8.4 identifies two such situations: when the overlaps are all about deep edges, and when the pre-template relates contraction trees only.

8.3 Observational equivalences on terms

The notion of observational refinement on terms, informally introduced in Sec. 4, can now be defined using the contextual refinement on hypernets as follows. Let (X, Y) be a pair (all, O) or

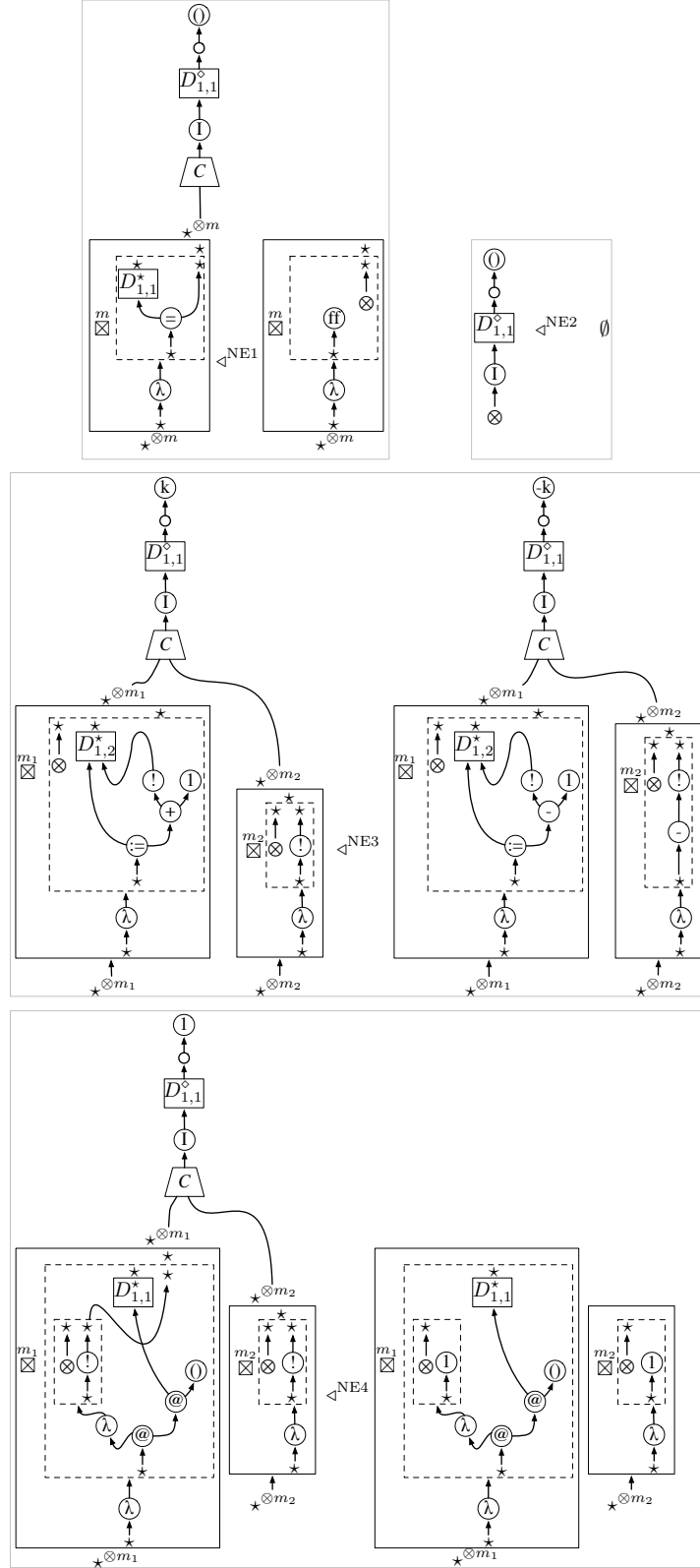


Figure 10: Name-exhaustive pre-templates (C is a contraction tree)

	template (input-safety)	robustness		dependency	implication of $H_1 \triangleleft H_2$
		of \triangleleft	of \triangleleft^{-1}		
$\triangleleft^{\otimes \text{Assoc}}$	$\mathbb{C}_{0\text{ex}}, =, =$	$\mathbb{C}_{0\text{ex}}, =, =, =$	$\mathbb{C}_{0\text{ex}}, =, =, =$	—	$H_1 \simeq_{=N}^{\mathbb{C}_{0\text{ex}}} H_2$
$\triangleleft^{\otimes \text{Comm}}$	$\mathbb{C}_{0\text{ex}}, =, =$	$\mathbb{C}_{0\text{ex}}, =, =, =$	$\mathbb{C}_{0\text{ex}}, =, =, =$	—	$H_1 \simeq_{=N}^{\mathbb{C}_{0\text{ex}}} H_2$
$\triangleleft^{\otimes \text{Idem}}$	$\square, \square, \square$	$\mathbb{C}_{0\text{ex}}, =, =, =$	$\mathbb{C}_{0\text{ex}}, =, =, =$	—	$H_1 \simeq_{=N}^{\mathbb{C}_{0\text{ex}}} H_2$
\triangleleft^{\otimes}	$\mathbb{C}_{0\text{ex}}, \geq, =$	$\mathbb{C}_{0\text{ex}}, =, =, =$	$\mathbb{C}_{0\text{ex}}, =, =, =$	$\triangleleft^{\otimes \text{Assoc}}$ $\triangleleft^{\otimes \text{Comm}}$	$H_1 \succeq_{\geq N}^{\mathbb{C}_{0\text{ex}}} H_2,$ $H_2 \succeq_{\leq N}^{\mathbb{C}_{0\text{ex}}} H_1$
$\triangleleft^{\text{GC}}$	$\square, \square, \square$	$\mathbb{C}_{0\text{ex}}, =, =, =$	$\mathbb{C}_{0\text{ex}}, =, =, =$	—	$H_1 \simeq_{=N}^{\mathbb{C}_{0\text{ex}}} H_2$
$\triangleleft^{\text{BPerm}}$	$\square, \square, \square$	$\mathbb{C}_{0\text{ex}}, =, =, =$	$\mathbb{C}_{0\text{ex}}, =, =, =$	—	$H_1 \simeq_{=N}^{\mathbb{C}_{0\text{ex}}} H_2$
$\triangleleft^{\text{BPullC}}$	$\square, \square, \square$	$\mathbb{C}_{0\text{ex}}, =, =, =$	$\mathbb{C}_{0\text{ex}}, =, =, =$	$\triangleleft^{\otimes \text{Assoc}}$ $\triangleleft^{\otimes \text{Comm}}$ $\triangleleft^{\otimes \text{Idem}}$	$H_1 \simeq_{=N}^{\mathbb{C}_{0\text{ex}}} H_2$
$\triangleleft^{\text{BPullW}}$	$\square, \square, \square$	$\mathbb{C}_{0\text{ex}}, =, =, =$	$\mathbb{C}_{0\text{ex}}, =, =, =$	$\triangleleft^{\otimes \text{Idem}}$	$H_1 \simeq_{=N}^{\mathbb{C}_{0\text{ex}}} H_2$
$\triangleleft^{\text{BPullD}}$	$\square, \square, \square$	$\mathbb{C}_{0\text{ex}}, =, =, \leq$	$\mathbb{C}_{0\text{ex}}, =, \geq, =$	$\triangleleft^{\otimes \text{Assoc}}$ $\triangleleft^{\otimes \text{Comm}}$	$H_1 \succeq_{\leq N}^{\mathbb{C}_{0\text{ex}}} H_2,$
		$\mathbb{C}_{0\text{ex}}, \leq, =, \leq$	$\mathbb{C}_{0\text{ex}}, \geq, \geq, \geq$	$\triangleleft^{\otimes \text{Idem}}$ \triangleleft^{\otimes} $\triangleleft^{\text{GC}}$	$H_2 \succeq_{\geq N}^{\mathbb{C}_{0\text{ex}}} H_1$
$\triangleleft^{\rightarrow \otimes}$	$\mathbb{C}_{0\text{ex}}, \geq, =$	$\mathbb{C}_{0\text{ex-bf}}, =, =, =$	$\mathbb{C}_{0\text{ex-bf}}, =, =, =$	—	$H_1 \succeq_{\geq N}^{\mathbb{C}_{0\text{ex-bf}}} H_2,$
	$\mathbb{C}_{0\text{ex-bf}}, \geq, =$				$H_2 \succeq_{\leq N}^{\mathbb{C}_{0\text{ex-bf}}} H_1$
$\triangleleft^{\text{ref}}$	$\mathbb{C}_{0\text{ex}}, \geq, =$	$\mathbb{C}_{0\text{ex-bf}}, =, =, =$	$\mathbb{C}_{0\text{ex-bf}}, =, =, =$	—	$H_1 \succeq_{\geq N}^{\mathbb{C}_{0\text{ex-bf}}} H_2,$
	$\mathbb{C}_{0\text{ex-bf}}, \geq, =$				$H_2 \succeq_{\leq N}^{\mathbb{C}_{0\text{ex-bf}}} H_1$
$\triangleleft^{\text{NE1}}$	$\mathbb{C}_{0\text{ex}}, =, =$	$\mathbb{C}_{0\text{ex}}, \geq, =, =$	—	$\triangleleft^{\otimes \text{Assoc}}$ $\triangleleft^{\otimes \text{Idem}}$ $\triangleleft^{\text{GC}}$	$H_1 \succeq_{\geq N}^{\mathbb{C}_{0\text{ex}}} H_2$
$\triangleleft^{\text{NE2}}$	$\square, \square, \square$	$\mathbb{C}_{0\text{ex}}, =, =, =$	$\mathbb{C}_{0\text{ex}}, =, =, =$	—	$H_1 \simeq_{=N}^{\mathbb{C}_{0\text{ex}}} H_2$
$\triangleleft^{\text{NE3}}$	$\mathbb{C}_{0\text{ex}}, =, =$	$\mathbb{C}_{0\text{ex}}, \leq, \geq, \leq$	$\mathbb{C}_{0\text{ex}}, \geq, \leq, \leq$	$\triangleleft^{\otimes \text{Assoc}}$ $\triangleleft^{\otimes \text{Idem}}$	$H_1 \simeq_{N \times N}^{\mathbb{C}_{0\text{ex}}} H_2$
		$\mathbb{C}_{0\text{ex}}, N \times N, \geq, N \times N$	$\mathbb{C}_{0\text{ex}}, N \times N, \geq, N \times N$	\triangleleft^{\otimes} $\triangleleft^{\text{GC}}$	
$\triangleleft^{\text{NE4}}$	$\mathbb{C}_{0\text{ex}}, =, =$	$\mathbb{C}_{0\text{ex}}, \geq, \geq, =$	$\mathbb{C}_{0\text{ex}}, \leq, =, \leq$	$\triangleleft^{\otimes \text{Assoc}}$ $\triangleleft^{\otimes \text{Idem}}$	$H_1 \succeq_{\geq N}^{\mathbb{C}_{0\text{ex}}} H_2$
		$\mathbb{C}_{0\text{ex}}, \geq, \geq, \geq$		\triangleleft^{\otimes} $\triangleleft^{\text{GC}}$	$H_2 \succeq_{\geq N}^{\mathbb{C}_{0\text{ex}}} H_1$

Table 1: Templates, with their robustness and implied contextual refinements/equivalences (\square denotes anything)

	\otimes Assoc	\otimes Comm	\otimes Idem	\otimes	GC	BPerm	BPullC	BPullW	BPullD	$\vec{\otimes}$	ref	NEi
Weakening			\circ					\circ				
Exchange						\circ						
Struct. (1) (2)	\circ	\circ	$\circ, (W)$			(Ex)		(W)				
Struct. (3) (4)	\circ	\circ	$\circ, (W)$					(W)				
Aux. Copy	\circ	\circ	\circ	\circ	\circ							
Aux. Subst.	\bullet	\bullet	\bullet	\bullet	\bullet	\circ	\circ	\circ	\circ			
Struct. (5)	\circ	\circ	$\circ, (W)$	\circ	\circ			(W)				
Struct. (6)	\circ	\circ	$\circ, (W)$	\circ	\circ	$\circ, (Ex)$	\circ	$\circ, (W)$	\circ			
Micro beta		\circ				(Ex)				\circ		
Freshness	\circ	\circ	$\circ, (W)$		\bullet	\circ	\circ	$\circ, (W)$		\circ	\circ	1
Locality	\circ	\circ	$\circ, (W)$					(W)		\circ	\circ	2
Param. 1	\circ	\circ	$\circ, (W)$	\bullet	\bullet	\circ	\circ	$\circ, (W)$		\circ	\circ	3
Param. 2	\circ	\circ	$\circ, (W)$	\bullet	\bullet	\circ	\circ	$\circ, (W)$		\circ	\circ	4

Table 2: Dependency of contextual refinements/equivalences on templates

(bf, \mathbb{O} -bf) of annotations. Given two derivable judgements $\vec{x} \mid \vec{a} \vdash t_1 : \star$ and $\vec{x} \mid \vec{a} \vdash t_2 : \star$, we write $B_{\mathbb{O}} \models (\vec{x} \mid \vec{a} \vdash t_1 \preceq^{\dagger_x} t_2 : \star)$ (resp. $B_{\mathbb{O}} \models (\vec{x} \mid \vec{a} \vdash t_1 \simeq^{\dagger_x} t_2 : \star)$) if $B_{\mathbb{O}} \models ((\vec{x} \mid \vec{a} \vdash t_1 : \tau)^{\dagger} \preceq_{\mathbb{N} \times \mathbb{N}}^{\mathbb{C}_Y} (\vec{x} \mid \vec{a} \vdash t_2 : \tau)^{\dagger})$ (resp. $B_{\mathbb{O}} \models ((\vec{x} \mid \vec{a} \vdash t_1 : \tau)^{\dagger} \simeq_{\mathbb{N} \times \mathbb{N}}^{\mathbb{C}_Y} (\vec{x} \mid \vec{a} \vdash t_2 : \tau)^{\dagger})$).

The refinements $\preceq^{\dagger_{\text{all}}}$ and $\preceq^{\dagger_{\text{bf}}}$ enjoy different congruence results, which can be described in terms of syntactical contexts. Let *term-contexts* and their *binding-free* restriction be defined by the following grammar:

$$\begin{aligned}
C &::= [] \mid \text{new } a \multimap t \text{ in } C \mid \text{bind } x \rightarrow t \text{ in } C \\
&\quad \mid \text{new } a \multimap C \text{ in } t \mid \text{bind } x \rightarrow C \text{ in } t \\
&\quad \mid \vec{y}.C \mid \phi(\vec{t}, C, \vec{t}'; \vec{s}) \quad (\text{term-contexts}) \\
\tilde{C} &::= [] \mid \text{new } a \multimap t \text{ in } \tilde{C} \mid \text{bind } x \rightarrow t \text{ in } \tilde{C} \\
&\quad \mid \vec{y}.\tilde{C} \mid \phi(\vec{t}, \tilde{C}, \vec{t}'; \vec{s}) \quad (\text{binding-free term-contexts})
\end{aligned}$$

The type system of SPARTAN (Fig. 1) can be extended to term-contexts, by annotating the hole ‘[]’ as ‘[] $_{\vec{x} \mid \vec{a}}$ ’ and adding a typing rule $\vec{x} \mid \vec{a} \vdash []_{\vec{x} \mid \vec{a}} : \star$. We write $C[]_{\vec{x} \mid \vec{a}}$ when the hole of C is annotated with $\vec{x} \mid \vec{a}$.

Lemma 8.2. *Let $\vec{x} \mid \vec{a} \vdash t_1 : \star$ and $\vec{x} \mid \vec{a} \vdash t_2 : \star$ be derivable judgements. Let C be a term-context, \vec{x}' be a sequence of variables and \vec{a}' be a sequence of atoms, such that $\vec{x}' \mid \vec{a}' \vdash C[]_{\vec{x} \mid \vec{a}} : \star$.*

1. *If $\vec{x} \mid \vec{a} \vdash t_1 \preceq^{\dagger_{\text{all}}} t_2 : \star$, then $\vec{x}' \mid \vec{a}' \vdash C[t_1] \preceq^{\dagger_{\text{all}}} C[t_2] : \star$.*
2. *If $\vec{x} \mid \vec{a} \vdash t_1 \simeq^{\dagger_{\text{all}}} t_2 : \star$, then $\vec{x}' \mid \vec{a}' \vdash C[t_1] \simeq^{\dagger_{\text{all}}} C[t_2] : \star$.*
3. *If $\vec{x} \mid \vec{a} \vdash t_1 \preceq^{\dagger_{\text{bf}}} t_2 : \star$, and C is binding-free, then $\vec{x}' \mid \vec{a}' \vdash C[t_1] \preceq^{\dagger_{\text{bf}}} C[t_2] : \star$.*
4. *If $\vec{x} \mid \vec{a} \vdash t_1 \simeq^{\dagger_{\text{bf}}} t_2 : \star$, and C is binding-free, then $\vec{x}' \mid \vec{a}' \vdash C[t_1] \simeq^{\dagger_{\text{bf}}} C[t_2] : \star$.*

Proof outline. The translation $(-)^{\dagger}$ of SPARTAN terms to hypernets (Fig. 2) can be extended to term-contexts, by translating the rule $\vec{x} \mid \vec{a} \vdash []_{\vec{x} \mid \vec{a}} : \star$ into a path hypernet $\chi : \star \Rightarrow \star^{\otimes |\vec{x}|} \otimes \diamond^{\otimes |\vec{a}|}$. Translating term-contexts indeed yields (graphical) contexts, and translating binding-free term-contexts yields (graphical) binding-free contexts (proof by induction on (binding-free) term-contexts).

For each $i \in \{1, 2\}$, a judgement $\vec{x}' \mid \vec{a}' \vdash C[t_i] : \star$ is derivable, and moreover,

$$(\vec{x}' \mid \vec{a}' \vdash C[t_i] : \star)^{\dagger} = (\vec{x}' \mid \vec{a}' \vdash C[]_{\vec{x} \mid \vec{a}} : \star)^{\dagger} [(\vec{x} \mid \vec{a} \vdash t_i : \star)^{\dagger}]$$

(proof by induction on C). The congruence property of contextual refinements concludes the proof. \square

Each law in Sec. 4 can be proved for the operation sets $\mathbb{O}_{\vec{Y}}^{\text{ex}} = \mathbb{N} \cup \{\lambda, \text{tt}, \text{ff}\}$ and

$$\mathbb{O}_i^{\text{ex}} = \{\vec{\otimes}, \text{ref}, =, :=, !, +, -, -_1\},$$

$$\begin{array}{c}
\frac{\star^{\otimes k_1} \star^{\otimes k_2} \diamond^{\otimes h}}{(\vec{x}, z, \vec{y} \mid \vec{a} \vdash t : \tau)^\dagger} \sim_{\text{Call}}^{\text{Call}} \frac{\star^{\otimes k_1} \star^{\otimes k_2} \diamond^{\otimes h}}{(\vec{x}, \vec{y} \mid \vec{a} \vdash t : \tau)^\dagger} \\
\frac{\star^{\otimes k} \diamond^{\otimes h_1} \diamond^{\otimes h_2}}{(\vec{x} \mid \vec{a}, b, \vec{c} \vdash t : \tau)^\dagger} \sim_{\text{Call}}^{\text{Call}} \frac{\star^{\otimes k} \diamond^{\otimes h_1} \diamond^{\otimes h_2}}{(\vec{x} \mid \vec{a}, \vec{c} \vdash t : \tau)^\dagger}
\end{array}$$

(a) Weakening ($z \notin fv(t)$, $b \notin fa(t)$)

$$\begin{array}{c}
\frac{\star^{\otimes k_1} \star^{\otimes k_2} \diamond^{\otimes h}}{(\vec{x}, z, z', \vec{y} \mid \vec{a} \vdash t : \tau)^\dagger} \sim_{\text{Call}}^{\text{Call}} \frac{\star^{\otimes k_1} \star^{\otimes k_2} \diamond^{\otimes h}}{(\vec{x}, z', z, \vec{y} \mid \vec{a} \vdash t : \tau)^\dagger} \\
\frac{\star^{\otimes k} \diamond^{\otimes h_1} \diamond^{\otimes h_2}}{(\vec{x} \mid \vec{a}, c, c', \vec{b} \vdash t : \tau)^\dagger} \sim_{\text{Call}}^{\text{Call}} \frac{\star^{\otimes k} \diamond^{\otimes h_1} \diamond^{\otimes h_2}}{(\vec{x} \mid \vec{a}, c', c, \vec{b} \vdash t : \tau)^\dagger}
\end{array}$$

(b) Exchange

$$\begin{array}{c}
\frac{\star^{\otimes k} \diamond^{\otimes h}}{(\vec{x} \mid \vec{a} \vdash \vec{t} : \star)^\dagger} \sim_{\text{Call}}^{\text{Call}} \frac{\star^{\otimes k} \diamond^{\otimes h}}{D_{k,n}^\star} \frac{\star^{\otimes n}}{D_{1,n}^\star} \sim_{\text{Call}}^{\text{Call}} \frac{\star^{\otimes k} \diamond^{\otimes h}}{D_{k,n}^\star} \frac{\star^{\otimes n}}{D_{h,n}^\star} \frac{\star^{\otimes n}}{\boxtimes^n (\vec{x} \mid \vec{a} \vdash \vec{t} : \star)^\dagger}
\end{array}$$

(c) Auxiliary Copy (\vec{t} is referentially transparent)

$$\begin{array}{c}
\frac{\star^{\otimes k_1} \star^{\otimes k} \star^{\otimes k_2} \diamond^{\otimes h} \diamond^{\otimes h_1}}{(\vec{x} \mid \vec{a} \vdash \vec{t} : \star)^\dagger} \sim_{\text{Call}}^{\text{Call}} \frac{\star^{\otimes k_1} \star^{\otimes k} \star^{\otimes k_2} \diamond^{\otimes h} \diamond^{\otimes h_1}}{(\vec{x} \mid \vec{a} \vdash \vec{t} : \star)^\dagger} \frac{\star^{\otimes (n+k_1)} \star^{\otimes k_2} \diamond^{\otimes h_1}}{G} \frac{\star^{\otimes n}}{T^n(\star)} \sim_{\text{Call}}^{\text{Call}} \frac{\star^{\otimes k_1} \star^{\otimes k} \star^{\otimes k_2} \diamond^{\otimes h} \diamond^{\otimes h_1}}{(\vec{x} \mid \vec{a} \vdash \vec{t} : \star)^\dagger} \frac{\star^{\otimes (n+k_1)} \star^{\otimes k_2} \diamond^{\otimes h_1}}{G} \frac{\star^{\otimes n}}{T^n(\star)}
\end{array}$$

(d) Auxiliary Substitution (\vec{t} referentially transparent)

Figure 11: Auxiliary laws

by combining the pre-templates. The assumption here is that the pre-templates imply contextual refinement as listed in Table 1. We describe below how each law depends on pre-templates, using Table 2. The (full) Beta law is simply the combination of the Micro Beta law and the substitution law (6), so it is omitted in the table.

Some rows of the table (namely: Weakening, Exchange, Auxiliary Copy and Auxiliary Substitution) represent auxiliary laws shown in Fig. 11. When a law can be proved using the Weakening or Exchange law, it is indicated by (W) or (E), respectively, in the table. The Auxiliary Copy law is used to prove both the Structural laws (5) and (6), while the Auxiliary Substitution law is used to prove the Structural law (6).

The symbol ‘o’ indicates *direct* dependency on pre-templates, in the sense that a law can be proved by combining contextual refinements implied by these pre-templates. For these pre-templates to imply contextual refinements, some other pre-templates may need to imply contextual refinements; these other pre-templates are *indirectly* depended by the law. The indirect dependency is indicated by the symbol ‘•’.

Thanks to the Weakening law, it suffices to check the minimum judgement to prove an observational equivalence. This is the case for binding-free contexts, too, because any context that consists of a hole with no target, and weakening edges, is binding-free. For example, Fig. 12 illustrates a proof of the Parametricity 2 law in the empty environment ($- \mid - \vdash \square : \star$), and this proof is

enough to show the law in any environment $(\vec{x} \mid \vec{a} \vdash \square : \star)$.

8.4 Local reasoning

For the particular operation set \mathbb{O}^{ex} , the following lemma identifies two typical situations, where a local rule simply preserves or duplicates edges contributed by a pre-template, without breaking them.

Lemma 8.3. *Assume any \mathbb{C} -specimen of an output-closed pre-template \triangleleft that has the form*

$$(\mathcal{C}_1[\vec{\chi}', \dot{\mathcal{C}}_2[\vec{\chi}'']]; \vec{G}', \vec{G}''; \vec{H}', \vec{H}''),$$

and any focussed hypernet \dot{N} , such that $\mathcal{C}_1[\vec{G}', \dot{\mathcal{C}}_2[\vec{G}''']]$ and $\mathcal{C}_1[\vec{H}', \dot{\mathcal{C}}_2[\vec{H}''']]$ are states, and $\dot{\mathcal{C}}_2[\vec{G}'] \mapsto \dot{N}$ is a contraction rule or a local rewrite rule of an operation of $\mathbb{O}_i^{\text{ex}} = \{\vec{\text{@}}, \text{ref}, =, :=, !, +, -, -_1\}$.

1. *If all holes of $\dot{\mathcal{C}}_2$ are deep, then there exist a focussed context $\dot{\mathcal{C}}'_2$ and two sequences \vec{G}''' and \vec{H}''' of focus-free hypernets, such that $\dot{\mathcal{C}}_2[\vec{G}'] \mapsto \dot{\mathcal{C}}'_2[\vec{G}'''] = \dot{N}$, $\dot{\mathcal{C}}_2[\vec{H}'] \mapsto \dot{\mathcal{C}}'_2[\vec{H}''']$, and $G_i''' \triangleleft H_i'''$ for each index i .*
2. *In the situation of (1), if additionally both states $\mathcal{C}_1[\vec{G}', \dot{\mathcal{C}}_2[\vec{G}''']]$ and $\mathcal{C}_1[\vec{H}', \dot{\mathcal{C}}_2[\vec{H}''']]$ are rooted, and $|\mathcal{C}_1[\vec{\chi}', \dot{\mathcal{C}}_2[\vec{\chi}''']]|$ is binding-free, then $\dot{\mathcal{C}}'_2$ can be taken so that $|\mathcal{C}_1[\vec{\chi}', \dot{\mathcal{C}}'_2[\vec{\chi}''']]|$ is also binding-free.*
3. *If the pre-template \triangleleft is a relation on contraction trees, then there exist a focussed context $\dot{\mathcal{C}}'_2$ and two sequences \vec{G}''' and \vec{H}''' of focus-free hypernets, such that $\dot{\mathcal{C}}_2[\vec{G}'] \mapsto \dot{\mathcal{C}}'_2[\vec{G}'''] = \dot{N}$, $\dot{\mathcal{C}}_2[\vec{H}'] \mapsto \dot{\mathcal{C}}'_2[\vec{H}''']$, and $G_i''' \triangleleft H_i'''$ for each index i .*

Proof of the point (1). The proof is by case analysis on the local rule $\dot{\mathcal{C}}_2[\vec{G}'] \mapsto \dot{N}$.

- When the rule is a contraction rule, the rule simply duplicates all box edges without changing any deep edges. Because all holes of $\dot{\mathcal{C}}_2$ are deep, there exists a focussed context $\dot{\mathcal{C}}'_2$, whose holes are all deep, such that $\dot{\mathcal{C}}_2[\vec{G}'] \mapsto \dot{\mathcal{C}}'_2[\vec{G}'', \vec{G}''']$ is a contraction rule. Moreover, $\dot{\mathcal{C}}_2[\vec{H}'] \mapsto \dot{\mathcal{C}}'_2[\vec{H}'', \vec{H}''']$ is also a contraction rule.
- When the rule is a beta rewrite rule, the rule replaces a box edge with the hypernet that labels the box. Because all holes of $\dot{\mathcal{C}}_2$ are deep, there exists a focussed context $\dot{\mathcal{C}}'_2$ such that $\dot{\mathcal{C}}_2[\vec{G}'] \mapsto \dot{\mathcal{C}}'_2[\vec{G}'']$ is a beta rewrite rule. Moreover, $\dot{\mathcal{C}}_2[\vec{H}'] \mapsto \dot{\mathcal{C}}'_2[\vec{H}'']$ is also a beta rewrite rule.
- Otherwise, the rule does not involve any deep edge. This means the focussed context $\dot{\mathcal{C}}_2$ must have no hole, and the sequences \vec{G}'' and \vec{H}'' must be empty. We can take a focussed context $\dot{\mathcal{C}}'_2$ with no hole, such that $\dot{\mathcal{C}}_2[] \mapsto \dot{\mathcal{C}}'_2[]$.

Because contraction rules and local rewrite rules are all deterministic, $\dot{\mathcal{C}}'_2[\vec{G}'''] = \dot{N}$ follows from $\dot{\mathcal{C}}_2[\vec{G}'] \mapsto \dot{\mathcal{C}}'_2[\vec{G}''']$. \square

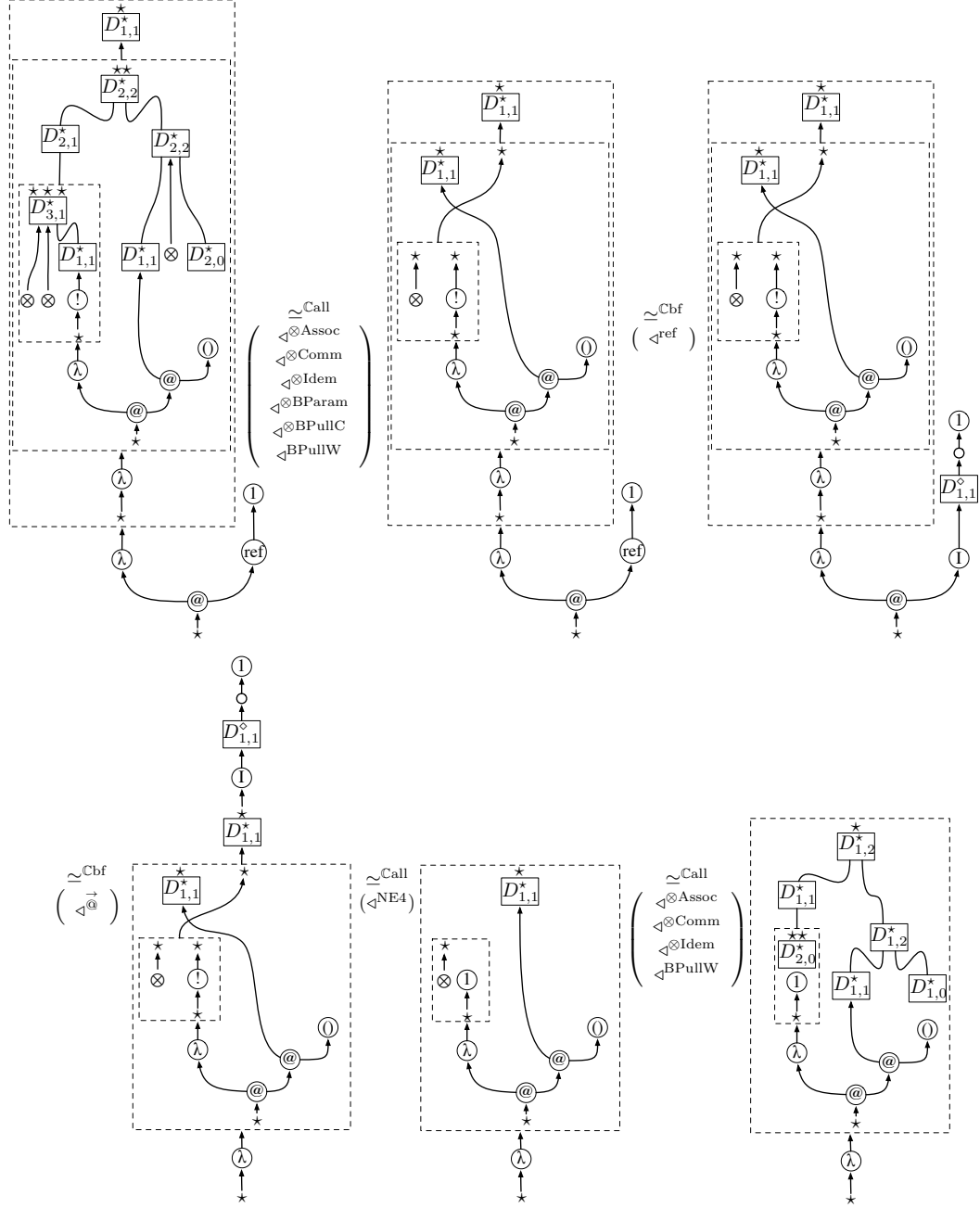
Proof of the point (2). The proof is built on top of the proof of the point (1). In particular, we assume that all holes of $\dot{\mathcal{C}}'_2$ are deep in the case of contraction rules, and $\dot{\mathcal{C}}'_2$ has no hole in the case of local rules of $\mathbb{O}_i^{\text{ex}} \setminus \{\vec{\text{@}}\}$. Under this assumption, our goal is to prove that $|\mathcal{C}_1[\vec{\chi}', \dot{\mathcal{C}}'_2]|$ is binding-free.

Let \mathcal{C} denote $\mathcal{C}_1[\vec{\chi}', |\dot{\mathcal{C}}_2|]$ and \mathcal{C}' denote $\mathcal{C}_1[\vec{\chi}', |\dot{\mathcal{C}}'_2|]$. Because $\mathcal{C} = |\mathcal{C}_1[\vec{\chi}', \dot{\mathcal{C}}_2]|$ and $\mathcal{C}' = |\mathcal{C}_1[\vec{\chi}', \dot{\mathcal{C}}'_2]|$, it suffices to prove that \mathcal{C}' is binding-free, given that \mathcal{C} is binding-free.

Firstly, because \mathcal{C} is binding-free, $|\dot{\mathcal{C}}_2|$ is necessarily binding-free. By inspecting the local rule $\dot{\mathcal{C}}_2[\vec{G}'] \mapsto \dot{\mathcal{C}}'_2[\vec{G}''']$, we check below that $|\dot{\mathcal{C}}'_2|$ is also binding-free.

- When the rule is a contraction rule, or a beta rewrite rule, any path that makes $|\dot{\mathcal{C}}'_2|$ not binding-free gives a path in $|\dot{\mathcal{C}}_2|$, which leads to a contradiction.
- Otherwise, $|\dot{\mathcal{C}}'_2|$ is trivially binding-free because it does not have any hole edges.

$$(- \mid - \vdash (\lambda x. \lambda f. (\lambda z. !x) \vec{\text{@}}(f \vec{\text{@}}())) \vec{\text{@}}(\text{ref } 1) : \star)^\dagger =$$



$$= (- \mid - \vdash \lambda f. (\lambda z. 1) \vec{\text{@}}(f \vec{\text{@}}()) : \star)^\dagger.$$

Figure 12: A proof outline of the Parametricity 2 law, in the empty environment

Now we prove that \mathcal{C}' is binding-free by contradiction; we assume that there exists a path P in \mathcal{C}' , from a source of a contraction, atom, box or hole edge e , to a source of a hole edge e' . Let χ be the last hole label of \mathcal{C}_1 (i.e. $\mathcal{C}_1[\vec{\chi}, \chi]$), and e_χ denote the hole edge of \mathcal{C}_1 labelled with χ . We derive a contradiction by case analysis on the path P .

- When e' comes from \mathcal{C}_1 , and the path P consists of edges from \mathcal{C}_1 only, the path P gives a path in \mathcal{C}_1 . Because P does not contain e_χ , it also gives a path in \mathcal{C} . This contradicts \mathcal{C} being binding-free.
- When e' comes from \mathcal{C}_1 , and the path P contains an edge from \mathcal{C}_2 , by finding the last edge from \mathcal{C}_2 in P , we can take a suffix of P that gives a path P' from a target of the hole edge e_χ to a source of the hole edge e' , in \mathcal{C}_1 . Because the suffix path P' does not contain e_χ , it gives a path in \mathcal{C} . We inspect the local rule $\dot{\mathcal{C}}_2[\vec{G}'] \mapsto \dot{\mathcal{C}}_2[\vec{G}''']$ as follows.
 - When the rule is a contraction rule, because each output of $|\dot{\mathcal{C}}_2|$ is reachable from a source of a contraction edge, adding the contraction edge at the beginning of P' gives a path in \mathcal{C} from a source of the contraction edge to a source of the hole edge e' . This contradicts \mathcal{C} being binding-free.
 - When the rule is a rewrite rule of operations $\{\vec{\text{@}}, \text{ref}, =, :=, !\}$, P' gives a path in \mathcal{C} to the hole edge e' , from either of the following: a target of an atom or box edge; or a source of a contraction, atom or a box edge (by typing). This contradicts \mathcal{C} being binding-free.
 - Otherwise, i.e. when the rule is a rewrite rule of operations $\{+, -, -_1\}$, the hole edge e_χ actually does not have any target. This contradicts, in the first place, the path P containing an edge from \mathcal{C}_2 .
- When both e and e' come from $|\dot{\mathcal{C}}_2|$, and the path P gives a path in $|\dot{\mathcal{C}}_2|$, this contradicts $|\dot{\mathcal{C}}_2|$ being binding-free.
- When both e and e' come from $|\dot{\mathcal{C}}_2|$, and the path P does not give a single path in $|\dot{\mathcal{C}}_2|$, we inspect the local rule $\dot{\mathcal{C}}_2[\vec{G}'] \mapsto \dot{\mathcal{C}}_2[\vec{G}''']$ as follows.
 - When the rule is a contraction rule, all holes of $\dot{\mathcal{C}}_2'$ are deep. It is impossible for P , which is to a target of a deep hole edge, not to give a path in $|\dot{\mathcal{C}}_2|$. This is a contradiction.
 - When the rule is a local rewrite rule of any operations but ' $\vec{\text{@}}$ ', $\dot{\mathcal{C}}_2'$ actually does not have any hole edge. It is impossible for the hole edge e' to be from $|\dot{\mathcal{C}}_2|$. This is a contradiction.
 - When the rule is a beta rewrite rule, the hole edge e_χ actually has only one source. If P consists of edges from $|\dot{\mathcal{C}}_2|$ only, the source must be also a target of e_χ ; otherwise, P has a sub-sequence that gives a path in \mathcal{C}_1 . In either case, there exists a path P' from the source of e_χ to the source of e_χ , i.e. a cycle around the source of e_χ , in \mathcal{C}_1 . This cycle gives a cycle in \mathcal{C} around the source of e_χ .
 In the case of a beta rewrite rule, outputs are all reachable from the input, which coincides with the token source, in $\dot{\mathcal{C}}_2[\vec{\chi}']$. Therefore, the cycle P' also gives a cycle in $\mathcal{C}_1[\vec{\chi}', \dot{\mathcal{C}}_2[\vec{\chi}']]$, around the token source. Because $(\mathcal{C}_1[\vec{\chi}', \dot{\mathcal{C}}_2[\vec{\chi}']]; \vec{G}', \vec{G}''; \vec{H}', \vec{H}'')$ is a \mathbb{C} -specimen of the output-closed pre-template \triangleleft , by Lem. A.12(3), at least one of the states $\mathcal{C}_1[\vec{G}', \dot{\mathcal{C}}_2[\vec{G}']]$ and $\mathcal{C}_1[\vec{H}', \dot{\mathcal{C}}_2[\vec{H}']]$ is not rooted. This is a contradiction.
- When e comes from \mathcal{C}_1 and e' comes from $|\dot{\mathcal{C}}_2|$, we inspect the local rule $\dot{\mathcal{C}}_2[\vec{G}'] \mapsto \dot{\mathcal{C}}_2[\vec{G}''']$ as follows.
 - When the rule is a contraction rule, by finding the last edge from \mathcal{C}_1 in P , we can take a suffix of P that gives a path P' from an input to a source of the hole edge e' in $|\dot{\mathcal{C}}_2|$. However, the path P' cannot exist because the hole edge e' is deep in $|\dot{\mathcal{C}}_2|$. This is a contradiction.

- When the rule is a local rewrite rule of any operations but ‘@’, $\dot{\mathcal{C}}'_2$ actually does not have any hole edge. It is impossible for the hole edge e' to be from $|\dot{\mathcal{C}}'_2|$. This is a contradiction.
- When the rule is a beta rewrite rule, by finding the first edge from $|\dot{\mathcal{C}}'_2|$ in P , we can take a prefix of P that gives a path P' from a source of the edge e to a source of the hole edge e_χ , in \mathcal{C}_1 . Because P does not contain e_χ , P' does not contain e_χ either.

In the case of a beta rewrite rule, e_χ has only one source, and the input of $\dot{\mathcal{C}}'_2[\chi'']$ coincides with the token source. Therefore, P' gives a path from a source of e to the token source in $\mathcal{C}_1[\vec{\chi}', \dot{\mathcal{C}}'_2[\chi'']]$, and this path is not an operation path, because of its first edge e . Because $(\mathcal{C}_1[\vec{\chi}', \dot{\mathcal{C}}'_2[\chi'']]; \vec{G}', \vec{G}''; \vec{H}', \vec{H}'')$ is a \mathbb{C} -specimen of the output-closed pre-template \triangleleft , by Lem. A.12(3), at least one of the states $\mathcal{C}_1[\vec{G}', \dot{\mathcal{C}}'_2[\vec{G}']]$ and $\mathcal{C}_1[\vec{H}', \dot{\mathcal{C}}'_2[\vec{H}']]$ is not rooted. This is a contradiction.

□

Proof of the point (3). The proof is by case analysis on the local rule $\dot{\mathcal{C}}'_2[\vec{G}'''] \mapsto \dot{N}$.

- When the rule is a contraction rule, the rule simply duplicates all box edges without changing any deep edges, and does not change any existing shallow contraction/weakening edges. Because \triangleleft relates only contraction trees, deep edges from \vec{G}'' are duplicated and shallow ones from \vec{G}'' are preserved, by the rule. There exist a sequence \vec{G}''' of contraction trees and a focussed context $\dot{\mathcal{C}}'_2$, such that $\vec{G}''' \subseteq \vec{G}''$ (as sets) and $\dot{\mathcal{C}}'_2[\vec{G}'''] \mapsto \dot{\mathcal{C}}'_2[\vec{G}'', \vec{G}''']$ is a contraction rule. Moreover, there exists a sequence \vec{H}''' of contraction trees that satisfies $\vec{H}''' \subseteq \vec{H}''$ (as sets) and corresponds to \vec{G}''' . Because \triangleleft relates only contraction trees, $\dot{\mathcal{C}}'_2[\vec{H}'''] \mapsto \dot{\mathcal{C}}'_2[\vec{H}'', \vec{H}''']$ is also a contraction rule.
- When the rule is a beta rewrite rule, the rule involves no shallow contraction trees. Because \triangleleft relates only contraction trees, all holes of $\dot{\mathcal{C}}'_2$ must be deep, and the proof is reduced to the point (1).
- When the rule is a rewrite rule of name-accessing operations $\{=, :=, !\}$, the rule preserves contraction trees at any depth. There exists a focussed context $\dot{\mathcal{C}}'_2$ such that $\dot{\mathcal{C}}'_2[\vec{G}'''] \mapsto \dot{\mathcal{C}}'_2[\vec{G}'']$ is a local rewrite rule. Moreover, because \triangleleft relates only contraction trees, $\dot{\mathcal{C}}'_2[\vec{H}'''] \mapsto \dot{\mathcal{C}}'_2[\vec{H}'']$ is also a local rewrite rule.
- Otherwise, the rule involves no contraction trees, which means the focussed context $\dot{\mathcal{C}}'_2$ must have no hole, and the sequences \vec{G}'' and \vec{H}'' must be empty. We can take a focussed context $\dot{\mathcal{C}}'_2$ with no hole, such that $\dot{\mathcal{C}}'_2[] \mapsto \dot{\mathcal{C}}'_2[]$.

Because contraction rules and local rewrite rules are all deterministic, $\dot{\mathcal{C}}'_2[\vec{G}'''] = \dot{N}$ follows from $\dot{\mathcal{C}}'_2[\vec{G}'''] \mapsto \dot{\mathcal{C}}'_2[\vec{G}''']$. □

8.5 Details of input-safety and robustness proofs

In this section we give some details of proving input-safety and robustness of the pre-templates.

Fig. 13 lists triggers that we use to prove some input-safety and robustness of the pre-templates. Table 3 shows contextual refinements/equivalences implied by these triggers, given that some pre-templates (shown in the “dependency” column) imply contextual refinement as shown in Table 1. All the implications can be proved simply using the congruence property and transitivity of contextual refinement. Table 3 shows which pre-template requires each trigger in its proof of input-safety or robustness (in the “used for” column). Note that the converse of any trigger is again a trigger.

Recall that there is a choice of contraction trees upon applying a contraction rule or some local rewrite rule. The minimum choice is to collect only contraction edges whose target is reachable from the token target. The maximum choice is to take the contraction tree(s) so that no contraction or weakening edge is incoming to the unique hole edge in a context.

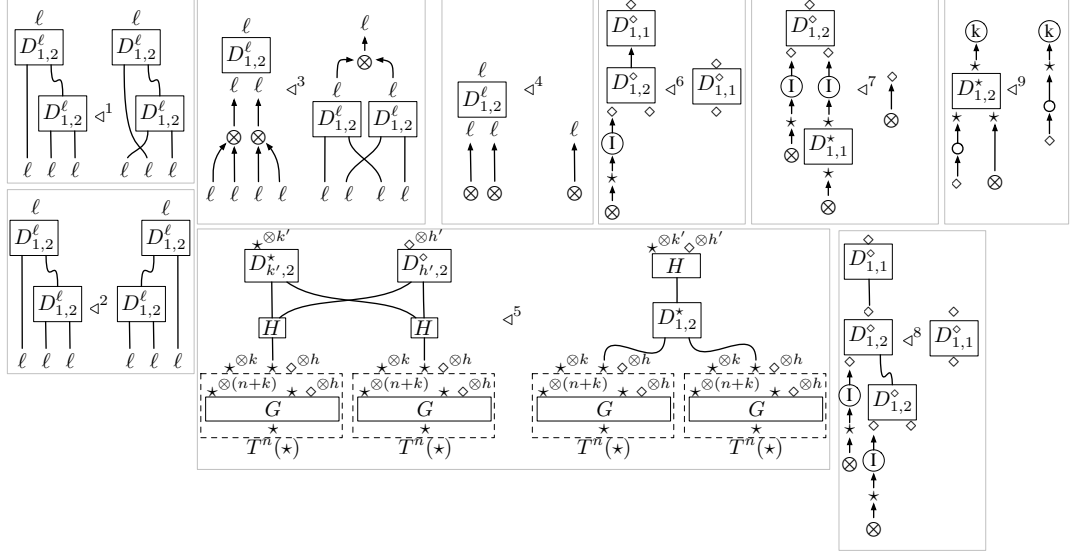


Figure 13: Triggers (H is a copyable hypernet, and G is a hypernet)

	dependency	implication of $H_1 \triangleleft H_2$	used for
\triangleleft^1	$\triangleleft^{\otimes} \text{Assoc}, \triangleleft^{\otimes} \text{Comm}$	$H_1 \simeq_{\mathbb{N}}^{\mathcal{C}_0^{\text{ex}}} H_2$	\triangleleft^{\otimes}
\triangleleft^2	$\triangleleft^{\otimes} \text{Assoc}, \triangleleft^{\otimes} \text{Comm}$	$H_1 \simeq_{\mathbb{N}}^{\mathcal{C}_0^{\text{ex}}} H_2$	\triangleleft^{\otimes}
\triangleleft^3	$\triangleleft^{\otimes} \text{Assoc}, \triangleleft^{\otimes} \text{Comm}, \triangleleft^{\otimes} \text{Idem}$	$H_1 \simeq_{\mathbb{N}}^{\mathcal{C}_0^{\text{ex}}} H_2$	$\triangleleft^{\text{BPullC}}$
\triangleleft^4	$\triangleleft^{\otimes} \text{Idem}$	$H_1 \simeq_{\mathbb{N}}^{\mathcal{C}_0^{\text{ex}}} H_2$	$\triangleleft^{\text{BPullW}}, \triangleleft^{\text{NE3}}$
\triangleleft^5	$\triangleleft^{\otimes} \text{Assoc}, \triangleleft^{\otimes} \text{Comm}, \triangleleft^{\otimes} \text{Idem}, \triangleleft^{\otimes}, \triangleleft^{\text{GC}}$	$H_1 \preceq_{\leq \mathbb{N}}^{\mathcal{C}_0^{\text{ex}}} H_2, H_2 \preceq_{\geq \mathbb{N}}^{\mathcal{C}_0^{\text{ex}}} H_1$	$\triangleleft^{\text{BPullD}}$
\triangleleft^6	$\triangleleft^{\otimes} \text{Assoc}, \triangleleft^{\otimes} \text{Idem}, \triangleleft^{\text{GC}}$	$H_1 \simeq_{\mathbb{N}}^{\mathcal{C}_0^{\text{ex}}} H_2$	$\triangleleft^{\text{NE1}}, \triangleleft^{\text{NE4}}$
\triangleleft^7	$\triangleleft^{\otimes} \text{Idem}, \triangleleft^{\otimes} \text{GC}$	$H_1 \simeq_{\mathbb{N}}^{\mathcal{C}_0^{\text{ex}}} H_2$	$\triangleleft^{\text{NE1}}$
\triangleleft^8	$\triangleleft^{\otimes} \text{Assoc}, \triangleleft^{\otimes} \text{Idem}, \triangleleft^{\text{GC}}$	$H_1 \simeq_{\mathbb{N}}^{\mathcal{C}_0^{\text{ex}}} H_2$	$\triangleleft^{\text{NE3}}$
\triangleleft^9	$\triangleleft^{\otimes}, \triangleleft^{\text{GC}}$	$H_1 \preceq_{\geq \mathbb{N}}^{\mathcal{C}_0^{\text{ex}}} H_2, H_2 \preceq_{\leq \mathbb{N}}^{\mathcal{C}_0^{\text{ex}}} H_1$	$\triangleleft^{\text{NE3}}, \triangleleft^{\text{NE4}}$

Table 3: Triggers and their implied contextual refinements/equivalences

8.5.1 Pre-templates on contraction trees

First we check input-safety and robustness of $\triangleleft^{\otimes \text{Assoc}}$, $\triangleleft^{\otimes \text{Comm}}$ and $\triangleleft^{\otimes \text{Idem}}$, which are all on contraction trees.

input-safety of $\triangleleft^{\otimes \text{Assoc}}$ and $\triangleleft^{\otimes \text{Comm}}$ can be checked as follows. Given a \mathbb{C}_{ex} -specimen $(\dot{\mathcal{C}}; \vec{H}^1; \vec{H}^2)$ with an entering search token, because any input of a contraction tree is a source of a contraction edge, we have:

$$\dot{\mathcal{C}}[\vec{H}^1] \xrightarrow{\bullet} \langle \dot{\mathcal{C}}[\vec{H}^1] \rangle_{\sharp/?}, \quad \dot{\mathcal{C}}[\vec{H}^2] \xrightarrow{\bullet} \langle \dot{\mathcal{C}}[\vec{H}^2] \rangle_{\sharp/?}.$$

It can be observed that a rewrite transition is possible in $\langle \dot{\mathcal{C}}[\vec{H}^1] \rangle_{\sharp/?}$ if and only if a rewrite transition is possible in $\langle \dot{\mathcal{C}}[\vec{H}^2] \rangle_{\sharp/?}$. When a rewrite transition is possible in both states, we can use Lem. 8.3(3), by considering a maximal possible contraction rule. The results of the rewrite transition can be given by a new quasi- \mathbb{C}_{ex} -specimen up to $(=, =)$ (here $=$ denotes equality on states). When no rewrite transition is possible, both of the states are not final but stuck.

Robustness of the three pre-templates and their converse can also be proved using Lem. 8.3(3), by considering a maximal possible local (contraction or rewrite) rule in each case.

8.5.2 Input-safety of pre-templates not on contraction trees

As mentioned in Sec. 8.2, pre-templates that relate hypernets with no input of type \star are trivially input-safety for any parameter (\mathbb{C}, Q, Q') . This leaves us pre-templates \triangleleft^{\otimes} , $\triangleleft^{\vec{\otimes}}$, $\triangleleft^{\text{ref}}$, $\triangleleft^{\text{NE1}}$, $\triangleleft^{\text{NE3}}$ and $\triangleleft^{\text{NE4}}$ to check.

As for \triangleleft^{\otimes} , note that the pre-template \triangleleft^{\otimes} relates hypernets with at least one input. Any \mathbb{C}_{ex} -specimen of \triangleleft^{\otimes} with an entering search token can be turned into the form $(\mathcal{C}[(?;_j \chi), \vec{\chi}]; H^1, \vec{H}^1; H^2, \vec{H}^2)$ where j is a positive number. The proof is by case analysis on the number j .

- When $j = 1$, we have:

$$\begin{aligned} \mathcal{C}[(?;_j H^1), \vec{H}^1] &\xrightarrow{\bullet} \mathcal{C}[(\sharp;_j H^1), \vec{H}^1] \\ &\rightarrow \mathcal{C}[(?;_j H^2), \vec{H}^1]. \end{aligned}$$

We can take $(\mathcal{C}[(?;_j H^2), \vec{\chi}]; \vec{H}^1; \vec{H}^2)$ as a \mathbb{C}_{ex} -specimen, and the token in $\mathcal{C}[(?;_j H^2), \vec{\chi}]$ is not entering.

- When $j > 1$, the token target must be a source of a contraction edge. There exist a focus-free context $\mathcal{C}'[\chi']$, two focus-free hypernets $H'^1 \triangleleft^{\otimes} H'^2$ and a focus-free hypernet G , such that

$$\begin{aligned} \mathcal{C}[(?;_j H^1), \vec{H}^1] &\xrightarrow{\bullet} \mathcal{C}[(\sharp;_j H^1), \vec{H}^1] \\ &\rightarrow \mathcal{C}[(?;_j \mathcal{C}'[H'^1]), \vec{H}^1], \\ \mathcal{C}[(?;_j H^2), \vec{H}^2] &\xrightarrow{\bullet} \mathcal{C}[(\sharp;_j H^2), \vec{H}^2] \\ &\rightarrow \mathcal{C}[(?;_j G), \vec{H}^2], \end{aligned}$$

and $\mathcal{C}'[H'^2] \dot{=}_{\mathbb{N}} G$ given by the trigger \triangleleft^1 via Lem. 6.19. The results of these sequences give a quasi- \mathbb{C}_{ex} -specimen up to $(=, \dot{=}_{\mathbb{N}})$.

A proof of input-safety of the operational pre-templates $\triangleleft^{\vec{\otimes}}$ and $\triangleleft^{\text{ref}}$ is a simpler version of that of \triangleleft^{\otimes} , because the operational pre-templates relate hypernets with only one input.

Let \mathbb{C} be either \mathbb{C}_{ex} or $\mathbb{C}_{\text{ex-bf}}$. Any \mathbb{C} -specimen of an operational pre-template with an entering search token can be turned into the form $(\mathcal{C}[(?; \chi), \vec{\chi}]; H^1, \vec{H}^1; H^2, \vec{H}^2)$; note that the parameter j that we had for \triangleleft^{\otimes} is redundant in $?; \chi$. We have:

$$\begin{aligned} \mathcal{C}[(?; H^1), \vec{H}^1] &\xrightarrow{\bullet} \mathcal{C}[(\sharp; H^1), \vec{H}^1] \\ &\rightarrow \mathcal{C}[(?; H^2), \vec{H}^1]. \end{aligned}$$

We can take $(\mathcal{C}[(?; H^2), \vec{\chi}]; \vec{H}^1; \vec{H}^2)$ as a \mathbb{C}_{ex} -specimen, and the token in $\mathcal{C}[(?; H^2), \vec{\chi}]$ is not entering. This data gives a $\mathbb{C}_{\text{ex-bf}}$ -specimen when $\mathbb{C} = \mathbb{C}_{\text{ex-bf}}$, which follows from the closedness

of $\mathbb{C}_{\mathbb{O}^{\text{ex-bf}}}$ with respect to plugging (Lem. A.21). Note that $?; H^1$ can be seen as a context with no holes, which is trivially binding-free.

Finally, we look at the name-exhaustive pre-templates $\triangleleft^{\text{NE1}}$, $\triangleleft^{\text{NE3}}$ and $\triangleleft^{\text{NE4}}$. Any $\mathbb{C}_{\mathbb{O}^{\text{ex}}}$ -specimen of one of these pre-templates, with an entering search token, can be turned into the form $(\mathcal{C}[(?;_j \chi), \vec{\chi}]; H^1, \vec{H}^1; H^2, \vec{H}^2)$ where j is a positive number. The token target is a source of an edge labelled with $\lambda \in \mathbb{O}^{\text{ex}}$, so we have:

$$\begin{aligned} \mathcal{C}[(?;_j H^1), \vec{H}^1] &\dot{\rightarrow} \mathcal{C}[(\checkmark;_j H^1), \vec{H}^1], \\ \mathcal{C}[(?;_j H^2), \vec{H}^2] &\dot{\rightarrow} \mathcal{C}[(\checkmark;_j H^2), \vec{H}^2]. \end{aligned}$$

The results of these sequences give a quasi- $\mathbb{C}_{\mathbb{O}^{\text{ex}}}$ -specimen up to $(=, =)$.

8.5.3 Robustness of pre-templates not on contraction trees: a principle

Robustness can be checked by inspecting rewrite transition $\dot{\mathcal{C}}[\vec{H}^1] \rightarrow \dot{N}'$ from the state given by a specimen $(\dot{\mathcal{C}}; \vec{H}^1; \vec{H}^2)$ of a pre-template, where the token of $\dot{\mathcal{C}}$ is not entering. We in particular consider the minimum local (contraction or rewrite) rule $\dot{G} \mapsto \dot{G}'$ applied in this transition. This means that, in the hypernet \dot{G} , every vertex is reachable from the token target.

The inspection boils down to analyse how the minimum local rule involves edges that come from the hypernets \vec{H}^1 . If all the involvement is deep, i.e. only deep edges from \vec{H}^1 are involved in the local rule, these deep edges must come via deep holes in the context $\dot{\mathcal{C}}$. We can use Lem. 8.3(1).

If the minimum local rule involves shallow edges that are from \vec{H}^1 , endpoints of these edges are reachable from the token target. This means that, in the context $\dot{\mathcal{C}}$, some holes are shallow and their sources are reachable from the token target. Moreover, given that the token is not entering in $\dot{\mathcal{C}}$, the context has a path from the token target to a source of a hole edge.

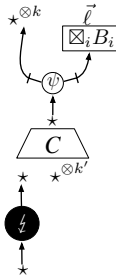


Figure 14: A shallow overlap (C is a contraction tree, B_i are box edges)

For example, in checking robustness of $\triangleleft^{\text{BPerm}}$ with respect to copy transitions, one situation of shallow overlaps is when \dot{G} is in the form in Fig. 14, and some of the box edges B_i are from \vec{H}^1 . Taking the minimum contraction rule means that C in the graph is a contraction tree that gives a path from the token target. This path C followed by the operation edge ϕ corresponds to paths from the token target to hole sources in the context $\dot{\mathcal{C}}$.

So, if the minimum local rule involves shallow edges that are from \vec{H}^1 , the context $\dot{\mathcal{C}}$ necessarily has a path P from the token target to a hole source. The path becomes a path in the state $\dot{\mathcal{C}}[\vec{H}^1]$, from the token target to a source of an edge e that is from \vec{H}^1 . The edge e is necessarily shallow, and also involved in the application of the minimum local rule, because of the connectivity of \dot{G} . Moreover, a source of the edge e is an input, in the relevant hypernet of \vec{H}^1 . By inspecting minimum local rules, we can enumerate possible labelling of the path P and the edge e , as summarised in Table 4. Explanation on the notation used in the table is to follow.

We use the regular-expression like notation in Table 4. For example, $(\otimes_{\mathcal{C}}^*)^+ \cdot \mathbb{O}^{\text{ex}}$ represents finite sequences of edge labels, where more than one occurrences of the label $\otimes_{\mathcal{C}}^*$ is followed by one operation $\phi \in \mathbb{O}^{\text{ex}}$. This characterises paths that inhabit the overlap shown in Fig. 14, i.e. the contraction tree C followed by the operation edge ϕ . Note that this regular-expression like notation is not a proper regular expression, because it is over the infinite alphabet $M_{\mathbb{O}^{\text{ex}}}$, the edge label set, and it accordingly admits infinite alternation (aka. union) implicitly.

$$\text{ref} \cdot (\mathbb{O}_{\checkmark}^{\text{ex}})^*, \quad !, \quad -_1$$

from the token target. All paths but $(\otimes_C^*)^+$ gives rise to a state that is not rooted, which can be checked using Lem. 6.17. This reduces the robustness check of $(\triangleleft^{\otimes})^{-1}$ to that of \triangleleft^{\otimes} .

8.5.5 Robustness of $\triangleleft^{\text{GC}}$ and $\triangleleft^{\text{NE2}}$, and their converse

These four pre-templates both relate hypernets with no inputs. Proofs of robustness of them and their converse always boils down to the use of Lem. 8.3(1), following the discussion in Sec. 8.5.3. Namely, it is impossible to find the path P in the context \check{C} from the token target to a hole source.

8.5.6 Robustness of $\triangleleft^{\text{BPerm}}$, $\triangleleft^{\text{BPullC}}$, $\triangleleft^{\text{BPullW}}$ and $\triangleleft^{\text{BPullD}}$, and their converse

These eight pre-templates all concern boxes. Using Table 4, shallow overlaps are caused by paths $(\otimes_C^*)^+ \cdot \mathbb{O}^{\text{ex}}$ and $\vec{\textcircled{a}} \cdot \lambda$ from the token target.

Robustness check with respect to compute transitions of operations $\mathbb{O}_{\checkmark}^{\text{ex}} \setminus \{\textcircled{a}\}$ always boil down to Lem. 8.3(1).

As for compute transitions of the operation ' $\vec{\textcircled{a}}$ ', either one path $\vec{\textcircled{a}} \cdot \lambda$ causes one shallow overlap, or all overlaps are deep. The latter situation boils down to Lem. 8.3(1). In the former situation, a beta rule involves one box that is contributed by a pre-template, and states given by a $\mathbb{C}_{\mathbb{O}^{\text{ex}}}$ -specimen are turned into a quasi- $\mathbb{C}_{\mathbb{O}^{\text{ex}}}$ -specimen up to $(=, =)$, by one rewrite transition.

As for copy transitions, there are two possible situations.

- Paths $(\otimes_C^*)^+ \cdot \mathbb{O}^{\text{ex}}$ cause some shallow overlaps and there are some deep overlaps too.
- All overlaps are deep, which boils down to Lem. 8.3(1).

In the first situation, some of the shallow boxes duplicated by a contraction rule are contributed by a pre-template, and other duplicated boxes may have deep edges contributed by the pre-template. By tracking these shallow and deep contributions in a contraction rule, it can be checked that one rewrite transition turns states given by a $\mathbb{C}_{\mathbb{O}^{\text{ex}}}$ -specimen into a quasi- $\mathbb{C}_{\mathbb{O}^{\text{ex}}}$ -specimen. This quasi-specimen is up to the following, depending on pre-templates:

- $(=, =)$ for $\triangleleft^{\text{BPerm}}$ and its converse,
- $(=, \dot{=}_{=_{\mathbb{N}}})$ for $\triangleleft^{\text{BPullC}}$, and $(\dot{=}_{=_{\mathbb{N}}}, =)$ for its converse, using the trigger \triangleleft^3 ,
- $(=, \dot{=}_{=_{\mathbb{N}}})$ for $\triangleleft^{\text{BPullW}}$, and $(\dot{=}_{=_{\mathbb{N}}}, =)$ for its converse, using the trigger \triangleleft^4 , and
- $(=, \dot{=}_{\leq_{\mathbb{N}}})$ for $\triangleleft^{\text{BPullD}}$, and $(\dot{=}_{\leq_{\mathbb{N}}}, =)$ for its converse, using the trigger \triangleleft^5 .

8.5.7 Robustness of operational pre-templates and their converse

For the operational pre-templates and their converse, we use the class $\mathbb{C}_{\mathbb{O}^{\text{ex-bf}}}$ of binding-free contexts. This restriction is crucial to rule out some shallow overlaps.

Using Table 4, shallow overlaps with the operational pre-templates $\triangleleft^{\vec{\textcircled{a}}}$ and $\triangleleft^{\text{ref}}$ are caused by paths $(\otimes_C^*)^+$ from the token context. However, the restriction to binding-free contexts makes this situation impossible, which means the robustness check always boils down to Lem. 8.3(1) and Lem. 8.3(2).

In checking robustness of the converse $(\triangleleft^{\vec{\textcircled{a}}})^{-1}$ and $(\triangleleft^{\text{ref}})^{-1}$, shallow overlaps are caused by paths:

$$\begin{array}{lll} (\otimes_C^*)^+, & =, & +, \\ \vec{\textcircled{a}} \cdot (\mathbb{O}_{\checkmark}^{\text{ex}})^*, & := \cdot (\mathbb{O}_{\checkmark}^{\text{ex}})^*, & -, \\ \text{ref} \cdot (\mathbb{O}_{\checkmark}^{\text{ex}})^*, & !, & -_1 \end{array}$$

from the token target. Like the case of $(\triangleleft^{\otimes})^{-1}$, all paths but $(\otimes_C^*)^+$ give rise to a state that is not rooted, which can be checked using Lem. 6.17. The paths $(\otimes_C^*)^+$ are impossible because of the binding-free restriction. As a result, this robustness check also boils down to Lem. 8.3(1) and Lem. 8.3(2).

8.5.8 Robustness of name-exhaustive pre-templates on lambda-abstractions, and their converse

We here look at the three name-exhaustive pre-templates $\triangleleft^{\text{NE1}}$, $\triangleleft^{\text{NE3}}$ and $\triangleleft^{\text{NE4}}$, and their converse. All these six pre-templates concern lambda-abstractions, and they give rather rare examples of robustness check where we compare different numbers of transitions.

Using Table 4, shallow overlaps with these pre-templates are caused by paths:

$$\begin{array}{ll} (\otimes_{\mathbb{C}}^*)^+, & \vec{\textcircled{a}} \cdot (\mathbb{O}_{\check{\vee}}^{\text{ex}})^*, \\ \text{ref} \cdot (\mathbb{O}_{\check{\vee}}^{\text{ex}})^*, & := \cdot (\mathbb{O}_{\check{\vee}}^{\text{ex}})^* \end{array}$$

from the token target.

As for compute transitions of operations $\mathbb{O}_{\check{\vee}}^{\text{ex}} \setminus \{\vec{\textcircled{a}}\}$, there are two possible situations.

- Shallow overlaps are caused by paths $\text{ref} \cdot (\mathbb{O}_{\check{\vee}}^{\text{ex}})^*$ or $:= \cdot (\mathbb{O}_{\check{\vee}}^{\text{ex}})^*$.
- There is no overlap at all, which boils down to Lem. 8.3(1).

In the first situation, a stable hypernet G_S of a local rewrite rule (see e.g. Fig. 5) contains shallow edges, labelled with $\lambda \in \mathbb{O}_{\check{\vee}}^{\text{ex}}$, that are contributed by a pre-template. The overlapped shallow contributions are not modified at all by the rewrite rule, and consequently, one rewrite transition results in a quasi- $\mathbb{C}_{\mathbb{O}^{\text{ex}}}$ -specimen up to $(=, =)$.

As for copy transitions, either one path $(\otimes_{\mathbb{C}}^*)^+$ causes one shallow overlap, or all overlaps are deep. The latter situation boils down to Lem. 8.3(1). In the former situation, one lambda-abstraction contributed by a pre-template gets duplicated. Namely, a $\mathbb{C}_{\mathbb{O}^{\text{ex}}}$ -specimen with a non-entering rewrite token can be turned into the form $(\mathcal{C}[\dot{\mathcal{C}}'[\chi'], \vec{\chi}]; H^1, \vec{H}^1; H^2, \vec{H}^2)$ where $\dot{\mathcal{C}}'$ is a focussed context in the form of Fig. 15. There exist a focussed context $\dot{\mathcal{C}}''$ and two hypernets $G^1 \triangleleft^{\text{NE1}} G^2$ such that:

$$\begin{aligned} \mathcal{C}[\dot{\mathcal{C}}'[H^1], \vec{H}^1] &\rightarrow \mathcal{C}[\dot{\mathcal{C}}''[G^1], \vec{H}^1], \\ \mathcal{C}[\dot{\mathcal{C}}'[H^2], \vec{H}^2] &\rightarrow \mathcal{C}[\dot{\mathcal{C}}''[G^2], \vec{H}^2]. \end{aligned}$$

Results of these rewrite transitions give a new quasi- $\mathbb{C}_{\mathbb{O}^{\text{ex}}}$ -specimen up to $(=, =)$.

As for compute transitions of the operation $\vec{\textcircled{a}}$, there are two possible situations.

- One path $\vec{\textcircled{a}}$ causes a shallow overlap of the edge that has label λ and gets eliminated by a beta rewrite rule, and possibly some other paths $\vec{\textcircled{a}} \cdot (\mathbb{O}_{\check{\vee}}^{\text{ex}})^*$ cause shallow overlaps in the stable hypernet G_S (see Fig. 4).
- There are possibly deep overlaps, and paths $\vec{\textcircled{a}} \cdot (\mathbb{O}_{\check{\vee}}^{\text{ex}})^*$ may cause shallow overlaps in the stable hypernet G_S .

In the second situation, all overlaps are not modified at all by the beta rewrite rule, except for some deep overlaps turned shallow. Consequently, one rewrite transition results in a quasi- $\mathbb{C}_{\mathbb{O}^{\text{ex}}}$ -specimen up to $(=, =)$.

In the first situation, one lambda-abstraction contributed by the pre-template is modified, while all the other shallow overlaps (if any) are not. We can focus on the lambda-abstraction. The beta rewrite acts on the lambda-abstraction, an edge labelled with $\vec{\textcircled{a}}$, and the stable hypernet G_S .

In the case of $\triangleleft^{\text{NE1}}$, application of the beta rule is followed by a few more transitions. When G_S is not an instance edge, we can prove that a token eventually gets stuck, failing to apply a rewrite rule of the equality operation $'=''$. When G_S is simply an instance edge, a token may still get stuck for the same reason, but if there is an applicable rewrite rule of the equality operation $'=''$, we obtain a quasi- $\mathbb{C}_{\mathbb{O}^{\text{ex}}}$ -specimen up to $(\dot{=}_{=\mathbb{N}}, \dot{=}_{=\mathbb{N}})$, using triggers \triangleleft^6 and \triangleleft^7 on the left, and $\triangleleft^{\text{GC}}$ on the right.

The case of $\triangleleft^{\text{NE3}}$ and $\triangleleft^{\text{NE4}}$, and their converse, is similar. The only difference is the triggers used to find a quasi-specimen, as summarised in Table 3.

9 Related and future work

Our Spartan UAM is motivated by a need for a flexible and expressive framework in which a wide variety of effects can be given a cost-accurate model. As discussed, this opens the door to a uniform study of operations and their interactions. Defining new styles of abstract machines is a rich and attractive vein of research. The “monoidal computer” of Pavlovic [2013] or the “evolving algebras” of Gurevich [2018] are such examples. What sets the Spartan UAM apart is the fact that it can be used, rather conveniently, for reasoning robustly about observational equivalence.

It is well known that observational equivalence is fragile, in the sense that the validity of an equation depends not only on the operations involved in the operation itself but also on the operations used in the context. Therefore, the addition of any new operation to a language may invalidate some or all of its equations even if they do not concern the operation itself. Consequently, reasoning about “real-life” languages must be constrained to a syntactically idealised subset which cannot support exotic extrinsics (e.g. `Gc.stat`, discussed earlier) or, more significantly, foreign function calls, which can be seen as a very general, programmable form of extrinsic operation. This is, in some sense, unavoidable.

An even more troubling aspect of observational equivalence is that the proof methods themselves are fragile, in the sense that language additions may violate the very principles on which proofs are constructed. This issue is addressed by Dreyer et al. [2012] by carefully distinguishing between various kinds of operations (state vs. control). More radical approaches are down to replacing the concept of a syntactic context with an “epistemic” context, akin to a Dolev-Yao-style attacker by Ghica and Tzevelekos [2012], or by characterising combinatorially the interaction between a term and its context as is the case with the game semantics of Abramsky et al. [2013], Hyland and Ong [2000] or the trace semantics of Jeffrey and Rathke [2005]. Our paper is a new approach along the same lines, distinguished by the graph-theoretic method and the notion of local reasoning it supports.

We argue that our approach is both flexible and elementary. A specific version of this formalism has been used to prove, for example, the soundness of exotic operations involved in (a functional version of) Google’s TensorFlow machine learning library by Muroya et al. [2018]. Even though the proofs can seem complicated, this is in part due to the graph-based formalism being new, and in part due to the fact that proofs of equivalences are lengthy case-based analyses. But herein lies the simplicity and the robustness of the approach, avoiding the discovery of clever-but-fragile language invariants which can be used to streamline proofs. Our tedious-but-elementary proofs on the other hand seem highly suitable for automation – this is for future work.

The original motivation of the DGoI machine given by Muroya and Ghica [2017] was to produce an abstract machine that expresses the computational intuitions of the GoI while correctly modelling the cost of evaluation, particularly for call-by-value and call-by-need. Although the Universal Abstract Machine framework does not aim at efficiency, one can think of a cost model of the machine in a similar way as the DGoI machine. Moreover, the indexing of observational equivalence with a preorder representing the number of steps gives a direct avenue for modelling and comparing computation costs. For example, the beta law (Tbl. 1) is indexed by the normal order on \mathbb{N} , which indicates that one side always requires fewer steps than the other in the evaluation process. The only details to be resolved are associating costs (time and space) with steps, in particular different costs for different operations.

Another strand of research related to ours is work on defining programming languages with effects. We are not so much interested in “simulated effects”, which are essentially the encoding of effectful behaviour into pure languages, and which can be achieved via monads [Wadler, 1998], but we are interested in genuine “native” effects which happen outside of the language. Semantically this has been introduced by Plotkin and Power [2008] and more recently developed by Møgelberg and Staton [2011]. SPARTAN in some sense takes the idea to the extreme by situating all operations (pure or effectful) outside of the program, in “the world”, and keeping as intrinsic to the language only the structural aspects of copying vs. sharing, and scheduling of computation via thunking.

Nested (hyper)graphs are inspired by the *exponential boxes* of proof nets, a graphical representation of linear logic proofs [Girard, 1987]. Exponential boxes can be formalised by parametrising an agent (which corresponds to an edge in our setting) by a net, as indicated by Lafont [1995]. In the framework of interaction nets [Lafont, 1990] that subsume proof nets, agents can be coordinated

to represent a boundary of a box, as suggested by Mackie [1998]. An alternative representation of boxes that use extra links between agents is proposed by Accattoli and Guerrini [2009]. Our graphical formulation of boxes shares the idea with the first parametrisation approach, but we have flexibility regarding types of a box edge itself and its content (i.e. the hypernet that labels it). We use box edges to represent thunks, and a box edge can have less targets than outputs of its contents, reflecting the number of bound variables a thunk has. This generalised box structure is also studied by Drewes et al. [2002] as *hierarchical graphs*, in the context of double-pushout graph transformation [Rozenberg, 1997], an well-established algebraic approach to graph rewriting. Investigating local reasoning in this context is an important future direction.

Interaction nets are another established framework of graph rewriting, in which various evaluations of pure lambda-calculus can be implemented [Sinot, 2005, 2006]. The idea of having the token to represent an evaluation strategy can be found in *loc. cit.*, which suggests that our focussed rewriting on hypernets could be implemented using interaction nets. However, the local reasoning we are aiming at with focussed rewriting does not seem easy in the setting of interaction nets, because of technical subtleties observed in *loc. cit.*; namely, a status of evaluation is remembered by not only the token but also some other agents around an interaction net.

Hypernets could also perhaps be formalised as string diagrams, with boxes modelled as “functorial boxes” by Melliès [2006]. However, this would mean accommodating some equations on hypernets *built in*. With SPARTAN terms to represent, it is not clear what should be, or can be, such a built-in equation. Our aim is rather to provide a framework in which we can *reason* about, and figure out, the equations. This is also why the use of Frobenius structures to model variables as in Bonchi et al. [2016] is not automatic in the case of *focussed* graph rewriting.

Another future work is the introduction of a more meaningful type system. The type system of SPARTAN is very weak, and we consider it a strength of the approach that equivalences can be proved without the aid of a powerful type infrastructure. On the other hand, in order to avoid stuck configurations and ensure safety of evaluation types are required. The usage of more expressive types is perfectly compatible with SPARTAN, and is something we intend to explore. In particular we would like to study notions of typing which are germane to SPARTAN, capturing its concepts of locality and robustness.

Beyond types, if we look at logics there are some appealing similarities between Spartan and *separation logic* [Reynolds, 2002]. The division of nodes into copying nodes via variables and sharing nodes via atoms is not accidental, and their different contraction properties match those from *bunched implications* [O’Hearn and Pym, 1999]. On a deeper level, the concepts of locality and in particular robustness developed here are related to the “frame” rule of separation logic.

Finally, our formulation of equivalence has some self-imposed limitations needed to limit the complexity of the technical presentation. We are hereby concerned with *sequential* and *deterministic* computation. Future work will show how these restrictions can be relaxed. Parallelism and concurrency can be naturally simulated using multi-token reductions, as inspired by the multi-token GOI machine of Lago et al. [2017], whereas nondeterminism (or probabilistic evaluation) requires no changes to the machinery but rather a new definition of observational equivalence. This is work we are aiming to undertake in the future.

References

- S. Abramsky. Game semantics for programming languages. In *22nd International Symposium on Mathematical Foundations of Computer Science, MFCS 1997, August 25-29 1997, Bratislava, Slovakia*, pages 3–4, 1997. URL <https://doi.org/10.1007/BFb0029944>.
- S. Abramsky, R. Jagadeesan, and P. Malacaria. Full abstraction for PCF. *CoRR*, abs/1311.6125, 2013. URL <http://arxiv.org/abs/1311.6125>.
- B. Accattoli and S. Guerrini. Jumping boxes. In *CSL 2009*, volume 5771 of *Lect. Notes Comp. Sci.*, pages 55–70. Springer, 2009.
- B. Accattoli, P. Barenbaum, and D. Mazza. Distilling abstract machines. In *19th ACM SIGPLAN International Conference on Functional Programming, ICFP 2014, September 1-3 2014, Gothenburg, Sweden*, pages 363–376, 2014. URL <https://doi.org/10.1145/2628136.2628154>.

- A. Ahmed, D. Dreyer, and A. Rossberg. State-dependent representation independence. In *36th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2009, January 21-23 2009, Savannah, GA, USA*, pages 340–353, 2009. URL <https://doi.org/10.1145/1480881.1480925>.
- F. Bonchi, F. Gadducci, A. Kissinger, P. Sobociński, and F. Zanasi. Rewriting modulo symmetric monoidal structure. In *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science, LICS '16, New York, NY, USA, July 5-8, 2016*, pages 710–719, 2016. doi: 10.1145/2933575.2935316. URL <https://doi.org/10.1145/2933575.2935316>.
- F. Bonchi, D. Petrisan, D. Pous, and J. Rot. A general account of coinduction up-to. *Acta Inf.*, 54(2):127–190, 2017. URL <https://doi.org/10.1007/s00236-016-0271-4>.
- B. Coecke and R. Duncan. Interacting quantum observables: categorical algebra and diagrammatics. *New Journal of Physics*, 13(4):043016, 2011. URL <https://doi.org/10.1088/1367-2630/13/4/043016>.
- O. Danvy and A. Filinski. Abstracting control. In *ACM Conference on LISP and Functional Programming, LFP 1990, June 27-29 1990, Nice, France*, pages 151–160, 1990. URL <https://doi.org/10.1145/91556.91622>.
- O. Danvy, K. Millikin, J. Munk, and I. Zerny. On inter-deriving small-step and big-step semantics: A case study for storeless call-by-need evaluation. *Theor. Comput. Sci.*, 435:21–42, 2012. URL <https://doi.org/10.1016/j.tcs.2012.02.023>.
- F. Drewes, B. Hoffmann, and D. Plump. Hierarchical graph transformation. *J. Comput. Syst. Sci.*, 64(2):249–283, 2002. doi: 10.1006/jcss.2001.1790. URL <https://doi.org/10.1006/jcss.2001.1790>.
- D. Dreyer, G. Neis, and L. Birkedal. The impact of higher-order state and control effects on local relational reasoning. *J. Funct. Program.*, 22(4-5):477–528, 2012. URL <https://doi.org/10.1017/S095679681200024X>.
- M. Felleisen and R. Hieb. The revised report on the syntactic theories of sequential control and state. *Theor. Comput. Sci.*, 103(2):235–271, 1992. URL [https://doi.org/10.1016/0304-3975\(92\)90014-7](https://doi.org/10.1016/0304-3975(92)90014-7).
- D. R. Ghica and N. Tzevelekos. A system-level game semantics. *Electr. Notes Theor. Comput. Sci.*, 286:191–211, 2012. URL <https://doi.org/10.1016/j.entcs.2012.08.013>.
- J. Girard. Linear logic. *Theor. Comput. Sci.*, 50:1–102, 1987. URL [https://doi.org/10.1016/0304-3975\(87\)90045-4](https://doi.org/10.1016/0304-3975(87)90045-4).
- Y. Gurevich. Evolving algebras 1993: Lipari guide. *arXiv preprint arXiv:1808.06255*, 2018.
- N. Hoshino, K. Muroya, and I. Hasuo. Memoryful geometry of interaction: from coalgebraic components to algebraic effects. In *Joint Meeting of the 23rd EACSL Annual Conference on Computer Science Logic (CSL) and the 29th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS), CSL-LICS 2014, July 14-18 2014, Vienna, Austria*, pages 52:1–52:10, 2014. URL <https://doi.org/10.1145/2603088.2603124>.
- J. M. E. Hyland and C. L. Ong. On full abstraction for PCF: I, II, and III. *Inf. Comput.*, 163(2): 285–408, 2000. URL <https://doi.org/10.1006/inco.2000.2917>.
- A. Jeffrey and J. Rathke. Java jr: Fully abstract trace semantics for a core java language. In *14th European Symposium on Programming, ESOP 2005, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2005, April 4-8 2005, Edinburgh, UK*, pages 423–438, 2005. URL https://doi.org/10.1007/978-3-540-31987-0_29.

- V. Koutavas and M. Wand. Small bisimulations for reasoning about higher-order imperative programs. In *33rd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2006, January 11-13 2006, Charleston, SC, USA*, pages 141–152, 2006. URL <https://doi.org/10.1145/1111037.1111050>.
- Y. Lafont. Interaction nets. In *17th Annual ACM Symposium on Principles of Programming Languages, POPL 1990, January 1990, San Francisco, California, USA*, pages 95–108, 1990. URL <https://doi.org/10.1145/96709.96718>.
- Y. Lafont. *From proof nets to interaction nets*, page 225248. London Mathematical Society Lecture Note Series. Cambridge University Press, 1995. URL <https://doi.org/10.1017/CB09780511629150.012>.
- U. D. Lago, R. Tanaka, and A. Yoshimizu. The geometry of concurrent interaction: Handling multiple ports by way of multiple tokens. In *32nd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2017, Reykjavik, Iceland, June 20-23, 2017*, pages 1–12, 2017. doi: 10.1109/LICS.2017.8005112. URL <https://doi.org/10.1109/LICS.2017.8005112>.
- I. Mackie. Linear logic *With boxes*. In *13th IEEE Symposium on Logic in Computer Science, June 21-24 1998, Indianapolis, IN, USA*, pages 309–320, 1998. doi: 10.1109/LICS.1998.705667. URL <https://doi.org/10.1109/LICS.1998.705667>.
- P. Mellies. Functorial boxes in string diagrams. In *20th International Workshop on Computer Science Logic, CSL 2006, 15th Annual Conference of the EACSL, September 25-29 2006, Szeged, Hungary*, pages 1–30, 2006. URL https://doi.org/10.1007/11874683_1.
- R. E. Møgelberg and S. Staton. Linearly-used state in models of call-by-value. In *4th International Conference on Algebra and Coalgebra in Computer Science, CALCO 2011, August 30 - September 2 2011, Winchester, UK*, pages 298–313, 2011. URL https://doi.org/10.1007/978-3-642-22944-2_21.
- J. H. Morris Jr. *Lambda-calculus models of programming languages*. PhD thesis, Massachusetts Institute of Technology, 1969. URL <https://doi.org/1721.1/64850>.
- K. Muroya and D. R. Ghica. The dynamic geometry of interaction machine: A call-by-need graph rewriter. In *26th EACSL Annual Conference on Computer Science Logic, CSL 2017, August 20-24 2017, Stockholm, Sweden*, pages 32:1–32:15, 2017. URL <https://doi.org/10.4230/LIPIcs.CSL.2017.32>.
- K. Muroya, N. Hoshino, and I. Hasuo. Memoryful geometry of interaction II: recursion and adequacy. In *Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2016, St. Petersburg, FL, USA, January 20-22 2016*, pages 748–760, 2016. URL <https://doi.org/10.1145/2837614.2837672>.
- K. Muroya, S. W. T. Cheung, and D. R. Ghica. The geometry of computation-graph abstraction. In *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2018, Oxford, UK, July 09-12 2018*, pages 749–758, 2018. URL <https://doi.org/10.1145/3209108.3209127>.
- P. O’Hearn and R. Tennent. *ALGOL-like Languages*. Springer Science & Business Media, 2013. URL doi.org/10.1007/978-1-4612-4118-8.
- P. W. O’Hearn and D. J. Pym. The logic of bunched implications. *Bulletin of Symbolic Logic*, 5(2):215–244, 1999.
- D. Pavlovic. Monoidal computer i: Basic computability by string diagrams. *Information and Computation*, 226:94–116, 2013.
- A. M. Pitts and I. D. B. Stark. Observable properties of higher order functions that dynamically create local names, or what’s new? In *18th International Symposium on Mathematical Foundations of Computer Science 1993, MFCS 1993, August 30 - September 3 1993, Gdansk, Poland*, pages 122–141, 1993. URL https://doi.org/10.1007/3-540-57182-5_8.

- G. D. Plotkin and J. Power. Algebraic operations and generic effects. *Applied Categorical Structures*, 11(1):69–94, 2003. URL <https://doi.org/10.1023/A:1023064908962>.
- G. D. Plotkin and J. Power. Tensors of comodels and models for operational semantics. *Electr. Notes Theor. Comput. Sci.*, 218:295–311, 2008. doi: 10.1016/j.entcs.2008.10.018. URL <https://doi.org/10.1016/j.entcs.2008.10.018>.
- G. D. Plotkin and M. Pretnar. Handling algebraic effects. *Logical Methods in Computer Science*, 9(4), 2013. URL [https://doi.org/10.2168/LMCS-9\(4:23\)2013](https://doi.org/10.2168/LMCS-9(4:23)2013).
- J. C. Reynolds. Separation logic: A logic for shared mutable data structures. In *Proceedings 17th Annual IEEE Symposium on Logic in Computer Science*, pages 55–74. IEEE, 2002.
- G. Rozenberg, editor. *Handbook of graph grammars and computing by graph transformations, volume 1: foundations*. World Scientific, 1997. ISBN 9810228848.
- D. S. Scott and C. Strachey. *Toward a mathematical semantics for computer languages*, volume 1. Oxford University Computing Laboratory, Programming Research Group, 1971.
- F. Sinot. Call-by-name and call-by-value as token-passing interaction nets. In *7th International Conference on Typed Lambda Calculi and Applications, TLCA 2005, April 21-23 2005, Nara, Japan*, pages 386–400, 2005. URL https://doi.org/10.1007/11417170_28.
- F. Sinot. Call-by-need in token-passing nets. *Mathematical Structures in Computer Science*, 16(4): 639–666, 2006. URL <https://doi.org/10.1017/S0960129506005408>.
- R. Statman. Logical relations and the typed lambda-calculus. *Information and Control*, 65(2/3): 85–97, 1985. URL [https://doi.org/10.1016/S0019-9958\(85\)80001-2](https://doi.org/10.1016/S0019-9958(85)80001-2).
- P. Wadler. The marriage of effects and monads. In *ACM SIGPLAN Notices*, volume 34:1, pages 63–74. ACM, 1998. URL <https://doi.org/10.1145/289423.289429>.
- A. K. Wright and M. Felleisen. A syntactic approach to type soundness. *Inf. Comput.*, 115(1): 38–94, 1994. URL <https://doi.org/10.1006/inco.1994.1093>.

A Technical appendix

A.1 Equivalent definitions of hypernets

Informally, hypernets are nested hypergraphs, and one hypernet can contain nested hypergraphs up to different depths. This intuition is reflected by Def. 3.5 of hypernets, in particular the big union in $\mathcal{H}_{k+1}(L, M) = \mathcal{H}\left(L, M \cup \bigcup_{i \leq k} \mathcal{H}_i(L, M)\right)$. In fact, the definition can be replaced by a simpler, but possibly less intuitive, definition below that does not explicitly deal with the different depths of nesting.

Definition A.1. Given sets L and M , a set $\mathcal{H}'_k(L, M)$ is defined by induction on $k \in \mathbb{N}$:

$$\begin{aligned}\mathcal{H}'_0(L, M) &:= \mathcal{H}(L, M) \\ \mathcal{H}'_{k+1}(L, M) &:= \mathcal{H}\left(L, M \cup \mathcal{H}'_k(L, M)\right)\end{aligned}$$

and hence a set $\mathcal{H}'_\omega(L, M) := \bigcup_{i \in \mathbb{N}} \mathcal{H}'_i(L, M)$.

Lemma A.2. *Given arbitrary sets L and M , any two numbers $k, k' \in \mathbb{N}$ satisfy $\mathcal{H}'_k(L, M) \subseteq \mathcal{H}'_{k+k'}(L, M)$.*

Proof. If $k' = 0$, the inclusion trivially holds. If not, i.e. $k' > 0$, it can be proved by induction on $k \in \mathbb{N}$. The key reasoning principle we use is that $M \subseteq M'$ implies $\mathcal{H}(L, M) \subseteq \mathcal{H}(L, M')$.

In the base case, when $k = 0$ (and $k' > 0$), we have

$$\begin{aligned}\mathcal{H}'_0(L, M) &= \mathcal{H}(L, M) \\ &\subseteq \mathcal{H}\left(L, M \cup \mathcal{H}'_{k'-1}(L, M)\right) = \mathcal{H}'_{k'}(L, M).\end{aligned}$$

In the inductive case, when $k > 0$ (and $k' > 0$), we have

$$\begin{aligned}\mathcal{H}'_k(L, M) &= \mathcal{H}\left(L, M \cup \mathcal{H}'_{k-1}(L, M)\right) \\ &\subseteq \mathcal{H}\left(L, M \cup \mathcal{H}'_{k-1+k'}(L, M)\right) = \mathcal{H}'_{k+k'}(L, M)\end{aligned}$$

where the inclusion is by induction hypothesis on $k - 1$. □

Proposition A.3. *Any sets L and M satisfy $\mathcal{H}_k(L, M) = \mathcal{H}'_k(L, M)$ for any $k \in \mathbb{N}$, and hence $\mathcal{H}_\omega(L, M) = \mathcal{H}'_\omega(L, M)$.*

Proof. We first prove $\mathcal{H}_k(L, M) \subseteq \mathcal{H}'_k(L, M)$ by induction on $k \in \mathbb{N}$. The base case, when $k = 0$, is trivial. In the inductive case, when $k > 0$, we have

$$\begin{aligned}\mathcal{H}_k(L, M) &= \mathcal{H}\left(L, M \cup \bigcup_{i \leq k-1} \mathcal{H}_i(L, M)\right) \\ &\subseteq \mathcal{H}\left(L, M \cup \bigcup_{i \leq k-1} \mathcal{H}'_i(L, M)\right) && \text{(by I.H.)} \\ &= \mathcal{H}\left(L, M \cup \mathcal{H}'_{k-1}(L, M)\right) && \text{(by Lem. A.2)} \\ &= \mathcal{H}'_k(L, M).\end{aligned}$$

The other direction, i.e. $\mathcal{H}'_k(L, M) \subseteq \mathcal{H}_k(L, M)$, can be also proved by induction on $k \in \mathbb{N}$. The base case, when $k = 0$, is again trivial. In the inductive case, we have

$$\begin{aligned}\mathcal{H}'_k(L, M) &= \mathcal{H}\left(L, M \cup \mathcal{H}'_{k-1}(L, M)\right) \\ &\subseteq \mathcal{H}\left(L, M \cup \mathcal{H}_{k-1}(L, M)\right) && \text{(by I.H.)} \\ &\subseteq \mathcal{H}\left(L, M \cup \bigcup_{i \leq k-1} \mathcal{H}_i(L, M)\right) \\ &= \mathcal{H}_k(L, M).\end{aligned}$$

□

Given a hypernet G , by Lem. A.2 and Prop. A.3, there exists a minimum number k such that $G \in \mathcal{H}'_k(L, M)$, which we call the “minimum level” of G .

Lemma A.4. *Any hypernet has a finite number of shallow edges, and a finite number of deep edges.*

Proof. Any hypernet has a finite number of shallow edges by definition. We prove that any hypernet G has a finite number of deep edges, by induction on minimum level k of the hypernet.

When $k = 0$, the hypernet has no deep edges.

When $k > 0$, each hypernet H that labels a shallow edge of G belongs to $\mathcal{H}'_{k-1}(L, M)$, and therefore its minimum level is less than k . By induction hypothesis, the labelling hypernet H has a finite number of deep edges, and also a finite number of shallow edges. Deep edges of G are given by edges, at any depth, of any hypernet that labels a shallow edge of G . Because there is a finite number of the hypernets that label the shallow edges of G , the number of deep edges of G is finite. \square

A.2 Plugging

An interfaced labelled monoidal hypergraph can be given by data of the following form: $((V \uplus I \uplus O, E), (S, T), (f_L, f_M))$ where I is the input list, O is the output list, V is the set of all the other vertices, E is the set of edges, (S, T) defines source and target lists, and (f_L, f_M) is labelling functions.

Definition A.5 (Plugging). Let $\mathcal{C}[\vec{\chi}^1, \chi, \vec{\chi}^2] = ((V \uplus I \uplus O, E), (S, T), (f_L, f_M))$ and $\mathcal{C}'[\vec{\chi}^3] = ((V' \uplus I' \uplus O', E'), (S', T'), (f'_L, f'_M))$ be contexts, such that the hole χ and the latter context \mathcal{C}' have the same type and $\vec{\chi}^1 \cap \vec{\chi}^2 \cap \vec{\chi}^3 = \emptyset$. The *plugging* $\mathcal{C}[\vec{\chi}^1, \mathcal{C}', \vec{\chi}^2]$ is a hypernet given by data $((\hat{V}, \hat{E}), (\hat{S}, \hat{T}), (\hat{f}_L, \hat{f}_M))$ such that:

$$\begin{aligned}\hat{V} &= V \uplus V' \uplus I \uplus O \\ \hat{E} &= (E \setminus \{e_\chi\}) \uplus E' \\ \hat{S}(e) &= \begin{cases} S(e) & (\text{if } e \in E \setminus \{e_\chi\}) \\ g^*(S'(e)) & (\text{if } e \in E') \end{cases} \\ \hat{T}(e) &= \begin{cases} T(e) & (\text{if } e \in E \setminus \{e_\chi\}) \\ g^*(T'(e)) & (\text{if } e \in E') \end{cases} \\ g(v) &= \begin{cases} v & (\text{if } v \in V') \\ (S(e_\chi))_i & (\text{if } v = (I')_i) \\ (T(e_\chi))_i & (\text{if } v = (O')_i) \end{cases} \\ \hat{f}_L(v) &= \begin{cases} f_L(v) & (\text{if } v \in V) \\ f'_L(v) & (\text{if } v \in V') \end{cases} \\ \hat{f}_M(e) &= \begin{cases} f_M(e) & (\text{if } e \in E \setminus \{e_\chi\}) \\ f'_M(e) & (\text{if } e \in E') \end{cases}\end{aligned}$$

where $e_\chi \in E$ is the hole edge labelled with χ , and $(-)_i$ denotes the i -th element of a list.

In the resulting context $\mathcal{C}[\vec{\chi}^1, \mathcal{C}', \vec{\chi}^2]$, each edge comes from either \mathcal{C} or \mathcal{C}' . If a path in \mathcal{C} does not contain the hole edge e_χ , the path gives a path in $\mathcal{C}[\vec{\chi}^1, \mathcal{C}', \vec{\chi}^2]$. Conversely, if a path in $\mathcal{C}[\vec{\chi}^1, \mathcal{C}', \vec{\chi}^2]$ consists of edges from \mathcal{C} only, the path gives a path in \mathcal{C} .

Any path in \mathcal{C}' gives a path in $\mathcal{C}[\vec{\chi}^1, \mathcal{C}', \vec{\chi}^2]$. However, if a path in $\mathcal{C}[\vec{\chi}^1, \mathcal{C}', \vec{\chi}^2]$ consists of edges from \mathcal{C}' only, the path does not necessarily give a path in \mathcal{C}' . The path indeed gives a path in \mathcal{C}' , if sources and targets of the hole edge e_χ are distinct in \mathcal{C} (i.e. the hole edge e_χ is not a self-loop).

A.3 Rooted states

Lemma A.6. *Let (X, \rightarrow) is an abstract rewriting system that is deterministic.*

1. *For any $x, y, y' \in X$ such that y and y' are normal forms, and for any $k, h \in \mathbb{N}$, if there exist two sequences $x \rightarrow^k y$ and $x \rightarrow^h y'$, then these sequences are exactly the same.*
2. *For any $x, y \in X$ such that y is a normal form, and for any $i, j, k \in \mathbb{N}$ such that $i \neq j$ and $i, j \in \{1, \dots, k\}$, if there exists a sequence $x \rightarrow^k y$, then its i -th rewrite $z \rightarrow z'$ and j -th rewrite $w \rightarrow w'$ satisfy $z \neq w$.*

Proof. The point (1) is proved by induction on $k + h \in \mathbb{N}$. In the base case, when $k + h = 0$ (i.e. $k = h = 0$), the two sequences are both the empty sequence, and $x = y = y'$. The inductive case, when $k + h > 0$, falls into one of the following two situations. The first situation, where $k = 0$ or $h = 0$, boils down to the base case, because x must be a normal form itself, which means $k = h = 0$. In the second situation, where $k > 0$ and $h > 0$, there exist elements $z, z' \in X$ such that $x \rightarrow z \rightarrow^{k-1} y$ and $x \rightarrow z' \rightarrow^{h-1} y'$. Because \rightarrow is deterministic, $z = z'$ follows, and hence by induction hypothesis on $(k-1) + (h-1)$, these two sequences are the same.

The point (2) is proved by contradiction. The sequence $x \rightarrow^k y$ from x to the normal form y is unique, by the point (1). If its i -th rewrite $z \rightarrow z'$ and j -th rewrite $w \rightarrow w'$ satisfy $z = w$, determinism of the system implies that these two rewrites are the same. This means that the sequence $x \rightarrow^k y$ has a cyclic sub-sequence, and by repeating the cycle different times, one can yield different sequences of rewrites $x \rightarrow^* y$ from x to y . This contradicts the uniqueness of the original sequence $x \rightarrow^k y$. \square

Lemma A.7. *If a state \dot{G} is rooted, a search sequence $?; |\dot{G}| \rightarrow^* \dot{G}$ from the initial state $?; |\dot{G}|$ to the state \dot{G} is unique. Moreover, for any i -th search transition and j -th search transition in the sequence such that $i \neq j$, these transitions do not result in the same state.*

Proof. Let X be the set of states with search or value token. We can define an abstract rewriting system (X, \rightarrow) of “reverse search” by: $\dot{H} \rightarrow \dot{H}'$ if $\dot{H}' \rightarrow \dot{H}$. Any search sequence corresponds to a sequence of rewrites in this rewriting system.

The rewriting system is deterministic, i.e. if $\dot{H}' \rightarrow \dot{H}$ and $\dot{H}'' \rightarrow \dot{H}$ then $\dot{H}' = \dot{H}''$, because the inverse \mapsto^{-1} of the interaction rules (Fig. 3) is deterministic.

If a search transition changes a token to a search token, the resulting search token always has an incoming operation edge. This means that, in the rewriting system (X, \rightarrow) , initial states are normal forms. Therefore, by Lem. A.6(1), if there exist two search sequences from the initial state $?; |\dot{G}|$ to the state \dot{G} , these search sequences are exactly the same. The rest is a consequence of Lem. A.6(2). \square

Lemma A.8. *For any hypernet N , if there exists an operation path from an input to a vertex, the path is unique. Moreover, no edge appears twice in the operation path.*

Proof. Given the hypernet N whose set of (shallow) vertices is X , we can define an abstract rewriting system (X, \rightarrow) of “reverse connection” by: $v \rightarrow v'$ if there exists an operation edge whose unique source is v' and targets include v . Any operation path from an input to a vertex in N corresponds to a sequence of rewrites in this rewriting system.

This rewriting system is deterministic, because each vertex can have at most one incoming edge in a hypergraph (Def. 3.2) and each operation edge has exactly one source. Because inputs of the hypernet N have no incoming edges, they are normal forms in this rewriting system. Therefore, by Lem. A.6(1), an operation path from any input to any vertex is unique.

The rest is proved by contradiction. We assume that, in an operation path P from an input to a vertex, the same operation edge e appears twice. The edge e has one source, which either is an input of the hypernet N or has an incoming edge. In the former case, the edge e can only appear as the first edge of the operation path P , which is a contradiction. In the latter case, the operation edge e has exactly one incoming edge e' in the hypernet N . In the operation path P , each appearance of the operation edge e must be preceded by this edge e' via the same vertex. This contradicts Lem. A.6(2). \square

Lemma A.9. *For any rooted state \dot{G} , if its token source (i.e. the source of the token) does not coincide with the unique input, then there exists an operation path from the input to the token source.*

Proof. By Lem. A.7, the rooted state \dot{G} has a unique search sequence $?; |\dot{G}| \xrightarrow{*} \dot{G}$. The proof is by the length k of this sequence.

In the base case, where $k = 0$, the state \dot{G} itself is an initial state, which means the input and token source coincide in \dot{G} .

In the inductive case, where $k > 0$, there exists a state \dot{G}' such that $?; |\dot{G}| \xrightarrow{k-1} \dot{G}' \xrightarrow{} \dot{G}$. The proof here is by case analysis on the interaction rule used in $\dot{G}' \xrightarrow{} \dot{G}$.

- When the interaction rules (1a), (1b), (2) or (5b) is used (see Fig. 3), the transition $\dot{G}' \xrightarrow{} \dot{G}$ only changes a token label.
- When the interaction rule (3) is used, the transition $\dot{G}' \xrightarrow{} \dot{G}$ turns the token and its outgoing operation edge $e_{G'}$ into an operation edge e_G and its outgoing token. By induction hypothesis on \dot{G}' , the token source coincides with its input, or there exists an operation path from the input to the token source, in \dot{G}' .

In the former case, in \dot{G} , the source of the operation edge e_G coincides with the input. The edge e_G itself gives the desired operation path in \dot{G} .

In the latter case, the operation path $P_{G'}$ from the input to the token source in \dot{G}' does not contain the outgoing operation edge $e_{G'}$ of the token; otherwise, the edge $e_{G'}$ must be preceded by the token edge in the operation path $P_{G'}$, which is a contradiction. Therefore, the operation path $P_{G'}$ in \dot{G}' is inherited in \dot{G} , becoming a path P_G from the input to the source of the incoming operation edge e_G of the token. In the state \dot{G} , the path P_G followed by the edge e_G yields the desired operation path.

- When the interaction rule (4) is used, the transition $\dot{G}' \xrightarrow{} \dot{G}$ changes the token from a $(k+1)$ -th outgoing edge of an operation edge e to a $(k+2)$ -th outgoing edge of the same operation edge e , for some $k \in \mathbb{N}$. In \dot{G}' , the token source is not an input, and therefore, there exists an operation path $P_{G'}$ from the input to the token source, by induction hypothesis.

The operation path $P_{G'}$ ends with the operation edge e , and no outgoing edge of the edge e is involved in the path $P_{G'}$; otherwise, the edge e must appear more than once in the path $P_{G'}$, which is a contradiction by Lem. A.8. Therefore, the path $P_{G'}$ is inherited exactly as it is in \dot{G} , and it gives the desired operation path.

- When the interaction rule (5a) is used, by the same reasoning as in the case of rule (4), \dot{G}' has an operation path $P_{G'}$ from the input to the token source, where the incoming operation edge $e_{G'}$ of the token appears exactly once, at the end. Removing the edge $e_{G'}$ from the path $P_{G'}$ yields another operation path P from the input in \dot{G}' , and it also gives an operation path from the input to the token source in \dot{G} .

□

Lemma A.10. *For any state \dot{G} with a \mathbf{t} -token such that $\mathbf{t} \neq ?$, if \dot{G} is rooted, then there exists a search sequence $?; |\dot{G}| \xrightarrow{*} \langle \dot{G} \rangle_{?/\mathbf{t}} \xrightarrow{+} \dot{G}$.*

Proof. By Lem. A.7, the rooted state \dot{G} has a unique search sequence $?; |\dot{G}| \xrightarrow{*} \dot{G}$. The proof is to show that a transition from the state $\langle \dot{G} \rangle_{?/\mathbf{t}}$ appears in this search sequence, and it is by the length k of the search sequence.

Because \dot{G} does not have a search token, $k = 0$ is impossible, and therefore the base case is when $k = 1$. The search transition $?; |\dot{G}| \xrightarrow{} \dot{G}$ must use one of the interaction rules (1a), (1b), (2) and (5b). This means $?; |\dot{G}| = \langle \dot{G} \rangle_{?/\mathbf{t}}$.

In the inductive case, where $k > 0$, there exists a state \dot{G}' such that $?; |\dot{G}| \xrightarrow{k-1} \dot{G}' \xrightarrow{} \dot{G}$. The proof here is by case analysis on the interaction rule used in $\dot{G}' \xrightarrow{} \dot{G}$.

- When the interaction rule (1a), (1b), (2) or (5b) is used, $?; |\dot{G}| = \langle \dot{G} \rangle_{?/\mathbf{t}}$.

- Because \dot{G} does not have a search token, the interaction rules (3) and (4) can be never used in $\dot{G}' \xrightarrow{\bullet} \dot{G}$.
- When the interaction rule (5a) is used, \dot{G}' has a value token, which is a $(k+1)$ -th outgoing edge of an operation edge e , for some $k \in \mathbb{N}$. The operation edge e becomes the outgoing edge of the token in \dot{G} . By induction hypothesis on \dot{G}' , we have

$$?; |\dot{G}| \xrightarrow{\bullet^*} \langle \dot{G}' \rangle_{\text{?}/\checkmark} \xrightarrow{\bullet^+} \dot{G}' \xrightarrow{\bullet} \dot{G}. \quad (\text{A})$$

If $k = 0$, in \dot{G}' , the token is the only outgoing edge of the operation edge e . Because $\langle \dot{G}' \rangle_{\text{?}/\checkmark}$ is not an initial state, it must be a result of the interaction rule (3), which means the search sequence (A) is factored through as:

$$?; |\dot{G}| \xrightarrow{\bullet^*} \langle \dot{G} \rangle_{\text{?}/\text{t}} \xrightarrow{\bullet} \langle \dot{G}' \rangle_{\text{?}/\checkmark} \xrightarrow{\bullet^+} \dot{G}' \xrightarrow{\bullet} \dot{G}.$$

If $k > 0$, for each $m \in \{0, \dots, k\}$, let \dot{N}_m be a state with a search token, such that $|\dot{N}_m| = |\dot{G}'|$ and the token is an $(m+1)$ -th outgoing edge of the operation edge e . This means $\dot{N}_k = \langle \dot{G}' \rangle_{\text{?}/\checkmark}$. The proof concludes by combining the following internal lemma with (A), taking k as m .

Lemma A.11. *For any $m \in \{0, \dots, k\}$, if there exists $h < k$ such that $?; |\dot{G}| \xrightarrow{\bullet^h} \dot{N}_m$, then it is factored through as $?; |\dot{G}| \xrightarrow{\bullet^*} \langle \dot{G} \rangle_{\text{?}/\text{t}} \xrightarrow{\bullet^+} \dot{N}_m$.*

Proof. By induction on m . In the base case, when $m = 0$, the token of \dot{N}_m is the first outgoing edge of the operation edge e . This state is not initial, and therefore must be a result of the interaction rule (3), which means

$$?; |\dot{G}| \xrightarrow{\bullet^*} \langle \dot{G} \rangle_{\text{?}/\text{t}} \xrightarrow{\bullet} \dot{N}_m.$$

In the inductive case, when $m > 0$, the state \dot{N}_m is not an initial state and must be a result of the interaction rule (4), which means

$$?; |\dot{G}| \xrightarrow{\bullet^*} \langle \dot{N}_{m-1} \rangle_{\checkmark/\text{?}} \xrightarrow{\bullet} \dot{N}_m.$$

The first half of this search sequence, namely $?; |\dot{G}| \xrightarrow{\bullet^*} \langle \dot{N}_{m-1} \rangle_{\checkmark/\text{?}}$, consists of $h-1 < k$ transitions. Therefore, by (outer) induction hypothesis on $h-1$, we have

$$?; |\dot{G}| \xrightarrow{\bullet^*} \dot{N}_{m-1} \xrightarrow{\bullet^+} \langle \dot{N}_{m-1} \rangle_{\checkmark/\text{?}} \xrightarrow{\bullet} \dot{N}_m.$$

The first part, namely $?; |\dot{G}| \xrightarrow{\bullet^*} \dot{N}_{m-1}$, consists of less than k transitions. Therefore, by (inner) induction hypothesis on $m-1$, we have

$$?; |\dot{G}| \xrightarrow{\bullet^*} \langle \dot{G} \rangle_{\text{?}/\text{t}} \xrightarrow{\bullet^+} \dot{N}_{m-1} \xrightarrow{\bullet^+} \langle \dot{N}_{m-1} \rangle_{\checkmark/\text{?}} \xrightarrow{\bullet} \dot{N}_m.$$

□

□

Lemma A.12.

1. For any state \dot{N} , if it has a path to the token source that is not an operation path, then it is not rooted.
2. For any focus-free hypernet H and any focussed context $\dot{C}[\chi]$ with one hole edge, such that $\dot{C}[H]$ is a state, if the hypernet H is one-way and the context \dot{C} has a path to the token source that is not an operation path, then the state $\dot{C}[H]$ is not rooted.

3. For any \mathbb{C} -specimen $(\dot{C}[\vec{\chi}]; \vec{G}; \vec{H})$ of an output-closed pre-template \triangleleft , if the context $\dot{C}[\vec{\chi}]$ has a path to the token source that is not an operation path, then at least one of the states $\dot{C}[\vec{G}]$ and $\dot{C}[\vec{H}]$ is not rooted.

Proof of the point (1). Let P be the path in \dot{N} to the token source that is not an operation path. The proof is by contradiction; we assume that \dot{N} is a rooted state.

Because of P , the token source is not an input. Therefore by Lem. A.9, the state \dot{N} has an operation path from its unique input to the token source. This operation path contradicts the path P , which is not an operation path, because each operation edge has only one source and each vertex has at most one incoming edge. \square

Proof of the point (2). Let P be the path in \dot{C} to the token source that is not an operation path.

If the path P contains no hole edge, it gives a path in the state $\dot{C}[H]$ to the token source that is not an operation path. By the point (1), the state is not rooted.

Otherwise, i.e. if the path P contains a hole edge, we give a proof by contradiction; we assume that the state $\dot{C}[H]$ is rooted. We can take a suffix of the path P , so that it gives a path from a target of a hole edge to the token source in \dot{C} , and moreover, gives a path P' from a source of an edge from H to the token source in $\dot{C}[H]$. This implies the token source is not an input, and therefore by Lem. A.9, the state $\dot{C}[H]$ has an operation path from its unique input to the token source. This operation path must have P' as a suffix, meaning P' is also an operation path, because each operation edge has only one source and each vertex has at most one incoming edge. Moreover, H must have an operation path from an input to an output, such that the input and the output have type \star and the path ends with the first edge of the path P' . This contradicts H being one-way. \square

Proof of the point (3). Let P be the path in \dot{C} to the token source that is not an operation path.

If the path P contains no hole edge, it gives a path in the states $\dot{C}[\vec{G}]$ and $\dot{C}[\vec{H}]$ to the token source that is not an operation path. By the point (1), the states are not rooted.

Otherwise, i.e. if the path P contains a hole edge, we can take a suffix of P that gives a path P' from a source of a hole edge e to the token source in \dot{C} , so that the path P' does not contain any hole edge. We can assume that the hole edge e is labelled with χ_1 , without loss of generality. The path P' gives paths P'_G and P'_H to the token source, in contexts $\dot{C}[\chi_1, \vec{G} \setminus \{G_1\}]$ and $\dot{C}[\chi_1, \vec{H} \setminus \{H_1\}]$, respectively. The paths P'_G and P'_H are not an operation path, because they start with the hole edge e labelled with χ_1 .

Because \triangleleft is output-closed, G_1 or H_1 is one-way. By the point (2), at least one of the states $\dot{C}[\vec{G}]$ and $\dot{C}[\vec{H}]$ is not rooted. \square

Lemma A.13. *If a rewrite transition $\dot{G} \rightarrow \dot{G}'$ is stationary, it preserves the rooted property, i.e. \dot{G} being rooted implies \dot{G}' is also rooted.*

Proof. The stationary rewrite transition $\dot{G} \rightarrow \dot{G}'$ is in the form of $\mathcal{C}[\downarrow;_i H] \rightarrow \mathcal{C}[\downarrow;_i H']$, where \mathcal{C} is a focus-free simple context, H is a focus-free one-way hypernet, H' is a focus-free hypernet and $i \in \mathbb{N}$. We assume $\mathcal{C}[\downarrow;_i H]$ is rooted, and prove that $\mathcal{C}[\downarrow;_i H']$ is rooted, i.e. $?\mathcal{C}[H'] \xrightarrow{*} \mathcal{C}[\downarrow;_i H']$. By Lem. A.10, there exists a number $k \in \mathbb{N}$ such that:

$$?\mathcal{C}[H] \xrightarrow{\bullet^k} \mathcal{C}[\downarrow;_i H] \xrightarrow{\bullet^+} \mathcal{C}[\downarrow;_i H].$$

The rest of the proof is by case analysis on the number k .

- When $k = 0$, i.e. $?\mathcal{C}[H] = \mathcal{C}[\downarrow;_i H]$, the unique input and the i -th source of the hole coincide in the simple context \mathcal{C} . Therefore, $?\mathcal{C}[H'] = \mathcal{C}[\downarrow;_i H']$, which means $\mathcal{C}[\downarrow;_i H']$ is rooted.
- When $k > 0$, there exists a state \dot{N} such that $?\mathcal{C}[H] \xrightarrow{\bullet^{k-1}} \dot{N} \xrightarrow{\bullet} \mathcal{C}[\downarrow;_i H]$. By the following internal lemma (Lem. A.14), there exists a focussed simple context \mathcal{C}_N , whose token is not entering nor exiting, and we have two search sequences:

$$\begin{aligned} ?\mathcal{C}[H] &\xrightarrow{\bullet^{k-1}} \dot{\mathcal{C}}_N[H] \xrightarrow{\bullet} \mathcal{C}[\downarrow;_i H], \\ ?\mathcal{C}[H'] &\xrightarrow{\bullet^{k-1}} \dot{\mathcal{C}}_N[H']. \end{aligned}$$

The last search transition $\dot{\mathcal{C}}_N[H] \xrightarrow{\bullet} \mathcal{C}[?;_i H]$, which yields a search token, must use the interaction rule (3) or (4). Because the token is not entering nor exiting in the simple context $\dot{\mathcal{C}}_N$, either of the two interaction rules acts on the token and an edge of the context. This means that the same interaction is possible in the state $\dot{\mathcal{C}}_N[H']$, yielding:

$$?; \mathcal{C}[H'] \xrightarrow{\bullet}^{k-1} \dot{\mathcal{C}}_N[H'] \xrightarrow{\bullet} \mathcal{C}[?;_i H'],$$

which means $\mathcal{C}[?;_i H']$ is rooted.

Lemma A.14. *For any $m \in \{0, \dots, k-1\}$ and any state \dot{N} such that $?; \mathcal{C}[H] \xrightarrow{\bullet}^m \dot{N} \xrightarrow{\bullet}^{k-m} \mathcal{C}[?;_i H]$, the following holds.*

(A) *If there exists a focussed simple context $\dot{\mathcal{C}}_N$ such that $\dot{N} = \dot{\mathcal{C}}_N[H]$, the token of the context $\dot{\mathcal{C}}_N$ is not entering.*

(B) *If there exists a focussed simple context $\dot{\mathcal{C}}_N$ such that $\dot{N} = \dot{\mathcal{C}}_N[H]$, the token of the context $\dot{\mathcal{C}}_N$ is not exiting.*

(C) *There exists a focussed simple context $\dot{\mathcal{C}}_N$ such that $\dot{N} = \dot{\mathcal{C}}_N[H]$, and $?; \mathcal{C}[H'] \xrightarrow{\bullet}^m \dot{\mathcal{C}}_N[H']$ holds.*

Proof. Firstly, because search transitions do not change an underlying hypernet, if there exists a focussed simple context $\dot{\mathcal{C}}_N$ such that $\dot{N} = \dot{\mathcal{C}}_N[H]$, $|\dot{\mathcal{C}}_N| = \mathcal{C}$ necessarily holds.

The point (A) is proved by contradiction; we assume that the context $\dot{\mathcal{C}}_N$ has an entering token. This means that there exist a number $p \in \mathbb{N}$ and a token label $t \in \{?, \checkmark, \cancel{\checkmark}\}$ such that $\dot{\mathcal{C}}_N = \mathcal{C}[t;_p H]$. By Lem. A.10, there exists a number h such that $h \leq m$ and:

$$?; \mathcal{C}[H] \xrightarrow{\bullet}^h \mathcal{C}[?;_p H] \xrightarrow{\bullet}^{k-h} \mathcal{C}[?;_i H]. \quad (\$)$$

We derive a contradiction by case analysis on the numbers p and h .

- If $p = i$ and $h = 0$, the state $\mathcal{C}[?;_i H]$ must be initial, but it is a result of a search transition because $k - h > 0$. This is a contradiction.
- If $p = i$ and $h > 0$, two different transitions in the search sequence (\$) result in the same state, because of $h > 0$ and $k - h > 0$, which contradicts Lem. A.7.
- If $p \neq i$, by Def. 6.2, there exists a state \dot{N}' with a rewrite token such that $\mathcal{C}[?;_p H] \xrightarrow{\bullet} \dot{N}'$. This contradicts the search sequence (\$), because $k - h > 0$ and search transitions are deterministic.

The point (B) follows from the contraposition of Lem. A.12(2), because H is one-way and \dot{N} is rooted. The rooted property of \dot{N} follows from the fact that search transitions do not change underlying hypernets.

The point (C) is proved by induction on $m \in \{0, \dots, k-1\}$. In the base case, when $m = 0$, we have $?; \mathcal{C}[H] = \dot{N}$, and therefore the context $?; \mathcal{C}$ can be taken as $\dot{\mathcal{C}}_N$. This means $?; \mathcal{C}[H'] = \dot{\mathcal{C}}_N[H']$.

In the inductive case, when $m > 0$, there exists a state \dot{N}' such that

$$?; \mathcal{C}[H] \xrightarrow{\bullet}^{m-1} \dot{N}' \xrightarrow{\bullet} \dot{N} \xrightarrow{\bullet}^{k-m} \mathcal{C}[?;_i H].$$

By the induction hypothesis, there exists a focussed simple context $\dot{\mathcal{C}}_{N'}$ such that $\dot{N}' = \dot{\mathcal{C}}_{N'}[H]$ and

$$\begin{aligned} ?; \mathcal{C}[H] &\xrightarrow{\bullet}^{m-1} \dot{\mathcal{C}}_{N'}[H] \xrightarrow{\bullet} \dot{N} \xrightarrow{\bullet}^{k-m} \mathcal{C}[?;_i H], \\ ?; \mathcal{C}[H'] &\xrightarrow{\bullet}^{m-1} \dot{\mathcal{C}}_{N'}[H']. \end{aligned}$$

Our goal here is to find a focussed simple context $\dot{\mathcal{C}}_N$, such that $\dot{N} = \dot{\mathcal{C}}_N[H]$ and $\dot{\mathcal{C}}_{N'}[H'] \xrightarrow{\bullet} \dot{\mathcal{C}}_N[H']$.

In the search transition $\dot{\mathcal{C}}_{N'}[H] \xrightarrow{\bullet} \dot{N}$, the only change happens to the token and its incoming or outgoing edge e in the state $\dot{\mathcal{C}}_{N'}[H]$. By the points (A) and (B), the token is not entering nor exiting in the context $\dot{\mathcal{C}}_{N'}$, which means the edge e must be from the context, not from H .

Now that no edge from H is changed in $\dot{\mathcal{C}}_{N'}[H] \xrightarrow{\bullet} \dot{N}$, there exists a focussed simple context $\dot{\mathcal{C}}_N$ such that $\dot{N} = \dot{\mathcal{C}}_N[H]$, and moreover, $\dot{\mathcal{C}}_{N'}[H'] \xrightarrow{\bullet} \dot{\mathcal{C}}_N[H']$. \square

\square

A.4 Accessible paths and stable hypernets

A stable hypernet always has at least one edge, and any non-output vertex is labelled with \star . It has a tree-like shape.

Lemma A.15 (Shape of Stable Hypernets). *1. In any stable hypernet, if a vertex v' is reachable from another vertex v such that $v \neq v'$, there exists a unique path from the vertex v to the vertex v' .*

2. Any stable hypernet has no cyclic path, i.e. a path from a vertex to itself.

3. Let $\mathcal{C} : \star \Rightarrow \otimes_{i=1}^m \ell_i$ be a simple context such that: its hole has one source and at least one outgoing edge; and its unique input is the hole's source. There are no two stable hypernets G and G' that satisfy $G = \mathcal{C}[G']$.

Proof. To prove the point (1), assume there are two different paths from the vertex v to the vertex v' . These paths, i.e. non-empty sequences of edges, have to involve an edge with more than one source, or two different edges that share the same target. However, neither of these is possible in a stable hypernet, because both a passive operation edge and an instance edge have only one source and vertices can have at most one incoming edge. The point (1) follows from this by contradiction.

If a stable hypernet has a cyclic path from a vertex v to itself, there must be infinitely many paths from the input to the vertex v , depending on how many times the cycle is included. This contradicts the point (1).

The point (3) is also proved by contradiction. Assume that there exist two stable hypernets G and G' that satisfy $G = \mathcal{C}[G']$ for the simple context \mathcal{C} . In the stable hypernet G , a vertex is always labelled with \star if it is not an output. However, in the simple context \mathcal{C} , there exists at least one target of the hole that is not an output of the context but not labelled with \star either. This contradicts $\mathcal{C}[G']$ being a stable hypernet. \square

A stable hypernet can be found as a part of representation of a value.

Lemma A.16. *Let \vec{x} be a sequence of k variables and \vec{a} be a sequence of h atoms. For any derivable type judgement $\vec{x} \mid \vec{a} \vdash v : \star$ where v is a value, its representation can be decomposed as $(\vec{x} \mid \vec{a} \vdash v : \star)^\dagger = \mathcal{C}[G]$ using a stable hypernet $G : \star \Rightarrow \otimes_{i=1}^m \ell_i$ and a simple context $\mathcal{C} : \star \Rightarrow \star^{\otimes k} \otimes \diamond^{\otimes h}$ whose unique input coincides with a (unique) source of its hole.*

Proof. By induction on the definition of value.

When the value v is an atom, in the representation $(\vec{x} \mid \vec{a} \vdash v : \star)^\dagger$, only an instance edge can comprise a stable hypernet.

When the value is $v \equiv \phi_\vee(v_1, \dots, v_m; \vec{s})$, by induction hypothesis, a stable hypernet G_i can be extracted from (a bottom part of) representation of each eager argument v_i . The stable hypernet G that decomposes the representation $(\vec{x} \mid \vec{a} \vdash v : \star)^\dagger$ can be given by all these stable hypernets G_1, \dots, G_m together with the passive operation edge ϕ_\vee that is introduced in the representation.

When the value is $v \equiv \text{bind } x \rightarrow t \text{ in } v'$, or $v \equiv \text{new } a \multimap t \text{ in } v'$, by induction hypothesis, representation of the value v' includes a stable hypernet G' . The stable hypernet itself decomposes the representation $(\vec{x} \mid \vec{a} \vdash v : \star)^\dagger$ in the required way. \square

Lemma A.17. *For any state \dot{N} , and its vertex v , such that the vertex v is not a target of an instance edge or a passive operation edge, if an accessible path from the vertex v is stable or active, then the path has no multiple occurrences of a single edge.*

Proof. Any stable or active path consists of edges that has only one source. As a consequence, except for the first edge, no edge appears twice in the stable path. If the stable path is from the vertex v , its first edge also does not appear twice, because v is not a target of an instance edge or a passive operation edge. \square

Lemma A.18. *For any state \dot{N} , and its vertex v , such that the vertex v is not a target of an instance edge or a passive operation edge, the following are equivalent.*

(A) *There exist a focussed simple context $\dot{C}[\chi]$ and a stable hypernet G , such that $\dot{N} = \dot{C}[G]$, where the vertex v of \dot{N} corresponds to a unique source of the hole edge in \dot{C} .*

(B) *Any accessible path from the vertex v in \dot{N} is a stable path.*

Proof of (A) \Rightarrow (B). Because no output of a stable hypernet has type \star , any path from the vertex v in $\dot{C}[G]$ gives a path from the unique input in G . In the stable hypernet G , any path from the unique input is a stable path. \square

Proof of (B) \Rightarrow (A). In the state \dot{N} , the token target has to be a source of an edge, which forms an accessible path itself. By Lem. A.17, in the state \dot{N} , we can take maximal stable paths from the vertex v , in the sense that appending any edge to these paths, if possible, does not give a stable path.

If any of these maximal stable paths is to some vertex, the vertex does not have type \star ; this can be confirmed as follows. If the vertex has type \star , it is not an output, so it is a source of an instance, token, operation or contraction edge. The case of an instance or passive operation edge contradicts the maximality. The other case yields a non-stable accessible path that contradicts the assumption (B).

Collecting all edges contained by the maximal stable paths, therefore, gives the desired hypernet G . These edges are necessarily all shallow, because of the vertex v of \dot{N} . The focussed context $\dot{C}[\chi]$, whose hole is shallow, can be made of all the other edges (at any depth) of the state \dot{N} . \square

Lemma A.19. *Let \dot{N} be a state, where the token is an incoming edge of an operation edge e , whose label ϕ takes at least one eager arguments. Let k denote the number of eager arguments of ϕ .*

For each $i \in \{1, \dots, k\}$, let $sw_i(\dot{N})$ be a state such that: both states $sw_i(\dot{N})$ and \dot{N} have the same token label and the same underlying hypernet, and the token in $sw_i(\dot{N})$ is the i -th outgoing edge of the operation edge e .

For each $i \in \{1, \dots, k\}$, the following are equivalent.

(A) *In \dot{N} , any accessible path from an i -th target of the operation edge e is a stable (resp. active) path.*

(B) *In $sw_i(\dot{N})$, any accessible path from the token target is a stable (resp. active) path.*

Proof. The only difference between \dot{N} and $sw_i(\dot{N})$ is the swap of the token with the operation edge e , and these two edges form an accessible path in the states \dot{N} and $sw_i(\dot{N})$, individually or together (in an appropriate order). Therefore, there is one-to-one correspondence between accessible paths from an i -th target of the edge e in \dot{N} , and accessible paths from the token target in $sw_i(\dot{N})$.

When (A) is the case, in \dot{N} , any accessible paths from an i -th target of the edge e does not contain the token nor the edge e ; otherwise there would be an accessible path that contains the token and hence not stable nor active, which is a contradiction. This means that, in $sw_i(\dot{N})$, any accessible path from the token target also does not contain the token nor the edge e , and the path must be a stable (resp. active) path.

When (B) is the case, the proof takes the same reasoning in the reverse way. \square

Lemma A.20. *Let \dot{N} be a rooted state with a search token, such that the token is not an incoming edge of a contraction edge.*

1. $\dot{N} \xrightarrow{+} \langle \dot{N} \rangle_{\checkmark/?}$, *if and only if any accessible path from the token target in \dot{N} is a stable path.*

2. $\dot{N} \xrightarrow{+} \langle \dot{N} \rangle_{\dagger/?}$, *if and only if any accessible path from the token target in \dot{N} is an active path.*

Proof of the forward direction. Let t be either ' \checkmark ' or ' \dagger '. The assumption is $\dot{N} \xrightarrow{*} \langle \dot{N} \rangle_{t/?}$. We prove the following, by induction on the length n of this search sequence:

- any accessible path from the token target in \dot{N} is a stable path, when $\mathbf{t} = \checkmark$, and
- any accessible path from the token target in \dot{N} is an active path, when $\mathbf{t} = \zeta$.

In the base case, where $n = 1$, because the token is not an incoming edge of a contraction edge, the token target is a source of an instance edge, or an operation edge labelled with $\phi \in \mathbb{O}_{\mathbf{t}}$ that takes no eager argument. In either situation, the outgoing edge of the token gives the only possible accessible path from the token target. The path is stable when $\mathbf{t} = \checkmark$, and active when $\mathbf{t} = \zeta$.

In the inductive case, where $n > 1$, the token target is a source of an operation edge e_ϕ labelled with an operation $\phi \in \mathbb{O}_{\mathbf{t}}$ that takes at least one eager argument.

Let k denote the number of eager arguments of $\phi_{\mathbf{t}}$, and i be an arbitrary number in $\{1, \dots, k\}$. Let $sw_i(\dot{N})$ be the state as defined in Lem. A.19. Because \dot{N} is rooted, by Lem. A.10, the given search sequence gives the following search sequence (proof by induction on $k - i$):

$$?; |\dot{N}| \xrightarrow{*} \dot{N} \xrightarrow{+} sw_i(\dot{N}) \xrightarrow{+} \langle sw_i(\dot{N}) \rangle_{\checkmark/?} \xrightarrow{+} \langle \dot{N} \rangle_{\mathbf{t}/?}.$$

By induction hypothesis on the intermediate sequence $sw_i(\dot{N}) \xrightarrow{+} \langle sw_i(\dot{N}) \rangle_{\checkmark/?}$, any accessible path from the token target in $sw_i(\dot{N})$ is a stable path. By Lem. A.19, any accessible path from an i -th target of the operation edge e_ϕ in \dot{N} is a stable path.

In \dot{N} , any accessible path from the token target is given by the operation edge e_ϕ followed by an accessible path, which is proved to be stable above, from a target of e_ϕ . Any accessible path from the token target is therefore stable when $\mathbf{t} = \checkmark$, and active when $\mathbf{t} = \zeta$. \square

Proof of the backward direction. Let \mathbf{t} be either ' \checkmark ' or ' ζ '. The assumption is the following:

- any accessible path from the token target in \dot{N} is a stable path, when $\mathbf{t} = \checkmark$, and
- any accessible path from the token target in \dot{N} is an active path, when $\mathbf{t} = \zeta$.

Our goal is to show $\dot{N} \xrightarrow{*} \langle \dot{N} \rangle_{\mathbf{t}/?}$.

In the state \dot{N} , the token target has to be a source of an edge, which forms an accessible path itself. By Lem. A.17, we can define $r(\dot{N})$ by the maximum length of stable paths from the token target. This number $r(\dot{N})$ is well-defined and positive. We prove $\dot{N} \xrightarrow{*} \langle \dot{N} \rangle_{\mathbf{t}/?}$ by induction on $r(\dot{N})$.

In the base case, where $r(\dot{N}) = 1$, the outgoing edge of the token is the only possible accessible path from the token target. The outgoing edge is not a contraction edge by the assumption, and hence it is an instance edge, or an operation edge labelled with $\phi \in \mathbb{O}_{\mathbf{t}}$ that takes no eager argument. We have $\dot{N} \xrightarrow{*} \langle \dot{N} \rangle_{\mathbf{t}/?}$.

In the inductive case, where $r(\dot{N}) > 1$, the outgoing edge of the token is an operation edge e_ϕ labelled with $\phi \in \mathbb{O}_{\mathbf{t}}$ that takes at least one eager argument. Any accessible path from the token target in \dot{N} is given by the edge e_ϕ followed by a stable path from a target of e_ϕ .

Let k denote the number of eager arguments of $\phi_{\mathbf{t}}$, and i be an arbitrary number in $\{1, \dots, k\}$. Let $sw_i(\dot{N})$ be the state as defined in Lem. A.19.

By the assumption, any accessible path from an i -th target of the operation edge e_ϕ in \dot{N} is a stable path. Therefore by Lem. A.19, in $sw_i(\dot{N})$, any accessible path from the token target is a stable path. Moreover, these paths in \dot{N} and $sw_i(\dot{N})$ correspond to each other. By Lem. A.17, we can define $r(sw_i(\dot{N}))$ by the maximum length of stable paths from the token target. This number $r(sw_i(\dot{N}))$ is well-defined, and satisfies $r(sw_i(\dot{N})) < r(\dot{N})$. By induction hypothesis on this number, we have:

$$sw_i(\dot{N}) \xrightarrow{*} \langle sw_i(\dot{N}) \rangle_{\checkmark/?}.$$

Combining this search sequence with the following possible search transitions concludes the proof:

$$\begin{aligned} \dot{N} &\xrightarrow{*} sw_1(\dot{N}), \\ \langle sw_i(\dot{N}) \rangle_{\checkmark/?} &\xrightarrow{*} sw_{i+1}(\dot{N}), \\ &\quad (\text{when } k \neq 1 \text{ and } i < k) \\ \langle sw_k(\dot{N}) \rangle_{\checkmark/?} &\xrightarrow{*} \langle \dot{N} \rangle_{\mathbf{t}/?}. \end{aligned}$$

\square

A.5 Parametrised (contextual) refinement and equivalence

Lemma A.21. *For any focus-free contexts $\mathcal{C}_1[\vec{\chi}', \chi, \vec{\chi}']$ and \mathcal{C}_2 such that $\mathcal{C}_1[\vec{\chi}', \mathcal{C}_2, \vec{\chi}']$ is defined, if both \mathcal{C}_1 and \mathcal{C}_2 are binding-free, then $\mathcal{C}_1[\vec{\chi}', \mathcal{C}_2, \vec{\chi}']$ is also binding-free.*

Proof. Let \mathcal{C} denote $\mathcal{C}_1[\vec{\chi}', \mathcal{C}_2, \vec{\chi}']$, and e_χ denote the hole edge of \mathcal{C}_1 labelled with χ .

The proof is by contradiction. We assume that there exists a path P in \mathcal{C} , from a source of a contraction, atom, box or hole edge e , to a source of a hole edge e' . We derive a contradiction by case analysis on the path P .

- When e' comes from \mathcal{C}_1 , and the path P consists of edges from \mathcal{C}_1 only, the path P gives a path in \mathcal{C}_1 that contradicts \mathcal{C}_1 being binding-free.
- When e' comes from \mathcal{C}_1 , and the path P contains an edge from \mathcal{C}_2 , by finding the last edge from \mathcal{C}_2 in P , we can take a suffix of P that gives a path from a target of the hole edge e_χ to a source of a hole edge, in \mathcal{C}_1 . Adding the hole edge e_χ at the beginning yields a path in \mathcal{C}_1 that contradicts \mathcal{C}_1 being binding-free.
- When both e and e' come from \mathcal{C}_2 , and the path P gives a path in \mathcal{C}_2 , this contradicts \mathcal{C}_2 being binding-free.
- When both e and e' come from \mathcal{C}_2 , and the path P does not give a single path in \mathcal{C}_2 , there exists a path from a source of the hole edge e_χ to a source of the hole edge e_χ , in \mathcal{C}_1 . This path contradicts \mathcal{C}_1 being binding-free.
- When e comes from \mathcal{C}_1 and e' comes from \mathcal{C}_2 , by finding the first edge from \mathcal{C}_2 in P , we can take a prefix of P that gives a path from a source of a contraction, atom, box or hole edge to a source of the hole edge e_χ , in \mathcal{C}_1 . This path contradicts \mathcal{C}_1 being binding-free.

□

Lemma A.22. *For any set \mathbb{C} of contexts that is closed under plugging, and any preorder Q on natural numbers, the following holds.*

- $\dot{\preceq}_Q$ and $\preceq_Q^{\mathbb{C}}$ are reflexive.
- $\dot{\preceq}_Q$ and $\preceq_Q^{\mathbb{C}}$ are transitive.
- \simeq_Q and $\simeq_Q^{\mathbb{C}}$ are equivalences.

Proof. Because \simeq_Q and $\simeq_Q^{\mathbb{C}}$ are defined as a symmetric subset of $\dot{\preceq}_Q$ and $\preceq_Q^{\mathbb{C}}$, respectively, \simeq_Q and $\simeq_Q^{\mathbb{C}}$ are equivalences if $\dot{\preceq}_Q$ and $\preceq_Q^{\mathbb{C}}$ are preorders.

Reflexivity and transitivity of $\dot{\preceq}_Q$ is a direct consequence of those of the preorder Q .

For any focus-free hypernet H , and any focus-free context $\mathcal{C}[\chi] \in \mathbb{C}$ such that $?\mathcal{C}[H]$ is a state, $?\mathcal{C}[H] \dot{\preceq}_Q ?\mathcal{C}[H]$ because of reflexivity of $\dot{\preceq}_Q$.

For any focus-free hypernets H_1, H_2 and H_3 , and any focus-free context $\mathcal{C}[\chi] \in \mathbb{C}$, such that $H_1 \preceq_Q^{\mathbb{C}} H_2$, $H_2 \preceq_Q^{\mathbb{C}} H_3$, and both $?\mathcal{C}[H_1]$ and $?\mathcal{C}[H_3]$ are states, our goal is to show $?\mathcal{C}[H_1] \dot{\preceq}_Q ?\mathcal{C}[H_3]$. Because $H_1 \preceq_Q^{\mathbb{C}} H_2$ and $H_2 \preceq_Q^{\mathbb{C}} H_3$, all three hypernets H_1, H_2 and H_3 have the same type, and hence $?\mathcal{C}[H_2]$ is also a state. Therefore, we have $?\mathcal{C}[H_1] \dot{\preceq}_Q ?\mathcal{C}[H_2]$ and $?\mathcal{C}[H_2] \dot{\preceq}_Q ?\mathcal{C}[H_3]$, and the transitivity of $\dot{\preceq}_Q$ implies $?\mathcal{C}[H_1] \dot{\preceq}_Q ?\mathcal{C}[H_3]$. □

Lemma A.23. *For any set \mathbb{C} of contexts that is closed under plugging, and any preorder Q on natural numbers, the following holds.*

1. *For any hypernets H_1 and H_2 , $H_1 \simeq_{Q \cap Q^{-1}}^{\mathbb{C}} H_2$ implies $H_1 \simeq_Q^{\mathbb{C}} H_2$.*
2. *If all compute transitions are deterministic, for any hypernets H_1 and H_2 , $H_1 \simeq_Q^{\mathbb{C}} H_2$ implies $H_1 \simeq_{Q \cap Q^{-1}}^{\mathbb{C}} H_2$.*

Proof. Because $(Q \cap Q^{-1}) \subseteq Q$, the point (1) follows from the monotonicity of contextual equivalence.

For the point (2), $H_1 \simeq_Q^{\mathbb{C}} H_2$ means that any focus-free context $\mathcal{C}[\chi] \in \mathbb{C}$, such that $?\mathcal{C}[H_1]$ and $?\mathcal{C}[H_2]$ are states, yields $?\mathcal{C}[H_1] \preceq_Q ?\mathcal{C}[H_2]$ and $?\mathcal{C}[H_2] \preceq_Q ?\mathcal{C}[H_1]$. If the state $?\mathcal{C}[H_1]$ terminates at a final state after k_1 transitions, there exists k_2 such that $k_1 Q k_2$ and the state $?\mathcal{C}[H_2]$ terminates at a final state after k_2 transitions. Moreover, there exists k_3 such that $k_2 Q k_3$ and the state $?\mathcal{C}[H_1]$ terminates at a final state after k_3 transitions.

Because search transitions and copy transitions are deterministic, if all compute transitions are deterministic, states and transitions comprise a deterministic abstract rewriting system, in which final states are normal forms. By Lem. A.6, $k_1 = k_3$ must hold. This means $k_1 Q \cap Q^{-1} k_2$, and $?\mathcal{C}[H_1] \preceq_{Q \cap Q^{-1}} ?\mathcal{C}[H_2]$. Similarly, we can infer $?\mathcal{C}[H_2] \preceq_{Q \cap Q^{-1}} ?\mathcal{C}[H_1]$, and hence $H_1 \simeq_{Q \cap Q^{-1}}^{\mathbb{C}} H_2$. \square

A.6 Proof for Sec. 6.4

Lemma A.24. *Let \mathbb{C} be a set of contexts, and Q' be a binary relation on \mathbb{N} such that, for any $k_0, k_1, k_2 \in \mathbb{N}$, $(k_0 + k_1) Q' (k_0 + k_2)$ implies $k_1 Q' k_2$. Let \triangleleft be a pre-template that is a trigger and implies contextual refinement $\preceq_{Q'}^{\mathbb{C}}$. For any single \mathbb{C} -specimen $(\dot{\mathcal{C}}[\chi]; H^1; H^2)$ of \triangleleft , the following holds.*

1. *For any $k \in \mathbb{N}$, $?\mathcal{C}[\dot{\mathcal{C}}[H^1]] \xrightarrow{\bullet^k} \dot{\mathcal{C}}[H^1]$ if and only if $?\mathcal{C}[\dot{\mathcal{C}}[H^2]] \xrightarrow{\bullet^k} \dot{\mathcal{C}}[H^2]$.*
2. *If compute transitions are all deterministic, and one of states $\dot{\mathcal{C}}[H^1]$ and $\dot{\mathcal{C}}[H^2]$ is rooted, then the other state is also rooted, and moreover, $\dot{\mathcal{C}}[H^1] \preceq_{Q'} \dot{\mathcal{C}}[H^2]$.*

Proof of the point (1). Let (p, q) be an arbitrary element of a set $\{(1, 2), (2, 1)\}$. We prove that, for any $k \in \mathbb{N}$, $?\mathcal{C}[\dot{\mathcal{C}}[H^p]] \xrightarrow{\bullet^k} \dot{\mathcal{C}}[H^p]$ implies $?\mathcal{C}[\dot{\mathcal{C}}[H^q]] \xrightarrow{\bullet^k} \dot{\mathcal{C}}[H^q]$. The proof is by case analysis on the number k .

- When $k = 0$, $\dot{\mathcal{C}}[H^p]$ is initial, and by Lem. 7.5(1), $\dot{\mathcal{C}}[H^q]$ is also initial. Note that \triangleleft is a trigger and hence output-closed.
- When $k > 0$, by the following internal lemma, $?\mathcal{C}[\dot{\mathcal{C}}[H^q]] \xrightarrow{\bullet^k} \dot{\mathcal{C}}[H^q]$ follows from $?\mathcal{C}[\dot{\mathcal{C}}[H^p]] \xrightarrow{\bullet^k} \dot{\mathcal{C}}[H^p]$.

Lemma A.25. *For any $m \in \{0, \dots, k\}$, there exists a focussed context $\dot{\mathcal{C}}'[\chi]$ such that $|\dot{\mathcal{C}}'| = |\dot{\mathcal{C}}|$ and the following holds:*

$$\begin{aligned} ?\mathcal{C}[\dot{\mathcal{C}}[H^p]] &\xrightarrow{\bullet^m} \dot{\mathcal{C}}'[H^p] \xrightarrow{\bullet^{k-m}} \dot{\mathcal{C}}[H^p], \\ ?\mathcal{C}[\dot{\mathcal{C}}[H^q]] &\xrightarrow{\bullet^m} \dot{\mathcal{C}}'[H^q]. \end{aligned}$$

Proof. By induction on m . In the base case, when $m = 0$, we can take $?\mathcal{C}[\dot{\mathcal{C}}]$ as $\dot{\mathcal{C}}'$.

In the inductive case, when $m > 0$, by induction hypothesis, there exists a focussed context $\dot{\mathcal{C}}'[\chi]$ such that $|\dot{\mathcal{C}}'| = |\dot{\mathcal{C}}|$ and the following holds:

$$\begin{aligned} ?\mathcal{C}[\dot{\mathcal{C}}[H^p]] &\xrightarrow{\bullet^{m-1}} \dot{\mathcal{C}}'[H^p] \xrightarrow{\bullet^{k-m+1}} \dot{\mathcal{C}}[H^p], \\ ?\mathcal{C}[\dot{\mathcal{C}}[H^q]] &\xrightarrow{\bullet^{m-1}} \dot{\mathcal{C}}'[H^q]. \end{aligned}$$

Because $|\dot{\mathcal{C}}'| = |\dot{\mathcal{C}}| \in \mathbb{C}$, $(\dot{\mathcal{C}}'; H^1; H^2)$ is a single \mathbb{C} -specimen of \triangleleft , which yields rooted states. Because $k - m + 1 > 0$, $\dot{\mathcal{C}}'$ cannot have a rewrite token. The rest of the proof is by case analysis on the token of $\dot{\mathcal{C}}'$.

- When $\dot{\mathcal{C}}'$ has an entering search token, because \triangleleft is a trigger, $\dot{\mathcal{C}}'[H^r] \rightarrow \langle \dot{\mathcal{C}}'[H^r] \rangle_{\sharp/?}$ for each $r \in \{p, q\}$. Because $\langle \dot{\mathcal{C}}'[H^r] \rangle_{\sharp/?} = \langle \dot{\mathcal{C}}' \rangle_{\sharp/?}[H^r]$, and search transitions are deterministic, we have the following:

$$\begin{aligned} ?\mathcal{C}[\dot{\mathcal{C}}[H^p]] &\xrightarrow{\bullet^{m-1}} \dot{\mathcal{C}}'[H^p] \xrightarrow{\bullet} \langle \dot{\mathcal{C}}' \rangle_{\sharp/?}[H^p] \xrightarrow{\bullet^{k-m}} \dot{\mathcal{C}}[H^p], \\ ?\mathcal{C}[\dot{\mathcal{C}}[H^q]] &\xrightarrow{\bullet^{m-1}} \dot{\mathcal{C}}'[H^q] \xrightarrow{\bullet} \langle \dot{\mathcal{C}}' \rangle_{\sharp/?}[H^q]. \end{aligned}$$

We also have $|\langle \dot{\mathcal{C}}' \rangle_{\neq/?}| = |\dot{\mathcal{C}}'| = |\dot{\mathcal{C}}|$.

- When $\dot{\mathcal{C}}'$ has a value token, or a non-entering search token, because \triangleleft is output-closed, by Lem. 7.5(3), there exists a focussed context $\dot{\mathcal{C}}''$ such that $|\dot{\mathcal{C}}''| = |\dot{\mathcal{C}}'|$ and $\dot{\mathcal{C}}'[H^r] \rightarrow \dot{\mathcal{C}}''[H^r]$ for each $r \in \{p, q\}$. The transition $\dot{\mathcal{C}}'[H^r] \rightarrow \dot{\mathcal{C}}''[H^r]$, for each $r \in \{p, q\}$, is a search transition, and by the determinism of search transitions, we have the following:

$$\begin{aligned} ?; |\dot{\mathcal{C}}|[H^p] &\xrightarrow{m-1} \dot{\mathcal{C}}'[H^p] \xrightarrow{\bullet} \dot{\mathcal{C}}''[H^p] \xrightarrow{k-m} \dot{\mathcal{C}}[H^p], \\ ?; |\dot{\mathcal{C}}|[H^q] &\xrightarrow{m-1} \dot{\mathcal{C}}'[H^q] \xrightarrow{\bullet} \dot{\mathcal{C}}''[H^q]. \end{aligned}$$

□

□

Proof of the point (2). If one of states $\dot{\mathcal{C}}[H^1]$ and $\dot{\mathcal{C}}[H^2]$ is rooted, by the point (1), the other state is also rooted, and moreover, there exists $k \in \mathbb{N}$ such that $?; |\dot{\mathcal{C}}|[H^r] \xrightarrow{k} \dot{\mathcal{C}}[H^r]$ for each $r \in \{1, 2\}$.

Our goal is to prove that, for any $k_1 \in \mathbb{N}$ and any final state \dot{N}_1 such that $\dot{\mathcal{C}}[H^1] \rightarrow^{k_1} \dot{N}_1$, there exist $k_2 \in \mathbb{N}$ and a final state \dot{N}_2 such that $k_1 \mathcal{Q}' k_2$ and $\dot{\mathcal{C}}[H^2] \rightarrow^{k_2} \dot{N}_2$. Assuming $\dot{\mathcal{C}}[H^1] \rightarrow^{k_1} \dot{N}_1$, we have the following:

$$\begin{aligned} ?; |\dot{\mathcal{C}}|[H^1] &\xrightarrow{k} \dot{\mathcal{C}}[H^1] \rightarrow^{k_1} \dot{N}_1, \\ ?; |\dot{\mathcal{C}}|[H^2] &\xrightarrow{k} \dot{\mathcal{C}}[H^2]. \end{aligned}$$

Because \triangleleft implies contextual refinement $\preceq_{\mathcal{Q}'}^{\mathbb{C}}$, and $|\dot{\mathcal{C}}| \in \mathbb{C}$, we have state refinement $?; |\dot{\mathcal{C}}|[H^1] \dot{\preceq}_{\mathcal{Q}'} ?; |\dot{\mathcal{C}}|[H^2]$. Therefore, there exist $l_2 \in \mathbb{N}$ and a final state \dot{N}_2 such that $(k + k_1) \mathcal{Q}' l_2$ and $?; |\dot{\mathcal{C}}|[H^2] \rightarrow^{l_2} \dot{N}_2$.

The assumption that compute transitions are all deterministic implies that all transitions, including intrinsic ones, are deterministic. Following from this are $l_2 \geq k$ and the following:

$$\begin{aligned} ?; |\dot{\mathcal{C}}|[H^1] &\xrightarrow{k} \dot{\mathcal{C}}[H^1] \rightarrow^{k_1} \dot{N}_1, \\ ?; |\dot{\mathcal{C}}|[H^2] &\xrightarrow{k} \dot{\mathcal{C}}[H^2] \rightarrow^{l_2-k} \dot{N}_2. \end{aligned}$$

By the assumption on \mathcal{Q}' , $(k + k_1) \mathcal{Q}' l_2$ implies $k_1 \mathcal{Q}' (l_2 - k)$.

□