Decentralized Execution of Constraint Handling Rules for Ensembles*

Edmund S. L. Lam and Iliano Cervesato

Apr 2013
CMU-CS-13-106
CMU-CS-QTR-118

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Carnegie Mellon University, Qatar campus.
The author can be reached at sllam@qatar.cmu.edu or iliano@cmu.edu.

Abstract

CHR is a declarative, concurrent and committed choice rule-based constraint programming language. In this paper, we adapt CHR to provide a decentralized execution model for parallel and distributed programs. Specifically, we consider an execution model consisting of an ensemble of computing entities, each with its own constraint store and each capable of communicating with its neighbors. We extend CHR into CHR^e, in which rewrite rules are executed at one location and are allowed to access the constraint store of its immediate neighbors. We give an operational semantics for CHR^e, denoted ω_0^e, that defines incremental and asynchronous decentralized rewriting for the class of CHR^e rules characterized by purely local matching (0-neighbor restricted rules). We show the soundness of the ω_0^e semantics with respect to the abstract CHR semantics. We then give a safe encoding of the more general 1-neighbor restricted rules as 0-neighbor restricted rules, and discuss how this encoding can be generalized to all CHR^e programs.

* Funded by the Qatar National Research Fund as project NPRP 09-667-1-100 (Effective Programming for Large Distributed Ensembles)

Keywords: Distributed Programming, Constraint Logic Programming, Multiset Rewriting

Contents

List of Figures

1 Introduction

In recent years, we have seen many advances in distributed systems, multicore architectures and cloud computing, drawing more research interest into better ways to harness and coordinate the combined power of distributed computation. While this has made distributed computing resources more readily accessible to main-stream audiences, the fact remains that implementing distributed applications that can exploit such resources via traditional distributed programming methodologies is an extremely difficult task. As such, finding effective means of programming distributed systems is more than ever an active and fruitful research and development endeavor.

In this paper, we propose an extension of the constraint programming language CHR [Frü94] (Constraint Handling Rules). This language, which we call CHR^e is designed specifically as a high-level distributed and parallel programming language for developing applications that operate in a decentralized manner over an *ensemble* of distributed computing entities, referred to as locations. Each rewrite rule executes at a location, enabling this location to read and write data held by its immediate neighbors. Specifically, we are interested in rules that can read data from up to n of their immediate neighbors for various values of n, but writes to any number of neighbors. We call them *n-neighbor restricted rules*. Such n-neighbor restricted rules are a generalization of link restriction [LCG$^+$06, ARLG$^+$09] adapted to the context of multiset rewriting. This gives us a highly expressive model for programming complex behaviors involving distributed ensembles in a declarative and concurrent manner. Figure 1 shows an example CHR^e program that contains three rewrite

$$
\begin{array}{l}
\mathit{base} : [X]\,\mathit{edge}(Y, D) \Longrightarrow [X]\,\mathit{path}(Y, D) \\
\mathit{elim} : [X]\,\mathit{path}(Y, D) \setminus [X]\,\mathit{path}(Y, D') \Longleftrightarrow D < D' \mid \mathit{true} \\
\mathit{trans} : [X]\,\mathit{edge}(Y, D), [Y]\,\mathit{path}(Z, D') \Longrightarrow X \neq Z \mid [X]\,\mathit{path}(Z, D + D')
\end{array}
$$

Figure 1: Distributed All Shortest Path

rules. This program computes all shortest paths of a graph in a distributed manner. An edge of length D from a location X to Y is represented by a constraint $\mathit{edge}(Y, D)$ found in X's constraint store, written $[X]\,\mathit{edge}(Y, D)$. Similarly $[X]\,\mathit{path}(Y, D)$ expresses a path of length D from X to Y. The location of a constraint is modeled by the $[l]$ operator that prefixes all constraints in the rewrite rules. The rules *base* and *elim* are 0-*neighbor restricted* rules because their left-hand sides involve constraints from exactly one location, X. Rule *trans* is a 1-neighbor restricted rule since its left-hand side involves X and a neighbor Y. We designate X as the *primary* location of this rewrite rule because it references Y in an argument (here, $[X]\,\mathit{edge}(Y, D)$). This means that Y is a topological neighbor of X[1]. Neighbor restriction brings aspects of the topology of the ensemble (X has an edge to Y) in the rewrite rules. Rule *base* adds $\mathit{path}(Y, D)$ at location X for every instance of constraint $\mathit{edge}(Y, D)$ at the same location. Rule *elim* looks for any instances of a pair of constraints, $\mathit{path}(Y, D)$ and $\mathit{path}(Y, D')$ involving the same location Y at location X and removes the longer of the two path ($\mathit{path}(Y, D')$ for $D < D'$). Rule *trans* adds $\mathit{path}(Z, D + D')$ at location X whenever there is $\mathit{edge}(Y, D)$ at location X and a matching $\mathit{path}(Z, D')$ at location Y for $X \neq Z$. This program is *declarative* because the programmer focuses on *which* distributed computations to synchronize (e.g. $[X]\,\mathit{edge}(Y, D)$, $[Y]\,\mathit{path}(Z, D')$) to produce *what* results ($[X]\,\mathit{path}(Z, D + D')$), rather than *how* synchronization is achieved. It is *concurrent* because while a rewrite rule applies to a fragment of the ensemble, many other rewritings can occur asynchronously in the rest of the ensemble.

In this technical report, we present CHR^e, a distributed programming language that extends CHR with *located constraints* and *n-neighbor restricted rewrite rules*, a generalized notion of link-restriction [LCG$^+$06]. We define the ω^e_α abstract semantics of CHR^e and prove its soundness with respect to the standard semantics of CHR. Following this, we extend the original CHR refined operational semantics [DSdlBH04] to

[1] Any location reference Y that appears in an *edge* predicate is this manner is considered a neighbor to X. This is how neighbor restriction enforces topological information to be embedded in predicates.

support decentralized incremental multiset matching for 0-neighbor restricted rules obtaining the ω_0^e operational semantics, and prove its soundness and the exhaustiveness of rule application, with respect to the ω_α^e semantics. We then give an optimized encoding of 1-neighbor restricted rules into 0-neighbor restricted rules of ω_0^e, prove the soundness of this encoding. Following this, we generalize this encoding work for n-neighbor restricted rules and prove its soundness.

The main contributions in this paper are as follows:

- We present CHR^e, a distributed programming language that extends CHR with *located constraints* and *n-neighbor restricted rewrite rules*, a generalized notion of link-restriction [LCG+06] in the context of multiset rewriting.
- We define the ω_α^e abstract semantics of CHR^e and prove its soundness with respect to the CHR abstract semantics.
- We present the ω_0^e operational semantics (based on the refined CHR operational semantics [DSdlBH04]) that supports decentralized incremental multiset matching for 0-neighbor restricted rewrite rules. We prove its soundness with respect to the ω_α^e abstract semantics.
- We give an optimized encoding of 1-neighbor restricted rules in ω_0^e and prove the soundness of this encoding.
- We generalize this encoding for n-neighbor restricted rules and prove the soundness of this encoding.

The rest of this report is organized as follows: Section 2 recalls notions used in this paper and the traditional CHR language. Section 3 defines the CHR^e language while Section 4 gives the ω_α^e abstract semantics, the ω_0^e operational semantics and relates them. Section 5 encodes 1-neighbor restricted rules into ω_0^e. In Section 6, we show the encoding of the full CHR^e language (n-neighbor restricted rules) into ω_0^e. We review related work in Section 7 and conclude in Section 8. Proof of all lemmas and theorems in this report are consolidated in Appendix A.

2 Preliminaries

In this section, we present the notations that will be used throughout this paper. We also give a brief introduction to the abstract CHR language and its semantics.

2.1 Notations

Let o be a generic syntactic construct or runtime object of our language. We write $\bar{o}$ for a multiset of objects o and $\vec{o}$ for a sequence of objects o. Relying on these notational accents for disambiguation, we write ',' to represent both multiset union and sequence concatenation, with $\emptyset$ as the identity element. For instance, '$o, \bar{o}$' is a multiset while '$o, \vec{o}$' is a sequence (the former is commutative while the latter is not). A set is treated as a multiset with single occurrences of each element. We use standard notations for sets: $\bar{o}_1 \cup \bar{o}_2$ for union, $o \in \bar{o}$ for membership, and $\bar{o}_1 \subseteq \bar{o}_2$ for subset. Given a set $\mathcal{I}$ of labels, we write $\biguplus_{i \in \mathcal{I}} o_i$ to denote the multiset of objects o_i, for $i \in \mathcal{I}$. It is inductively defined as follows:

$$\left(\biguplus_{i \in \{j\} \cup \mathcal{I}} o_i\right) = o_j, \left(\biguplus_{i \in \mathcal{I}} o_i\right) \qquad \left(\biguplus_{i \in \emptyset} o_i\right) = \emptyset$$

The substitution of all occurrences of variable x in o with the term expression t is denoted as $[t/x]o$. It is inductively defined on all syntactic constructs and runtime objects o in the usual manner. We assume that substitution is capture-avoiding and rely on implicit α-renaming. Given a sequence of terms $\vec{t}$ and a sequence of variables $\vec{x}$, we write $[\vec{t}/\vec{x}]o$ as the simultaneous substitution of each variable in $\vec{x}$ with the corresponding term in $\vec{t}$. We will use this notion with sets as well (i.e, $\bar{t}$ and $\bar{x}$). We write $\mathrm{FV}(o)$ for the set of free-variables

CHR Abstract Syntax

Variables x Values v Predicate names p Rule names r Rule guard G

Term	t	$::=$	$x \mid v \mid ...$				
Constraint	c	$::=$	$p(\vec{t})$	Rewrite Rule	R	$::=$	$r : H \setminus H \iff G \mid B$
Rule Heads	H	$::=$	$\cdot \mid c, H$	Program	$\mathcal{P}$	$::=$	$R \mid R\,\mathcal{P}$
Rule Body	B	$::=$	$\exists \bar{x}.\ D$	Store	$\bar{\mathcal{S}}$	$::=$	$\emptyset \mid \bar{\mathcal{S}}, c$
Rule Body Elements	D	$::=$	$true \mid c, D$				

Simplification and Propagation Rule Short-hands

$$\text{Simpagation Rule} \quad \begin{cases} r : P \setminus S \iff G \mid B \\ r : P \setminus S \iff B \quad = \quad r : P \setminus S \iff true \mid B \end{cases}$$

$$\text{Propagation Rule} \quad \begin{cases} r : P \Longrightarrow G \mid B \quad = \quad r : P \setminus \cdot \iff G \mid B \\ r : P \Longrightarrow B \quad = \quad r : P \setminus \cdot \iff true \mid B \end{cases}$$

$$\text{Simplification Rule} \quad \begin{cases} r : S \iff G \mid B \quad = \quad r : \cdot \setminus S \iff G \mid B \\ r : S \iff B \quad = \quad r : \cdot \setminus S \iff true \mid B \end{cases}$$

ω_α Abstract Semantics of CHR

$$\frac{r : P' \setminus S' \iff G \mid B \in \mathcal{P} \qquad \models \theta G \qquad P = \theta P' \qquad S = \theta S'}{\mathcal{P} \quad \triangleright \quad (\bar{\mathcal{S}}, P, S) \quad \mapsto_{\omega_\alpha} \quad (\bar{\mathcal{S}}, P, \mathrm{NF}(\mathrm{Inst}(\theta B)))}$$

Figure 2: Constraint Handling Rules, Language and Semantics

in o and say that o is *ground* if $\mathrm{FV}(o) = \emptyset$. We write meta level operators in upright font (e.g. $\mathrm{FV}(-)$) and CHR constraint predicates in italics (e.g. *edge*, *path*).

2.2 CHR Language and Semantics

Figure 2 illustrates the abstract syntax of CHR. CHR is a high-level multiset rewriting language built on top of a *term language*, which we will keep mostly abstract. We shall only assume that it has variables and that each term t has a normal form, denoted $\mathrm{NF}(t)$. A rule guard G is a set of relations among term expressions, called *built-in constraints*. We assume that built-in constraints contain equality. The judgment $\models G$ decides the validity of ground built-in constraint G.

A CHR constraint $p(\vec{t})$ is a first-order predicate symbol p applied to a sequence of terms $\vec{t}$. A CHR rule $r : P \setminus S \iff G \mid B$ is a rewrite rule with a unique name r, such that P, S and B are multisets of CHR constraints and G is a set of built-in constraints. Each CHR rule specifies a rewriting that can occur over fragments of a multiset of constraints, known as the *CHR store*. Specifically, a rule like this states that we can replace any matching instance of 'P, S' in the CHR store with the corresponding instance of B, if G is valid when applied to the matching substitution ($\models \theta G$). We denote *true* as the identity built-in constraint such that $\models true$ is universally true. We will refer to P and S as the *propagate* and *simplify* rule heads, G as the rule *guard* and B as the *rule body*. This general form of CHR rule is known as a *simpagation rule*.

The short-hands $r : P \Longrightarrow G \mid B$ and $r : S \Longleftrightarrow G \mid B$, known as *propagation* and *simplification* rules, have empty simplified and propagated rule heads respectively. We will omit the rule guard component if it is empty as well. We take a few benign deviations from traditional treatments of CHR (e.g. [Frü94]): we require that all constraints in a CHR store be ground and that built-in constraints only appear in the guards G. This allows us to focus on the distributed multiset rewriting problem in this paper and set aside features like explicit built-in constraints as orthogonal extensions. Also, a rule body B can be prefixed by zero or more existential variable declarations $(\exists \bar{x})$. Such variables, which do not appear in the scope of the rule heads, are instantiated to new constants (that do not appear in the store) during rule application, allowing the creation of new *destinations* in destination passing-style programming [Pfe04]. For a rule body $\exists \bar{x}.\ D$ such that $\bar{x}$ is an empty set, we simply write it as D. A CHR rule $r : P \setminus S \Longleftrightarrow G \mid B$ is *well-formed* if rule guards and rule body are grounded by the rule heads ($\mathrm{FV}(G) \cup \mathrm{FV}(B) \subseteq \mathrm{FV}(P) \cup \mathrm{FV}(S)$[2]) and all term expressions that appear in the rule are well-formed. A CHR program $\mathcal{P}$ is *well-formed* if all CHR rules in $\mathcal{P}$ are well-formed and have a unique rule name. A CHR store $\bar{\mathcal{S}}$ is a multiset of constraints. It is *well-formed* if $\mathrm{FV}(\bar{\mathcal{S}}) = \emptyset$ and all terms that appear in constraints are well-formed.

We inductively extend the normalization function to rule bodies. Given a rule body that contains no existentials D, $\mathrm{NF}(D)$ denotes the normalized rule body with all term expressions in D in normal form. Since this operation only applies to rule bodies that contain no existentials, we define a complimentary meta operator that instantiates existential variables to fresh constants: Given a rule body $\exists \bar{x}.D$, $\mathrm{Inst}(\exists \bar{x}.D)$ denotes an instance of D such that each existential variable $\bar{x}$ that appear in D is replaced by a fresh constant a. These meta operation is inductively defined as follows:

$$\text{Normalize} \qquad \begin{cases} \mathrm{NF}(p(\vec{t}), D) &= p(\mathrm{NF}(\vec{t})), \mathrm{NF}(D) \\ \mathrm{NF}(\mathit{true}) &= \mathrm{NF}(\mathit{true}) \end{cases}$$

$$\text{Instantiate} \qquad \mathrm{Inst}(\exists \bar{x}.D) = [\bar{a}/\bar{x}]D \qquad \text{for each } a \in \bar{a} \text{ is a fresh constant}$$

Figure 2 also shows the abstract semantics ω_α of CHR. It defines the transitions of well-formed CHR stores via the derivation step $\mathcal{P} \rhd \bar{\mathcal{S}} \mapsto_{\omega_\alpha} \bar{\mathcal{S}}'$ for a given well-formed CHR program $\mathcal{P}$. A derivation step models the application of a CHR rule: it checks that rule heads P' and S' match fragments P and S of the store under a matching substitution θ and that the corresponding instance of guard G is valid ($\models \theta G$), resulting to the rewrite of S with θB with existential variables instantiated and then all term expressions evaluated to normal form (i.e, $\mathrm{NF}(\mathrm{Inst}(\theta B))$). Derivation steps preserves the well-formedness of constraint stores. We denote the reflexive and transitive application of derivation steps as $\mathcal{P} \rhd \bar{\mathcal{S}} \mapsto^*_{\omega_\alpha} \bar{\mathcal{S}}'$. A CHR store $\bar{\mathcal{S}}$ is *terminal* for $\mathcal{P}$ if no derivation steps of ω_α applies in $\bar{\mathcal{S}}$.

While the semantics in Figure 2 models CHR rewritings abstractly, actual CHR implementations implement some variant of the refined operational semantics [DSdlBH04]. This semantics computes new matches incrementally from newly added constraints. Additionally, rule heads are implicitly ordered by an *occurrence index* (typically in textual order of appearance) and matches are attempted in that order, thus providing programmatic idioms that assume a textual ordering of rule application. More details of these will be provided in Section 4.2.

3 The CHR^e Language

In this section, we introduce the CHR^e language. CHR^e extends CHR in several ways and here we focus on the syntactic ramifications of these extensions. *Locations* l in CHR^e rewriting rules are name annotations to a CHR constraint c, written as $[l]\,c$ and read as 'c is located at l'. We call $[l]$ the *localization operator*. Locations can be variables and in this case are subjected to substitution and other free variable meta operations as if they were an additional argument in constraints. Operationally, $[l]\,c$ indicates that constraint c is held at location l (details in Section 4.2). Figure 3 shows the abstract syntax of CHR^e rules and programs.

[2]Recall that the existential variables are not free.

Location names k

Location	l	::=	$x \mid k$				
Rule Heads	H	::=	$\cdot \mid [l]\,c, H$	Rewrite Rule	R	::=	$r : H \setminus H \iff G \mid B$
Rule Body	B	::=	$\exists \bar{x}.D$	Program	$\mathcal{P}$	::=	$R \mid R\,\mathcal{P}$
Rule Body Elements	D	::=	$true \mid [l]\,c, D$				

Figure 3: Abstract syntax of CHR^e

All constraints in a rule are now explicitly localized by the operator $[l]$. A localization operator in a rule head indicates the location where the constraint is to be matched, while a localization operator in a rule body indicates the location where that constraint is to be delivered. We call the former locations *matching locations* of the rule while the latter are *forwarding locations*.

CHR^e rules and programs are *well-formed* similarly to CHR rules and programs. Like all variables, location variables of rule bodies must appear as term arguments or localization operators of rule heads, or otherwise be existentially quantified. These "existential locations" represent new locations to be created during rule application and allows the specification of dynamically growing ensembles. We will only consider well-formed CHR^e rules from now on. Given a multiset of located constraints H, $\text{Locs}(H)$ denotes the set of locations that appear in the localization operators of H (e.g. $\text{Locs}([X]\,a(Y,Z), [Y]\,b(Z,2)) = \{X, Y\}$) and $\text{Args}(H)$ denotes the set of all term arguments of constraints in H (e.g. $\text{Args}([X]\,a(Y,Z), [Y]\,b(Z,2)) = \{Y, Z, 2\}$). We write $H_{|l}$ for the *location restriction* on H, which denotes the multiset of all constraints in H that is located at l (e.g. $([X]\,a(Y,Z), [Y]\,b(Z,2))_{|X} = a(Y,Z)$). We write $[l]\,H$ for the multiset of all constraints in H each prefixed with $[l]$. These operators are inductively defined as follows:

Retrieve Locations

$$\begin{cases} \text{Locs}([l]\,c, H) &= l \cup \text{Locs}(H) \\ \text{Locs}(\cdot) &= \emptyset \end{cases}$$

Retrieve Arguments

$$\begin{cases} \text{Args}([l]\,p(\vec{t}), H) &= \vec{t} \cup \text{Args}(H) \\ \text{Args}(\cdot) &= \emptyset \end{cases}$$

Location Restrict (Rule Heads)

$$\begin{cases} ([l]\,c, H)_{|l} &= c, H_{|l} \\ ([l']\,c, H)_{|l} &= H_{|l} \quad \text{if } l \neq l' \\ \cdot_{|l} &= \emptyset \end{cases}$$

Link restriction in distributed rule based languages [CARG$^+$12, LCG$^+$06] constrains how locations are used. We generalize it to the notion of n-neighbor restriction. A CHR^e rule $r : P \setminus S \iff G \mid B$ is *n-neighbor restricted* (where $n = |\text{Locs}(P,S)| - 1$) if we can select $l \in \text{Locs}(P,S)$ such that it is *directly connected* to each other n locations that appear in the rule heads. Furthermore, rule heads of locations other than l are *isolated* in that they do not contain common variables that do not appear at l. Rule guards also need to be *isolated* in a manner such that each atomic rule guard is grounded by the rule heads of location l and at most one other location of the rule head. Such a matching location l is called the *primary location* of the n-neighbor restricted rule, while all other matching locations of the rule are called *neighbor locations*. We assume that rule guards G have no side-effects and hence are pure boolean assertions. Formally, a CHR^e rule $r : P \setminus S \iff G \mid B$ is *n-neighbor restricted* (for $n = |\text{Locs}(P,S)| - 1$) if there exists $l \in \text{Locs}(P,S)$ such that the following three conditions are satisfied:

- *Directly connected:* $\text{Locs}(P,S) \subseteq \{l\} \cup \text{Args}((P,S)_{|l})$

- *Neighbor isolated rule heads:* For all other distinct $l', l'' \in \text{Locs}(P, S)$, for each $x \in \text{FV}((P, S)_{|l'}, (P, S)_{|l''})$, we have $x \in \text{FV}((P, S)_{|l})$.
- *Neighbor isolated rule guards:* For each guard condition $g \in G$, either we have $\text{FV}(g) \subseteq \text{FV}((P, S)_{|l})$ or for some other $l' \in \text{Locs}(P, S)$, we have $\text{FV}(g) \in (\text{FV}((P, S)_{|l}) \cup \text{FV}((P, S)_{|l'}))$.

n-neighbor restricted rules characterizes multiset rewritings across constraint stores of $n + 1$ connected locations in a 'star' topology with the *primary* location in the center, directly connected to each n neighbors. The neighbor isolation conditions (for rule heads and guards) are defined so as to make the matching problem decomposable into partial match problems between the primary location l and each of its n-neighbors separately (details in Section 6). In general, an n-neighbor restricted rule is of the following form:

$$r : \left(\biguplus_{i\in\mathcal{I}_n} [k_i]\, P_i\right) \setminus \left(\biguplus_{i\in\mathcal{I}_n} [k_i]\, S_i\right) \Longleftrightarrow G \mid \exists \bar{x}.\ \left(\biguplus_{i\in\mathcal{I}_n} [k_i]\, D_i\right), \left(\biguplus_{j\in\mathcal{I}_m} [k_j]\, D_j\right), \left(\biguplus_{l\in\mathcal{I}_e} [k_l]\, D_l\right)$$

where k_i for $i \in \mathcal{I}_n$ are *matching locations.*
k_j for $j \in \mathcal{I}_m$, $k_j \in \left(\text{FV}(\biguplus_{i\in\mathcal{I}_n} P_i) \cup \text{FV}(\biguplus_{i\in\mathcal{I}_n} S_i)\right)$ are *non-matched forwarding locations.*
k_l for $l \in \mathcal{I}_e$, $k_l \in \bar{x}$ are *existential forwarding locations.*

The matching locations of the rule r are the set of locations k_i labeled by $i \in \mathcal{I}_n$. These are the locations that will be involved in rule matching and correspond to the set of locations $\text{Locs}(\left(\biguplus_{i\in\mathcal{I}_n} [k_i]\, P_i, [k_i]\, S_i\right))$. We call each "$P_i, S_i$" belonging to a location k_i the *matching obligations* of k_i and assume that each location k_i has a non-empty matching obligation (i.e, either $P_i \neq \emptyset$ or $S_i \neq \emptyset$). All locations that appear in localization operators of the right-hand side of rule r are called forwarding locations. We classify them into three types: first we have *matched forwarding locations* in the rule body $\left(\biguplus_{i\in\mathcal{I}_n} [k_i]\, D_i\right)$ are such that each k_i is a matching location as well. Next, *non-matched forwarding locations* within $\left(\biguplus_{j\in\mathcal{I}_m} [k_j]\, D_j\right)$ are such that k_j is not a matching location, but appear as a term argument of some constraint in the rule head[3]. Finally, *existential forwarding locations* within $\left(\biguplus_{l\in\mathcal{I}_e} [k_l]\, D_l\right)$ are such that $k_l \in \bar{x}$, the set of existential variables. Note that this implicitly means that k_l neither is a matching location nor appear as a term argument of a rule head, hence it is a reference to a *new* location. We assume that the body of matched forwarding locations (i.e, D_i) may be empty but that of non-matched and existential forwarding locations must be non-empty (i.e, D_j and D_l). This assumption provide a more concise notation for n-neighbor restricted rules without superfluous references to locations that are not involved in the rule application[4]. We will refer to a n-neighbor restricted program as a general CHR^e program, or simply a CHR^e program. A well-formed CHR^e rule where heads are located at exactly one location are, by definition, 0-neighbor restricted. We call these *local rules.* A CHR^e program $\mathcal{P}$ is *n-neighbor restricted* if each rule in $\mathcal{P}$ are m-neighbor restricted for $m \leq n$.

As an example, consider the example program in Figure 1. Rules *base* and *elim* are examples of 0-neighbor restricted rules or local rules. This is because the each have rule heads that specify constraints from a single location. Rule *trans* however, has rule heads from two distinct locations X and Y. Specifically, rule heads $P = [X]\, edge(Y, D)$, $[Y]\, path(Z, D')$ and $S = \emptyset$, satisfying the directly connected condition because we have $edge(Y, D)$ located at X, while satisfying the neighbor isolation condition by default because the rule only has one neighboring matching location Y. Hence it is a 1-neighbor restricted rule. An interested observation is that the 1-neighbor restriction corresponds directly to the link restriction of [CARG$^+$12, LCG$^+$06].

[3]The *directly connected* condition of neighbor restriction dictates that the primary matching location would possess one such constraint.

[4]Note that a matched forwarding location k_i is allowed to have an empty rule body D_i because its presence is justified by its appearance as a matching location.

$$\begin{array}{lrcl} \text{Constraint Store} & \bar{\mathcal{S}} & ::= & \emptyset \mid \bar{\mathcal{S}}, c \\ \text{Abstract Ensemble} & \mathcal{A} & ::= & \emptyset \mid \langle \bar{\mathcal{S}} \rangle_k, \mathcal{A} \end{array}$$

$$\frac{\begin{array}{c} r : \left(\uplus_{i\in\mathcal{I}_n} [k'_i]\, P'_i\right) \setminus \left(\uplus_{i\in\mathcal{I}_n} [k'_i]\, S'_i\right) \Longleftrightarrow G \mid \exists \bar{x}.\ \bar{D} \in \mathcal{P} \\ \models \theta G \quad \left(\uplus_{i\in\mathcal{I}_n} P_i\right) = \theta\left(\uplus_{i\in\mathcal{I}_n} P'_i\right) \quad \left(\uplus_{i\in\mathcal{I}_n} S_i\right) = \theta\left(\uplus_{i\in\mathcal{I}_n} S'_i\right) \quad \left(\uplus_{i\in\mathcal{I}_n\cup\mathcal{I}_m} k_i\right) = \theta\left(\uplus_{i\in\mathcal{I}_n\cup\mathcal{I}_m} k'_i\right) \\ \left(\uplus_{i\in\mathcal{I}_n} [k_i]\, D_i, \uplus_{j\in\mathcal{I}_m} [k_j]\, D_j, \uplus_{l\in\mathcal{I}_e} [k_l]\, D_l\right) = \mathrm{NF}(\mathrm{Inst}(\theta(\exists \bar{x}.\ \bar{D}))) \end{array}}{\mathcal{P} \rhd \mathcal{A}, \left(\begin{array}{l} \uplus_{i\in\mathcal{I}_n} \langle \bar{\mathcal{S}}_i, P_i, S_i\rangle_{k_i}, \\ \uplus_{j\in\mathcal{I}_m} \langle \bar{\mathcal{S}}_j\rangle_{k_j} \end{array}\right) \mapsto_{\omega^e_\alpha} \mathcal{A}, \left(\begin{array}{l} \uplus_{i\in\mathcal{I}_n} \langle \bar{\mathcal{S}}_i, P_i, D_i\rangle_{k_i}, \\ \uplus_{j\in\mathcal{I}_m} \langle \bar{\mathcal{S}}_j, D_j\rangle_{k_j}, \\ \uplus_{l\in\mathcal{I}_e} \langle D_l\rangle_{k_l} \end{array}\right)}$$

$$\begin{array}{lll} \text{where} & k'_j \text{ for } j \in \mathcal{I}_m & \text{such that } k'_j \in \left(\mathrm{FV}(\uplus_{i\in\mathcal{I}_n} P_i, \uplus_{i\in\mathcal{I}_n} S_i)\right) \\ & k'_l \text{ for } l \in \mathcal{I}_e & \text{such that } k'_l \in \bar{x} \\ & \bar{D} \quad = & \left(\uplus_{i\in\mathcal{I}_n} [k'_i]\, D'_i\right), \left(\uplus_{j\in\mathcal{I}_m} [k'_j]\, D'_j\right), \left(\uplus_{l\in\mathcal{I}_e} [k'_l]\, D'_l\right) \end{array}$$

Figure 4: ω^e_α Abstract Semantics of CHR^e

4 Semantics of CHR^e

We define the ω^e_α abstract semantics of CHR^e in Section 4.1, which introduces a decentralized execution of CHR^e programs, and prove its soundness with respect to the ω_α semantics. Using ω^e_α as a stepping stone, Section 4.2 defines and proves the soundness of the ω^e_0 operational semantics that provides a more operational view of decentralized execution of CHR^e programs. Although this operational semantics only supports 0-neighbor restricted rules, Section 5 will show how we encode the arbitral CHR^e programs into 0-neighbor restricted programs. Note that in this work, we assume a lossless network, meaning that communication between locations are always eventually delivered and never lost.

4.1 ω^e_α Abstract Semantics

In this section, we introduce ω^e_α, an abstract decentralized semantics for CHR^e. This semantics accounts for the distributed nature of CHR^e, where each location has its own constraint store. This abstract semantics models a state transition system between abstract ensemble states, which are multisets of local stores $\langle \bar{\mathcal{S}} \rangle_k$ where $\bar{\mathcal{S}}$ is a constraint store and k a location name. An abstract ensemble states $\mathcal{A}$ is well-formed if all constraint stores $\bar{\mathcal{S}}$ that appear in it are well-formed and location names k are unique.

Figure 4 shows the ω^e_α semantics. Given a CHR^e program $\mathcal{P}$, we write a derivation step of ω^e_α as $\mathcal{P} \rhd \mathcal{A} \mapsto_{\omega^e_\alpha} \mathcal{A}'$ for abstract states $\mathcal{A}, \mathcal{A}'$. A derivation step defines the application of an n-neighbor restricted rules: Each of the $n+1$ locations k_i for $i \in \mathcal{I}_n$ provides a partial match P_i and S_i in their respective stores to their respective matching obligations (i.e, P'_i and S'_i), resulting in the combined substitution θ. If guard θG is valid, we apply the rule instance by removing S_i from the respective store of matching location k_i and replace them with the respective normalized rule body fragment (D_i of matching location k_i). For non-matching forwarding locations k_j for $j \in \mathcal{I}_m$, we simply add rule body fragment D_j to their stores. Finally, for existential forwarding locations, we create new location names k_l that contains just the rule body fragments D_l[5]. Since we assume that rule guards G have no side effects, $\models \theta G$ is a global assertion. Note that 0-neighbor restriction is the special case where matching is localized ($\mathcal{I}_n$ is a singleton set). We denote the reflexive and transitive application of derivation steps by $\mathcal{P} \rhd \mathcal{A} \mapsto^*_{\omega^e_\alpha} \mathcal{A}'$. ω^e_α derivation steps

[5]New location names are implicitly created by the instantiation of existential variables (i.e, Inst($-$)) applied to the rule body after substitution θ is applied.

$$\text{Abstract Ensembles} \quad \llbracket \mathcal{A}, \langle \bar{\mathcal{S}} \rangle_k \rrbracket = \llbracket \mathcal{A} \rrbracket, \llbracket \bar{\mathcal{S}} \rrbracket^k \qquad \llbracket \emptyset \rrbracket = \emptyset$$

$$\text{CHR Program} \quad \llbracket R\ \mathcal{P} \rrbracket = \llbracket R \rrbracket\ \llbracket \mathcal{P} \rrbracket \qquad \llbracket \cdot \rrbracket = \cdot$$

$$\text{CHR Rule} \quad \llbracket r : P \setminus S \iff G \mid B \rrbracket = r : \llbracket P \rrbracket \setminus \llbracket S \rrbracket \iff G \mid \llbracket B \rrbracket$$

$$\text{Constraints} \quad \llbracket p(\vec{t}) \rrbracket^l = p(l, \vec{t}) \qquad \text{Stores} \quad \begin{cases} \llbracket c, \bar{\mathcal{S}} \rrbracket^l = \llbracket c \rrbracket^l, \llbracket \bar{\mathcal{S}} \rrbracket^l \\ \llbracket \emptyset \rrbracket^l = \emptyset \end{cases}$$

$$\text{Body} \quad \begin{cases} \llbracket \exists \bar{x}.\ D \rrbracket = \exists \bar{x}. \llbracket D \rrbracket \\ \llbracket\, [l]\, c\,,\ D \rrbracket = \llbracket c \rrbracket^l\,,\ \llbracket D \rrbracket \\ \llbracket true \rrbracket = true \end{cases} \qquad \text{Head} \quad \begin{cases} \llbracket\, [l]\, c,\ H \rrbracket = \llbracket c \rrbracket^l,\ \llbracket H \rrbracket \\ \llbracket \cdot \rrbracket = \cdot \end{cases}$$

Figure 5: CHR Interpretation of CHR^e

preserves the well-formedness of states $\mathcal{A}$, provided that CHR^e programs are well-formed. Lemma 1 states this property of ω_α^e derivations.

Lemma 1 (Well-Formedness Preservation of ω_α^e Derivations) *Given a well-formed CHR^e program $\mathcal{P}$, a well-formed state $\mathcal{A}$ and state $\mathcal{A}'$, if $\mathcal{P} \rhd \mathcal{A} \mapsto^*_{\omega_\alpha^e} \mathcal{A}'$ then $\mathcal{A}'$ is well-formed.*

From here on, we will implicitly assume the well-formedness of $\mathcal{P}$, $\mathcal{A}$ and $\mathcal{A}'$, when writing $\mathcal{P} \rhd \mathcal{A} \mapsto^*_{\omega_\alpha^e} \mathcal{A}'$.

We now relate the CHR^e abstract semantics ω_α^e to the CHR abstract semantics ω_α. Figure 5 defines a function $\llbracket - \rrbracket$ that inductively traverses the structure of a CHR^e syntactic construct and translates it into a CHR construct that represents its CHR interpretation. For a CHR^e rule, this function translates located constraints $[l]\, p(\vec{t})$ to standard CHR constraints by inserting location l as the first (leftmost) term argument of the predicate (i.e, $p(l, \vec{t})$). We call these *location interpreted CHR constraints*. An abstract ensemble state $\mathcal{A}$ is interpreted in CHR simply by collapsing all constraint stores in $\mathcal{A}$ into a single global constraint store, containing location interpreted CHR constraints. The translation function is defined so that given a well-formed CHR^e syntactic object o, $\llbracket o \rrbracket$ is a well-formed syntactic object of CHR. Lemma 2 states this property of the translation function $\llbracket - \rrbracket$.

Lemma 2 (Well-Formedness Preservation of $\llbracket - \rrbracket$ Translation) *Given a well-formed CHR^e object o, $\llbracket o \rrbracket$ is a well-formed CHR object.*

Theorem 3 states the soundness of ω_α^e, namely the translation function $\llbracket - \rrbracket$ preserves deriviability.

Theorem 3 (Soundness of ω_α^e) *Given a CHR^e program $\mathcal{P}$ and abstract states $\mathcal{A}$ and $\mathcal{A}'$, if $\mathcal{P} \rhd \mathcal{A} \mapsto^*_{\omega_\alpha^e} \mathcal{A}'$, then $\llbracket \mathcal{P} \rrbracket \rhd \llbracket \mathcal{A} \rrbracket \mapsto^*_{\omega_\alpha} \llbracket \mathcal{A}' \rrbracket$.*

4.2 ω_0^e Operational Semantics

This section introduces the operational semantics ω_0^e. Similarly to ω_α^e, this semantics specifies a distributed execution for CHR^e programs. Unlike ω_α^e, it is *operational* in that it describes the execution of CHR^e programs in a procedural manner with a clear and concise execution strategy for each location of the ensemble. As such, it comprises of more derivation rules, each of which is dedicated to a specific sub-task of decentralized multiset rewriting. The ω_0^e semantics is an extension of the refined CHR operational semantics [DSdlBH04], adapted to describe an execution model for decentralized and incremental multiset matching[6].

[6]It is incremental in that multiset matches are processed incrementally from new information (constraints). This is a property that is crucial for any effective execution model for distributed rule-based systems.

Stored constraint id d Rule Occurrence Index i

$$\begin{array}{llcl} \text{Goals} & g & ::= & b \mid c \mid c\#d:i \\ \text{Ids} & \vec{D} & ::= & \emptyset \mid d, \vec{D} \\ \text{Buffers} & \vec{\mathcal{U}} & ::= & \emptyset \mid c, \vec{\mathcal{U}} \end{array} \qquad \begin{array}{llcl} \text{Goals} & \vec{\mathcal{G}} & ::= & \emptyset \mid g, \vec{\mathcal{G}} \\ \text{Numbered Store} & \bar{\mathcal{S}} & ::= & \emptyset \mid \bar{\mathcal{S}}, c\#d \\ \text{History} & \bar{\mathcal{H}} & ::= & \emptyset \mid \bar{\mathcal{H}}, (\vec{D}) \end{array}$$

$$\text{Operational Ensembles} \quad \Omega ::= \emptyset \mid \Omega, \langle \vec{\mathcal{U}} \ ; \ \vec{\mathcal{G}} \ ; \ \bar{\mathcal{S}} \ ; \ \bar{\mathcal{H}} \rangle_k$$

$$\text{Retrieve Locations} \quad \begin{cases} \text{Locs}(\Omega, \langle \vec{\mathcal{U}} \ ; \ \vec{\mathcal{G}} \ ; \ \bar{\mathcal{S}} \ ; \ \bar{\mathcal{H}} \rangle_k) & = \quad \text{Locs}(\Omega) \cup \{k\} \\ \text{Locs}(\emptyset) & = \quad \emptyset \end{cases}$$

Figure 6: ω_0^e Ensemble States

As such, it shares many meta constructs with the refined CHR operational semantics (e.g. rule head occurrence index, goals, numbered constraints, history). We refer the interested reader to [DSdlBH04] for a more detailed treatment of the refined operational semantics of CHR. The ω_0^e semantics applies only to 0-neighbor restricted rules. While we could easily have included n-neighbor restricted rule execution as a derivation step of ω_0^e, doing so would not capture the operational challenges of synchronizing multiple locations during rule application. Instead we compile n-neighbor restricted rules for $n > 1$ into 0-neighbor restricted rules in Section 5.

Figure 6 defines the states of the ω_0^e semantics. An ω_0^e ensemble Ω is a set of tuples of the form $\langle \vec{\mathcal{U}} \ ; \ \vec{\mathcal{G}} \ ; \ \bar{\mathcal{S}} \ ; \ \bar{\mathcal{H}} \rangle_k$ which represent the state of the computing entity at location k. The *buffer* $\vec{\mathcal{U}}$ is a sequence of the constraints that have been sent to location k. The *goals* $\vec{\mathcal{G}}$ is a sequence of the constraints c or active constraints $c\#d:i$. The *numbered store* $\bar{\mathcal{S}}$ is a multiset of numbered constraints $c\#d$. The index d serves as a reference link between $c\#d$ in the store and an active constraint $c\#d:i$ in the goals. The *history* $\bar{\mathcal{H}}$ is a set of indices where each element is a unique set of constraint ids $(\vec{D})$. Note the use of accents to explicitly indicate the nature of each collection of objects. Buffers is a novel extension of this semantics while goals, numbered store and history are artifacts of the refined operational semantics of CHR. Also, like the refined operational semantics of CHR, we assume that each rule head in a CHR program have a unique rule occurrence index i, representing the sequence (typically textual order of appearance) in which the rule head is matched to an active constraint. We write occurrence index i as a subscript of the rule head (i.e, $\texttt{[}l\texttt{]}\, c_i$). A state Ω is well-formed if each $\langle \vec{\mathcal{U}} \ ; \ \vec{\mathcal{G}} \ ; \ \bar{\mathcal{S}} \ ; \ \bar{\mathcal{H}} \rangle_k \in \Omega$ has a unique location name k, all objects in $\vec{\mathcal{U}}$, $\vec{\mathcal{G}}$ and $\bar{\mathcal{S}}$ are ground and all term expressions that appear in them are well-formed. Furthermore we require that each goals $\vec{\mathcal{G}}$ has at most one active constraint $(c\#d:i)$ found at the head of $\vec{\mathcal{G}}$ with a corresponding $c\#d$ found in the store $\bar{\mathcal{S}}$. Ω is *initial* if all $\vec{\mathcal{U}} = \emptyset$, $\bar{\mathcal{S}} = \emptyset$ and $\bar{\mathcal{H}} = \emptyset$ and *terminal* if all $\vec{\mathcal{U}} = \emptyset$ and $\vec{\mathcal{G}} = \emptyset$. We extend the meta operation $\text{Locs}(-)$ to the domain of operational states Ω, namely that $\text{Locs}(\Omega)$ returns the set of all locations k that appear in Ω.

We define three meta operations that will be used in the ω_0^e semantics: Given a CHR numbered store $\bar{\mathcal{S}}$, $\text{DropIds}(\bar{\mathcal{S}})$ returns the multiset of all constraints in $\bar{\mathcal{S}}$ without their numbered ids; $\text{Ids}(\bar{\mathcal{S}})$ returns the set of all constraint ids that appear in $\bar{\mathcal{S}}$; given a CHR program $\mathcal{P}$, $\text{OccIds}(\mathcal{P})$ returns the set of all rule head occurrence indices that appear in each rule $R \in \mathcal{P}$. These meta operations are inductively defined as follows:

$$\text{(Flush)}\quad \frac{\vec{\mathcal{U}} \neq \emptyset}{\mathcal{P} \rhd \Omega, \langle \vec{\mathcal{U}} \;;\; \emptyset \;;\; \bar{\mathcal{S}} \;;\; \bar{\mathcal{H}} \rangle_k \mapsto_{\omega_0^e} \Omega, \langle \emptyset \;;\; \vec{\mathcal{U}} \;;\; \bar{\mathcal{S}} \;;\; \bar{\mathcal{H}} \rangle_k}$$

$$\text{(Loc 1)}\quad \frac{}{\mathcal{P} \rhd \Omega, \left(\begin{array}{l} \langle \vec{\mathcal{U}} \;;\; ([k']c, \vec{\mathcal{G}}) \;;\; \bar{\mathcal{S}} \;;\; \bar{\mathcal{H}} \rangle_k, \\ \langle \vec{\mathcal{U}}' \;;\; \vec{\mathcal{G}}' \;;\; \bar{\mathcal{S}}' \;;\; \bar{\mathcal{H}}' \rangle_{k'} \end{array} \right) \mapsto_{\omega_0^e} \Omega, \left(\begin{array}{l} \langle \vec{\mathcal{U}} \;;\; \vec{\mathcal{G}} \;;\; \bar{\mathcal{S}} \;;\; \bar{\mathcal{H}} \rangle_k, \\ \langle (\vec{\mathcal{U}}', [c]) \;;\; \vec{\mathcal{G}}' \;;\; \bar{\mathcal{S}}' \;;\; \bar{\mathcal{H}}' \rangle_{k'} \end{array} \right)}$$

$$\text{(Loc 2)}\quad \frac{}{\mathcal{P} \rhd \Omega, \langle \vec{\mathcal{U}} \;;\; ([k]c, \vec{\mathcal{G}}) \;;\; \bar{\mathcal{S}} \;;\; \bar{\mathcal{H}} \rangle_k \mapsto_{\omega_0^e} \Omega, \langle \vec{\mathcal{U}} \;;\; (c, \vec{\mathcal{G}}) \;;\; \bar{\mathcal{S}} \;;\; \bar{\mathcal{H}} \rangle_k}$$

$$\text{(Loc 3)}\quad \frac{k \neq k' \qquad k' \notin \mathrm{Locs}(\Omega)}{\mathcal{P} \rhd \Omega, \langle \vec{\mathcal{U}} \;;\; ([k']c, \vec{\mathcal{G}}) \;;\; \bar{\mathcal{S}} \;;\; \bar{\mathcal{H}} \rangle_k \mapsto_{\omega_0^e} \Omega, \langle \vec{\mathcal{U}} \;;\; \vec{\mathcal{G}} \;;\; \bar{\mathcal{S}} \;;\; \bar{\mathcal{H}} \rangle_k, \langle c \;;\; \emptyset \;;\; \emptyset \;;\; \emptyset \rangle_{k'}}$$

$$\text{(Act)}\quad \frac{d \text{ is a fresh id}}{\mathcal{P} \rhd \Omega, \langle \vec{\mathcal{U}} \;;\; (p(\vec{t}), \vec{\mathcal{G}}) \;;\; \bar{\mathcal{S}} \;;\; \bar{\mathcal{H}} \rangle_k \mapsto_{\omega_0^e} \Omega, \langle \vec{\mathcal{U}} \;;\; (p(\vec{t})\#d : 1, \vec{\mathcal{G}}) \;;\; (\bar{\mathcal{S}}, p(\vec{t})\#d) \;;\; \bar{\mathcal{H}} \rangle_k}$$

$$\text{(Simp)}\quad \frac{\begin{array}{c} r : [l]\,P' \setminus [l]\,(S', c_i', S'') \Longleftrightarrow G \mid B \in \mathcal{P} \qquad \models \theta \wedge G \qquad k = \theta l \\ \mathrm{DropIds}(P) = \theta P' \qquad \mathrm{DropIds}(S) = \theta(S', S'') \qquad c = \theta c' \end{array}}{\mathcal{P} \rhd \Omega, \langle \vec{\mathcal{U}} \;;\; (c\#d : i, \vec{\mathcal{G}}) \;;\; (\bar{\mathcal{S}}, P, S, c\#d) \;;\; \bar{\mathcal{H}} \rangle_k \mapsto_{\omega_0^e} \Omega, \langle \vec{\mathcal{U}} \;;\; (\mathrm{NF}(\mathrm{Inst}(\theta B)), \vec{\mathcal{G}}) \;;\; (\bar{\mathcal{S}}, P) \;;\; \bar{\mathcal{H}} \rangle_k}$$

$$\text{(Prop)}\quad \frac{\begin{array}{c} r : [l]\,(P', c_i', P'') \setminus [l]\,S' \Longleftrightarrow G \mid B \in \mathcal{P} \qquad \models \theta \wedge G \qquad k = \theta l \\ \mathrm{DropIds}(P) = \theta(P', P'') \qquad \mathrm{DropIds}(S) = \theta S' \qquad c = \theta c' \qquad (d, \mathrm{Ids}(P, S)) \notin \bar{\mathcal{H}} \end{array}}{\begin{array}{rl} \mathcal{P} \rhd & \Omega, \langle \vec{\mathcal{U}} \;;\; (c\#d : i, \vec{\mathcal{G}}) \;;\; (\bar{\mathcal{S}}, P, S, c\#d) \;;\; \bar{\mathcal{H}} \rangle_k \\ \mapsto_{\omega_0^e} & \Omega, \langle \vec{\mathcal{U}} \;;\; (\mathrm{NF}(\mathrm{Inst}(\theta B)), c\#d : i, \vec{\mathcal{G}}) \;;\; (\bar{\mathcal{S}}, P, c\#d) \;;\; (\bar{\mathcal{H}}, (d, \mathrm{Ids}(P, S))) \rangle_k \end{array}}$$

$$\text{(Next)}\quad \frac{\text{(Simp) and (Prop) do not apply for } c\#d : i}{\mathcal{P} \rhd \Omega, \langle \vec{\mathcal{U}} \;;\; (c\#d : i, \vec{\mathcal{G}}) \;;\; \bar{\mathcal{S}} \;;\; \bar{\mathcal{H}} \rangle_k \mapsto_{\omega_0^e} \Omega, \langle \vec{\mathcal{U}} \;;\; (c\#d : (i+1), \vec{\mathcal{G}}) \;;\; \bar{\mathcal{S}} \;;\; \bar{\mathcal{H}} \rangle_k}$$

$$\text{(Drop)}\quad \frac{i \notin \mathrm{OccIds}(\mathcal{P})}{\mathcal{P} \rhd \Omega, \langle \vec{\mathcal{U}} \;;\; (c\#d : i, \vec{\mathcal{G}}) \;;\; \bar{\mathcal{S}} \;;\; \bar{\mathcal{H}} \rangle_k \mapsto_{\omega_0^e} \Omega, \langle \vec{\mathcal{U}} \;;\; \vec{\mathcal{G}} \;;\; \bar{\mathcal{S}} \;;\; \bar{\mathcal{H}} \rangle_k}$$

Figure 7: ω_0^e Operational Semantics for CHR^e

$$\text{Drop Ids}\quad \begin{cases} \mathrm{DropIds}(\bar{\mathcal{S}}, c\#d) &= \mathrm{DropIds}(\bar{\mathcal{S}}), c \\ \mathrm{DropIds}(\emptyset) &= \emptyset \end{cases}$$

$$\text{Retrieve Ids}\quad \begin{cases} \mathrm{Ids}(\bar{\mathcal{S}}, c\#d) &= \mathrm{Ids}(\bar{\mathcal{S}}), d \\ \mathrm{Ids}(\emptyset) &= \emptyset \end{cases}$$

$$\text{Retrieve Occ Indices}\quad \begin{cases} \mathrm{OccIds}(R\ \mathcal{P}) &= \mathrm{OccIds}(R), \mathrm{OccIds}(\mathcal{P}) \\ \mathrm{OccIds}(r : P \setminus S \Longleftrightarrow G \mid B) &= \mathrm{OccIds}(P), \mathrm{OccIds}(S) \\ \mathrm{OccIds}([l]\,c_i,\ H) &= i, \mathrm{OccIds}(H) \\ \mathrm{OccIds}(\cdot) &= \emptyset \end{cases}$$

Figure 7 defines the ω_0^e operational semantics. Given a 0-neighbor restricted CHR^e program $\mathcal{P}$, a ω_0^e derivation step expresses a transition between ensemble states, written $\mathcal{P} \rhd \Omega \mapsto_{\omega_0^e} \Omega'$. Execution in a

location $\langle \vec{\mathcal{U}} \ ; \ \vec{\mathcal{G}} \ ; \ \bar{\mathcal{S}} \ ; \ \bar{\mathcal{H}} \rangle_k \in \Omega$ is mainly driven by the goals $\vec{\mathcal{G}}$ which function as a stack of procedures waiting to be executed. By contrast $\vec{\mathcal{U}}$ buffers the constraints sent to the location. The (Flush) step states that constraints in a non-empty buffer $\vec{\mathcal{U}}$ are to be moved into the goals if the current goal is empty. (Loc 1), (Loc 2) and (Loc 3) model the delivery of body constraint $[k']c$ to forwarding location k': (Loc 1) applies if the leading goal is $[k']c$ and k' is distinct from the origin location k, and sends c to the buffer $\vec{\mathcal{U}}'$ of k'. For (Loc 2) the forwarding location is the origin location, hence no actual transmission occur and the localization operator $[k]$ is simply stripped away. Finally for (Loc 3) with localization operator $[k']$, k' does not appear anywhere in the ensemble, hence we create a new location k' with just c in the buffer and all other collections empty[7]. (Act) applies to a leading goal of the form $p(\vec{t})$. It introduces $p(\vec{t}) : d$ into the store, where d is a fresh constraint identifier, and puts the active constraint $p(\vec{t})\#d : 1$ as the new leading goal. This represents the initialization of rule matching rooted at $p(\vec{t})\#d$ starting from the rule head in the program $\mathcal{P}$ that corresponds to the first occurrence index. The last four derivation steps apply to leading goals of the form $c\#d : i$. (Prop) and (Simp) model the application of a 0-neighbor restricted CHR rule instance $R \in \mathcal{P}$ such that the i^{th} rule head occurrence matches c and respective partner constraints are found in the store. (Prop) additionally enforces a history check similar to the traditional CHR semantics [DSdlBH04, Sch05]. The purpose of this is to disallow multiple applications of rule instances that originate from the same multiset of constraints in the store[8]. The derivation (Next) increments the occurrence index of the active constraint, while (Drop) removes the active constraint from the goal once it has been tried on all rule head occurrences. We define the transitive and reflexive application of ω_0^e derivation steps as $\mathcal{P} \rhd \Omega \mapsto^*_{\omega_0^e} \Omega'$. A state Ω' is *reachable* by a program $\mathcal{P}$ under the ω_0^e semantics if there exists some initial state Ω such that $\mathcal{P} \rhd \Omega \mapsto^*_{\omega_0^e} \Omega'$.

We are interested in a class of CHR^e programs known as *locally quiescent* CHR^e programs. We say that a CHR^e program $\mathcal{P}$ is locally quiescent if given any well-formed reachable state Ω, we cannot have any infinite derivation sequences that *does not include* the (Flush) derivation step. This specifically means that each location k in a Ω must always (eventually and asynchronously) execute to a state where its goals are empty, during which the (Flush) step is applicable and the constraints in the buffer will be pushed into the goals. Locally quiescent programs have the property that constraints sent across locations are not left to "starve" in a location's buffer, because local execution of a location k is guaranteed to be terminating. Hence it is a form of progress guarantee that each location will eventually process (activate, store and match) each constraint delivered to it. While we expect that it is reasonable that distributed applications can possibly behave in a non *globally quiescent* manner [9], we expect that each location executes in a locally quiescent manner as described here. We will explicitly consider locally quiescent programs in Section 5.

Lemma 4 states the property that ω_0^e derivations preserves the well-formedness of the operational states Ω.

Lemma 4 (Well-Formedness Preservation of ω_0^e Derivations) *Given a well-formed 0-neighbor restricted CHR^e program $\mathcal{P}$, a well-formed state Ω and state Ω', if $\mathcal{P} \rhd \Omega \mapsto^*_{\omega_0^e} \Omega'$ then Ω' is well-formed.*

Given a state Ω, we define a meta operation Goals(Ω) that denotes the consolidated sequence of all goals in Ω. We extend the notion of location retrieval and location restriction on goals: Locs($\vec{\mathcal{G}}$) denotes the set of distinct locations k such that $[k]c \in \vec{\mathcal{G}}$ for some c. Given location k, $\vec{\mathcal{G}}_{|k}$ denotes the sequence containing all constraints c where $[k]c \in \vec{\mathcal{G}}$. These are inductively defined by the following:

[7] New locations are introduced by existential forwarding locations in rule bodies.

[8] History checking is only required on propagation rules ($r : P \setminus S \iff G \mid B$ where $S = \emptyset$). For brevity, we conservatively apply history checking to all states that applies to the (Prop) derivation step.

[9] Globally quiescent: collective execution of the ensemble terminates. In other words, there exists a reachable state where all locations of the ensemble reaches quiescence.

$$\text{Ensembles} \quad \begin{cases} \lceil \Omega \rceil = \lceil \Omega \rceil^{\vec{\mathcal{G}}'}, \biguplus_{k \in (\text{Locs}(\vec{\mathcal{G}}') - \text{Locs}(\Omega))} \langle \vec{\mathcal{G}}'_{|k} \rangle_k & \text{where } \vec{\mathcal{G}}' = \text{Goals}(\Omega) \\ \lceil \Omega, \langle \vec{\mathcal{U}} \ ; \ \vec{\mathcal{G}} \ ; \ \bar{\mathcal{S}} \ ; \ \bar{\mathcal{H}} \rangle_k \rceil^{\vec{\mathcal{G}}'} = \lceil \Omega \rceil^{\vec{\mathcal{G}}'}, \langle \lceil \vec{\mathcal{U}} \rceil, \lceil \vec{\mathcal{G}} \rceil, \lceil \bar{\mathcal{S}} \rceil, \vec{\mathcal{G}}'_{|k} \rangle_k \\ \lceil \emptyset \rceil^{\vec{\mathcal{G}}'} = \emptyset \end{cases}$$

$$\begin{array}{rl} \text{Identity} & \lceil \emptyset \rceil = \emptyset \\ \text{Buffers} & \lceil c, \vec{\mathcal{U}} \rceil = c, \lceil \vec{\mathcal{U}} \rceil \\ \text{Stores} & \lceil c\#d, \bar{\mathcal{S}} \rceil = c, \lceil \bar{\mathcal{S}} \rceil \end{array} \qquad \text{Goals} \quad \begin{cases} \lceil c, \vec{\mathcal{G}} \rceil = c, \lceil \vec{\mathcal{G}} \rceil \\ \lceil c\#d : i, \vec{\mathcal{G}} \rceil = \lceil \vec{\mathcal{G}} \rceil \\ \lceil \texttt{[}k\texttt{]}\, c, \vec{\mathcal{G}} \rceil = \lceil \vec{\mathcal{G}} \rceil \end{cases}$$

Figure 8: Abstract Ensemble State Interpretation of Operational Ensemble States

$$\text{Consolidate Goals} \quad \begin{cases} \text{Goals}(\Omega, \langle \vec{\mathcal{U}} \ ; \ \vec{\mathcal{G}} \ ; \ \bar{\mathcal{S}} \ ; \ \bar{\mathcal{H}} \rangle_k) = \text{Goals}(\Omega), \vec{\mathcal{G}} \\ \text{Goals}(\emptyset) = \emptyset \end{cases}$$

$$\text{Retrieve Locations (Goals)} \quad \begin{cases} \text{Locs}(c, \vec{\mathcal{G}}) = \text{Locs}(\vec{\mathcal{G}}) \\ \text{Locs}(\texttt{[}k\texttt{]}\, c, \vec{\mathcal{G}}) = \{k\} \cup \text{Locs}(\vec{\mathcal{G}}) \\ \text{Locs}(\emptyset) = \emptyset \end{cases}$$

$$\text{Location Restrict (Goals)} \quad \begin{cases} c, \vec{\mathcal{G}}_{|k} = \vec{\mathcal{G}}_{|k} \\ \texttt{[}k'\texttt{]}\, c, \vec{\mathcal{G}}_{|k} = \vec{\mathcal{G}}_{|k} & \text{if } k \neq k' \\ \texttt{[}k\texttt{]}\, c, \vec{\mathcal{G}}_{|k} = c, \vec{\mathcal{G}}_{|k} \\ \emptyset_{|k} = \emptyset \end{cases}$$

Figure 8 gives the translation function $\lceil - \rceil$ that inductively traverses the structure of an operational ensemble state Ω and translates it to a corresponding fragment of an abstract state $\mathcal{A}$. At the top-most level, $\lceil \Omega \rceil$ is equivalent to $\lceil \Omega \rceil^{\vec{\mathcal{G}}'}, \biguplus_{k \in (\text{Locs}(\vec{\mathcal{G}}') - \text{Locs}(\Omega))} \langle \vec{\mathcal{G}}'_{|k} \rangle_k$ such that $\vec{\mathcal{G}}' = \text{Goals}(\Omega)$ is the consolidated sequence of all goals in the ensemble. The first component ($\lceil \Omega \rceil^{\vec{\mathcal{G}}'}$) takes each location in Ω and collapses $\vec{\mathcal{U}}$, $\vec{\mathcal{G}}$ and $\bar{\mathcal{S}}$ together while dropping the history $\bar{\mathcal{H}}$ entirely. Kept as a superscript is the sequence of all goals $\vec{\mathcal{G}}'$. Each location k also extracts from $\vec{\mathcal{G}}'$ all located constraints that belongs to k (i.e, $\vec{\mathcal{G}}'_{|k}$). Buffer $\vec{\mathcal{U}}$ is interpreted as a multiset of constraints. For $\vec{\mathcal{G}}$, we discard active constraints $c\#d : i$ because well-formed states have a corresponding $c\#d$ in $\bar{\mathcal{S}}$. We also discard located constraints $\texttt{[}k\texttt{]}\, c$. For $\bar{\mathcal{S}}$, we strip away constraint ids $\#d$. We implicitly convert sequences ($\vec{\mathcal{U}}$ and $\vec{\mathcal{G}}$) into multisets. The second component $\biguplus_{k \in (\text{Locs}(\vec{\mathcal{G}}') - \text{Locs}(\Omega))} \langle \vec{\mathcal{G}}'_{|k} \rangle_k$ retrieves the set of all locations k that are referenced in the goals but are not known locations in Ω, and "creates" these new locations each containing their respective fragment of the goals (i.e, $\vec{\mathcal{G}}'_{|k}$). Lemma 5 states the property that this translation maps a well-formed operational state into a well-formed abstract state.

Lemma 5 (Well-Formedness Preservation of $\lceil - \rceil$ Translation) *Given a well-formed ω_0^e operational state Ω, $\lceil \Omega \rceil$ is a well-formed ω_α^e abstract state.*

Theorem 6 states that ω_0^e derivations can be mapped to corresponding ω_α^e derivations via the $\lceil - \rceil$ interpretation of operational states.

Theorem 6 (Soundness of ω_0^e) *Given 0-neighbor restricted CHR^e program $\mathcal{P}$ and states Ω and Ω', if $\mathcal{P} \rhd \Omega \mapsto^*_{\omega_0^e} \Omega'$, then $\mathcal{P} \rhd \lceil \Omega \rceil \mapsto^*_{\omega_\alpha^e} \lceil \Omega' \rceil$.*

Note that a terminal state Ω (where all $\vec{\mathcal{U}}_i = \emptyset$ and $\vec{\mathcal{G}}_i = \emptyset$) is in a state of *quiescence* where no ω_0^e

$$\text{(Single)} \qquad \frac{\mathcal{P} \ \rhd \ \Omega \mapsto_{\omega_0^e} \Omega'}{\mathcal{P} \ \rhd \ \Omega \ \mapsto^{||}_{\omega_0^e} \ \Omega'}$$

$$\text{(Concurrent)} \qquad \frac{\mathcal{P} \ \rhd \ (\Omega_1, \Omega_2) \ \mapsto^{||}_{\omega_0^e} \ (\Omega_1', \Omega_2) \qquad \mathcal{P} \ \rhd \ (\Omega_1, \Omega_2) \ \mapsto^{||}_{\omega_0^e} \ (\Omega_1, \Omega_2')}{\mathcal{P} \ \rhd \ (\Omega_1, \Omega_2) \ \mapsto^{||}_{\omega_0^e} \ (\Omega_1', \Omega_2')}$$

Figure 9: Concurrent ω_0^e Derivation Steps

derivation steps can apply. The ω_0^e semantics guarantees that when we reach quiescence from a well-formed initial state, all rules in the program have been *exhaustively applied*.

To prove this guarantee of exhaustive rule application in quiescence of ω_0^e derivations, we first define the notion of rule applicability: Given a 0-neighbor restricted rule $R = r : P' \setminus S' \iff G \mid B$, we say that R *is applicable* in a state Ω, if there exists some $\langle \vec{\mathcal{U}} \ ; \ \vec{\mathcal{G}} \ ; \ \bar{\mathcal{S}} \ ; \ \bar{\mathcal{H}} \rangle_k \in \Omega$ and some substitution θ where there exists $P, S \in \bar{\mathcal{S}}$ such that $\text{DropIds}(P) = \theta P'$ and $\text{DropIds}(S) = \theta S'$, $\models \theta G$ and $\text{Ids}(P, S) \notin \bar{\mathcal{H}}$. The rule instance of R in Ω is said to be *active* if for its rule head instance, P and S, there exists some '$c\#d \in P, S$' that matches the j^{th} occurrence index of R, such that $c\#d : i \in \vec{\mathcal{G}}$ and $i \leq j$. Lemma 7 states the property that all rule instances in a reachable state are active.

Lemma 7 (Always Active Rule Head Instances) *Given a* 0*-neighbor restricted CHR^e program $\mathcal{P}$ and an initial state Ω, for any reachable state Ω' such that $\mathcal{P} \rhd \Omega \mapsto^*_{\omega_0^e} \Omega'$, all rule instances in Ω' must be active.*

Theorem 8 states the exhaustiveness of rule application in the ω_0^e semantics. Specifically, it state that starting from an initial state Ω, if we derive a terminal state from Ω, this terminal state has no rule instances. This is an important property because it means that exhaustive execution of ω_0^e derivations of a CHR^{e}program $\mathcal{P}$ to a state of quiescence results to the exhaustive application of all rule instances of $\mathcal{P}$.

Theorem 8 (Exhaustiveness of Rule Application in ω_0^e) *Given a* 0*-neighbor restricted CHR^e program $\mathcal{P}$ and reachable states Ω and Ω', if $\mathcal{P} \ \rhd \ \Omega \mapsto^*_{\omega_0^e} \Omega'$ and Ω' is terminal, then there exists no rule instance $R \in \mathcal{P}$ such that R is applicable.*

Figure 9 define the concurrent ω_0^e derivation step, denoted $\mapsto^{||}_{\omega_0^e}$. It explicitly defines concurrent execution of ω_0^e semantics derivation steps that modifies non-overlapping portions of an operational state. The (Single) rule states that a standard ω_0^e derivation (i.e, $\mapsto_{\omega_0^e}$) maps naively to a concurrent derivation step, $\mapsto^{||}_{\omega_0^e}$. The rule (Concurrent) states that we can compose concurrent derivation steps together, as long as they operate on non-overlapping portions of an operational state. We write $\mathcal{P} \ \rhd \ \Omega \mapsto^{||*}_{\omega_0^e} \Omega'$ for the reflexive and transitive application of $\mapsto^{||}_{\omega_0^e}$ derivation steps. The composibility of ω_0^e concurrent derivation steps stems from a more fundamental property of the abstract CHR semantics known as *monotonicity* [Frü94]. Specifically, it states that CHR derivations are valid even in a larger context of constraints. The ω_0^e semantics has this property as well, but in a slightly different context: ω_0^e derivations are *monotonic* with-respect-to locations, namely in a larger context of locations[10]. Lemma 9 states this property in the ω_0^e semantics in its various forms. We assume implicit α-renaming of locations when composing ensemble states together.

Lemma 9 (Monotonicity of ω_0^e semantics) *Given a* 0*-neighbor restricted program $\mathcal{P}$, and reachable states $\Omega, \Omega', \Omega''$,*

1. *if $\mathcal{P} \ \rhd \ \Omega \ \mapsto_{\omega_0^e} \ \Omega'$, then $\mathcal{P} \ \rhd \ \Omega, \Omega'' \ \mapsto_{\omega_0^e} \ \Omega', \Omega''$*

[10]Note that the refined operational semantics [DSdlBH04] does not have this property with-respect-to constraint stores, but it is monotonic with-respect-to its goals.

2. *if* $\mathcal{P} \ \rhd \ \Omega \ \mapsto^*_{\omega_0^e} \ \Omega'$, *then* $\mathcal{P} \ \rhd \ \Omega, \Omega'' \ \mapsto^*_{\omega_0^e} \ \Omega', \Omega''$

3. *if* $\mathcal{P} \ \rhd \ \Omega \ \mapsto^{||}_{\omega_0^e} \ \Omega'$, *then* $\mathcal{P} \ \rhd \ \Omega, \Omega'' \ \mapsto^{||}_{\omega_0^e} \ \Omega', \Omega''$

4. *if* $\mathcal{P} \ \rhd \ \Omega \ \mapsto^{||*}_{\omega_0^e} \ \Omega'$, *then* $\mathcal{P} \ \rhd \ \Omega, \Omega'' \ \mapsto^{||*}_{\omega_0^e} \ \Omega', \Omega''$

With the monotonicity property stated in Lemma 9, we can prove Theorem 10, that states that every concurrent ω_0^e derivation can be simulated using sequential ω_0^e derivation steps.

Theorem 10 (Serializability of Concurrent ω_0^e Derivations) *Given a* 0*-neighbor restricted program* $\mathcal{P}$*, and reachable states* Ω, Ω'*, if we have* $\mathcal{P} \ \rhd \ \Omega \ \mapsto^{||*}_{\omega_0^e} \ \Omega'$ *then we must have* $\mathcal{P} \ \rhd \ \Omega \ \mapsto^*_{\omega_0^e} \ \Omega'$

Theorem 10 has far reaching implications on the concurrent behavior of ω_0^e: Specifically that 0-neighbor restricted CHR^{e}rule application *need not be* executed entirely *atomically*, since the sequence of ω_0^e derivation steps that models a rule application can be interleaved between multiple ω_0^e derivation steps. Rather, *rule head observation and deletion* (observing presence of all rule head constraints in the store and deleting the matching simplified rule heads) is the critical component of rule application that needs to be atomic[11]. More concretely, it is an application of either the (Simp) or (Prop) ω_0^e derivation step followed by a sequence of (Act), (Next) and (Drop) derivation steps that incrementally "moves" the body B of the rule instance into the constraint store. Between this sequence of derivation steps, we can possibly execute other concurrent derivation steps that involve rewritings on non-overlapping portions of the local constraint store or ensemble. Yet these concurrent derivations of non-atomic rule applications can be serialized to interleaving sequential ω_0^e derivation steps. We will exploit this observation in our encoding of 1-neighbor restricted rules into ω_0^e (Section 5).

5 Encoding 1-Neighbor Restricted Programs for ω_0^e

In this section, we define a translation that transforms a 1-neighbor restricted program into a 0-neighbor restricted program. Given a 1-neighbor restricted rule $r : \texttt{[}X\texttt{]}P_x\texttt{,[}Y\texttt{]}P_y \setminus \texttt{[}X\texttt{]}S_x\texttt{,[}Y\texttt{]}S_y \iff G \mid B$, we designate X as the primary location and Y as the neighbor location. We define two properties of a 1-neighbor restricted rule r, namely that r is *primary propagated* if $S_x = \emptyset$ and that r is *neighbor propagated* if $S_y = \emptyset$. We call P_x and S_x the *primary matching obligations*, while P_y and S_y are the *neighbor matching obligations*. We only consider locally quiescent CHR^e programs from this section.

5.1 Basic Encoding Scheme

Figure 10 illustrates our basic encoding scheme, $\rightsquigarrow^{\text{basic}}_{\text{1Nb}}$ on an example. It applies to a 1-neighbor restricted rule that is neither primary nor neighbor propagated. We assume that constraints of the predicate *neighbor* are never deleted, i.e, no other rules in a program with the *swap* rule have a *neighbor* constraint as a simplified head. We call such a constraint a *persistent* constraint. A constraint c is *persistent* if there is no substitution θ such that $\theta c = \theta c'$ for some rule $r : P \setminus S \iff G \mid B \in \mathcal{P}$ and constraint $c' \in S$. Note that the rule heads demand that we must *atomically observe* that in some location X we have $neighbor(Y)$ and $color(C)$, while in Y we have $color(C')$. This observation must be *atomic* in the sense that the observation of X's and Y's matching obligations should not be interrupted by an interleaving concurrent derivation.

The 0-neighbor restricted encoding of the *swap* rule (Figure 10) recovers this atomicity: In *swap_1* X sends a swap request $swap\_req(X, Y, C)$ to Y if it possesses the primary matching obligation of the

[11]This property is first observed in [SL08, LS11] and we call this atomic fragment of rule application *atomic rule head verification*.

$$\mathit{swap} : \left(\begin{array}{c} [X]\,\mathit{neighbor}(Y), \\ \cdot \end{array} \right) \setminus \left(\begin{array}{l} [X]\,\mathit{color}(C), \\ [Y]\,\mathit{color}(C') \end{array} \right) \Longleftrightarrow \left(\begin{array}{l} [X]\,\mathit{color}(C'), \\ [Y]\,\mathit{color}(C) \end{array} \right)$$

$$\downarrow \text{basic 1Nb}$$

$$\left(\begin{array}{l} \mathit{swap\_1} : [X]\,\mathit{neighbor}(Y), [X]\,\mathit{color}(C) \Longrightarrow [Y]\,\mathit{swap\_req}(X,Y,C) \\ \mathit{swap\_2} : [Y]\,\mathit{color}(C') \setminus [Y]\,\mathit{swap\_req}(X,Y,C) \Longleftrightarrow [X]\,\mathit{swap\_match}(X,Y,C,C') \\ \mathit{swap\_3} : [X]\,\mathit{neighbor}(Y) \setminus [X]\,\mathit{color}(C), [X]\,\mathit{swap\_match}(X,Y,C,C') \\ \qquad\qquad \Longleftrightarrow [Y]\,\mathit{swap\_commit}(X,Y,C,C') \\ \mathit{swap\_4a} : [Y]\,\mathit{swap\_commit}(X,Y,C,C'), [Y]\,\mathit{color}(C') \Longleftrightarrow [X]\,\mathit{color}(C'), [Y]\,\mathit{color}(C) \\ \mathit{swap\_4b} : [Y]\,\mathit{swap\_commit}(X,Y,C,C') \Longleftrightarrow [X]\,\mathit{color}(C) \end{array} \right)$$

Figure 10: Color Swapping Example: 1-Neighbor Restricted Rule

swap rule. In *swap_2* if Y observes this request together with a $\mathit{color}(C')$, it responds to X by sending $\mathit{swap\_match}(X,Y,C,C')$. Note that in this example, $\mathit{swap\_req}(X,Y,C)$ is simplified since X's matching obligation is not primary propagated[12]. In *swap_3*, X must observe that it has a response from Y ($\mathit{swap\_match}(X,Y,C,C')$) and that its matching obligations are still valid[13], then it sends a commit request to Y $\mathit{swap\_commit}(X,Y,C,C')$. This is the point where X actually *commits* to the match by consuming $\mathit{color}(C)$. From here there are two possibilities: *swap_4a* considers that, if Y still possesses the matching instance of $\mathit{color}(C')$, we complete the execution of *swap* by delivering its rule body. *swap_4b* considers the alternative case where Y no longer has $\mathit{color}(C')$ hence we cannot commit to the rule instance. Hence we *roll back* X's commitment by returning $\mathit{color}(C)$ to X. Note that ω_0^e's sequencing of rule occurrences is vital to ensure that given a *swap_commit* instance, rule matching for *swap_4a* is always attempted before *swap_4b*. We call the predicates introduced in the encoding (like *swap_req*, *swap_match*) *synchronizing predicates* and constraints they form *synchronizing constraints*.

The five 0-neighbor restricted rules in Figure 10 implement an *asynchronous* and *optimistic* synchronization protocol between two locations of the ensemble. It is asynchronous because neither primary X nor neighbor Y ever "blocks" or busy-waits for responses. Rather they communicate asynchronously via the synchronizing constraints, while potentially interleaving with other derivation steps. It is optimistic because non-synchronizing constraints are only ever consumed after both X and Y have independently observed their respective fragment of the rule head instance.

Figure 11 defines another example of a 1-neighbor restricted rule that differs from the *swap* example in two ways: It is *primary propagated* and it contain a primary propagated head (namely $\mathit{color}(C)$) that is not *persistent*. We highlight in boxes the fragments which differ by the fact that the rule is *primary propagated*, and with an underline the fragments which differ by the fact that propagated head $\mathit{color}(C)$ is not *persistent*.

Since *prop* is primary propagated, it is possible that a single instance of this rule head fragment be applied to multiple instances of $[Y]\,\mathit{color}(C')$ for a particular location Y. If we follow a similar encoding to the *swap_2* rule for the previous example (Figure 10) we cannot guarantee exhaustiveness of 1-neighbor restricted rule application. Therefore, as highlighted in a box in figure 11, rule *prop_2* is defined such that the synchronizing constraint $\mathit{prop\_req}(X,Y,C)$ is propagated as opposed to being simplified (highlighted in a box).

[12]If it was primary propagated (by making $[X]\,\mathit{color}(C)$ of the *swap* rule propagated instead), we would have to propagate $\mathit{swap\_req}(X,Y,C)$ instead, since X's matching obligation can match and apply to multiple instances of *swap* (see Figure 11 for such an example).

[13]This revalidation ensures that Y's observation of its matching obligation has not been invalidated by an interleaving rule application that consumed any part of X's obligations.

$$prop : \left(\begin{array}{c} [X]\, neighbor(Y), \underline{[X]\, color(C)} \\ \cdot \end{array} \right) \setminus \left(\begin{array}{c} \boxed{\cdot} \\ [Y]\, color(C') \end{array} \right) \Longleftrightarrow [Y]\, color(C)$$

$$\downarrow \rightsquigarrow^{\text{basic}}_{\text{1Nb}}$$

$$\left(\begin{array}{l} prop1 : [X]\, neighbor(Y), [X]\, color(C) \Longrightarrow [Y]\, prop\_req(X,Y,C) \\ prop2 : \boxed{[Y]\, color(C'), [Y]\, prop\_req(X,Y,C) \Longrightarrow} [X]\, prop\_match(X,Y,C,C') \\ prop3 : [X]\, neighbor(Y) \setminus \underline{[X]\, color(C)}, [X]\, prop\_match(X,Y,C,C') \\ \qquad\qquad \Longleftrightarrow [Y]\, prop\_commit(X,Y,C,C') \\ prop4a : [Y]\, prop\_commit(X,Y,C,C'), [Y]\, color(C') \Longleftrightarrow \underline{[X]\, color(C)}, [Y]\, color(C) \\ prop4b : [Y]\, prop\_commit(X,Y,C,C') \Longleftrightarrow \underline{[X]\, color(C)} \end{array} \right)$$

Figure 11: Color Propagation Example: Primary Propagated 1-Neighbor Restricted Rule

Since *prop* has a non persistent propagated head, in order to safely guarantee that the observation of X and Y matching obligations are done independently, *prop3* commits its obligation by deleting its non persistent propagated constraint(s), in this case $color(C)$, while in *prop4a* and *prop4b*, this constraint is returned to X. While this possibly introduces additional overhead, it is crucial to ensuring the safety of this rule application. Figure 12 that shows an example of the same rule *prop* of Figure 11 but with an encoding that treats non-persistent propagated head $color(C)$ as persistent. This encoding is flawed because we can derive *prop* rule applications from the program encoding $\mathcal{P}_0$, while $\mathcal{P}_1$ has no valid derivation. Specifically, Figure 12 highlights the steps that leads to a wrong derivation. We consider a contrived program with two additional rules *cheat1* and *cheat2* that shuffles a constraint *color* between two neighboring locations. The initial state starts with only location k_1 having a $color(blue)$ constraint, but with proper timing of the *cheat* rules, we can shuffle the single *color* constraint and derive the application of rule *prop*.

Figure 13 defines the basic encoding scheme for 1-neighbor restricted rule. It is denoted by $R_1 \rightsquigarrow^{\text{basic}}_{\text{1Nb}} \mathcal{P}_0$, where R_1 is a well-formed 1-neighbor restricted rule while $\mathcal{P}_0$ a well-formed 0-neighbor restricted program. The propagated rule heads of the primary location X are split into two, namely P_x containing all rule heads which are persistent in $\mathcal{P}_1$ and P'_x containing all those which are non-persistent. P'_x and S_x will be consumed in *r_3* when the primary location commits, this effectively "locks" the primary matching obligation. The neighbor location Y applies *r_4a* to complete the rule application r, specifically adding the rule body D and "unlocking" P'_x, the primary propagated rule heads which are not persistent. If it is unable to complete this rule application, *r_4b* is applied to roll-back location X's commit attempt by returning P'_x and S_x to X. MatchRule$(\cdot)$ characterizes the fragment of the encoding unique to primary propagated 1-neighbor restricted rules, while MatchRule(S_x) for a non-empty S_x represents the corresponding fragment of non-primary propagated rules. Xs and Ys are the set of variables of the primary and neighbor location rule heads while Rs is the union of the two. Rule guards are divided into two parts, namely G_x the primary rule guards and G_y the neighbor rule guards. Primary rule guards G_x are all the guard conditions that are grounded by Xs, while G_y are the rest of the guards. We refer to these rules (r_i) as *encoding rules*. We define the meta operator DropSyncs$(\bar{\mathcal{S}}; \mathcal{P})$ that returns the multiset of all constraints that appear in $\bar{\mathcal{S}}$ that are not synchronizing constraints of $\mathcal{P}$:

$$\mathcal{P}_1 = \left\{ \begin{array}{l} \mathit{prop} : \left(\begin{array}{c} \texttt{[}X\texttt{]}\,\mathit{neighbor}(Y)\,\texttt{,}\,\underline{\texttt{[}X\texttt{]}\,\mathit{color}(C)} \\ \cdot \end{array} \right) \setminus \left(\begin{array}{c} \boxed{\cdot} \\ \texttt{[}Y\texttt{]}\,\mathit{color}(C') \end{array} \right) \Longleftrightarrow \texttt{[}Y\texttt{]}\,\mathit{color}(C) \\ \mathit{cheat1} : \texttt{[}X\texttt{]}\,\mathit{neighbor}(Y) \setminus \texttt{[}X\texttt{]}\,\mathit{color}(\mathit{blue}) \Longleftrightarrow \texttt{[}Y\texttt{]}\,\mathit{color}(\mathit{red}) \\ \mathit{cheat2} : \texttt{[}X\texttt{]}\,\mathit{neighbor}(Y) \setminus \texttt{[}X\texttt{]}\,\mathit{color}(\mathit{red}) \Longleftrightarrow \texttt{[}Y\texttt{]}\,\mathit{color}(\mathit{blue}) \end{array} \right.$$

$$\mathcal{P}_0 = \left\{ \begin{array}{l} \mathit{prop1} : \texttt{[}X\texttt{]}\,\mathit{neighbor}(Y)\,\texttt{,}\,\texttt{[}X\texttt{]}\,\mathit{color}(C) \Longrightarrow \texttt{[}Y\texttt{]}\,\mathit{prop\_req}(X,Y,C) \\ \mathit{prop2} : \texttt{[}Y\texttt{]}\,\mathit{color}(C')\,\texttt{,}\,\texttt{[}Y\texttt{]}\,\mathit{prop\_req}(X,Y,C) \Longrightarrow \texttt{[}X\texttt{]}\,\mathit{prop\_match}(X,Y,C,C') \\ \mathit{prop3} : \texttt{[}X\texttt{]}\,\mathit{neighbor}(Y)\,\texttt{,}\,\underline{\texttt{[}X\texttt{]}\,\mathit{color}(C)} \setminus \texttt{[}X\texttt{]}\,\mathit{prop\_match}(X,Y,C,C') \\ \qquad\qquad \Longleftrightarrow \texttt{[}Y\texttt{]}\,\mathit{prop\_commit}(X,Y,C,C') \\ \mathit{prop4a} : \texttt{[}Y\texttt{]}\,\mathit{prop\_commit}(X,Y,C,C')\,\texttt{,}\,\texttt{[}Y\texttt{]}\,\mathit{color}(C') \Longleftrightarrow \underline{\texttt{[}Y\texttt{]}\,\mathit{color}(C)} \\ \mathit{prop4b} : \texttt{[}Y\texttt{]}\,\mathit{prop\_commit}(X,Y,C,C') \Longleftrightarrow \underline{\mathit{true}} \\ \mathit{cheat1} : \texttt{[}X\texttt{]}\,\mathit{neighbor}(Y) \setminus \texttt{[}X\texttt{]}\,\mathit{color}(\mathit{blue}) \Longleftrightarrow \texttt{[}Y\texttt{]}\,\mathit{color}(\mathit{red}) \\ \mathit{cheat2} : \texttt{[}X\texttt{]}\,\mathit{neighbor}(Y) \setminus \texttt{[}X\texttt{]}\,\mathit{color}(\mathit{red}) \Longleftrightarrow \texttt{[}Y\texttt{]}\,\mathit{color}(\mathit{blue}) \end{array} \right.$$

$$\begin{array}{ll} \mathcal{P}_0 \ \triangleright & \langle \boxed{\mathit{neighbor}(k_2), \mathit{color}(\mathit{blue})} \rangle_{k_1}, \langle \mathit{neighbor}(k_1) \rangle_{k_2} \\ \mapsto^{\mathit{prop1}}_{\omega^e_\alpha} & \langle \boxed{\mathit{neighbor}(k_2), \mathit{color}(\mathit{blue})} \rangle_{k_1}, \langle \mathit{neighbor}(k_1), \mathit{prop\_req}(k_1, k_2, \mathit{blue}) \rangle_{k_2} \\ \mapsto^{\mathit{cheat1}}_{\omega^e_\alpha} & \langle \mathit{neighbor}(k_2) \rangle_{k_1}, \langle \mathit{neighbor}(k_1), \boxed{\mathit{prop\_req}(k_1, k_2, \mathit{blue}), \mathit{color}(\mathit{red})} \rangle_{k_2} \\ \mapsto^{\mathit{prop2}}_{\omega^e_\alpha} & \langle \mathit{neighbor}(k_2), \mathit{prop\_match}(k_1, k_2, \mathit{blue}, \mathit{red}) \rangle_{k_1}, \langle \boxed{\mathit{neighbor}(k_1)}, \mathit{prop\_req}(k_1, k_2, \mathit{blue}), \boxed{\mathit{color}(\mathit{red})} \rangle_{k_2} \\ \mapsto^{\mathit{cheat2}}_{\omega^e_\alpha} & \langle \boxed{\mathit{neighbor}(k_2), \mathit{prop\_match}(k_1, k_2, \mathit{blue}, \mathit{red}), \mathit{color}(\mathit{blue})} \rangle_{k_1}, \langle \mathit{neighbor}(k_1), \mathit{prop\_req}(k_1, k_2, \mathit{blue}) \rangle_{k_2} \\ \mapsto^{\mathit{prop3}}_{\omega^e_\alpha} & \langle \boxed{\mathit{neighbor}(k_2), \mathit{color}(\mathit{blue})} \rangle_{k_1}, \langle \mathit{neighbor}(k_1), \mathit{prop\_req}(k_1, k_2, \mathit{blue}), \mathit{prop\_commit}(k_1, k_2, \mathit{blue}, \mathit{red}) \rangle_{k_2} \\ \mapsto^{\mathit{cheat1}}_{\omega^e_\alpha} & \langle \mathit{neighbor}(k_2) \rangle_{k_1}, \langle \mathit{neighbor}(k_1), \mathit{prop\_req}(k_1, k_2, \mathit{blue}), \boxed{\mathit{prop\_commit}(k_1, k_2, \mathit{blue}, \mathit{red}), \mathit{color}(\mathit{red})} \rangle_{k_2} \\ \mapsto^{\mathit{prop4a}}_{\omega^e_\alpha} & \langle \mathit{neighbor}(k_2) \rangle_{k_1}, \langle \mathit{neighbor}(k_1), \mathit{prop\_req}(k_1, k_2, \mathit{blue}), \mathit{color}(\mathit{blue}) \rangle_{k_2} \end{array}$$

Figure 12: Example of Flawed Encoding of Rule with Non-persistent Propagated Head

$$\begin{array}{lcl} \text{SyncPred}((r : P \setminus S \Longleftrightarrow G \mid B)\ \mathcal{P}) & = & r\_req, r\_match, r\_commit, r\_abort, \text{SyncPred}(\mathcal{P}) \\ \text{SyncPred}(\cdot) & = & \emptyset \end{array}$$

$$\text{DropSyncs}(p(\bar{t}), \bar{\mathcal{S}}; \mathcal{P}) = \begin{cases} p(\bar{t}), \text{DropSyncs}(\bar{\mathcal{S}}; \mathcal{P}) & \text{if } p \notin \text{SyncPred}(\mathcal{P}) \\ \text{DropSyncs}(\bar{\mathcal{S}}; \mathcal{P}) & \text{otherwise} \end{cases}$$

$$\text{DropSyncs}(\emptyset; \mathcal{P}) = \emptyset$$

As illustrated in Figure 13, we generalize the application of this translation to 1-neighbor restricted *programs*, thus given a 1-neighbor restricted program $\mathcal{P}_1$, we have its encoding via $\mathcal{P}_1 \rightsquigarrow^{\text{basic}}_{\text{1Nb}} \mathcal{P}_0$, such that the encoding operation is applied to each 1-neighbor restricted rule in $\mathcal{P}_1$ while 0-neighbor restricted rules are simply left unmodified. All rule encodings (each a 0-neighbor program) are then concatenated into a

$$(r : [X]P_x, [X]P'_x, [Y]P_y \setminus [X]S_x, [Y]S_y \Longleftrightarrow G_x, G_y \mid \exists \bar{z}.\ D)$$

$$\overset{\sim}{\rightarrow}_{\text{1Nb}}^{\text{basic}}$$

$$\left(\begin{array}{l} r\_1 : [X]P_x, [X]S_x \Longrightarrow G_x \mid [Y]r\_req(Xs) \\ \text{MatchRule}(S_x) \\ r\_3 : [X]P_x \setminus [X]P'_x, [X]S_x, [X]r\_match(Rs) \Longleftrightarrow [Y]r\_commit(Rs) \\ r\_4a : [Y]P_y \setminus [Y]S_y, [Y]r\_commit(Rs) \Longleftrightarrow \exists \bar{z}.\ [X]P'_x, D \\ r\_4b : [Y]r\_commit(Rs) \Longleftrightarrow [X]P'_x, [X]S_x \end{array}\right)$$

where
All $c \in P_x$ are persistent constraints and all $c' \in P'_x$ are non-persistent constraints
$\text{MatchRule}(.) = r\_2 : [Y]P_y, [Y]S_y, [Y]r\_req(Xs) \Longrightarrow G_y \mid [X]r\_match(Rs)$
$\text{MatchRule}(S_x) = r\_2 : [Y]P_y, [Y]S_y \setminus [Y]r\_req(Xs) \Longleftrightarrow G_y \mid [X]r\_match(Rs) \quad \text{if } S_x \neq .$
$Xs = \text{FV}(P_x, P'_x, S_x) \qquad Ys = \text{FV}(P_y, S_y) \qquad Rs = \text{FV}(P_x, P'_x, S_x, P_y, S_y)$
$\text{FV}(G_x) \subseteq \text{FV}(P_x, P'_x, S_x) \qquad \text{FV}(G_y) \subseteq \text{FV}(P_x, P'_x, S_x, P_y, S_y)$

$$\text{(1-neighbor)} \quad \frac{\mathcal{P} \rightsquigarrow_{\text{1Nb}}^{\text{basic}} \mathcal{P}' \qquad R_1 \rightsquigarrow_{\text{1Nb}}^{\text{basic}} \mathcal{P}_0 \qquad R_1 \text{ is 1-neighbor restricted}}{(\mathcal{P}\ R_1) \rightsquigarrow_{\text{1Nb}}^{\text{basic}} (\mathcal{P}'\ \mathcal{P}_0)}$$

$$\text{(0-neighbor)} \quad \frac{\mathcal{P} \rightsquigarrow_{\text{1Nb}}^{\text{basic}} \mathcal{P}' \qquad R_0 \text{ is 0-neighbor restricted}}{(\mathcal{P}\ R_0) \rightsquigarrow_{\text{1Nb}}^{\text{basic}} (\mathcal{P}'\ R_0)} \qquad \text{(identity)} \quad \frac{}{. \rightsquigarrow_{\text{1Nb}}^{\text{basic}} .}$$

Figure 13: Basic Encoding of 1-Neighbor Restricted Rules

single 0-neighbor restricted program $\mathcal{P}_0$. We assume that unique rule head occurrence indices are issued in order of rule head appearance in $\mathcal{P}_0$. When required for specific discussions, we will denote a (Simp) or (Prop) derivation step that involves the application of a r_i encoding rule instance as $\mathcal{P} \rhd \Omega \mapsto_{\omega_0^e}^{r\_i} \Omega'$, where Ω and Ω' are the states before and after the application of r_i.

It is possible that a partial sequences of encoding rules (r_i) is executed in a ω_0^e derivation. For instance, primary location X can apply *r_1* but never receives a reply from neighbor location Y with *r_2* because Y does not possess the matching obligations required to complete the rule instance. Or similarly, Y can apply *r_2* in response to X's instance of *r_1*, but never receives a reply from X with *r_3* because X no longer possess matching obligations. Such partial sequences of execution are the side-effect of asynchrony in this synchronization protocol, and are *benign* in that they do not rewrite (delete or insert) non-synchronizing constraints. Furthermore, their only observable effects are the introduction of synchronizing constraints *r_req* and *r_match* whose only purpose and effect is the sequencing and staging of the flow of consensus building between locations X and Y. Lemma 11 states this property that derivation steps of each r_i encoding rule must have been preceded by derivation steps of matching instances of encoding rules that come before i.

Lemma 11 (Prefix Executions of Encoding) *Given a locally quiescent 1-neighbor restricted program $\mathcal{P}_1$ and a 0-neighbor restricted program $\mathcal{P}_0$ such that $\mathcal{P}_1 \rightsquigarrow_{\text{1Nb}}^{\text{basic}} \mathcal{P}_0$, if we have reachable states Ω and Ω' such that $\mathcal{P}_0 \rhd \Omega \mapsto_{\omega_0^e} \Omega'$, then we have the following:*

1. *If $\mathcal{P}_0 \rhd \Omega \mapsto_{\omega_0^e}^{r\_2} \Omega'$, then there exists some reachable states Ω'' and Ω''', such that $\mathcal{P}_0 \rhd \Omega'' \mapsto_{\omega_0^e}^{r\_1} \Omega'''$ and $\mathcal{P}_0 \rhd \Omega''' \mapsto_{\omega_0^e}^{*} \Omega$*
2. *If $\mathcal{P}_0 \rhd \Omega \mapsto_{\omega_0^e}^{r\_3} \Omega'$, then there exists some reachable states Ω'' and Ω''', such that $\mathcal{P}_0 \rhd \Omega'' \mapsto_{\omega_0^e}^{r\_2} \Omega'''$ and $\mathcal{P}_0 \rhd \Omega''' \mapsto_{\omega_0^e}^{*} \Omega$*

3. *If* $\mathcal{P}_0 \rhd \Omega \mapsto^{r\_4a}_{\omega_0^e} \Omega'$, *then there exists some reachable states* Ω'' *and* Ω''', *such that* $\mathcal{P}_0 \rhd \Omega'' \mapsto^{r\_3}_{\omega_0^e} \Omega'''$ *and* $\mathcal{P}_0 \rhd \Omega''' \mapsto^*_{\omega_0^e} \Omega$

4. *If* $\mathcal{P}_0 \rhd \Omega \mapsto^{r\_4b}_{\omega_0^e} \Omega'$, *then there exists some reachable states* Ω'' *and* Ω''', *such that* $\mathcal{P}_0 \rhd \Omega'' \mapsto^{r\_3}_{\omega_0^e} \Omega'''$ *and* $\mathcal{P}_0 \rhd \Omega''' \mapsto^*_{\omega_0^e} \Omega$

We now consider the soundness of this encoding. Specifically, the soundness condition that we need is that the ω_0^e derivations of the 0-neighbor restricted encodings of a 1-neighbor restricted program $\mathcal{P}$ derive valid states computable by $\mathcal{P}$ in the ω_α^e semantics. However, not all states derived by our encodings are such valid states: It is possible that the ω_0^e derivations of the 0-neighbor restricted encodings derive intermediate states in which a 1-neighbor restricted rule instance is partially applied. Specifically, after application of *r_3* only the X matching obligation is consumed, hence the rule instance is only partially applied. For this reason, the encodings are defined such that these intermediate states always contain the *r_commit* synchronizing constraint. We call such states *non-commit free* states and *commit free* states are all other states that do not contain *r_commit*. A state Ω is *commit free* if and only if for all $\langle \bar{\mathcal{S}} \rangle_k \in \lceil \Omega \rceil$, all $p(\vec{t}) \in \bar{\mathcal{S}}$ is such that $p \neq r\_commit$. Note we use $\lceil \Omega \rceil$ for the convenience of collapsing the contents of buffers, goals and stores of the ω_0^e semantics into one abstract store of the ω_α^e semantics. By definition of the $\lceil - \rceil$ translation, this effectively means that all buffers, goals and store in the operational state Ω must not contain any commit synchronizing constraints in order for Ω to qualify as a commit-free state. An important property of this encoding is that non-commit free states can always be eventually returned to a commit free state by applying either *r_4a* or *r_4b*, resulting to the complete execution of r or a roll-back to the state before its execution attempt, respectively. Note that this is only true for CHR^e programs which are locally quiescent: The reason is because if a location's execution is not locally quiescent, it might never push buffered constraints into the goals (via the (Flush) derivation step of ω_0^e semantics) and hence might remain non-commit free indefinitely.

Lemma 12 states that the encoding operation $\rightsquigarrow^{\text{basic}}_{\text{1Nb}}$ preserves local quiescence of CHR^e programs.

Lemma 12 ($\rightsquigarrow^{\text{basic}}_{\text{1Nb}}$ Preserves Local Quiescence) *Given a locally quiescent* 1*-neighbor restricted program* $\mathcal{P}_1$ *and a* 0*-neighbor restricted program* $\mathcal{P}_0$ *such that* $\mathcal{P}_1 \rightsquigarrow^{\text{basic}}_{\text{1Nb}} \mathcal{P}_0$, *then* $\mathcal{P}_0$ *is also locally quiescent.*

Lemma 13 states the property that non-commit free states can always eventually derive commit free state.

Lemma 13 (1-Neighbor Commit-Free Reachability) *Given a locally quiescent* 1*-neighbor restricted program* $\mathcal{P}_1$ *and a* 0*-neighbor restricted program* $\mathcal{P}_0$ *such that* $\mathcal{P}_1 \rightsquigarrow^{\text{basic}}_{\text{1Nb}} \mathcal{P}_0$ *and states* Ω *reachable by* $\mathcal{P}_0$, *if* Ω *is not commit-free, then there exists some commit-free state* Ω' *such that* $\mathcal{P}_0 \rhd \Omega \mapsto^*_{\omega_0^e} \Omega'$.

Lemma 14 states that given any ω_α^e derivation between two commit free state $\mathcal{P}_0 \rhd \mathcal{A} \mapsto^*_{\omega_\alpha^e} \mathcal{A}'$, we can safely permute it such that all applications of *r_3* encoding rules are immediately followed by either *r_4a* or *r_4b*. This lemma is proven by using the monotonicity property of the ω_α^e semantics (Lemma 9).

Lemma 14 (Basic Encoding Rule Serializability) *Given a* 1*-neighbor restricted and locally quiescent CHR*^{e} *program* $\mathcal{P}_1$ *and a* 0*-neighbor restricted CHR*^{e} *program* $\mathcal{P}_0$ *such that* $\mathcal{P}_1 \rightsquigarrow_{\text{1Nb}} \mathcal{P}_0$ *and commit free abstract states* $\mathcal{A}_1$ $\mathcal{A}_2$, $\mathcal{A}_3$ *and* $\mathcal{A}_4$, *given that:*

1. *For some basic encoding rule instances r_3 and r_4a, we have* $\mathcal{P}_0 \rhd \mathcal{A}_1 \mapsto^{r\_3}_{\omega_\alpha^e} \mathcal{A}_2 \mapsto^*_{\omega_\alpha^e} \mathcal{A}_3 \mapsto^{r\_4a}_{\omega_\alpha^e} \mathcal{A}_4$ *then there exists some* $\mathcal{A}'_2$ *and* $\mathcal{A}'_3$ *such that* $\mathcal{P}_0 \rhd \mathcal{A}_1 \mapsto^*_{\omega_\alpha^e} \mathcal{A}'_2 \mapsto^{r\_3}_{\omega_\alpha^e} \mathcal{A}_3 \mapsto^{r\_4a}_{\omega_\alpha^e} \mathcal{A}'_3 \mapsto^*_{\omega_\alpha^e} \mathcal{A}_4$

2. *For some basic encoding rule instances r_3 and r_4b, we have* $\mathcal{P}_0 \rhd \mathcal{A}_1 \mapsto^{r\_3}_{\omega_\alpha^e} \mathcal{A}_2 \mapsto^*_{\omega_\alpha^e} \mathcal{A}_3 \mapsto^{r\_4b}_{\omega_\alpha^e} \mathcal{A}_4$ *then there exists some* $\mathcal{A}'_2$ *and* $\mathcal{A}'_3$ *such that* $\mathcal{P}_0 \rhd \mathcal{A}_1 \mapsto^*_{\omega_\alpha^e} \mathcal{A}'_2 \mapsto^{r\_3}_{\omega_\alpha^e} \mathcal{A}_3 \mapsto^{r\_4b}_{\omega_\alpha^e} \mathcal{A}'_3 \mapsto^*_{\omega_\alpha^e} \mathcal{A}_4$

Theorem 15 asserts the soundness of the basic encoding: It states that ω_0^e derivations between commit free states of 0-neighbor restricted encodings have a mapping to ω_α^e derivations of its original 1-neighbor restricted program.

$$blend : \left(\begin{array}{c} [X]\, neighbor(Y), \\ [Y]\, pallete(C') \end{array} \right) \setminus \left(\begin{array}{c} [X]\, color(C) \\ \boxed{\quad \cdot \quad} \end{array} \right) \Longleftrightarrow [Y]\, color(C + C')$$

$$\downarrow \overset{\text{n-persist}}{\underset{\text{1Nb}}{\rightsquigarrow}}$$

$$\left(\begin{array}{l} blend1 : [X]\, neighbor(Y), [X]\, color(C) \Longrightarrow [Y]\, blend\_req(X, Y, C) \\ blend2 : [Y]\, pallete(C') \setminus [Y]\, blend\_req(X, Y, C) \Longleftrightarrow [X]\, blend\_match(X, Y, C, C') \\ blend3 : [X]\, neighbor(Y) \setminus [X]\, color(C), [X]\, blend\_match(X, Y, C, C') \Longleftrightarrow [X]\, color(C + C') \end{array} \right)$$

$$(r : [X]\, P_x, [Y]\, P_y \setminus [X]\, S_x \Longleftrightarrow G_x, G_y \mid B)$$

$$\downarrow \overset{\text{n-persist}}{\underset{\text{1Nb}}{\rightsquigarrow}}$$

$$\left(\begin{array}{l} r\_1 : [X]\, P_x, [X]\, S_x \Longrightarrow G_x \mid [Y]\, r\_req(Xs) \\ \text{MatchRule}(S_x) \\ r\_3 : [X]\, P_x \setminus [X]\, S_x, [X]\, r\_match(Rs) \Longleftrightarrow B \end{array} \right)$$

where
All $c \in P_y$ are persistent constraints
$\text{MatchRule}(.) = r\_2 : [Y]\, P_y, [Y]\, r\_req(Xs) \Longrightarrow G_y \mid [X]\, r\_match(Rs)$
$\text{MatchRule}(S_x) = r\_2 : [Y]\, P_y \setminus [Y]\, r\_req(Xs) \Longleftrightarrow G_y \mid [X]\, r\_match(Rs) \quad \text{if } S_x \neq .$
$Xs = \text{FV}(P_x, S_x)$ and $\text{FV}(G_x) \subseteq \text{FV}(P_x, S_x)$ and $\text{FV}(G_y) \subseteq \text{FV}(P_x, P_y, S_y)$

Figure 14: Optimized Encoding for Neighbor Persistent Rule

Theorem 15 (Soundness of Basic Encoding) *Given a 1-neighbor restricted and locally quiescent CHR^e program $\mathcal{P}_1$ and a 0-neighbor restricted CHR^e program $\mathcal{P}_0$ such that $\mathcal{P}_1 \rightsquigarrow_{\text{1Nb}}^{\text{basic}} \mathcal{P}_0$, for any reachable states Ω and Ω', if $\mathcal{P}_0 \rhd \Omega \mapsto_{\omega_0^e}^* \Omega'$, then we have either Ω' is not commit free or $\mathcal{P}_1 \rhd \text{DropSyncs}(\lceil \Omega \rceil; \mathcal{P}_1) \mapsto_{\omega_\alpha^e}^* \text{DropSyncs}(\lceil \Omega'' \rceil; \mathcal{P}_1)$.*

5.2 Optimizations

We define optimized encoding schemes for two special cases. Consider the basic encoding scheme in Figure 13. Suppose that we apply to it a 1-neighbor restricted rule that is neighbor propagated (i.e, $S_y = \emptyset$) and furthermore P_y is persistent. We call such rules *neighbor persistent*. This encoding executes an unnecessary indirection of sending *r_commit* to neighbor Y in *r_3* and completing the rule instance r with either *r_4a* or *r_4b*. However if $S_y = \emptyset$, primary location X can immediately complete the rule instance at *r_3* without further communications with Y. This specialized encoding scheme is denoted by $\rightsquigarrow_{\text{1Nb}}^{\text{n-persist}}$ is shown in Figure 15. We also show an example rule *blend* which has this property ($S_y = \emptyset$ and we assume constraints *pallete*(C) are persistent).

The next optimized encoding scheme applies specifically to 1-neighbor restricted rules which are not only primary propagated, but furthermore all propagated matching obligations are persistent[14]. We call such rules *primary persistent* rules. An example of this is the *trans* rule of the program of figure 1: Its primary matching obligation consists of only one propagated constraint *edge*(Y, D) which is never deleted by any rule of the program. Figure 15 shows this specialized encoding as the function $\rightsquigarrow_{\text{1Nb}}^{\text{p-persist}}$. This optimized

[14]This corresponds to a rule in Figure 13 such that $P_x = \emptyset$ and $S_x = \emptyset$, hence we only have P_x' the persistent propagated rule heads.

$$trans : \boxed{[X]\, edge(Y,D)}, [Y]\, path(Z,D') \Longrightarrow X \neq Z \mid [X]\, path(Z, D+D')$$

$$\downarrow \overset{\text{p-persist}}{\rightsquigarrow_{1\text{Nb}}}$$

$$\left(\begin{array}{l} trans\_1 : [X]\, edge(Y,D) \Longrightarrow [Y]\, trans\_req(X,D) \\ trans\_2 : [Y]\, trans\_req(X,D), [Y]\, path(Z,D') \Longrightarrow X \neq Z \mid [X]\, path(Z, D+D') \end{array} \right)$$

$$(r : [X]\, P_x, [Y]\, P_y \setminus [Y]\, S_y \Longleftrightarrow G_x, G_y \;\mid\; B)$$

$$\downarrow \overset{\text{p-persist}}{\rightsquigarrow_{1\text{Nb}}}$$

$$\left(\begin{array}{l} r\_1 : [X]\, P_x \Longrightarrow G_x \;\mid\; [Y]\, r\_req(Xs) \\ r\_2 : [Y]\, r\_req(Xs), [Y]\, P_y \setminus [Y]\, S_y \Longleftrightarrow G_y \;\mid\; B \end{array} \right)$$

where All $c \in P_x$ are persistent constraints
$Xs = \mathrm{FV}(P_x)$ and $\mathrm{FV}(G_x) \subseteq \mathrm{FV}(P_x)$ and $\mathrm{FV}(G_y) \subseteq \mathrm{FV}(P_x, P_y, S_y)$

Figure 15: Optimized Encodings for Primary Persistent Rules

translation scheme is very similar to *rule localization* of link-restricted rule in distributed Datalog [LCG+06] and indeed it is a special case of rewriting where left-hand sides are not removed as a result of rule application.

Given a 1-neighbor restricted program $\mathcal{P}_1$, we define the translation function $\mathcal{P}_1 \rightsquigarrow_{1\text{Nb}} \mathcal{P}_0$, where $\mathcal{P}_0$ is the 0-neighbor restricted program encoding in which we apply the optimized encodings where possible: We apply $\rightsquigarrow_{1\text{Nb}}^{\text{n-persist}}$ for 1-neighbor restricted rules in $\mathcal{P}_1$ that are neighbor persistent, while we apply $\rightsquigarrow_{1\text{Nb}}^{\text{p-persist}}$ for those that are primary persistent, and $\rightsquigarrow_{1\text{Nb}}^{\text{basic}}$ for all other 1-neighbor restricted rule. This is inductively defined by the following rules:

$$\text{(Program)} \quad \frac{R \;\rightsquigarrow_{1\text{Nb}}\; \mathcal{P}_0 \qquad \mathcal{P}_1 \;\rightsquigarrow_{1\text{Nb}}\; \mathcal{P}_0'}{(R\ \mathcal{P}_1) \;\rightsquigarrow_{1\text{Nb}}\; (\mathcal{P}_0\ \mathcal{P}_0')} \qquad (0-\text{N}) \quad \frac{R \text{ is 0-neighbor restricted}}{R \;\rightsquigarrow_{1\text{Nb}}\; R}$$

$$\text{(N-Persist)} \quad \frac{\begin{array}{c} R \text{ is 1-neighbor restricted} \\ R \text{ is neighbor persistent} \\ R \rightsquigarrow_{1\text{Nb}}^{\text{n-persist}} R' \end{array}}{R \rightsquigarrow_{1\text{Nb}} R'} \qquad \text{(P-Persist)} \quad \frac{\begin{array}{c} R \text{ is 1-neighbor restricted} \\ R \text{ is primary persistent} \\ R \rightsquigarrow_{1\text{Nb}}^{\text{p-persist}} R' \end{array}}{R \rightsquigarrow_{1\text{Nb}} R'}$$

$$\text{(Basic)} \quad \frac{\begin{array}{c} R \text{ is 1-neighbor restricted} \\ \text{(N-Persist) and (P-Persist) does not apply} \\ R \rightsquigarrow_{1\text{Nb}}^{\text{basic}} R' \end{array}}{R \rightsquigarrow_{1\text{Nb}} R'}$$

We now consider the soundness of the optimized encoding (the soundness of the basic encoding has been shown in the previous section). In general, both the neighbor and primary persistent encoding schemes exploit a similar property: one location's matching obligation M of the 1-neighbor restricted rules is completely persistent and we do not need to lock and reserve any constraint in M. This simplifies the application of such 1-neighbor restricted rules and as long as we observe that persistent matching obligations M are fulfilled, the rule application can be completed entirely at the other location's discretion. Lemma 16 and 17

states this property for the neighbor persistent and primary persistent encoding schemes respectively.

Lemma 16 (Neighbor Persistent Encoding Rule Serializability) *Given a 1-neighbor restricted and locally quiescent CHR^e program $\mathcal{P}_1$ and a 0-neighbor restricted CHR^e program $\mathcal{P}_0$ such that $\mathcal{P}_1 \rightsquigarrow_{1\text{Nb}} \mathcal{P}_0$ and commit free abstract states $\mathcal{A}_1$ $\mathcal{A}_2$, $\mathcal{A}_3$ and $\mathcal{A}_4$, given that for some matching neighbor persistent encoding rule instances r_2 and r_3, we have $\mathcal{P}_0 \rhd \mathcal{A}_1 \mapsto^{r\_2}_{\omega^e_\alpha} \mathcal{A}_2 \mapsto^{*}_{\omega^e_\alpha} \mathcal{A}_3 \mapsto^{r\_3}_{\omega^e_\alpha} \mathcal{A}_4$ then there exists some $\mathcal{A}'_2$ such that $\mathcal{P}_0 \rhd \mathcal{A}_1 \mapsto^{*}_{\omega^e_\alpha} \mathcal{A}'_2 \mapsto^{r\_2}_{\omega^e_\alpha} \mathcal{A}_3 \mapsto^{r\_3}_{\omega^e_\alpha} \mathcal{A}_4$*

Lemma 17 (Primary Persistent Encoding Rule Serializability) *Given a 1-neighbor restricted and locally quiescent CHR^e program $\mathcal{P}_1$ and a 0-neighbor restricted CHR^e program $\mathcal{P}_0$ such that $\mathcal{P}_1 \rightsquigarrow_{1\text{Nb}} \mathcal{P}_0$ and commit free abstract states $\mathcal{A}_1$ $\mathcal{A}_2$, $\mathcal{A}_3$ and $\mathcal{A}_4$, given that for some matching primary persistent encoding rule instances r_1 and r_2, we have $\mathcal{P}_0 \rhd \mathcal{A}_1 \mapsto^{r\_1}_{\omega^e_\alpha} \mathcal{A}_2 \mapsto^{*}_{\omega^e_\alpha} \mathcal{A}_3 \mapsto^{r\_2}_{\omega^e_\alpha} \mathcal{A}_4$ then there exists some $\mathcal{A}'_2$ such that $\mathcal{P}_0 \rhd \mathcal{A}_1 \mapsto^{*}_{\omega^e_\alpha} \mathcal{A}'_2 \mapsto^{r\_1}_{\omega^e_\alpha} \mathcal{A}_3 \mapsto^{r\_2}_{\omega^e_\alpha} \mathcal{A}_4$*

Theorem 18 states the soundness of the optimized encoding.

Theorem 18 (Soundness of Optimized Encoding) *Given a 1-neighbor restricted and locally quiescent CHR^e program $\mathcal{P}_1$ and a 0-neighbor restricted CHR^e program $\mathcal{P}_0$ such that $\mathcal{P}_1 \rightsquigarrow_{1\text{Nb}} \mathcal{P}_0$, for any reachable states Ω and Ω', if $\mathcal{P}_0 \rhd \Omega \mapsto^{*}_{\omega^e_0} \Omega'$, then we have either Ω' is not commit free or $\mathcal{P}_1 \rhd \text{DropSyncs}(\lceil\Omega\rceil; \mathcal{P}_1) \mapsto^{*}_{\omega^e_\alpha} \text{DropSyncs}(\lceil\Omega''\rceil; \mathcal{P}_1)$.*

6 Generalized Encoding for n-Neighbor Restricted Rules

In this section, we show a generalized encoding scheme for n-neighbor restricted rules. The basic 1-neighbor restricted encoding of Figure 13 implements a consensus protocol between two nodes. Specifically, this encoding implements a *two-phase commit protocol* [SS83] lead by an initial round of matching. Rules *r_1* and *r_2* represent the *matching phase*, while *r_3* the *voting phase*, and *r_4a* and *r_4b* the *commit phase*. The encoding of n-neighbor restricted rules is then an implementation of a general consensus protocol that establishes consensus of a rule application among the primary location X (acting as the *coordinator* of the consensus) and n other directly connected and isolated neighbor locations Y_i for $i \in [1, n]$ (acting as the *cohorts* of the consensus).

We first show an example of a 2-neighbor restricted rule encoding into 0-neighbor restricted rules. Figure 16 shows an example of a 2-neighbor restricted rule that sums up all values in an acyclic graph and distributes the value to all nodes of the graph. The primary location X must synchronize with neighbor locations Y and Z to attempt to "atomically" observe each location's matching obligations and unanimously decide to commit or abort the rule instance. The translation function $\rightsquigarrow_{\text{nNb}}$ decomposes the rule into a sequence of 0-neighbor restricted rules that implement the consensus protocol that coordinates the application of rule s in the following manner: Rule *s_1* models the broadcast of the requests (to each matching neighbor Y and Z) to match on rule s should the primary location X possess its rule head obligations. We assume that constraints *child* are persistent and hence never deleted. Note that in the body, we introduce an existential variable E and each request is parameterized by it. E is known as a *destination* [Pfe04], used to group together all synchronizing constraints that originate from a particular request (*r_req*) from primary location X to synchronize a rule application of s. This provides a form of logical isolation from synchronizing constraints that belongs to other instances of this consensus building[15]. Specifically in this context, we will refer to destinations like E as *consensus destinations*. Rules *s_2Y* and *s_2Z* implements neighbors Y's and Z's respective match response to X (*s_match_Y* and *s_match_Z*) only if they possess their rule head obligations. Destination E is passed on for reasons discussed earlier. Rule *s_3* implements primary location X attempt to *commit* if it has confirmation from each neighbor that each possesses matching instances of their rule

[15]For the 1-neighbor restricted encoding detailed in Section 5 we can omit destinations because isolation is achieved simply by passing all free-variables between primary location X and its only neighbor location Y.

$$s : [X]\,child(Y), [X]\,child(Z) \setminus [X]\,val(A), [Y]\,val(B), [Z]\,val(C) \iff [X]\,val(A{+}B{+}C), [Y]\,val(A{+}B{+}C), [Z]\,val(A{+}B{+}C)$$

$$\rightsquigarrow_{\mathrm{nNb}}$$

$$\left(\begin{array}{l}
s_1 : [X]\,child(Y), [X]\,child(Z), [X]\,val(A) \\
\qquad \Longrightarrow \exists E.\ [Y]\,s_req_Y(X,Y,Z,A,E), [Z]\,s_req_Z(X,Y,Z,A,E) \\
\hline
s_2Y : [Y]\,val(B) \setminus [Y]\,s_req_Y(X,Y,Z,A,E) \iff [X]\,s_match_Y(B,E) \\
s_2Z : [Z]\,val(C) \setminus [Z]\,s_req_Z(X,Y,Z,A,E) \iff [X]\,s_match_Z(C,E) \\
\hline
s_3 : [X]\,child(Y), [X]\,child(Z) \setminus [X]\,val(A), [X]\,s_match_Y(B,E), [X]\,s_match_Z(C,E) \\
\qquad \iff [X]\,s_commit_X(Y,Z,A,B,C,E), [Y]\,s_vote_Y(X,B,E), [Z]\,s_vote_Z(X,C,E) \\
\hline
s_4aY : [Y]\,val(B), [Y]\,s_vote_Y(X,B,E) \iff [X]\,s_commit_Y(Y,B,E) \\
s_4bY : [Y]\,s_vote_Y(X,B,E) \iff [X]\,s_abort(E) \\
s_4aZ : [Z]\,val(C), [Z]\,s_vote_Z(X,C,E) \iff [X]\,s_commit_Z(Z,C,E) \\
s_4bZ : [Z]\,s_vote_Z(X,C,E) \iff [X]\,s_abort(E) \\
\hline
s_5a : [X]\,s_commit_X(Y,Z,A,B,C,E), [X]\,s_commit_Y(Y,B,E), [X]\,s_commit_Z(Z,C,E) \\
\qquad \iff [X]\,val(A{+}B{+}C), [Y]\,val(A{+}B{+}C), [Z]\,val(A{+}B{+}C) \\
\hline
s_5bX : [X]\,s_abort(E) \setminus [X]\,s_commit_X(Y,Z,A,B,C,E) \iff [X]\,val(A) \\
s_5bY : [X]\,s_abort(E) \setminus [X]\,s_commit_Y(Y,B,E) \iff [Y]\,val(B) \\
s_5bZ : [X]\,s_abort(E) \setminus [X]\,s_commit_Z(Z,C,E) \iff [Z]\,val(C)
\end{array}\right)$$

Figure 16: Graph Sum Example: 2-Neighbor Restricted Rule

head obligations, and its own obligations are still valid. Upon applying this rule instance, X consumes the simplified components of its rule head obligations (i.e, $[X]\,val(A)$), produces the synchronizing constraint *s_commit_X* as a witness to this partial rule application of s and broadcasts to each neighbor Y and Z a call to vote for commitment to this rule instance. Rules *s_4aY* and *s_4aZ* implement successful commits in Y and Z only if each still possesses their respective rule head obligations, producing *r_commit_Y* or *r_commit_Z* synchronizing constraints as witnesses to their respective partial rule applications. Rules *s_4bY* and *s_4bZ* implements the alternative scenario where either neighbor has to vote to abort the rule instance application. Rule *s_5a* implements the successful commitment for the rule instance of s. Specifically, if primary location X receives matching commit synchronizing constraints from its neighbors Y and Z, we rewrite with the body of s and thus completing the rule application of s. Finally, rules *r_5bX*, *r_5bY* and *r_5bZ* implements the roll back procedure for the cases that any neighbor Y or Z vote for aborting the rule instance because it can no longer fulfill its matching obligation. Specifically, each rule replaces a *commit* synchronizing constraint of a given location (that witnesses the partial rule application of that location) with the simplified rule head obligation of that particular location (e.g. $[X]\,val(A)$), thereby rolling back a failed non commit free state into a commit free consistent state.

This encoding of rule s is effectively an implementation of a consensus protocol [AD76] in 0-neighbor restricted rules. It establishes a consensus between the primary location X as the *coordinator* of the consensus and neighboring location Y and Z as *cohorts* of the consensus. This encoding implements a *two-phase commit protocol* [SS83] that is lead by an initial round of matching. Specifically, the *s_1* rule and *s_2x* rules collectively represents the *matching phase*, while *s_3* rule and *s_4xx* rules represent the *voting phase*, and *s_5a* rule and

$$r : [X]P_x, [X]P'_x, \left(\biguplus_{i\in\mathcal{I}_n} [Y_i]P_i, [Y_i]P'_i\right) \setminus [X]S_x, \left(\biguplus_{i\in\mathcal{I}_n} [Y_i]S_i\right) \Longleftrightarrow G_x, \left(\biguplus_{i\in\mathcal{I}_n} G_i\right) \mid \exists \bar{Z}.D$$

$$\Big\downarrow \text{nNb}$$

$$\left(\begin{array}{l}
r_1 : [X]P_x, [X]P'_x, [X]S_x \Longrightarrow G_x \mid \exists E.\ \left(\biguplus_{i\in\mathcal{I}_n} [Y_i]\, r_req_i(Xs, E)\right) \\
\hline
\left(\biguplus_{i\in\mathcal{I}_n} \text{MatchRule}(i, S_x)\right) \\
\hline
r_3 : [X]P_x \setminus [X]P'_x, [X]S_x, \left(\biguplus_{i\in\mathcal{I}_n} [X]\, r_match_i(Ys_i, E)\right) \\
\qquad \Longleftrightarrow [X]\, r_commit_X(Rs, E), \left(\biguplus_{i\in\mathcal{I}_n} [Y_i]\, r_vote_i(X, Ys_i, E)\right) \\
\hline
\left(\biguplus_{i\in\mathcal{I}_n} \text{VoteRules}(i)\right) \\
\hline
r_5a : [X]\, r_commit_X(Rs, E), \left(\biguplus_{i\in\mathcal{I}_n} [X]\, r_commit_i(Y_i, Ys_i, E)\right) \\
\qquad \Longleftrightarrow \exists \bar{Z}.\, [X]P'_x, \left(\biguplus_{i\in\mathcal{I}_n} [Y_i]P'_i\right), D \\
\hline
r_5bx : [X]\, r_abort(E) \setminus [X]\, r_commit_X(Rs, E) \Longleftrightarrow [X]P'_x, [X]S_x \\
\hline
\left(\biguplus_{i\in\mathcal{I}_n} r_5bi : [X]\, r_abort(E) \setminus [X]\, r_commit_i(Y_i, Ys_i, E) \Longleftrightarrow [Y_i]P'_i, [Y_i]S_i\right)
\end{array}\right)$$

where
All $c \in P_x$ are persistent constraints and all $c' \in P'_x$ are non-persistent constraints
All $c \in \left(\biguplus_{i\in\mathcal{I}_n} P_i\right)$ are persistent constraints and all $c' \in \left(\biguplus_{i\in\mathcal{I}_n} P'_i\right)$ are non-persistent constraints

$\text{MatchRule}(i, \cdot) = r\_2i : [Y_i]P_i, [Y_i]P'_i, [Y_i]S_i, r\_req\_i(Xs, E) \Longrightarrow G_i \mid [X]\, r\_match\_i(Ys_i, E)$
$\text{MatchRule}(i, S) = r\_2i : [Y_i]P_i, [Y_i]P'_i, [Y_i]S_i \setminus r\_req\_i(Xs, E) \Longleftrightarrow G_i \mid [X]\, r\_match\_i(Ys_i, E) \quad \text{if } S \neq \cdot$

$$\text{VoteRules}(i) = \begin{cases} r\_4ai : [Y_i]P_i \setminus [Y_i]P'_i, [Y_i]S_i, [Y_i]\, r\_vote\_i(X, Ys_i, E) \Longleftrightarrow [X]\, r\_commit\_i(Y_i, Ys_i, E) \\ r\_4bi : [Y_i]\, r\_vote\_i(X, Ys_i, E) \Longleftrightarrow [X]\, r\_abort(E) \end{cases}$$

$Xs = \text{FV}(P_x, S_x) \quad Ys_i = \text{FV}(P_i, S_i) \quad Rs = \text{FV}(P_x, S_x, \left(\biguplus_{i\in\mathcal{I}_n} P_i, S_i\right)) \quad \text{FV}(G_x) \subseteq Xs \quad \text{FV}(G_i) \subseteq Xs \cup Ys_i$

Figure 17: 0-Neighbor Restricted Encoding for n-Neighbor Restricted Rules

s_5bx rules the *commit phase*.

The generalized form of this n-neighbor restricted encoding is shown in Figure 17. Similarly to the 1-neighbor restricted encoding shown in Figure 13, this encoding comprises a set of 0-neighbor restricted rules, referred to as *encoding rules*, which collectively implements a n-consensus protocol that allows $n + 1$ locations (one primary and n neighbor locations) to unanimously decide on the application of an instance of a n-neighbor restricted rule. Specifically, each encoding rule implements a sub-routine of this n-consensus protocol in the following manner:

1. Encoding rule *r_1* models the initiation of the consensus by the primary location X: If location X fulfills the primary matching obligations (P_x, P'_x, S_x where G_x), it sends a set of requests $r\_req_i(Xs, E)$ to n candidate locations which will participate in a round of consensus building, uniquely identified by the existential variable E.

2. Each encoding rule *r_2i* models the asynchronous response of each neighbor location if its matching obligation is fulfilled (P_i, P'_i, S_i where G_i), replying to primary location X with a $r\_match_i(Ys_i, E)$ constraint.

3. Encoding rule *r_3* states that if location X's matching obligation is still available and if X has received

a match response $r\_match_i(Ys_i, E)$, from each neighbor involved in consensus E, it commits its non-persistent matching obligations (P'_x and S_x) and calls each neighbor location of consensus E to vote for this rule application ($r\_vote\_i(X, Ys_i, E)$).

4. Each encoding rule r_4ai states that if the neighbor location still has its matching obligation (P_i,P'_i, S_i where G_i) and receives a call to vote for the rule application of r ($r\_vote\_i(X, Ys_i, E)$), it commits its non-persistent matching obligations (P'_i and S_i) and sends its decision to commit to location X ($r\_commit\_i(Y_i, Ys_i, E)$). Otherwise, encoding rule r_4b states that if the neighbor location no longer has its matching obligation, it sends its decision to abort consensus E to location X ($r\_abort(E)$).

5. Encoding rule r_5a models the completion of rule application r in the event that all neighbor locations of consensus E have decided to commit to the rule application. Specifically, X receives a full set of commit synchronizing constraints of consensus E, during which it completes the rule application by applying the rule body of the n-neighbor restricted rule r (i.e, $\exists \bar{Z}.D$) and return all non-persistent propagated rule head matching obligations of the rule r (i.e, P'_x and $\biguplus_{i \in \mathcal{I}_n} P'_i$)

6. Encoding rule r_5bx and each encoding rule r_5bi models the abortion of consensus E. Specifically, upon receive $r\_abort(E)$ from any neighbor location of consensus E, r_5bx rolls back location X's commitment to E by return its non-persistent matching obligations (i.e, P'_x and S_x) while each r_5bi does the same of each neighbor location.

Note that each derivation step of an encoding rule that models this consensus protocol is designed so that it can be interleaved with concurrent and non-overlapping derivations without compromising the correctness of the consensus building. Since we assume a lossless network communication between locations, this encoding implements a two-phase commit consensus protocol, which is sufficed to guarantee correctness in a lossless network. Given a n-neighbor restricted program encoding $\mathcal{P}_0$, we write $\mathcal{P}_0 \rhd \Omega \mapsto^{r\_j(E)}_{\omega^e_0} \Omega'$ as a derivation step that involves the application of an instance of the encoding rule r_j with the consensus destination instantiated to some E. We will refer to such derivation steps as *E derivation steps*.

We now establish the soundness of the n-neighbor restricted rule encoding. The steps of the proof of the soundness theorem for this encoding are similar to the steps shown in Section 5. Firstly, Lemma 19 states that the encoding operation $\rightsquigarrow_{\text{nNb}}$ preserves local quiescence of CHR^e programs.

Lemma 19 ($\rightsquigarrow_{\text{nNb}}$ Preserves Local Quiescence) *Given a locally quiescent n-neighbor restricted program $\mathcal{P}_n$ and a 0-neighbor restricted program $\mathcal{P}_0$ such that $\mathcal{P}_n \rightsquigarrow_{\text{nNb}} \mathcal{P}_0$, then $\mathcal{P}_0$ is also locally quiescent.*

Lemma 20 states the property that all applications of encoding rules are part of a derivation of some prefix sequence of a consensus building routine of a specific consensus destination E.

Lemma 20 (Prefix Executions of n-Neighbor Encoding) *Given a locally quiescent n-neighbor restricted program $\mathcal{P}_1$ and a 0-neighbor restricted program $\mathcal{P}_0$ such that $\mathcal{P}_n \rightsquigarrow_{\text{nNb}} \mathcal{P}_0$, if we have reachable states Ω, Ω' such that $\mathcal{P}_0 \rhd \Omega \mapsto_{\omega^e_0} \Omega'$, then we have the following:*

1. *If $\mathcal{P}_0 \rhd \Omega \mapsto^{r\_2i(E)}_{\omega^e_0} \Omega'$ for r_2i encoding rule of some consensus destination E, then there exists some reachable states Ω'' and Ω''', such that $\mathcal{P}_0 \rhd \Omega'' \mapsto^{r\_1(E)}_{\omega^e_0} \Omega'''$ and $\mathcal{P}_0 \rhd \Omega''' \mapsto^*_{\omega^e_0} \Omega$*

2. *If $\mathcal{P}_0 \rhd \Omega \mapsto^{r\_3(E)}_{\omega^e_0} \Omega'$ for r_3 encoding rule of some consensus destination E, then there exists some reachable derivation $\mathcal{P}_0 \rhd \Omega'' \mapsto^*_{\omega^e_0} \Omega'''$ which contains n derivation steps $r\_2i(E)$ for each neighbor Y_i of $i \in \mathcal{I}_n$, and $\mathcal{P}_0 \rhd \Omega''' \mapsto^*_{\omega^e_0} \Omega$*

3. *If $\mathcal{P}_0 \rhd \Omega \mapsto^{r\_4ai(E)}_{\omega^e_0} \Omega'$, for r_4ai encoding rule of some consensus destination E, then there exists some reachable states Ω'' and Ω''', such that $\mathcal{P}_0 \rhd \Omega'' \mapsto^{r\_3(E)}_{\omega^e_0} \Omega'''$ and $\mathcal{P}_0 \rhd \Omega''' \mapsto^*_{\omega^e_0} \Omega$*

4. *If* $\mathcal{P}_0 \rhd \Omega \mapsto^{r\_4bi(E)}_{\omega^e_0} \Omega'$, *for* r_4bi *encoding rule of some consensus destination* E, *then there exists some reachable states* Ω'' *and* Ω''', *such that* $\mathcal{P}_0 \rhd \Omega'' \mapsto^{r\_3(E)}_{\omega^e_0} \Omega'''$ *and* $\mathcal{P}_0 \rhd \Omega''' \mapsto^*_{\omega^e_0} \Omega$
5. *If* $\mathcal{P}_0 \rhd \Omega \mapsto^{r\_5a(E)}_{\omega^e_0} \Omega'$ *for* r_5a *encoding rule of some consensus destination* E, *then there exists some reachable derivation* $\mathcal{P}_0 \rhd \Omega'' \mapsto^*_{\omega^e_0} \Omega'''$ *which contains* n *derivation steps* $r\_2i(E)$ *for each neighbor* Y_i *of* $i \in \mathcal{I}_n$, *and* $\mathcal{P}_0 \rhd \Omega''' \mapsto^*_{\omega^e_0} \Omega$
6. *If* $\mathcal{P}_0 \rhd \Omega \mapsto^{r\_5bx(E)}_{\omega^e_0} \Omega'$ *of* $\mathcal{P}_0 \rhd \Omega \mapsto^{r\_5bi(E)}_{\omega^e_0} \Omega'$ *for* r_5bx *or* r_5bi *encoding rule of some consensus destination* E, *then there exists some reachable states* Ω'' *and* Ω''', *such that* $\mathcal{P}_0 \rhd \Omega'' \mapsto^{r\_4bj(E)}_{\omega^e_0} \Omega'''$ *and* $\mathcal{P}_0 \rhd \Omega''' \mapsto^*_{\omega^e_0} \Omega$

Lemma 21 states the property that non-commit free states that contain partial applications of n-neighbor restricted rules, can always eventually derive commit free state that only contain complete applications of n-neighbor restricted rules.

Lemma 21 (n-Neighbor Commit-Free Reachability) *Given a locally quiescent* n*-neighbor restricted program* $\mathcal{P}_n$ *and a* 0*-neighbor restricted program* $\mathcal{P}_0$ *such that* $\mathcal{P}_n \rightsquigarrow_{\text{nNb}} \mathcal{P}_0$ *and states* Ω *reachable by* $\mathcal{P}_0$, *if* Ω *is not commit-free, then there exists some commit-free state* Ω' *such that* $\mathcal{P}_0 \rhd \Omega \mapsto^*_{\omega^e_0} \Omega'$.

Similarly to Lemma 14, Lemma 22 states that given any ω^e_α derivation between two commit free state $\mathcal{P}_0 \rhd \mathcal{A} \mapsto^*_{\omega^e_\alpha} \mathcal{A}'$ labeled $\mathcal{I}$, we can safely permute $\mathcal{I}$ so that for each consensus destination E, derivation step of $r\_3(E)$ is immediately proceeded by all derivation steps of consensus destination E that occur after it in $\mathcal{I}$. This simply allows us to focus on derivations which are controlled and orderly.

Lemma 22 (n-Neighbor Encoding Rule Serializability) *Given an* n*-neighbor restricted and locally quiescent CHR*^{e} *program* $\mathcal{P}_n$ *and a* 0*-neighbor restricted CHR*^{e} *program* $\mathcal{P}_0$ *such that* $\mathcal{P}_n \rightsquigarrow_{\text{nNb}} \mathcal{P}_0$ *and commit free abstract states* $\mathcal{A}$,$\mathcal{A}'$ *such that* $\mathcal{P}_0 \rhd \mathcal{A} \mapsto^*_{\omega^e_\alpha} \mathcal{A}'$ *labeled as derivation* $\mathcal{I}$, *if* $\mathcal{I}$ *contain derivations of a* n*-neighbor restricted encoding rule instance* r *of consensus destination* E *such that:*

- *We have a successful application of rule instance* r *of* E, *then there exists a valid derivation* $\mathcal{I}' = \mathcal{P}_0 \rhd \mathcal{A} \mapsto^*_{\omega^e_\alpha} \mathcal{A}_E \mapsto^*_{\omega^e_\alpha} \mathcal{A}'_E \mapsto^*_{\omega^e_\alpha} \mathcal{A}'$ *such that* $\mathcal{P}_0 \rhd \mathcal{A}_E \mapsto^*_{\omega^e_\alpha} \mathcal{A}'_E$ *consists the following non-interleaving derivation sequence:*
  1. *The derivation step* $r\_3(E)$.
  2. *All* n *derivation steps of* $r\_4ai(E)$ *each of* Y_i *for* $i \in \mathcal{I}_n$.
  3. *The derivation step* $r\_5a(E)$.
- *We have an aborted attempt to apply rule instance* r *of* E, *then there exists a valid derivation* $\mathcal{I}' = \mathcal{P}_0 \rhd \mathcal{A} \mapsto^*_{\omega^e_\alpha} \mathcal{A}_E \mapsto^*_{\omega^e_\alpha} \mathcal{A}'_E \mapsto^*_{\omega^e_\alpha} \mathcal{A}'$ *such that* $\mathcal{P}_0 \rhd \mathcal{A}_E \mapsto^*_{\omega^e_\alpha} \mathcal{A}'_E$ *consists a non-interleaving derivation sequences of derivation step* $r\_3(E)$ *followed by all* n *derivation steps of* $r\_4ai(E)$ *or* r_4bi *each of* Y_i *for* $i \in \mathcal{I}_n$, *followed by*
  1. *The derivation step* $r\_3(E)$.
  2. *All* m *derivation steps of either* $r\_4ai(E)$ *each of* Y_i *for* $i \in \mathcal{I}_n$ *where* $m \leq n$.
  3. *All* $n - m$ *derivation steps of either* $r\_4bi(E)$ *each of* Y_i *for* $i \in \mathcal{I}_n$.
  4. *The derivation step* $r\_5b(E)$.
  5. *All* $n - m$ *derivation steps of* $r\_5bi(E)$ *of* Y_i *for* $i \in \mathcal{I}_n$.

Theorem 23 states the soundness of the n-neighbor restricted encoding scheme.

Theorem 23 (Soundness of n-Neighbor Restricted Encoding) *Given an* n*-neighbor restricted and locally quiescent CHR*^{e} *program* $\mathcal{P}_n$ *and a* 0*-neighbor restricted CHR*^{e} *program* $\mathcal{P}_0$ *such that* $\mathcal{P}_n \rightsquigarrow_{\text{nNb}} \mathcal{P}_0$, *for any reachable states* Ω *and* Ω', *if* $\mathcal{P}_0 \rhd \Omega \mapsto^*_{\omega^e_0} \Omega'$, *then we have either* Ω' *is not commit free or* $\mathcal{P}_n \rhd \text{DropSyncs}(\lceil \Omega \rceil; \mathcal{P}_n) \mapsto^*_{\omega^e_\alpha} \text{DropSyncs}(\lceil \Omega'' \rceil; \mathcal{P}_n)$.

7 Related Works

An extension of Datalog for implementing network protocols is introduced in [LCG+06]. It defines *link restricted Datalog rules* along with the idea of *rule localization* which encodes link restricted rules into local Datalog rules. Our work adapts, expands and generalizes these ideas to the context of distributed multiset rewriting. A distributed and incremental algorithm for computing rule matchings involved in such distributed Datalog systems is presented in [NJLS11] and is analogous to what ω_0^e provides for CHR^e. Our work draws inspiration from the programming language Meld [ARLG+09]. Meld introduces a multitude of extensions to Datalog, including linearity, aggregates and comprehensions. Work in [CARG+12] adapts Meld for the context of general purpose multicore parallel programming. The ω_0^e semantics extends the refined CHR operational semantics [DSdlBH04]. A parallel execution model of CHR is introduced in [LS11], which assumes a more tightly coupled model of concurrent execution, where multiple execution threads compute rewritings concurrently over a *global constraint store*.

8 Conclusion and Future Works

We introduced CHR^e, an extension of CHR with located constraints and n-neighbor restricted rewrite rules for programming an ensemble of distributed computing entities. We defined the ω_α^e abstract semantics and ω_0^e operational semantics of CHR^e and showed their soundness. We gave an optimized encoding for 1-neighbor restricted rules into 0-neighbor restricted rules. Following this, we generalize this encoding scheme for n-neighbor restricted rules. We have developed a prototype implementation of CHR^e in Python with MPI (Message Passing Interface) as a proof of concept and demonstrated its relative scalability in distributed execution. In the future we intend to develop a more practical and competitive implementation of CHR^e in C or C++, imbued with existing CHR optimization techniques [Sch05]. We also intend to explore an implementation over higher-level distributed graph processing frameworks like Google's Pregel [MAB+10]. Additionally, we intend to explore using ω_0^e to serve as an operational semantics that describes the core multiset rewriting fragment of Meld and derive extensions like aggregates and comprehensions as higher-level language encodings into ω_0^e. Our works on encoding n-neighbor restricted rules can be also applied to extend Meld.

References

[AD76] P. A. Alsberg and J. D. Day. A Principle for Resilient Sharing of Distributed Resources. In *Proc. of the 2nd Int. Conf. on Software Engineering*, ICSE '76, pages 562–570, Los Alamitos, CA, USA, 1976. IEEE Computer Society Press.

[ARLG+09] M. P. Ashley-Rollman, P. Lee, S. C. Goldstein, P. Pillai, and J. D. Campbell. A Language for Large Ensembles of Independently Executing Nodes. In *Proc. of ICLP'09*, volume 5649, pages 265–280. Springer-Verlag, 2009.

[CARG+12] F. Cruz, M. P. Ashley-Rollman, S. C. Goldstein, Ricardo Rocha, and F. Pfenning. Bottom-Up Logic Programming for Multicores. In Vítor Santos Costa, editor, *Proc. of DAMP 2012*. ACM Digital Library, January 2012.

[DSdlBH04] G. J. Duck, P. J. Stuckey, M. Garcia de la Banda, and C. Holzbaur. The Refined Operational Semantics of Constraint Handling Rules. In *In 20th Int. Conf. on Logic Programming ICLP'04*, pages 90–104. Springer, 2004.

[Frü94] T. Frühwirth. Constraint handling rules. In *Constraint Programming*, pages 90–107, 1994.

[LCG+06] B. T. Loo, T. Condie, M. Garofalakis, D. E. Gay, J. M. Hellerstein, P. Maniatis, R. Ramakrishnan, T. Roscoe, and I. Stoica. Declarative Networking: Language, Execution and Optimization. In *Proc. of SIGMOD '06*, pages 97–108. ACM, 2006.

[LS11] E. S. L. Lam and M. Sulzmann. Concurrent Goal-based Execution of Constraint Handling Rules. *TPLP*, 11(6):841–879, 2011.

[MAB+10] G. Malewicz, M. H. Austern, A. J.C Bik, J. C. Dehnert, I. Horn, N. Leiser, and G. Czajkowski. Pregel: A System for Large-Scale Graph Processing. In *Proc. of Int. Conf. on Management of data*, SIGMOD, pages 135–146, USA, 2010. ACM.

[NJLS11] V. Nigam, L. Jia, B. T. Loo, and A. Scedrov. Maintaining distributed logic programs incrementally. In *Proc. of PPDP'11*, pages 125–136. ACM, 2011.

[Pfe04] F. Pfenning. Substructural Operational Semantics and Linear Destination-Passing Style (Invited Talk). In *APLAS*, page 196, 2004.

[Sch05] T. Schrijvers. Analyses, Optimizations and Extensions of Constraint Handling Rules: Ph.D. Summary. In *ICLP*, pages 435–436, 2005.

[SL08] M. Sulzmann and E. S. L. Lam. Parallel Execution of Multi-set Constraint Rewrite Rules. In *PPDP*, pages 20–31, 2008.

[SS83] D. Skeen and M. Stonebraker. A Formal Model of Crash Recovery in a Distributed Systems. *IEEE Transactions on Software Engineering*, pages 219–228, 1983.

A Proofs

A.1 Proofs for the ω_α^e Semantics

Lemma 1 (Well-Formedness Preservation of ω_α^e Derivations) *Given a well-formed CHR^e program $\mathcal{P}$, a well-formed state $\mathcal{A}$ and state $\mathcal{A}'$, if $\mathcal{P} \rhd \mathcal{A} \mapsto^*_{\omega_\alpha^e} \mathcal{A}'$ then $\mathcal{A}'$ is well-formed.*

Proof: We proof by induction on ω_α^e derivation steps. We consider the base case such that $\mathcal{P} \rhd \mathcal{A} \mapsto^*_{\omega_\alpha^e} \mathcal{A}$ of zero ω_α^e derivation step, hence trivially we have $\mathcal{A}' = \mathcal{A}$ is well-formed. Now suppose well-formedness preservation holds for derivations $\mathcal{P} \rhd \mathcal{A} \mapsto^*_{\omega_\alpha^e} \mathcal{A}''$ of m number of derivation steps, hence state $\mathcal{A}''$ is well-formed. We consider the successor state $\mathcal{A}'$ such that $\mathcal{P} \rhd \mathcal{A}'' \mapsto_{\omega_\alpha^e} \mathcal{A}'$. Specifically, we consider this derivation step in its most general form as specified in Figure 4: Since all rules in $\mathcal{P}$ are well-formed, we are guaranteed that all fragments of rule body (i.e, D'_i, D'_j and D'_k) consist of well-formed constraints, hence instantiated rule body fragments D_i, D_j and D_k are well-formed as well. Constraint stores $\bar{\mathcal{S}}_i$ and $\bar{\mathcal{S}}_j$ extended with well-formed constraints result to well-formed stores as well, hence new components of the state (i.e, $\biguplus_{i\in\mathcal{I}_n}\langle\bar{\mathcal{S}}_i, P_i, D_i\rangle_{k_i}$, $\biguplus_{j\in\mathcal{I}_m}\langle\bar{\mathcal{S}}_j, D_j\rangle_{k_j}$, $\biguplus_{l\in\mathcal{I}_m}\langle\bar{\mathcal{S}}_l, D_l\rangle_{k_l}$) are all well-formed. We show that new location names k_l (of existential forwarding locations) are unique and do not exist in $\mathcal{A}$: location variables k'_l are defined such that they do not appear as term arguments of rule heads (i.e, $k'_l \notin \big(\mathrm{FV}(\biguplus_{i\in\mathcal{I}_n} P_i) \cup \mathrm{FV}(\biguplus_{i\in\mathcal{I}_n} S_i)\big)$) nor are they matching locations (i.e, $\forall i \in \mathcal{I}_n.\ k'_l \neq k'_i$). Therefore, variables k'_l are treated as existential variables and the meta operation $\mathrm{Inst}(-)$ would simply assign it a fresh constant k_l that has never appeared before. Hence the state $\mathcal{A}'$ is well-formed. □

Lemma 2 (Well-Formedness Preservation of $[\![-]\!]$ Translation) *Given a well-formed CHR^e object o, $[\![o]\!]$ is a well-formed CHR object.*

Proof: We prove by structural induction over well-formed CHR^{e}objects, where CHR constraints $p(\vec{t})$ are the base elements:

- *Constraints:* $[\![p(\vec{t})]\!]^l = p(l,\vec{t})$. $p(\vec{t})$ is well-formed, namely terms $\vec{t}$ are well-formed. l is either a variable or location name (a constant) which is a well-formed term, hence $p(l,\vec{t})$ is well-formed.
- *Constraint Store:* For $[\![\emptyset]\!]^l = \emptyset$, an empty set which is by default a well-formed store. For $[\![c,\bar{\mathcal{S}}]\!]^l$, suppose $[\![c]\!]^l$ and $[\![\bar{\mathcal{S}}]\!]^l$ are well-formed constraint stores (via constraint and constraint store cases respectively), then the multiset union $[\![c]\!]^l, [\![\bar{\mathcal{S}}]\!]^l$ is also a well-formed constraint store.
- *Abstract Ensemble States:* For $[\![\emptyset]\!] = \emptyset$, an empty set is by default a well-formed constraint store. For $[\![\mathcal{A}, \langle\bar{\mathcal{S}}\rangle_k]\!] = [\![\mathcal{A}]\!], [\![\bar{\mathcal{S}}]\!]^k$, suppose that $[\![\mathcal{A}]\!]$ is a well-formed (this case) and furthermore, it is a constraint store. Also suppose that $[\![\bar{\mathcal{S}}]\!]^k$ is a well-formed constraint store (constraint store case), then the multiset union of the two is also a well-formed constraint store.
- *Rule Body:* At the top-level, $[\![\exists\bar{x}.D]\!]$ is simply $\exists\bar{x}.[\![D]\!]$ and is clearly well-formed, if $[\![D]\!]$ is well-formed. For $[\![true]\!] = true$, *true* is by default a well-formed CHR rule body. For $[\![\,[l]\,c, D]\!] = [\![c]\!]^l, [\![D]\!]$, suppose that $[\![c]\!]^l$ is a well-formed CHR constraint (constraint case) and that $[\![D]\!]$ is a well-formed rule body (this case), hence the composition $[\![c]\!]^l, [\![D]\!]$ is indeed a well-formed CHR rule body.
- *Rule Head:* For $[\![\cdot]\!] = \cdot$, '$\cdot$' is by default a well-formed CHR rule head. For $[\![\,[l]\,c, H]\!] = [\![c]\!]^l, [\![H]\!]$, suppose that $[\![c]\!]^l$ is a well-formed CHR constraint (constraint case) and that $[\![H]\!]$ is a well-formed rule head (this case), hence the composition $[\![c]\!]^l, [\![H]\!]$ is indeed a well-formed CHR rule head.
- *Rule:* $[\![r : P \setminus S \iff G \mid B]\!] = r : [\![P]\!] \setminus [\![S]\!] \iff G \mid [\![B]\!]$, Given multisets of well-formed located constraints P, S and B, we have $[\![P]\!]$, $[\![S]\!]$ and $[\![B]\!]$ as well-formed CHR constraints. Hence the resultant CHR rule is well-formed.
- *Program:* For $[\![\cdot]\!] = \cdot$, empty '$\cdot$' is by default a well-formed CHR program. For $[\![R\ \mathcal{P}]\!]$, suppose $[\![R]\!]$ is a well-formed CHR rule (rule case) and $[\![\mathcal{P}]\!]$ is a well-formed CHR program (this case), then the composition of the two is a well-formed CHR program.

Hence we have shown that given a well-formed CHR^e object o, $[\![o]\!]$ is a well-formed CHR object. □

Theorem 3 (Soundness of ω_α^e) *Given a CHR^e program $\mathcal{P}$ and abstract states $\mathcal{A}$ and $\mathcal{A}'$, if $\mathcal{P} \rhd \mathcal{A} \mapsto^*_{\omega_\alpha^e} \mathcal{A}'$, then $[\![\mathcal{P}]\!] \rhd [\![\mathcal{A}]\!] \mapsto^*_{\omega_\alpha} [\![\mathcal{A}']\!]$.*

Proof: We prove by induction on ω_α^e derivation steps: We consider the base case such that $\mathcal{P} \rhd \mathcal{A} \mapsto^*_{\omega_\alpha^e} \mathcal{A}$ of zero derivation steps. With Lemma 2, since $[\![\mathcal{P}]\!]$ and $[\![\mathcal{A}]\!]$ are well-formed CHR program and store, we trivially have $[\![\mathcal{P}]\!] \rhd [\![\mathcal{A}]\!] \mapsto^*_{\omega_\alpha} [\![\mathcal{A}]\!]$. We now consider the inductive case: Suppose we have $\mathcal{P} \rhd \mathcal{A} \mapsto^*_{\omega_\alpha^e} \mathcal{A}''$ such that $[\![\mathcal{P}]\!] \rhd [\![\mathcal{A}]\!] \mapsto^*_{\omega_\alpha} [\![\mathcal{A}'']\!]$ for some derivation of m steps. We consider the successor derivation step that extends this by one, in its most general form as specified in Figure 4:

$$\mathcal{P} \rhd \mathcal{A}', \left(\begin{array}{l} \uplus_{i\in\mathcal{I}_n}\langle\bar{\mathcal{S}}_i, P_i, S_i\rangle_{k_i}, \\ \uplus_{j\in\mathcal{I}_m}\langle\bar{\mathcal{S}}_j\rangle_{k_j} \end{array}\right) \mapsto_{\omega_\alpha^e} \mathcal{A}', \left(\begin{array}{l} \uplus_{i\in\mathcal{I}_n}\langle\bar{\mathcal{S}}_i, P_i, D_i\rangle_{k_i}, \\ \uplus_{j\in\mathcal{I}_m}\langle\bar{\mathcal{S}}_j, D_j\rangle_{k_j}, \\ \uplus_{l\in\mathcal{I}_e}\langle D_l\rangle_{k_l} \end{array}\right)$$

$$\text{such that } \mathcal{A}'' = \mathcal{A}', \left(\uplus_{i\in\mathcal{I}_n}\langle\bar{\mathcal{S}}_i, P_i, S_i\rangle_{k_i}, \uplus_{j\in\mathcal{I}_m}\langle\bar{\mathcal{S}}_j\rangle_{k_j}\right)$$

and from this we can infer (for some rule name r and guard G) the rule instance in $\mathcal{P}$ that corresponds to this derivation step is:

$$r : \uplus_{i\in\mathcal{I}_n}[k_i]\,P_i \setminus \uplus_{i\in\mathcal{I}_n}[k_i]\,S_i \iff G \mid \exists\bar{x}.\ \uplus_{i\in\mathcal{I}_n}[k_i]\,D_i, \uplus_{j\in\mathcal{I}_m}[k_j]\,D_j, \uplus_{l\in\mathcal{I}_e}[k_l]\,D_l$$

Hence, to complete the induction proof, we need to show that the following holds:

$$\llbracket \mathcal{P} \rrbracket \;\rhd\; \llbracket \mathcal{A}' \rrbracket, \left(\begin{array}{l} \biguplus_{i \in \mathcal{I}_n} \llbracket \bar{\mathcal{S}}_i \rrbracket^{k_i}, \llbracket P_i \rrbracket^{k_i}, \llbracket S_i \rrbracket^{k_i}, \\ \biguplus_{j \in \mathcal{I}_m} \llbracket \bar{\mathcal{S}}_j \rrbracket^{k_j} \end{array} \right) \;\mapsto_{\omega_\alpha}\; \llbracket \mathcal{A}' \rrbracket, \left(\begin{array}{l} \biguplus_{i \in \mathcal{I}_n} \llbracket \bar{\mathcal{S}}_i \rrbracket^{k_i}, \llbracket P_i \rrbracket^{k_i}, \llbracket D_i \rrbracket^{k_i}, \\ \biguplus_{j \in \mathcal{I}_m} \llbracket \bar{\mathcal{S}}_j \rrbracket^{k_j}, \llbracket D_j \rrbracket^{k_j}, \\ \biguplus_{l \in \mathcal{I}_e} \llbracket D_l \rrbracket^{k_l} \end{array} \right) \qquad (A)$$

We apply the translation function on rule instance r, which we know to be an instance of some rule $R \in \mathcal{P}$, hence we have the CHR interpretation of rule instance r:

$$\begin{array}{rl} & \llbracket r : \biguplus_{i \in \mathcal{I}_n} [k_i]\, P_i \setminus \biguplus_{i \in \mathcal{I}_n} [k_i]\, S_i \iff G \mid \exists \bar{x}.\ \biguplus_{i \in \mathcal{I}_n} [k_i]\, D_i, \biguplus_{j \in \mathcal{I}_m} [k_j]\, D_j, \biguplus_{l \in \mathcal{I}_e} [k_l]\, D_l \rrbracket \\ = & r : \biguplus_{i \in \mathcal{I}_n} \llbracket P_i \rrbracket^{k_i} \setminus \biguplus_{i \in \mathcal{I}_n} \llbracket S_i \rrbracket^{k_i} \iff G \mid \exists \bar{x}.\ \biguplus_{i \in \mathcal{I}_n} \llbracket D_i \rrbracket^{k_i}, \biguplus_{j \in \mathcal{I}_m} \llbracket D_j \rrbracket^{k_j}, \biguplus_{l \in \mathcal{I}_e} \llbracket D_l \rrbracket^{k_l} \end{array}$$

Note that this CHR rule instance corresponds the rule instance applied in CHR derivation (A), thus proving that derivation (A) is valid and thus, ω_α^e is sound and preserves deriviability. □

A.2 Proofs for the ω_0^e Semantics

Lemma 4 (Well-Formedness Preservation of ω_0^e Derivations) *Given a well-formed 0-neighbor restricted CHR^e program $\mathcal{P}$, a well-formed state Ω and state Ω', if $\mathcal{P} \rhd \Omega \mapsto^*_{\omega_0^e} \Omega'$ then Ω' is well-formed.*

Proof: We proof by induction on ω_0^e derivation steps. We consider the base case such that $\mathcal{P} \rhd \Omega \mapsto^*_{\omega_0^e} \Omega$ of zero ω_0^e derivation step, hence trivially we have $\Omega' = \Omega$ is well-formed. Now suppose well-formedness preservation holds for derivations $\mathcal{P} \rhd \Omega \mapsto^*_{\omega_0^e} \Omega''$ of m number of derivation steps, hence state Ω'' is well-formed. We consider all possible forms (each rule of Figure 7) of successor derivation step that maps Ω'' to some state Ω' (i.e, $\mathcal{P} \rhd \mathcal{A}'' \mapsto_{\omega_\alpha^e} \mathcal{A}'$):

- (Flush) Step: This step modifies a location $k \in \text{Locs}(\Omega'')$ by moving its buffer $\vec{\mathcal{U}}$ to its empty goals. Since $\vec{\mathcal{U}}$ (a sequence of constraints) is by definition a valid sequence of goals, hence successor state Ω' is well-formed.
- (Loc 1) This step modifies two distinct locations $k, k' \in \text{Locs}(\Omega'')$ by moving leading goal in k $[k']c$ into the buffer of k' as c. Hence goals of k and buffer of k' are still well-formed and therefore successor state Ω' is well-formed.
- (Loc 2) Step: This step modifies the goals of $k \in \text{Locs}(\Omega'')$ from $[k]c, \vec{\mathcal{G}}$ to $c, \vec{\mathcal{G}}$ which is clearly still well-formed. Therefore successor state Ω' is well-formed.
- (Loc 3) Step: This step modifies the goals of $k \in \text{Locs}(\Omega'')$ from $[k]c, \vec{\mathcal{G}}$ to $c, \vec{\mathcal{G}}$ and extends the successor state Ω' with $\langle c \;;\; \emptyset \;;\; \emptyset \;;\; \emptyset \rangle_{k'}.\ c, \vec{\mathcal{G}}$ is clearly still well-formed, and premise of the premise of the rule restricts k' to be unique from any other state that appear in Ω''. Therefore Ω' is well-formed.
- (Act) Step: This step modifies a location $k \in \text{Locs}(\Omega'')$ by changing its leading goal from $p(\vec{t})$ to $p(\vec{t})\#d : 1$ and adding $p(\vec{t})\#d$ to its store. Since $p(\vec{t})\#d : 1$ is a valid goal and d is a fresh id, successor state Ω' is well-formed.
- (Simp) Step: This step modifies a location $k \in \text{Locs}(\Omega'')$ by removing a fragment '$S, c\#d$' from the store, removing leading goal $c\#d : i$ from the goals and adding rule body instance $\text{NF}(\text{Inst}(\theta B))$ to the goals. Removing elements from store and goal have benign effects on well-formedness and since CHR^{e}program $\mathcal{P}$ contains only well-formed rules, rule body instance $\text{NF}(\text{Inst}(\theta B))$ is a ground and well-form multiset of constraints. We treat this multiset of constraints as an arbitrarily ordered sequence of constraints when appending to the goals. Therefore successor state Ω' is well-formed.

- (Prop) Step: This step is similar to (Simp) with exception that the active constraint $c\#d : i$ is retained in the goal, $c\#d$ is retained in the store and $(d, \text{Ids}(P, S))$ is added to the history. Since $(d, \text{Ids}(P, S))$ is a valid history element and premise restricts that it must not already appear in the current history (thus enforcing the set semantics of the history), successor state Ω' is well-formed.
- (Next) Step: This step modifies a location $k \in \text{Locs}(\Omega'')$ by changing its leading goal from $c\$d : i$ to $c\#d : i + 1$, which is a valid goal. Hence successor state Ω' is well-formed.
- (Drop) Step: This step modifies a location $k \in \text{Locs}(\Omega'')$ by removing its leading goal. Therefore successor state is well-formed.

Therefore we have shown that all derivation steps of ω_0^e preserves well-formedness. □

Lemma 5 (Well-Formedness Preservation of $\lceil - \rceil$ Translation) *Given a well-formed ω_0^e operational state Ω, $\lceil \Omega \rceil$ is a well-formed ω_α^e abstract state.*

Proof: We prove by structural induction over well-formed ω_0^e operational state and sub-structural fragments:

- *Ensembles:* $\lceil \Omega \rceil$, it is well-formed assuming that its two components $\lceil \Omega \rceil^{\vec{\mathcal{G}}'}$ and $\biguplus_{k \in (\text{Locs}(\vec{\mathcal{G}}') - \text{Locs}(\Omega))} \langle \vec{\mathcal{G}}'_{|k} \rangle_k$ are well-formed sets of store and location pairs ($\langle \bar{\mathcal{S}} \rangle_k$) and all k are unique. Indeed, by definition of both components each are sets of store and location pairs and uniqueness of each k is guaranteed by the assumption that each sub-structure is well-formed and that the second explicitly quantify over locations $k \notin \text{Locs}(\Omega)$ (i.e, $k \in \text{Locs}(\vec{\mathcal{G}}') - \text{Locs}(\Omega)$). For $\lceil \Omega, \langle \vec{\mathcal{U}} \ ; \ \vec{\mathcal{G}} \ ; \ \bar{\mathcal{S}} \ ; \ \bar{\mathcal{H}} \rangle_k \rceil^{\vec{\mathcal{G}}'}$, we show that it is well-formed by assuming that the sub-structure $\lceil \Omega \rceil^{\vec{\mathcal{G}}'}$ is well-formed and that $\lceil \vec{\mathcal{U}} \rceil$, $\lceil \vec{\mathcal{G}} \rceil$, $\lceil \bar{\mathcal{S}} \rceil$ and $\vec{\mathcal{G}}'_{|k}$ are well-formed multisets of constraints. The latter four sub-structures are collapsed into one multiset of constraints, hence $\lceil \Omega, \langle \vec{\mathcal{U}} \ ; \ \vec{\mathcal{G}} \ ; \ \bar{\mathcal{S}} \ ; \ \bar{\mathcal{H}} \rangle_k \rceil^{\vec{\mathcal{G}}'}$ is well-formed. Finally, $\lceil \emptyset \rceil^{\vec{\mathcal{G}}'}$ is the empty set $\emptyset$ which is by default a well-formed abstract state.
- *Buffers:* For $\lceil \emptyset \rceil = \emptyset$, $\emptyset$ is by default a well-formed constraint store. For $\lceil c, \vec{\mathcal{U}} \rceil$, it is well-formed assuming that $\lceil \vec{\mathcal{U}} \rceil$ is a well-formed constraint store. Since c is a valid element of a constraint store, unioning $\lceil \vec{\mathcal{U}} \rceil$ with c forms a well-formed constraint store.
- *Stores:* For $\lceil \emptyset \rceil = \emptyset$, $\emptyset$ is by default a well-formed constraint store. For $\lceil c\#d, \bar{\mathcal{S}} \rceil$, it is well-formed assuming that $\lceil \bar{\mathcal{S}} \rceil$ is a well-formed constraint store. Since we drop d and only retain c, a valid element of a constraint store, unioning $\lceil \bar{\mathcal{S}} \rceil$ with c forms a well-formed constraint store.
- For $\lceil \emptyset \rceil = \emptyset$, $\emptyset$ is by default a well-formed constraint store. For $\lceil c, \vec{\mathcal{G}} \rceil$, it is well-formed assuming that $\lceil \vec{\mathcal{G}} \rceil$ is a well-formed constraint store. Since c is a valid element of a constraint store, unioning $\lceil \vec{\mathcal{G}} \rceil$ with c forms a well-formed constraint store. For $\lceil c\#d : i, \vec{\mathcal{G}} \rceil$ and $\lceil [c], \vec{\mathcal{G}} \rceil$, we drop both respective atoms and only rely on the assumption that $\lceil \vec{\mathcal{G}} \rceil$ is well-formed.

Hence we have shown that given a well-formed ω_0^e operational state Ω, $\lceil \Omega \rceil$ is a well-formed ω_α^e abstract state. □

Theorem 6 (Soundness of ω_0^e) *Given 0-neighbor restricted CHR^e program $\mathcal{P}$ and states Ω and Ω', if $\mathcal{P} \rhd \Omega \mapsto^*_{\omega_0^e} \Omega'$, then $\mathcal{P} \rhd \lceil \Omega \rceil \mapsto^*_{\omega_\alpha^e} \lceil \Omega' \rceil$.*

Proof: We proof by induction on ω_0^e derivation steps. We consider the base case such that $\mathcal{P} \rhd \Omega \mapsto^*_{\omega_0^e} \Omega$ of zero ω_0^e derivation step, hence trivially we have $\Omega' = \Omega$ and that $\mathcal{P} \rhd \lceil \Omega \rceil \mapsto^*_{\omega_\alpha^e} \lceil \Omega \rceil$ for zero ω_α^e derivation step. Now suppose that for derivations $\mathcal{P} \rhd \Omega \mapsto^*_{\omega_0^e} \Omega''$ of m number of steps, we have $\mathcal{P} \rhd \lceil \Omega \rceil \mapsto^*_{\omega_\alpha^e} \lceil \Omega'' \rceil$. We consider all possible forms (each rule of Figure 7) of successor derivation step that maps Ω'' to some state Ω' (i.e, $\mathcal{P} \rhd \Omega'' \mapsto_{\omega_\alpha^e} \Omega'$), and show that for each instance we have $\mathcal{P} \rhd \lceil \Omega \rceil \mapsto^*_{\omega_\alpha^e} \lceil \Omega' \rceil$:

- (Flush) Step: Successor derivation step $\mathcal{P} \rhd \Omega'' \mapsto_{\omega_\alpha^e} \Omega'$ is of the form:

$$\mathcal{P} \ \rhd \ \Omega, \langle \vec{\mathcal{U}} \ ; \ \emptyset \ ; \ \bar{\mathcal{S}} \ ; \ \bar{\mathcal{H}} \rangle_k \ \mapsto_{\omega_0^e} \ \Omega, \langle \emptyset \ ; \ \vec{\mathcal{U}} \ ; \ \bar{\mathcal{S}} \ ; \ \bar{\mathcal{H}} \rangle_k$$

  Since $\lceil \langle \vec{\mathcal{U}} \ ; \ \emptyset \ ; \ \bar{\mathcal{S}} \ ; \ \bar{\mathcal{H}} \rangle_k \rceil = \lceil \langle \emptyset \ ; \ \vec{\mathcal{U}} \ ; \ \bar{\mathcal{S}} \ ; \ \bar{\mathcal{H}} \rangle_k \rceil$, hence $\lceil \Omega'' \rceil = \lceil \Omega' \rceil$ and we trivially have $\mathcal{P} \rhd \lceil \Omega'' \rceil \mapsto^*_{\omega_\alpha^e} \lceil \Omega' \rceil$ of zero ω_α^e derivation step. Therefore $\mathcal{P} \rhd \lceil \Omega \rceil \mapsto^*_{\omega_\alpha^e} \lceil \Omega' \rceil$.

- (Loc 1) Step: Successor derivation step $\mathcal{P} \rhd \Omega'' \mapsto_{\omega_\alpha^e} \Omega'$ is of the form:

$$\mathcal{P} \ \rhd \ \Omega, \left(\begin{array}{l} \langle \vec{\mathcal{U}} \ ; \ (\texttt{[}k'\texttt{]}\, c, \vec{\mathcal{G}}) \ ; \ \bar{\mathcal{S}} \ ; \ \bar{\mathcal{H}} \rangle_k, \\ \langle \vec{\mathcal{U}}' \ ; \ \vec{\mathcal{G}}' \ ; \ \bar{\mathcal{S}}' \ ; \ \bar{\mathcal{H}}' \rangle_{k'} \end{array} \right) \ \mapsto_{\omega_0^e} \ \Omega, \left(\begin{array}{l} \langle \vec{\mathcal{U}} \ ; \ \vec{\mathcal{G}} \ ; \ \bar{\mathcal{S}} \ ; \ \bar{\mathcal{H}} \rangle_k, \\ \langle (\vec{\mathcal{U}}', c) \ ; \ \vec{\mathcal{G}}' \ ; \ \bar{\mathcal{S}}' \ ; \ \bar{\mathcal{H}}' \rangle_{k'} \end{array} \right)$$

  By definition, $\lceil \texttt{[}k'\texttt{]}\, c, \vec{\mathcal{G}} \rceil = \lceil \vec{\mathcal{G}} \rceil$ and since $k \neq k'$, we have $\text{Goals}(\Omega'')_{|k} = \text{Goals}(\Omega')_{|k}$. Thus for location k we have $\langle \lceil \vec{\mathcal{U}} \rceil, \lceil (\texttt{[}k'\texttt{]}\, c, \vec{\mathcal{G}}) \rceil, \lceil \bar{\mathcal{S}} \rceil, \text{Goals}(\Omega'')_{|k} \rangle_k = \langle \lceil \vec{\mathcal{U}} \rceil, \lceil \vec{\mathcal{G}} \rceil, \lceil \bar{\mathcal{S}} \rceil, \text{Goals}(\Omega')_{|k} \rangle_k$. For location k', $c, \lceil \vec{\mathcal{U}}' \rceil = \lceil \vec{\mathcal{U}}', c \rceil$ but since $\text{Goals}(\Omega'')_{|k} = c, \text{Goals}(\Omega')_{|k}$, we have $\langle \lceil \vec{\mathcal{U}}' \rceil, \lceil \vec{\mathcal{G}}' \rceil, \lceil \bar{\mathcal{S}}' \rceil, \text{Goals}(\Omega'')_{|k} \rangle_{k'} = \langle \lceil \vec{\mathcal{U}}', c \rceil, \lceil \vec{\mathcal{G}}' \rceil, \lceil \bar{\mathcal{S}}' \rceil, \text{Goals}(\Omega')_{|k} \rangle_{k'}$. These imply that $\lceil \Omega'' \rceil = \lceil \Omega' \rceil$. We trivially have $\mathcal{P} \rhd \lceil \Omega'' \rceil \mapsto^*_{\omega_\alpha^e} \lceil \Omega' \rceil$ of zero ω_α^e derivation step. Therefore $\mathcal{P} \rhd \lceil \Omega \rceil \mapsto^*_{\omega_\alpha^e} \lceil \Omega' \rceil$.

- (Loc 2) Step: Successor derivation step $\mathcal{P} \rhd \Omega'' \mapsto_{\omega_\alpha^e} \Omega'$ is of the form:

$$\mathcal{P} \ \rhd \ \Omega, \langle \vec{\mathcal{U}} \ ; \ (\texttt{[}k\texttt{]}\, c, \vec{\mathcal{G}}) \ ; \ \bar{\mathcal{S}} \ ; \ \bar{\mathcal{H}} \rangle_k \ \mapsto_{\omega_0^e} \ \Omega, \langle \vec{\mathcal{U}} \ ; \ (c, \vec{\mathcal{G}}) \ ; \ \bar{\mathcal{S}} \ ; \ \bar{\mathcal{H}} \rangle_k$$

  By definition, we have $c, \lceil \texttt{[}k\texttt{]}\, c, \vec{\mathcal{G}} \rceil = \lceil c, \vec{\mathcal{G}} \rceil$ and $\text{Goals}(\Omega'')_{|k} = c, \text{Goals}(\Omega')_{|k}$. These implies that we have $\langle \lceil \vec{\mathcal{U}} \rceil, \lceil \texttt{[}k\texttt{]}\, c, \vec{\mathcal{G}} \rceil, \lceil \bar{\mathcal{S}} \rceil, \text{Goals}(\Omega'')_{|k} \rangle_k = \langle \lceil \vec{\mathcal{U}} \rceil, \lceil c, \vec{\mathcal{G}} \rceil, \lceil \bar{\mathcal{S}} \rceil, \text{Goals}(\Omega')_{|k} \rangle_k$ and hence $\lceil \Omega'' \rceil = \lceil \Omega' \rceil$. We trivially have $\mathcal{P} \rhd \lceil \Omega'' \rceil \mapsto^*_{\omega_\alpha^e} \lceil \Omega' \rceil$ of zero ω_α^e derivation step. Therefore $\mathcal{P} \rhd \lceil \Omega \rceil \mapsto^*_{\omega_\alpha^e} \lceil \Omega' \rceil$.

- (Loc 3) Step: Successor derivation step $\mathcal{P} \rhd \Omega'' \mapsto_{\omega_\alpha^e} \Omega'$ is of the form:

$$\mathcal{P} \ \rhd \ \Omega, \langle \vec{\mathcal{U}} \ ; \ (\texttt{[}k'\texttt{]}\, c, \vec{\mathcal{G}}) \ ; \ \bar{\mathcal{S}} \ ; \ \bar{\mathcal{H}} \rangle_k \ \mapsto_{\omega_0^e} \ \Omega, \langle \vec{\mathcal{U}} \ ; \ \vec{\mathcal{G}} \ ; \ \bar{\mathcal{S}} \ ; \ \bar{\mathcal{H}} \rangle_k, \langle c \ ; \ \emptyset \ ; \ \emptyset \ ; \ \emptyset \rangle_{k'}$$

  By definition, we have $\lceil \texttt{[}k'\texttt{]}\, c, \vec{\mathcal{G}} \rceil = \lceil \vec{\mathcal{G}} \rceil$. However, since Ω' has the additional location $\langle c \ ; \ \emptyset \ ; \ \emptyset \ ; \ \emptyset \rangle_{k'}$, hence $c, \lceil \Omega'' \rceil^{\text{Goals}(\Omega'')} = \lceil \Omega' \rceil^{\text{Goals}(\Omega')}$. Yet we have the following in the second component of the $\lceil - \rceil$ definition:

$$\biguplus_{k \in (\text{Locs}(\text{Goals}(\Omega'')) - \text{Locs}(\Omega))} \langle k_{|\text{Goals}(\Omega'')} \rangle_k = c, \biguplus_{k \in (\text{Locs}(\text{Goals}(\Omega')) - \text{Locs}(\Omega))} \langle k_{|\text{Goals}(\Omega')} \rangle_k$$

  Hence we have $\lceil \Omega'' \rceil = \lceil \Omega' \rceil$ and thus $\mathcal{P} \rhd \lceil \Omega'' \rceil \mapsto^*_{\omega_\alpha^e} \lceil \Omega' \rceil$ of zero ω_α^e derivation step. Therefore $\mathcal{P} \rhd \lceil \Omega \rceil \mapsto^*_{\omega_\alpha^e} \lceil \Omega' \rceil$.

- (Act) Step: Successor derivation step $\mathcal{P} \rhd \Omega'' \mapsto_{\omega_\alpha^e} \Omega'$ is of the form:

$$\mathcal{P} \ \rhd \ \Omega, \langle \vec{\mathcal{U}} \ ; \ (p(\vec{t}), \vec{\mathcal{G}}) \ ; \ \bar{\mathcal{S}} \ ; \ \bar{\mathcal{H}} \rangle_k \ \mapsto_{\omega_0^e} \ \Omega, \langle \vec{\mathcal{U}} \ ; \ (p(\vec{t}) \# d : 1, \vec{\mathcal{G}}) \ ; \ (\bar{\mathcal{S}}, p(\vec{t}) \# d) \ ; \ \bar{\mathcal{H}} \rangle_k$$

  We have $\lceil p(\vec{t}), \vec{\mathcal{G}} \rceil = p(\vec{t}), \lceil p(\vec{t}) \# d : 1, \vec{\mathcal{G}} \rceil$, $p(\vec{t}), \lceil \bar{\mathcal{S}} \rceil = \lceil \bar{\mathcal{S}}, p(\vec{t}) \# d \rceil$ and $\text{Goals}(\Omega'')_{|k} = \text{Goals}(\Omega')_{|k}$. Hence we have:

$$\langle \lceil \vec{\mathcal{U}} \rceil, \lceil p(\vec{t}), \vec{\mathcal{G}} \rceil, \lceil \bar{\mathcal{S}} \rceil, \text{Goals}(\Omega'')_{|k} \rangle_k = \langle \lceil \vec{\mathcal{U}} \rceil, \lceil p(\vec{t}) \# d : 1, \vec{\mathcal{G}} \rceil, \lceil \bar{\mathcal{S}}, p(\vec{t}) \# d \rceil, \text{Goals}(\Omega')_{|k} \rangle_k$$

  Hence we have $\lceil \Omega'' \rceil = \lceil \Omega' \rceil$ and thus $\mathcal{P} \rhd \lceil \Omega'' \rceil \mapsto^*_{\omega_\alpha^e} \lceil \Omega' \rceil$ of zero ω_α^e derivation step. Therefore $\mathcal{P} \rhd \lceil \Omega \rceil \mapsto^*_{\omega_\alpha^e} \lceil \Omega' \rceil$.

- (Simp) Step: Successor derivation step $\mathcal{P} \rhd \Omega'' \mapsto_{\omega_\alpha^e} \Omega'$ is of the form:

$$\mathcal{P} \ \rhd \ \Omega, \langle \vec{\mathcal{U}} \ ; \ (c\#d : i, \vec{\mathcal{G}}) \ ; \ (\bar{\mathcal{S}}, P, S, c\#d) \ ; \ \bar{\mathcal{H}} \rangle_k \ \mapsto_{\omega_0^e} \ \Omega, \langle \vec{\mathcal{U}} \ ; \ (\mathrm{NF}(\mathrm{Inst}(\theta B)), \vec{\mathcal{G}}) \ ; \ (\bar{\mathcal{S}}, P) \ ; \ \bar{\mathcal{H}} \rangle_k$$

for some substitution θ. Applying the translation function and simplifying, we need to prove that:

$$\begin{array}{ll} & \mathcal{P} \ \rhd \ \lceil \Omega \rceil^{\vec{\mathcal{G}}''}, \langle \lceil \vec{\mathcal{U}} \rceil, \lceil \vec{\mathcal{G}} \rceil, \lceil \bar{\mathcal{S}} \rceil, \lceil P \rceil, \lceil S \rceil, c, \vec{\mathcal{G}}''_{|k} \rangle_k, \mathcal{A}'' \\ \mapsto_{\omega_\alpha^e} & \lceil \Omega \rceil^{\vec{\mathcal{G}}'}, \langle \lceil \vec{\mathcal{U}} \rceil, \mathrm{NF}(\mathrm{Inst}(\theta B))_{|k}, \lceil \vec{\mathcal{G}} \rceil, \lceil \bar{\mathcal{S}} \rceil, \lceil P \rceil, \vec{\mathcal{G}}'_{|k} \rangle_k, \mathcal{A}' \end{array}$$

$$\begin{array}{ll} \text{where} & \vec{\mathcal{G}}'' = \mathrm{Goals}(\Omega'') \text{ and } \vec{\mathcal{G}}' = \mathrm{Goals}(\Omega') \\ & \mathcal{A}'' = \biguplus_{k \in \mathrm{Locs}(\mathrm{Goals}(\Omega'')) - \mathrm{Locs}(\Omega'')} \langle \mathrm{Goals}(\Omega'')_{|k} \rangle_k \\ & \mathcal{A}' = \biguplus_{k \in \mathrm{Locs}(\mathrm{Goals}(\Omega')) - \mathrm{Locs}(\Omega')} \langle \mathrm{Goals}(\Omega')_{|k} \rangle_k \end{array}$$

more precisely, we want to show that this ω_α^e derivation step involves the application of the 0-neighbor restricted rule instance:

$$\texttt{[}k\texttt{]}\, P \setminus \texttt{[}k\texttt{]}\, S, \texttt{[}k\texttt{]}\, c \iff G \mid \theta B_k, \biguplus_{j \in \mathcal{I}_m} \theta B_j, \biguplus_{l \in \mathcal{I}_e} \theta B_l \qquad (A)$$

We known that this rule instance (A) with rule heads $\texttt{[}k\texttt{]}\, P$ and $\texttt{[}k\texttt{]}\, S$ and rule guard G such that $\models G$ exists in $\mathcal{P}$ because it has been assumed to apply in $\mathcal{P} \rhd \Omega'' \mapsto_{\omega_0^e} \Omega'$. We now need to show that the rule body $\theta B_k, \biguplus_{j \in \mathcal{I}_m} \theta B_j, \biguplus_{l \in \mathcal{I}_e} \theta B_l$ corresponds to new constraints that appear in $\lceil \Omega' \rceil$ but not $\lceil \Omega'' \rceil$. Specifically, we need to show the following:

  - That $\mathrm{NF}(\mathrm{Inst}(\theta B)) = \theta B_k, \biguplus_{j \in \mathcal{I}_m} \theta B_j, \biguplus_{l \in \mathcal{I}_e} \theta B_l$ such that $\mathrm{NF}(\mathrm{Inst}(\theta B))_{|k} = \theta B_k$, for each $j \in \mathcal{I}_m$ $\mathrm{NF}(\mathrm{Inst}(\theta B))_{|k_j} = \theta B_j$ and for each $l \in \mathcal{I}_e$ $\mathrm{NF}(\mathrm{Inst}(\theta B))_{|k_l} = \theta B_l$
  - For each $j \in \mathcal{I}_m$ such that $\langle \bar{\mathcal{S}}_j \rangle_{k_j} \in \lceil \Omega'' \rceil^{\vec{\mathcal{G}}''}$ for some $\bar{\mathcal{S}}_j$, we have $\langle \bar{\mathcal{S}}_j, \theta B_j \rangle_{k_j} \in \lceil \Omega' \rceil^{\vec{\mathcal{G}}'}$
  - For each $l \in \mathcal{I}_e$ such that $\langle \bar{\mathcal{S}}_l \rangle_{k_l} \notin \mathcal{A}''$ for any $\bar{\mathcal{S}}_l$, we have $\langle \theta B_l \rangle_{k_l} \in \mathcal{A}'$

By definition of the translation function $\lceil - \rceil$, each translated location component k_j of $\lceil \Omega'' \rceil^{\vec{\mathcal{G}}''}$, contains $\vec{\mathcal{G}}''_{|k_j}$. By comparison of new constraints in Ω' but not in Ω'', we have for each $j \in \mathcal{I}_m$ $\mathrm{NF}(\mathrm{Inst}(\theta B))_{|k_j} = \theta B_j$. Hence the difference of $\lceil \Omega' \rceil^{\vec{\mathcal{G}}'}$ and $\lceil \Omega'' \rceil^{\vec{\mathcal{G}}''}$ is captured by $\biguplus_{j \in \mathcal{I}_m} \mathrm{NF}(\mathrm{Inst}(\theta B))_{|j}$. For each $l \in \mathcal{I}_e$ such that $k_l \notin \mathrm{Locs}(\Omega'')$, k_l will appear in $\mathcal{A}' = \biguplus_{k \in \mathrm{Locs}(\mathrm{Goals}(\Omega')) - \mathrm{Locs}(\Omega')} \langle \mathrm{Goals}(\Omega')_{|k} \rangle_k$, i.e, $\langle \bar{\mathcal{S}}_l \rangle_{k_l}$ for some $\bar{\mathcal{S}}_l$. This is because the only change in goals between Ω'' to Ω' is the introduction of $\mathrm{NF}(\mathrm{Inst}(\theta B))$ and all locations not found in Ω'' will appear in $\mathcal{A}'$, by its definition. Furthermore, $\bar{\mathcal{S}}_l = \mathrm{NF}(\mathrm{Inst}(\theta B))_{|k_l}$. Hence we have shown that $\mathrm{NF}(\mathrm{Inst}(\theta B))$ captures all constraints in $\lceil \Omega' \rceil$ that does not appear in $\lceil \Omega'' \rceil$, and furthermore with the constraints distributed to the corresponding locations. Since constraints that appear in $\lceil \Omega'' \rceil$ but not in $\lceil \Omega' \rceil$ is exactly 'S, c' in location k, these corresponds to the effects of applying rule instance (A) via the ω_α^e derivation step, in other words: $\mathcal{P} \rhd \lceil \Omega'' \rceil \mapsto^*_{\omega_\alpha^e} \lceil \Omega' \rceil$.

- (Prop) Step: The prove of this step is similar to (Simp), with the exception that active constraint $c\#d$ is kept in the store in successor state $\lceil \Omega' \rceil$. This is consistent to the application of a rule instance where c matches with a propagated rule head, hence it does not invalidate the proof. Ω' does still contain the active goal constraint $c\#d : i$, but this is not visible in $\lceil \Omega' \rceil$ as defined by the $\lceil - \rceil$ translation of goals. Lastly, history of k in Ω' is extended by $(d, \mathrm{Ids}(P, S))$, but histories are simply discarded in $\lceil \Omega' \rceil$. Therefore we still have $\mathcal{P} \rhd \lceil \Omega'' \rceil \mapsto^*_{\omega_\alpha^e} \lceil \Omega' \rceil$.

- (Next) Step: Successor derivation step $\mathcal{P} \rhd \Omega'' \mapsto_{\omega_\alpha^e} \Omega'$ is of the form:

$$\mathcal{P} \ \rhd \ \Omega, \langle \vec{\mathcal{U}} \ ; \ (c\#d : i, \vec{\mathcal{G}}) \ ; \ \bar{\mathcal{S}} \ ; \ \bar{\mathcal{H}} \rangle_k \ \mapsto_{\omega_0^e} \ \Omega, \langle \vec{\mathcal{U}} \ ; \ (c\#d : (i+1), \vec{\mathcal{G}}) \ ; \ \bar{\mathcal{S}} \ ; \ \bar{\mathcal{H}} \rangle_k$$

Since we have $\lceil c\#d : i, \vec{\mathcal{G}} \rceil = \lceil c\#d : (i+1), \vec{\mathcal{G}} \rceil$, we have $\lceil \Omega'' \rceil = \lceil \Omega' \rceil$ and thus $\mathcal{P} \rhd \lceil \Omega'' \rceil \mapsto^*_{\omega_\alpha^e} \lceil \Omega' \rceil$ of zero ω_α^e derivation step. Therefore $\mathcal{P} \rhd \lceil \Omega \rceil \mapsto^*_{\omega_\alpha^e} \lceil \Omega' \rceil$.

- (Drop) Step: Successor derivation step $\mathcal{P} \rhd \Omega'' \mapsto_{\omega_\alpha^e} \Omega'$ is of the form:

$$\mathcal{P} \ \rhd \ \Omega, \langle \vec{\mathcal{U}} \ ; \ (c\#d : i, \vec{\mathcal{G}}) \ ; \ \bar{\mathcal{S}} \ ; \ \bar{\mathcal{H}} \rangle_k \ \mapsto_{\omega_0^e} \ \Omega, \langle \vec{\mathcal{U}} \ ; \ \vec{\mathcal{G}} \ ; \ \bar{\mathcal{S}} \ ; \ \bar{\mathcal{H}} \rangle_k$$

  Since we have $\lceil c\#d : i, \vec{\mathcal{G}} \rceil = \lceil \vec{\mathcal{G}} \rceil$, we have $\lceil \Omega'' \rceil = \lceil \Omega' \rceil$ and thus $\mathcal{P} \rhd \lceil \Omega'' \rceil \mapsto^*_{\omega_\alpha^e} \lceil \Omega' \rceil$ of zero ω_α^e derivation step. Therefore $\mathcal{P} \rhd \lceil \Omega \rceil \mapsto^*_{\omega_\alpha^e} \lceil \Omega' \rceil$.

Therefore, we have shown $\mathcal{P} \rhd \lceil \Omega \rceil \mapsto^*_{\omega_\alpha^e} \lceil \Omega'' \rceil \mapsto^*_{\omega_\alpha^e} \lceil \Omega' \rceil$ for all forms of derivation step $\mathcal{P} \rhd \lceil \Omega'' \rceil \mapsto^*_{\omega_\alpha^e} \lceil \Omega' \rceil$ and thus completing the inductive proof of $\mathcal{P} \rhd \lceil \Omega \rceil \mapsto^*_{\omega_\alpha^e} \lceil \Omega' \rceil$. □

Lemma 7 (Always Active Rule Head Instances) *Given a 0-neighbor restricted CHR^e program $\mathcal{P}$ and an initial state Ω, for any reachable state Ω' such that $\mathcal{P} \rhd \Omega \mapsto^*_{\omega_0^e} \Omega'$, all rule instances in Ω' must be active.*

Proof: We proof this by induction on derivation steps between initial state Ω and reachable state Ω'. For the base case, we consider $\mathcal{P} \rhd \Omega \mapsto^*_{\omega_0^e} \Omega'$ of zero derivation step, hence $\Omega' = \Omega$. Since Ω' is an initial state, all stores $\bar{\mathcal{S}}$ in Ω' are empty, therefore we have no rule instances in Ω' and the property holds by default. For inductive case, we assume that we have some derivation of m steps to some reachable state Ω'' (i.e, $\mathcal{P} \rhd \Omega \mapsto^*_{\omega_0^e} \Omega''$) and that all rule instances in Ω'' are active. We will now show that for any successor derivation step $\mathcal{P} \rhd \Omega'' \mapsto^*_{\omega_0^e} \Omega'$, reachable state Ω' is such that all rule instances are active. We show this by considering all possible forms of this successor derivation:

- (Flush) Step: This step modifies location k in Ω'' by moving its buffers into the goals. Constraint store of k remains the same, hence rule instances in Ω' are similar to those in Ω''. Therefore all rule instances in Ω' are active.

- (Loc 1) Step: This step modifies two locations k and k' in Ω'' by moving leading goal of k, $\texttt{[}k'\texttt{]}c$ to goals of k'. Constraint stores of k and k' remain the same, hence rule instances in Ω' are similar to those in Ω''. Therefore all rule instances in Ω' are active.

- (Loc 2) Step: This step modifies location k in Ω'' by changing leading goal from $\texttt{[}k\texttt{]}c$ to c. Constraint store of k remains the same, hence rule instances in Ω' are similar to those in Ω''. Therefore all rule instances in Ω' are active.

- (Loc 3) Step: This step modifies location k in Ω'' by removing its leading goal $\texttt{[}k'\texttt{]}c$, and creating a new location k' with c in its buffer and everything else empty. Constraint store of k remains the same and constraint store of k' is empty, hence rule instances in Ω' are similar to those in Ω''. Therefore all rule instances in Ω' are active.

- (Act) Step: This step modifies location k in Ω'' by changing leading goal $p(\vec{t})$ into $p(\vec{t})\#d : i$ and also adding $p(\vec{t})\#d$ into the store $\bar{\mathcal{S}}$. This possibly adds new rule instances R in Ω' which has $p(\vec{t})\#d$ as a rule head. Since $p(\vec{t})\#d : i$ is in the goals, all these newly added rule instances are active. Therefore all rule instances in Ω' are active.

- (Simp) Step: This step modifies location k in Ω'' by applying an active rule instance R which has the leading goal $c\#d : i$ as one of its rule heads. Constraints matching the simplified heads S are removed from the store: the removes rule instances but does not affect activeness of other rule instances that remains. Rule body of R is added to goals, hence does not remove or add new rule instances. Leading goal $c\#d : i$ is removed from the goals, but since constraint $c\#d$ in the constraint store is simplified as well, all active rule instances that depend on $c\#d : i$ to be active will not be found in Ω'. Therefore all rule instances that remain in Ω' are still active.

- (Prop) Step: This step is similar to the (Simp) step. Except that leading goal $c\#d : i$ is matched to a propagated rule head of R. Since constraint $c\#d$ is not removed from the store, hence there may exist other rule instances that contain $c\#d$ and depend on its corresponding goal $c\#d : i$ to be active. But leading goal $c\#d : i$ is retained in Ω', therefore these rule instances remain active. This step also extends the history with the applied rule instances rule head ids. While this renders R no longer a rule instance in Ω', no new rule instances are added. Therefore all rule instances that remain in Ω' are still active.
- (Next) Step: This step modifies a location k in Ω by changing leading goal $c\#d : i$ to $c\#d : (i + 1)$. The premise of this rule explicitly asserts that no (Simp) and (Prop) steps must apply for this goal $c\#d : i$. This effectively enforces that this step only have applied because Ω'' contains no rule instances that contain $c\#d : i$ hence $c\#d : i$ can be removed without making any rule instances inactive. Adding $c\#d : (i + 1)$ possibly introduces new rule instances, but since it is part of the goals, all these newly added rule instances are active. Therefore all rule instances that remain in Ω' are still active.
- (Drop) Step: This step modifies a location k in Ω by removing its leading goal $c\#d : i$. The premise of this rule restricts it to only cases where i is not a occurrence index that appear in the program $\mathcal{P}$, hence there can never be any rule instances that contain $c\#d : i$ and it can be safely removed without rendering any rule instances inactive. Therefore all rule instances that remain in Ω' are still active.

Therefore, we have completed the inductive step proof for $\mathcal{P} \rhd \Omega \mapsto^*_{\omega_0^e} \Omega'$. Hence all rule instances in any reachable state Ω' are active. □

Theorem 8 (Exhaustiveness of Rule Application in ω_0^e) *Given a 0-neighbor restricted CHR^e program $\mathcal{P}$ and reachable states Ω and Ω', if $\mathcal{P} \rhd \Omega \mapsto^*_{\omega_0^e} \Omega'$ and Ω' is terminal, then there exists no rule instance $R \in \mathcal{P}$ such that R is applicable.*

Proof: We proof this by negation, showing that if we assume otherwise, we contradict Lemma 7. Assume that we have $\mathcal{P} \rhd \Omega \mapsto^*_{\omega_0^e} \Omega'$ and Ω' is terminal, but there exists some rule instance $R \in \mathcal{P}$ such that R is applicable. Suppose this rule instance is applicable in some location $k \in \text{Locs}(\Omega')$. Since Lemma 7 states that any rule instance R must be active, hence there must exists some $c\#d : i$ in the goals of k which matches the i^{th} rule head occurrence of rule instance R. However Ω' is terminal, in other words, its goals and buffer are suppose to be empty. Hence we have a contradiction. Therefore, it must be the case that in Ω' there exists no rule instance $R \in \mathcal{P}$ such that R is applicable. □

Lemma 9 (Monotonicity of ω_0^e semantics) *Given a 0-neighbor restricted program $\mathcal{P}$, and reachable states $\Omega, \Omega', \Omega''$,*

1. *if $\mathcal{P} \rhd \Omega \mapsto_{\omega_0^e} \Omega'$, then $\mathcal{P} \rhd \Omega, \Omega'' \mapsto_{\omega_0^e} \Omega', \Omega''$*
2. *if $\mathcal{P} \rhd \Omega \mapsto^*_{\omega_0^e} \Omega'$, then $\mathcal{P} \rhd \Omega, \Omega'' \mapsto^*_{\omega_0^e} \Omega', \Omega''$*
3. *if $\mathcal{P} \rhd \Omega \mapsto^{||}_{\omega_0^e} \Omega'$, then $\mathcal{P} \rhd \Omega, \Omega'' \mapsto^{||}_{\omega_0^e} \Omega', \Omega''$*
4. *if $\mathcal{P} \rhd \Omega \mapsto^{||*}_{\omega_0^e} \Omega'$, then $\mathcal{P} \rhd \Omega, \Omega'' \mapsto^{||*}_{\omega_0^e} \Omega', \Omega''$*

Proof:

1. We prove by structural induction over all forms of $\mapsto_{\omega_0^e}$ derivation steps. For the (Loc 3) step, we assume implicit α-renaming of location k' during composition of a larger context Ω''. All other cases are straight-forward, each simply to show that unmodified fragment of the ensemble state Ω can be extended without affecting the derivation step.

2. We prove by induction on $\mapsto_{\omega_0^e}$ derivation steps and also relying on the proof of (1).

3. We prove by structural induction over the (Single) and (Concurrent) derivation steps of $\mapsto^{||}_{\omega_0^e}$ and also relying on the proof of (1).

4. We prove by induction on $\mapsto^{||}_{\omega_0^e}$ derivation steps and also relying on the proof of (3).

□

Theorem 10 (Serializability of Concurrent ω_0^e Derivations) *Given a* 0*-neighbor restricted program* $\mathcal{P}$*, and reachable states* Ω, Ω'*, if we have* $\mathcal{P} \rhd \Omega \mapsto^{||*}_{\omega_0^e} \Omega'$ *then we must have* $\mathcal{P} \rhd \Omega \mapsto^{*}_{\omega_0^e} \Omega'$

Proof: We first proof this for one $\mapsto^{||}_{\omega_0^e}$ derivation step, $\mathcal{P} \rhd \Omega \mapsto^{||}_{\omega_0^e} \Omega'$: By structural induction on all forms of $\mapsto^{||}_{\omega_0^e}$ derivation step, we consider the base structure (Single). Hence $\mathcal{P} \rhd \Omega \mapsto^{||}_{\omega_0^e} \Omega'$ consist of a single $\mapsto_{\omega_0^e}$ derivation (i.e, $\mathcal{P} \rhd \Omega \mapsto_{\omega_0^e} \Omega'$) and we naively have $\mathcal{P} \rhd \Omega \mapsto^{*}_{\omega_0^e} \Omega'$. For the (Concurrent) step, we have the derivation step form:

$$\mathcal{P} \rhd (\Omega_1, \Omega_2) \mapsto^{||}_{\omega_0^e} (\Omega_1', \Omega_2') \text{ such that } \Omega = (\Omega_1, \Omega_2) \text{ and } \Omega' = (\Omega_1', \Omega_2')$$

Hence, assuming that we have $\mathcal{P} \rhd \Omega_1 \mapsto^{||}_{\omega_0^e} \Omega_1'$ and $\mathcal{P} \rhd \Omega_1 \mapsto^{||}_{\omega_0^e} \Omega_2'$ such that $\mathcal{P} \rhd \Omega_1 \mapsto_{\omega_0^e} \Omega_1'$ and $\mathcal{P} \rhd \Omega_1 \mapsto_{\omega_0^e} \Omega_2'$ by Lemma 9, we have:

$$\mathcal{P} \rhd \Omega_1, \Omega_2 \mapsto_{\omega_0^e} \Omega_1', \Omega_2 \mapsto_{\omega_0^e} \Omega_1', \Omega_2'$$

Therefore we have $\mathcal{P} \rhd (\Omega_1, \Omega_2) \mapsto^{*}_{\omega_0^e} (\Omega_1'', \Omega_2'')$.

We have shown that given $\mathcal{P} \rhd \Omega \mapsto^{||}_{\omega_0^e} \Omega'$, we have $\mathcal{P} \rhd \Omega \mapsto^{*}_{\omega_0^e} \Omega'$, now we prove the same for transitive $\mapsto^{||}_{\omega_0^e}$ derivation steps by induction on $\mapsto^{||*}_{\omega_0^e}$ derivation steps. For base case, we have $\mathcal{P} \rhd \Omega \mapsto^{||*}_{\omega_0^e} \Omega'$ of zero derivation step, hence $\Omega = \Omega'$ and we naively have $\mathcal{P} \rhd \Omega \mapsto^{*}_{\omega_0^e} \Omega'$. For the inductive case, we assume that we have the property for concurrent derivations of m steps, i.e, $\mathcal{P} \rhd \Omega \mapsto^{||*}_{\omega_0^e} \Omega''$ of m steps implies $\mathcal{P} \rhd \Omega \mapsto^{*}_{\omega_0^e} \Omega''$. We consider Ω' the successor of Ω'' after one derivation step, $\mathcal{P} \rhd \Omega'' \mapsto^{||}_{\omega_0^e} \Omega'$. Since we show that for a single concurrent derivation step, we have $\mathcal{P} \rhd \Omega'' \mapsto^{*}_{\omega_0^e} \Omega'$, hence we also have $\mathcal{P} \rhd \Omega \mapsto^{*}_{\omega_0^e} \Omega'$.

Therefore, we have shown that given $\mathcal{P} \rhd \Omega \mapsto^{||*}_{\omega_0^e} \Omega'$ then we must have $\mathcal{P} \rhd \Omega \mapsto^{*}_{\omega_0^e} \Omega'$ □

A.3 Proofs for 1-Neighbor Restricted Rule Basic Encoding Scheme

Lemma 11 (Prefix Executions of Encoding) *Given a locally quiescent* 1*-neighbor restricted program* $\mathcal{P}_1$ *and a* 0*-neighbor restricted program* $\mathcal{P}_0$ *such that* $\mathcal{P}_1 \rightsquigarrow^{\text{basic}}_{\text{1Nb}} \mathcal{P}_0$*, if we have reachable states* Ω, Ω' *such that* $\mathcal{P}_0 \rhd \Omega \mapsto_{\omega_0^e} \Omega'$*, then we have the following:*

1. *If* $\mathcal{P}_0 \rhd \Omega \mapsto^{r\_2}_{\omega_0^e} \Omega'$*, then there exists some reachable states* Ω'' *and* Ω'''*, such that* $\mathcal{P}_0 \rhd \Omega'' \mapsto^{r\_1}_{\omega_0^e} \Omega'''$ *and* $\mathcal{P}_0 \rhd \Omega''' \mapsto^{*}_{\omega_0^e} \Omega$

2. *If* $\mathcal{P}_0 \rhd \Omega \mapsto^{r\_3}_{\omega_0^e} \Omega'$*, then there exists some reachable states* Ω'' *and* Ω'''*, such that* $\mathcal{P}_0 \rhd \Omega'' \mapsto^{r\_2}_{\omega_0^e} \Omega'''$ *and* $\mathcal{P}_0 \rhd \Omega''' \mapsto^{*}_{\omega_0^e} \Omega$

3. *If* $\mathcal{P}_0 \rhd \Omega \mapsto^{r\_4a}_{\omega_0^e} \Omega'$*, then there exists some reachable states* Ω'' *and* Ω'''*, such that* $\mathcal{P}_0 \rhd \Omega'' \mapsto^{r\_3}_{\omega_0^e} \Omega'''$ *and* $\mathcal{P}_0 \rhd \Omega''' \mapsto^{*}_{\omega_0^e} \Omega$

4. *If* $\mathcal{P}_0 \rhd \Omega \mapsto^{r\_4b}_{\omega_0^e} \Omega'$, *then there exists some reachable states* Ω'' *and* Ω''', *such that* $\mathcal{P}_0 \rhd \Omega'' \mapsto^{r\_3}_{\omega_0^e} \Omega'''$ *and* $\mathcal{P}_0 \rhd \Omega''' \mapsto^{*}_{\omega_0^e} \Omega$

Proof: These are easily proved by reasoning about the definition of encoding rules of Figure 13:

1. By definition of encoding rule *r_1* and *r_2*, the former has *r_req*(Xs) as a rule body, while the latter has *r_req*(Xs) as a rule head. Since the synchronizing predicate *r_req* appears in no where else, Application of *r_2* encoding rule must always be preceded by application of a matching instance of *r_1*.

2. By definition of encoding rule *r_2* and *r_3*, the former has *r_match*(Rs) as a rule body, while the latter has *r_match*(Rs) as a rule head. Since the synchronizing predicate *r_match* appears in no where else, Application of *r_3* encoding rule must always be preceded by application of a matching instance of *r_2*.

3. By definition of encoding rule *r_3* and *r_4a*, the former has *r_commit*(Rs) as a rule body, while the latter has *r_commit*(Rs) as a rule head. Since the synchronizing predicate *r_commit* appears in no where else, Application of *r_4a* encoding rule must always be preceded by application of a matching instance of *r_3*.

4. By definition of encoding rule *r_3* and *r_4b*, the former has *r_commit*(Rs) as a rule body, while the latter has *r_commit*(Rs) as a rule head. Since the synchronizing predicate *r_commit* appears in no where else, Application of *r_4b* encoding rule must always be preceded by application of a matching instance of *r_3*.

□

Lemma 12 ($\rightsquigarrow^{\text{basic}}_{\text{1Nb}}$ Preserves Local Quiescence) *Given a locally quiescent* 1*-neighbor restricted program* $\mathcal{P}_1$ *and a* 0*-neighbor restricted program* $\mathcal{P}_0$ *such that* $\mathcal{P}_1 \rightsquigarrow^{\text{basic}}_{\text{1Nb}} \mathcal{P}_0$, *then* $\mathcal{P}_0$ *is also locally quiescent.*

Proof: By definition of the $\rightsquigarrow^{\text{basic}}_{\text{1Nb}}$ translation operation, encoding rules *r_1*, *r_2*, *r_3* and *r_4b* are guaranteed to be locally quiescent. This is because their rule bodies only include constraints to be delivered to its partner location, hence the rules are conservatively locally quiescent. Local quiescence of encoding rule *r_4a* however, depends on the local quiescence of the original rule in $\mathcal{P}_1$. Since we assume that $\mathcal{P}_1$ is locally quiescent, then *r_4a* must be locally quiescent. □

Lemma 13 (1-Neighbor Commit-Free Reachability) *Given a* 1*-neighbor restricted program* $\mathcal{P}_1$ *and a* 0*-neighbor restricted program* $\mathcal{P}_0$ *such that* $\mathcal{P}_1 \rightsquigarrow^{\text{basic}}_{\text{1Nb}} \mathcal{P}_0$ *and states* Ω *reachable by* $\mathcal{P}_0$, *if* Ω *is not commit-free, then there exists some commit-free state* Ω' *such that* $\mathcal{P}_0 \rhd \Omega \mapsto^{*}_{\omega_0^e} \Omega'$.

Proof: We prove this by considering all possible forms of derivations that lead to non-commit free states. We consider all forms of 0-neighbor restricted encodings that contains the *r_3*, *r_4a* and *r_4b* encoding rules that rewrites (add or delete) *r_commit* synchronizing constraints. By definition of the encodings, the only rule that produces *r_commit* constraints is *r_3*, hence derivations that reach a non commit-free state Ω must consist of several applications of the *r_3* rule. Suppose that one of rule application is contained in the

following derivation:

$$\mathcal{P}_0 \rhd \quad \Omega, \left(\begin{array}{c} \langle \vec{\mathcal{U}}_x \ ; \ \vec{\mathcal{G}}_x \ ; \ \left(\bar{\mathcal{S}}_x, \boxed{P_x, S_x, r\_match(Rs)\#d_1} \right) \ ; \ \bar{\mathcal{H}}_x \rangle_x \\ \langle \vec{\mathcal{U}}_y \ ; \ \vec{\mathcal{G}}_y \ ; \ \bar{\mathcal{S}}_y \ ; \ \bar{\mathcal{H}}_y \rangle_y \end{array} \right) \qquad -(1)$$

$$\mapsto_{\omega_0^e} \quad \Omega, \left(\begin{array}{c} \langle \vec{\mathcal{U}}_x \ ; \ \left(\texttt{[}y\texttt{]}\, r\_commit(Rs), \vec{\mathcal{G}}_x \right) \ ; \ (\bar{\mathcal{S}}_x, P_x) \ ; \ \bar{\mathcal{H}}_x \rangle_x \\ \langle \vec{\mathcal{U}}_y \ ; \ \vec{\mathcal{G}}_y \ ; \ \bar{\mathcal{S}}_y \ ; \ \bar{\mathcal{H}}_y \rangle_y \end{array} \right) \qquad -(2)$$

$$\mapsto_{\omega_0^e} \quad \Omega, \left(\begin{array}{c} \langle \vec{\mathcal{U}}_x \ ; \ \vec{\mathcal{G}}_x \ ; \ (\bar{\mathcal{S}}_x, P_x) \ ; \ \bar{\mathcal{H}}_x \rangle_x \\ \langle \left(\vec{\mathcal{U}}_y, \texttt{[}y\texttt{]}\, r\_commit(Rs) \right) \ ; \ \vec{\mathcal{G}}_y \ ; \ \bar{\mathcal{S}}_y \ ; \ \bar{\mathcal{H}}_y \rangle_y \end{array} \right) \qquad -(3)$$

$$\mapsto^*_{\omega_0^e} \quad \Omega, \left(\begin{array}{c} \langle \vec{\mathcal{U}}'_x \ ; \ \vec{\mathcal{G}}'_x \ ; \ \bar{\mathcal{S}}'_x \ ; \ \bar{\mathcal{H}}'_x \rangle_x \\ \langle \vec{\mathcal{U}}'_y \ ; \ \left(\boxed{r\_commit(Rs)\#d_2 : j}, \vec{\mathcal{G}}'_y \right) \ ; \ \left(\bar{\mathcal{S}}'_y, \boxed{r\_commit(Rs)\#d_2} \right) \ ; \ \bar{\mathcal{H}}'_y \rangle_y \end{array} \right) \qquad -(4)$$

Suppose that x and y the are primary and neighbor locations of the *r_3* rule instance. By the definition of ω_0^e derivation steps, we are guaranteed that from an initial state such that a rule instance of *r_3* is active in location x (highlighted in the derivation step (*1*) in above derivation) is applied (step (*2*) via (Simp) or (Prop) ω_0^e derivation step) and delivered to location y (step (*3*) via (Loc 1) ω_0^e derivation step), there exists some successor state ((*4*)) such that $r\_commit(Rs)\#d_2 : j$ is an active constraint of y. This is guaranteed because we assume that CHR^{e}programs are locally terminating, hence local execution eventually reach quiescences (goals are emptied) and buffer of y containing $r\_commit(Rs)$ is moved into the goals. Eventually, $r\_commit(Rs)\#d_2 : j$ will be the active constraint. From (*4*), there are two possibilities: If matching obligation of y is still valid in (*4*) (i.e, $P_y, S_y \subseteq \bar{\mathcal{S}}'_y$), then we have the following derivation:

$$\mathcal{P}_0 \rhd \quad \left(\begin{array}{c} \langle \vec{\mathcal{U}}'_x \ ; \ \vec{\mathcal{G}}'_x \ ; \ \bar{\mathcal{S}}'_x \ ; \ \bar{\mathcal{H}}'_x \rangle_x \\ \langle \vec{\mathcal{U}}'_y \ ; \ \left(r\_commit(Rs)\#d_2 : j, \vec{\mathcal{G}}'_y \right) \ ; \ \left(\bar{\mathcal{S}}''_y \boxed{, P_y, S_y, r\_commit(Rs)\#d_2} \right) \ ; \ \bar{\mathcal{H}}'_y \rangle_y \end{array} \right) \qquad -(4)$$

$$\mapsto_{\omega_0^e} \quad \left(\begin{array}{c} \langle \vec{\mathcal{U}}'_x \ ; \ \vec{\mathcal{G}}'_x \ ; \ \bar{\mathcal{S}}'_x \ ; \ \bar{\mathcal{H}}'_x \rangle_x \\ \langle \vec{\mathcal{U}}'_y \ ; \ \left(B, \vec{\mathcal{G}}'_y \right) \ ; \ \left(\bar{\mathcal{S}}''_y, P_y \right) \ ; \ \bar{\mathcal{H}}'_y \rangle_y \end{array} \right) \qquad -(5)$$

This results to a state which is commit free. If matching obligation of y is no longer valid in (*4*), *r_4b* applies, consuming the commit synchronizing constraint $r\_commit(Rs)$. Since application of *r_4b* *universal* in that it always applies if *r_4a* doesn't, hence any non-commit free state can reach a commit free state by a series of applications of either *r_4a* or *r_4b*. □

Lemma 14 (Basic Encoding Rule Serializability) *Given a* 1*-neighbor restricted and locally quiescent CHR^e program $\mathcal{P}_1$ and a* 0*-neighbor restricted CHR^e program $\mathcal{P}_0$ such that $\mathcal{P}_1 \rightsquigarrow_{1\text{Nb}} \mathcal{P}_0$ and commit free abstract states $\mathcal{A}_1$ $\mathcal{A}_2$, $\mathcal{A}_3$ and $\mathcal{A}_4$, given that:*

1. *For some basic encoding rule instances r_3 and r_4a, we have $\mathcal{P}_0 \rhd \mathcal{A}_1 \mapsto^{r\_3}_{\omega^e_\alpha} \mathcal{A}_2 \mapsto^*_{\omega^e_\alpha} \mathcal{A}_3 \mapsto^{r\_4a}_{\omega^e_\alpha} \mathcal{A}_4$ then there exists some $\mathcal{A}'_2$ and $\mathcal{A}'_3$ such that $\mathcal{P}_0 \rhd \mathcal{A}_1 \mapsto^*_{\omega^e_\alpha} \mathcal{A}'_2 \mapsto^{r\_3}_{\omega^e_\alpha} \mathcal{A}_3 \mapsto^{r\_4a}_{\omega^e_\alpha} \mathcal{A}'_3 \mapsto^*_{\omega^e_\alpha} \mathcal{A}_4$*
2. *For some basic encoding rule instances r_3 and r_4b, we have $\mathcal{P}_0 \rhd \mathcal{A}_1 \mapsto^{r\_3}_{\omega^e_\alpha} \mathcal{A}_2 \mapsto^*_{\omega^e_\alpha} \mathcal{A}_3 \mapsto^{r\_4b}_{\omega^e_\alpha} \mathcal{A}_4$ then there exists some $\mathcal{A}'_2$ and $\mathcal{A}'_3$ such that $\mathcal{P}_0 \rhd \mathcal{A}_1 \mapsto^*_{\omega^e_\alpha} \mathcal{A}'_2 \mapsto^{r\_3}_{\omega^e_\alpha} \mathcal{A}_3 \mapsto^{r\_4b}_{\omega^e_\alpha} \mathcal{A}'_3 \mapsto^*_{\omega^e_\alpha} \mathcal{A}_4$*

Proof: Let r be any 1-neighbor restricted rule instance of the general form:

$$r : [X]\,P_x, [X]\,P'_x, [y]\,P_y \setminus [X]\,S_x, [y]\,S_y \Longleftrightarrow G \mid \exists\, \bar{z}.\ [X]\,D_x, [Y]\,D_y, \biguplus_{j\in\mathcal{I}_m} [k_j]\,D_j, \biguplus_{l\in\mathcal{I}_e} [k_l]\,D_l \in \mathcal{P}_1$$

where k_j for $j \in \mathcal{I}_m$, $k_j \in \mathrm{FV}(P_x, P'_x, P_y, S_x, S_y)$ are *non-matched forwarding locations.*
k_l for $l \in \mathcal{I}_e$, $k_l \in \bar{z}$ are *existential forwarding locations.*

for some location X and Y. Hence this instance rule is translated into the following r_3, r_4a and r_4b encoding rule instances:

$r\_3 : [X]\,P_x \setminus [X]\,P'_x, [X]\,S_x, [X]\,r\_match(Rs) \Longleftrightarrow [Y]\,r\_commit(Rs)$
$r\_4a : [Y]\,P_y \setminus [Y]\,S_y, [Y]\,r\_commit(Rs) \Longleftrightarrow \exists \bar{z}\,[X]\,P'_x, [X]\,D_x, [Y]\,D_y, \biguplus_{j\in\mathcal{I}_m} [k_j]\,D_j, \biguplus_{l\in\mathcal{I}_e} [k_l]\,D_l$
$r\_4b : [Y]\,r\_commit(Rs) \Longleftrightarrow [X]\,P'_x, [X]\,S_x$

We now will show that derivation steps in the derivation $\mathcal{P}_0 \rhd \mathcal{A} \mapsto^*_{\omega^e_\alpha} \mathcal{A}'$ can be permutated such that an application of r_3 instance is immediately followed by an application r_4a or r_4b. We first consider any matching pairs of r_3 and r_4a. For this case the derivation $\mathcal{P}_0 \rhd \mathcal{A}_1 \mapsto^{r\_3}_{\omega^e_\alpha} \mathcal{A}_2 \mapsto^*_{\omega^e_\alpha} \mathcal{A}_3 \mapsto^{r\_4a}_{\omega^e_\alpha} \mathcal{A}_4$ is of the form:

$$\begin{array}{lll}
\mathcal{P} \rhd & \mathcal{A}'', \biguplus_{j\in\mathcal{I}_m}\langle \bar{\mathcal{S}}_j\rangle_{k_j}, \left(\begin{array}{c} \langle \bar{\mathcal{S}}_x, \boxed{P_x, P'_x, S_x, r_match(Rs)}\rangle_X, \\ \langle \mathcal{S}_y\rangle_Y \end{array}\right) & \text{Apply } r_3 \text{ instance} \\
\mapsto^{r_3}_{\omega^e_0} & \boxed{\mathcal{A}'', \biguplus_{j\in\mathcal{I}_m}\langle \bar{\mathcal{S}}_j\rangle_{k_j}}, \left(\begin{array}{c} \langle \boxed{\bar{\mathcal{S}}_x}, P_x\rangle_X, \\ \langle \boxed{\bar{\mathcal{S}}_y}, r_commit(Rs)\rangle_Y \end{array}\right) & \text{Unspecified interleaving derivations} \\
\mapsto^*_{\omega^e_0} & \mathcal{A}'''', \biguplus_{j\in\mathcal{I}_m}\langle \bar{\mathcal{S}}''_j\rangle_{k_j}, \left(\begin{array}{c} \langle \bar{\mathcal{S}}''_x, P_x\rangle_X, \\ \langle \bar{\mathcal{S}}'_y, \boxed{P_y, S_y, r_commit(Rs)}\rangle_Y \end{array}\right) & \text{Apply } r_4a \text{ instance} \\
\mapsto^*_{\omega^e_0} & \mathcal{A}'''', \biguplus_{j\in\mathcal{I}_m}\langle \bar{\mathcal{S}}''_j, D_j\rangle_{k_j}, \biguplus_{l\in\mathcal{I}_e}\langle D_l\rangle_{k_l}, \left(\begin{array}{c} \langle \bar{\mathcal{S}}''_x, P_x, P'_x, D_x\rangle_X, \\ \langle \bar{\mathcal{S}}'_y, P_y, D_y\rangle_Y \end{array}\right) &
\end{array}$$

We highlight the boxes the portions of the state which is responsible for the subsequent derivation step. By Lemma 9 that states monotonicity of the ω^e_α semantics, we can permutate derivation steps that rewrite over non-overlapping portions of the state. Specifically we want to push all derivation steps between r_3 and r_4a either before or after their subsequent derivation steps. Derivation steps that are responsible for introducing Y's matching obligation for r_4a will be pushed before r_3 while all others will be pushed after This is illustrated by the following:

$$\begin{array}{lll}
\mathcal{P} \rhd & \boxed{\mathcal{A}'', \biguplus_{j\in\mathcal{I}_m}\langle \bar{\mathcal{S}}_j\rangle_{k_j}}, \left(\begin{array}{c} \langle \boxed{\bar{\mathcal{S}}_x, \underline{P_x}}, \underline{P'_x}, \underline{S_x}, r_match(Rs)\rangle_X, \\ \langle \boxed{\bar{\mathcal{S}}_y}\rangle_Y \end{array}\right) & \text{Interleaving derivations resulting to } Y\text{'s matching obligation} - (I1) \\
\mapsto^*_{\omega^e_\alpha} & \mathcal{A}''', \biguplus_{j\in\mathcal{I}_m}\langle \bar{\mathcal{S}}'_j\rangle_{k_j}, \left(\begin{array}{c} \langle \bar{\mathcal{S}}'_x, \boxed{P_x, P'_x, S_x, r_match(Rs)}\rangle_X, \\ \langle \mathcal{S}'_y, P_y, S_y\rangle_Y \end{array}\right) & \text{Apply } r_3 \text{ instance} \\
\mapsto^{r_3}_{\omega^e_0} & \boxed{\mathcal{A}''', \biguplus_{j\in\mathcal{I}_m}\langle \bar{\mathcal{S}}'_j\rangle_{k_j}}, \left(\begin{array}{c} \langle \bar{\mathcal{S}}'_x, P_x\rangle_X, \\ \langle \boxed{\bar{\mathcal{S}}'_y, P_y, S_y, r_commit(Rs)}\rangle_Y \end{array}\right) & \text{Apply } r_4a \text{ instance} \\
\mapsto^{r_4a}_{\omega^e_0} & \boxed{\mathcal{A}''', \biguplus_{j\in\mathcal{I}_m}\langle \bar{\mathcal{S}}'_j, D_j\rangle_{k_j}}, \biguplus_{l\in\mathcal{I}_e}\langle D_l\rangle_{k_l}, \left(\begin{array}{c} \langle \boxed{\bar{\mathcal{S}}'_x}, P_x, P'_x, D_x\rangle_X, \\ \langle \mathcal{S}'_y, P_y, D_y\rangle_Y \end{array}\right) & \text{Other interleaving Derivations} - (I2) \\
\mapsto^*_{\omega^e_0} & \mathcal{A}'''', \biguplus_{j\in\mathcal{I}_m}\langle \bar{\mathcal{S}}''_j, D_j\rangle_{k_j}, \biguplus_{l\in\mathcal{I}_e}\langle D_l\rangle_{k_l}, \left(\begin{array}{c} \langle \bar{\mathcal{S}}''_x, P_x, P'_x, D_x\rangle_X, \\ \langle \bar{\mathcal{S}}'_y, P_y, D_y\rangle_Y \end{array}\right) &
\end{array}$$

Consider derivation steps $(I1)$: We highlight *conservatively* the largest portion of the state which possible contributes to the derivation of Y's matching obligations (i.e, P_y and S_y) that were originally interleaving

between r_3 and r_4a. Note that this includes fragments of location X's store including the P_x matching obligations of encoding rule r_3 (underline in the derivation, P_x, P'_x and S_x). Matching obligations P'_x and S_x are excluded because the original derivations are scheduled after r_3 where P'_x and S_x do not exists. P_x is include but by definition they are persistent and any derivations that depend on any constraint of P_x are guaranteed not to be simplified. Hence we can safely permutate these rules upwards. For derivation steps in (*I2*), since they involve portions of the state that are not overlapping with r_4a, we can safely push them downwards. Hence, we have proven the lemma for the r_4a case.

We now consider r_3 encoding rule instances that are paired with matching r_4b applications. Specifically, they are of the form:

$$
\begin{array}{lll}
\mathcal{P} \rhd & \mathcal{A}'', \left(\begin{array}{c} \langle \bar{\mathcal{S}}_x, \boxed{P_x, P'_x, S_x, r\_match(Rs)} \rangle_X, \\ \langle \mathcal{S}_y \rangle_Y \end{array} \right) & \text{Apply } r\_3 \text{ instance} \\
\mapsto^{r\_3}_{\omega^e_0} & \boxed{\mathcal{A}''}, \left(\begin{array}{c} \langle \boxed{\bar{\mathcal{S}}_x}, P_x \rangle_X, \\ \langle \boxed{\bar{\mathcal{S}}_y}, r\_commit(Rs) \rangle_Y \end{array} \right) & \text{Unspecified interleaving derivations} \\
\mapsto^*_{\omega^e_0} & \mathcal{A}'''', \left(\begin{array}{c} \langle \bar{\mathcal{S}}''_x, P_x \rangle_X, \\ \langle \bar{\mathcal{S}}'_y, \boxed{r\_commit(Rs)} \rangle_Y \end{array} \right) & \text{Apply } r\_4b \text{ instance} \\
\mapsto^*_{\omega^e_0} & \mathcal{A}'''', \left(\begin{array}{c} \langle \bar{\mathcal{S}}''_x, P_x, P'_x, S_x \rangle_X, \\ \langle \bar{\mathcal{S}}'_y \rangle_Y \end{array} \right) &
\end{array}
$$

Since matching obligation of r_4b only consist of the constraint $r\_commit(Rs)$ we can straightforwardly push all interleaving derivations downwards:

$$
\begin{array}{lll}
\mathcal{P} \rhd & \mathcal{A}'', \left(\begin{array}{c} \langle \bar{\mathcal{S}}_x, \boxed{P_x, P'_x, S_x, r\_match(Rs)} \rangle_X, \\ \langle \mathcal{S}_y \rangle_Y \end{array} \right) & \text{Apply } r\_3 \text{ instance} \\
\mapsto^*_{\omega^e_0} & \mathcal{A}'', \left(\begin{array}{c} \langle \bar{\mathcal{S}}_x, P_x \rangle_X, \\ \langle \bar{\mathcal{S}}_y, \boxed{r\_commit(Rs)} \rangle_Y \end{array} \right) & \text{Apply } r\_4b \text{ instance} \\
\mapsto^*_{\omega^e_0} & \boxed{\mathcal{A}''}, \left(\begin{array}{c} \langle \boxed{\bar{\mathcal{S}}_x, P_x}, P'_x, S_x \rangle_X, \\ \langle \boxed{\bar{\mathcal{S}}_y} \rangle_Y \end{array} \right) & \text{Interleaving Derivations} \\
\mapsto^*_{\omega^e_0} & \mathcal{A}'''', \left(\begin{array}{c} \langle \bar{\mathcal{S}}''_x, P_x, P'_x, S_x \rangle_X, \\ \langle \bar{\mathcal{S}}'_y \rangle_Y \end{array} \right) &
\end{array}
$$

Hence we have proven this lemma for all cases. □

Theorem 15 (Soundness of Basic Encoding) *Given a* 1*-neighbor restricted and locally quiescent CHR^e program $\mathcal{P}_1$ and a* 0*-neighbor restricted CHR^e program $\mathcal{P}_0$ such that $\mathcal{P}_1 \rightsquigarrow^{\text{basic}}_{1\text{Nb}} \mathcal{P}_0$, for any reachable states Ω and Ω', if $\mathcal{P}_0 \rhd \Omega \mapsto^*_{\omega^e_0} \Omega'$, then we have either Ω' is not commit free or $\mathcal{P}_1 \rhd \text{DropSyncs}(\lceil\Omega\rceil; \mathcal{P}_1) \mapsto^*_{\omega^e_\alpha} \text{DropSyncs}(\lceil\Omega''\rceil; \mathcal{P}_1)$.*

Proof: Given that we have $\mathcal{P}_0 \rhd \Omega \mapsto^*_{\omega^e_0} \Omega'$, by Theorem 6, we have $\mathcal{P}_0 \rhd \lceil\Omega\rceil \mapsto^*_{\omega^e_\alpha} \lceil\Omega'\rceil$, labeling this as $\mathcal{D}$. By Lemma 14, we can safely permutate $\mathcal{D}$ such that all applications of r_3 are followed immediately by r_4a or r_4b. We shall consider such a permutation of $\mathcal{D}$. We will now prove this theorem by induction on ω^e_α derivation steps. For base case we consider $\mathcal{P}_0 \rhd \lceil\Omega\rceil \mapsto^*_{\omega^e_\alpha} \lceil\Omega'\rceil$ of zero derivation step, hence $\lceil\Omega\rceil = \lceil\Omega'\rceil$ and we naively have $\mathcal{P}_1 \rhd \text{DropSyncs}(\lceil\Omega\rceil; \mathcal{P}_1) \mapsto^*_{\omega^e_\alpha} \text{DropSyncs}(\lceil\Omega''\rceil; \mathcal{P}_1)$. for $\lceil\Omega'\rceil$ being commit free and hence $\lceil\Omega'\rceil = \lceil\Omega''\rceil$. For the inductive case, we assume we have this property for derivations $\mathcal{P}_0 \rhd \lceil\Omega\rceil \mapsto^*_{\omega^e_\alpha} \lceil\Omega'''\rceil$ of m derivation steps. We now consider the successor derivation step $\lceil\Omega''\rceil$ such that $\mathcal{P}_0 \rhd \lceil\Omega'''\rceil \mapsto^r_{\omega^e_\alpha} \lceil\Omega''\rceil$. We consider all possible forms rules r applied:

- r is a non-encoding rule instance. Therefore we have $r \in \mathcal{P}_1$ as well. Furthermore, since synchronizing

constraints are reserved for encoding rules, and hence r does not have rule heads or rule bodies that are synchronizing constraints. From this, we have $\mathcal{P}_1 \rhd \text{DropSyncs}(\lceil\Omega'''\rceil; \mathcal{P}_1) \mapsto^{r}_{\omega^e_\alpha} \text{DropSyncs}(\lceil\Omega''\rceil; \mathcal{P}_1)$

- r is an instance of the encoding rule *r_1*. By definition of *r_1* encoding rules, this derivation simply adds synchronizing constraint $r\_req(Xs)$ to some location x in $\lceil\Omega'''\rceil$. Therefore, $\text{DropSyncs}(\lceil\Omega'''\rceil; \mathcal{P}_1) = \text{DropSyncs}(\lceil\Omega''\rceil; \mathcal{P}_1)$ and we naively have $\mathcal{P}_1 \rhd \text{DropSyncs}(\lceil\Omega'''\rceil; \mathcal{P}_1) \mapsto^{*}_{\omega^e_\alpha} \text{DropSyncs}(\lceil\Omega''\rceil; \mathcal{P}_1)$ of zero derivation step.

- r is an instance of the encoding rule *r_2*. By definition of *r_1* encoding rules, this derivation simply removes synchronizing constraint $r\_req(Xs)$ from some location x add synchronizing constraint $r\_match(Rs)$ to some location y in $\lceil\Omega'''\rceil$. Therefore, $\text{DropSyncs}(\lceil\Omega'''\rceil; \mathcal{P}_1) = \text{DropSyncs}(\lceil\Omega''\rceil; \mathcal{P}_1)$ and we naively have $\mathcal{P}_1 \rhd \text{DropSyncs}(\lceil\Omega'''\rceil; \mathcal{P}_1) \mapsto^{*}_{\omega^e_\alpha} \text{DropSyncs}(\lceil\Omega''\rceil; \mathcal{P}_1)$ of zero derivation step.

- r is an instance of the encoding rule *r_3*. By definition of *r_3* encoding rules, this derivation adds synchronizing constraint $r\_commit(Rs)$, hence making Ω'' non commit free. By Lemma 14, This derivation is followed by a derivation $\mathcal{P}_0 \rhd \lceil\Omega''\rceil \mapsto^{r'}_{\omega^e_\alpha} \lceil\Omega''''\rceil$ such that r' is either *r_4a* or *r_4b* encoding rule instance. Furthermore, these encoding rules are of the same matching instance and of the form:

$$r\_3 : [X]\,P_x \setminus [X]\,P'_x, [X]\,S_x, [X]\,r\_match(Rs) \Longleftrightarrow [Y]\,r\_commit(Rs)$$
$$r\_4a : [Y]\,P_y \setminus [Y]\,S_y, [Y]\,r\_commit(Rs) \Longleftrightarrow \exists\ \bar{z}.[X]\,P'_x, D$$
$$r\_4b : [Y]\,r\_commit(Rs) \Longleftrightarrow [X]\,P'_x, [X]\,S_x$$

  and these encoding rules corresponding to some 1-neighbor restricted rule instance $r_1 \in \mathcal{P}_1$. Let this rule instance be $r_1 : [X]\,P_x, [X]\,P'_x, [Y]\,P_y \setminus [X]\,S_x, [Y]\,S_y \Longleftrightarrow G \mid \exists\ \bar{z}.\ D$. We now consider the case that $r' = r\_4a$: This instance of *r_4a* removes S_y and $r\_commit(Rs)$ from location Y and adds P'_x to location X and D to various locations of the state, while its preceding *r_3* derivation step removes S_x and P'_x from location X. Hence we have that Ω'''' is commit free, and the net effect of these two derivation is the application of the r_1 1-neighbor restricted rule instance. Therefore $\mathcal{P}_1 \rhd \text{DropSyncs}(\Omega'''; \mathcal{P}_1) \mapsto^{r_1}_{\omega^e_\alpha} \text{DropSyncs}(\Omega''''; \mathcal{P}_1)$.

  We now consider the case that $r' = r\_4b$: This instance of *r_4b* removes $r\_commit(Rs)$ from location Y, and adds P'_x and S_x to location X, while its preceding *r_3* derivation step removes P'_x and S_x from location X. The net effect of these two derivation is that we return to the state before *r_3* was applied. Therefore, Therefore, $\text{DropSyncs}(\lceil\Omega'''\rceil; \mathcal{P}_1) = \text{DropSyncs}(\lceil\Omega''''\rceil; \mathcal{P}_1)$ and we naively have $\mathcal{P}_1 \rhd \text{DropSyncs}(\lceil\Omega'''\rceil; \mathcal{P}_1) \mapsto^{*}_{\omega^e_\alpha} \text{DropSyncs}(\lceil\Omega''''\rceil; \mathcal{P}_1)$ of zero derivation step.

Therefore we have proven the soundness of the basic encoding scheme. □

A.4 Proofs for 1-Neighbor Restricted Rule Optimized Encoding Scheme

Lemma 16 (Neighbor Persistent Encoding Rule Serializability) *Given a 1-neighbor restricted and locally quiescent CHR^e program $\mathcal{P}_1$ and a 0-neighbor restricted CHR^e program $\mathcal{P}_0$ such that $\mathcal{P}_1 \rightsquigarrow_{1\text{Nb}} \mathcal{P}_0$ and commit free abstract states $\mathcal{A}_1$ $\mathcal{A}_2$, $\mathcal{A}_3$ and $\mathcal{A}_4$, given that for some matching neighbor persistent encoding rule instances r_2 and r_3, we have $\mathcal{P}_0 \rhd \mathcal{A}_1 \mapsto^{r\_2}_{\omega^e_\alpha} \mathcal{A}_2 \mapsto^{*}_{\omega^e_\alpha} \mathcal{A}_3 \mapsto^{r\_3}_{\omega^e_\alpha} \mathcal{A}_4$ then there exists some $\mathcal{A}'_2$ such that $\mathcal{P}_0 \rhd \mathcal{A}_1 \mapsto^{*}_{\omega^e_\alpha} \mathcal{A}'_2 \mapsto^{r\_2}_{\omega^e_\alpha} \mathcal{A}_3 \mapsto^{r\_3}_{\omega^e_\alpha} \mathcal{A}_4$*

Proof: By definition, encoding rule *r_2* are of the form: $[Y]\,P_y, [Y]\,r\_req(Xs) \Longrightarrow G_y \mid [X]\,r\_match(Rs)$ or $[Y]\,P_y \setminus [Y]\,r\_req(Xs) \Longleftrightarrow G_y \mid [X]\,r\_match(Rs)$ Given that P_y are all persistent constraints, and that $r\_req(Xs$ strictly matches to this encoding rule only, by Lemma 9 and since P_y will never be deleted and $r\_req(Xs)$ is exclusive for the application of *r_2* and furthermore its rule body $r\_match(Rs)$ is only ever required for the application of the matching instance of *r_3* encoding rule, we can safely push a derivation step of *r_2* downwards after any derivation step that comes after it, except for its matching *r_3* encoding

rule. □

Lemma 17 (Primary Persistent Encoding Rule Serializability) *Given a* 1*-neighbor restricted and locally quiescent CHR^e program* $\mathcal{P}_1$ *and a* 0*-neighbor restricted CHR^e program* $\mathcal{P}_0$ *such that* $\mathcal{P}_1 \rightsquigarrow_{1\text{Nb}} \mathcal{P}_0$ *and commit free abstract states* $\mathcal{A}_1$ $\mathcal{A}_2$, $\mathcal{A}_3$ *and* $\mathcal{A}_4$*, given that for some matching primary persistent encoding rule instances r_1 and r_2, we have* $\mathcal{P}_0 \rhd \mathcal{A}_1 \mapsto^{r\_1}_{\omega^e_\alpha} \mathcal{A}_2 \mapsto^*_{\omega^e_\alpha} \mathcal{A}_3 \mapsto^{r\_2}_{\omega^e_\alpha} \mathcal{A}_4$ *then there exists some* $\mathcal{A}'_2$ *such that* $\mathcal{P}_0 \rhd \mathcal{A}_1 \mapsto^*_{\omega^e_\alpha} \mathcal{A}'_2 \mapsto^{r\_1}_{\omega^e_\alpha} \mathcal{A}_3 \mapsto^{r\_2}_{\omega^e_\alpha} \mathcal{A}_4$

Proof: By definition, encoding rule *r_1* are of the form: $\texttt{[}X\texttt{]}\,P_x \Longrightarrow G_x \mid \texttt{[}X\texttt{]}\,r\_req(Xs)$ Given that P_x are all persistent constraints, by Lemma 9 and since P_x will never be deleted and furthermore its rule body $r\_req(Xs)$ is only ever required for the application of the matching instance of *r_2* encoding rule, we can safely push a derivation step of *r_1* downwards after any derivation step that comes after it, except for its matching *r_2* encoding rule. □

Theorem 18 (Soundness of Optimized Encoding) *Given a* 1*-neighbor restricted and locally quiescent CHR^e program* $\mathcal{P}_1$ *and a* 0*-neighbor restricted CHR^e program* $\mathcal{P}_0$ *such that* $\mathcal{P}_1 \rightsquigarrow_{1\text{Nb}} \mathcal{P}_0$*, for any reachable states* Ω *and* Ω'*, if* $\mathcal{P}_0 \rhd \Omega \mapsto^*_{\omega^e_0} \Omega'$*, then we have either* Ω' *is not commit free or* $\mathcal{P}_1 \rhd \text{DropSyncs}(\lceil\Omega\rceil; \mathcal{P}_1) \mapsto^*_{\omega^e_\alpha} \text{DropSyncs}(\lceil\Omega''\rceil; \mathcal{P}_1)$*.*

Proof: The proof of this is similar to the proof of Theorem 15. Specifically, we first consider the ω^e_α derivations of $\mathcal{P}_0$, i.e, $\mathcal{P}_0 \rhd \lceil\Omega\rceil \mapsto^*_{\omega^e_\alpha} \lceil\Omega'\rceil$, Next we use Lemma 14, 16 and 17 to consider only $\mathcal{P}_0 \rhd \lceil\Omega\rceil \mapsto^*_{\omega^e_\alpha} \lceil\Omega'\rceil$ derivations such that

- For basic encoding rules, all applied *r_3* rule instances are immediately followed by a matching application of *r_4a* or *r_4b* (Lemma 14).
- For neighbor persistent encoding rules, all applied *r_2* rule instances are immediately followed by a matching application of *r_3* (Lemma 16).
- For primary persistent encoding rules, all applied *r_1* rule instances are immediately followed by a matching application of *r_2* (Lemma 17).

Given a derivation $\mathcal{P}_0 \rhd \lceil\Omega\rceil \mapsto^*_{\omega^e_\alpha} \lceil\Omega'\rceil$ with the above property, we prove by induction on the derivation steps: For derivation steps corresponding to the application of basic encoding rules and 0-neighbor restricted rules we defer to Theorem 15 for details, here we consider the inductive step derivation $\mathcal{P}_0 \rhd \lceil\Omega''\rceil \mapsto^r_{\omega^e_\alpha} \lceil\Omega'\rceil$ for neighbor and primary persistent encoding rules:

- r is an instance of the neighbor persistent encoding rule *r_2*. By Lemma 16, this derivation is followed by a derivation $\mathcal{P}_0 \rhd \lceil\Omega''\rceil \mapsto^{r\_3}_{\omega^e_0} \lceil\Omega''''\rceil$. These encoding rules are of the form:

$$r\_2 : \texttt{[}Y\texttt{]}\,P_y \setminus r\_req(Xs) \Longleftrightarrow G_y \mid \texttt{[}X\texttt{]}\,r\_match(Rs)$$
$$r\_3 : \texttt{[}X\texttt{]}\,P_x \setminus \texttt{[}X\texttt{]}\,S_x, \texttt{[}X\texttt{]}\,r\_match(Rs) \Longleftrightarrow B$$

  and these encoding rules corresponds to some 1-neighbor restricted rule instance $r_1 \in \mathcal{P}_1$. Let this rule instance be $r_1 : \texttt{[}X\texttt{]}\,P_x, \texttt{[}Y\texttt{]}\,P_y \setminus \texttt{[}X\texttt{]}\,S_x \Longleftrightarrow G_x, G_y \mid B$ and the net effects of the two derivation step is the application of the r_1 rule instance. Therefore $\mathcal{P}_1 \rhd \text{DropSyncs}(\Omega''; \mathcal{P}_1) \mapsto^{r_1}_{\omega^e_0} \text{DropSyncs}(\Omega'; \mathcal{P}_1)$.

- r is an instance of the primary persistent encoding rule *r_1*. By Lemma 17, this derivation is followed by a derivation $\mathcal{P}_0 \rhd \lceil\Omega''\rceil \mapsto^{r\_3}_{\omega^e_0} \lceil\Omega''''\rceil$. These encoding rules are of the form:

$$r\_1 : \texttt{[}X\texttt{]}\,P_x \Longrightarrow G_x \mid \texttt{[}Y\texttt{]}\,r\_req(Xs)$$
$$r\_2 : \texttt{[}Y\texttt{]}\,r\_req(Xs), \texttt{[}Y\texttt{]}\,P_y \setminus \texttt{[}Y\texttt{]}\,S_y \Longleftrightarrow G_y \mid B$$

and these encoding rules corresponds to some 1-neighbor restricted rule instance $r_1 \in \mathcal{P}_1$. Let this rule instance be $r_1 : \texttt{[}X\texttt{]}\,P_x\,\texttt{,}\,\texttt{[}Y\texttt{]}\,P_y \setminus \texttt{[}Y\texttt{]}\,S_y \Longleftrightarrow G_x, G_y \;\texttt{|}\; B$ and the net effects of the two derivation step is the application of the r_1 rule instance. Therefore $\mathcal{P}_1 \rhd \text{DropSyncs}(\Omega''; \mathcal{P}_1) \mapsto^{r_1}_{\omega^e_0} \text{DropSyncs}(\Omega'; \mathcal{P}_1)$.

Hence we have proven the soundness of the optimized encoding scheme. □

A.5 Proofs for n-Neighbor Restricted Rule Encoding Scheme

Lemma 19 ($\rightsquigarrow_{\text{nNb}}$ Preserves Local Quiescence) *Given a locally quiescent n-neighbor restricted program $\mathcal{P}_n$ and a 0-neighbor restricted program $\mathcal{P}_0$ such that $\mathcal{P}_n \rightsquigarrow_{\text{nNb}} \mathcal{P}_0$, then $\mathcal{P}_0$ is also locally quiescent.*

Proof: By the definition form of the encoding rules of $\rightsquigarrow_{\text{nNb}}$, encoding rules *r_1*, *r_2i*, *r_3*, *r_4ai*, *r_4bi* and *r_5bi* are guaranteed to be locally quiescent. This is because their rule bodies only include constraints to be delivered to its partner location, hence the rules are conservatively locally quiescent. Local quiescence of encoding rule *r_5a* and *r_5bx* depends on the local quiescence of the original rule r in $\mathcal{P}_1$. Since we assume that $\mathcal{P}_1$ is locally quiescent, then *r_5a* and *r_5bx* must be locally quiescent. □

Lemma 20 (Prefix Executions of n-Neighbor Encoding) *Given a locally quiescent n-neighbor restricted program $\mathcal{P}_1$ and a 0-neighbor restricted program $\mathcal{P}_0$ such that $\mathcal{P}_n \rightsquigarrow_{\text{nNb}} \mathcal{P}_0$, if we have reachable states Ω, Ω' such that $\mathcal{P}_0 \rhd \Omega \mapsto_{\omega^e_0} \Omega'$, then we have the following:*

1. *If $\mathcal{P}_0 \rhd \Omega \mapsto^{r\_2i(E)}_{\omega^e_0} \Omega'$ for r_2i encoding rule of some consensus destination E, then there exists some reachable states Ω'' and Ω''', such that $\mathcal{P}_0 \rhd \Omega'' \mapsto^{r\_1(E)}_{\omega^e_0} \Omega'''$ and $\mathcal{P}_0 \rhd \Omega''' \mapsto^*_{\omega^e_0} \Omega$*
2. *If $\mathcal{P}_0 \rhd \Omega \mapsto^{r\_3(E)}_{\omega^e_0} \Omega'$ for r_3 encoding rule of some consensus destination E, then there exists some reachable derivation $\mathcal{P}_0 \rhd \Omega'' \mapsto^*_{\omega^e_0} \Omega'''$ which contains n derivation steps $r\_2i(E)$ for each neighbor Y_i of $i \in \mathcal{I}_n$, and $\mathcal{P}_0 \rhd \Omega''' \mapsto^*_{\omega^e_0} \Omega$*
3. *If $\mathcal{P}_0 \rhd \Omega \mapsto^{r\_4ai(E)}_{\omega^e_0} \Omega'$, for r_4ai encoding rule of some consensus destination E, then there exists some reachable states Ω'' and Ω''', such that $\mathcal{P}_0 \rhd \Omega'' \mapsto^{r\_3(E)}_{\omega^e_0} \Omega'''$ and $\mathcal{P}_0 \rhd \Omega''' \mapsto^*_{\omega^e_0} \Omega$*
4. *If $\mathcal{P}_0 \rhd \Omega \mapsto^{r\_4bi(E)}_{\omega^e_0} \Omega'$, for r_4bi encoding rule of some consensus destination E, then there exists some reachable states Ω'' and Ω''', such that $\mathcal{P}_0 \rhd \Omega'' \mapsto^{r\_3(E)}_{\omega^e_0} \Omega'''$ and $\mathcal{P}_0 \rhd \Omega''' \mapsto^*_{\omega^e_0} \Omega$*
5. *If $\mathcal{P}_0 \rhd \Omega \mapsto^{r\_5a(E)}_{\omega^e_0} \Omega'$ for r_5a encoding rule of some consensus destination E, then there exists some reachable derivation $\mathcal{P}_0 \rhd \Omega'' \mapsto^*_{\omega^e_0} \Omega'''$ which contains n derivation steps $r\_2i(E)$ for each neighbor Y_i of $i \in \mathcal{I}_n$, and $\mathcal{P}_0 \rhd \Omega''' \mapsto^*_{\omega^e_0} \Omega$*
6. *If $\mathcal{P}_0 \rhd \Omega \mapsto^{r\_5bx(E)}_{\omega^e_0} \Omega'$ of $\mathcal{P}_0 \rhd \Omega \mapsto^{r\_5bi(E)}_{\omega^e_0} \Omega'$ for r_5bx or r_5bi encoding rule of some consensus destination E, then there exists some reachable states Ω'' and Ω''', such that $\mathcal{P}_0 \rhd \Omega'' \mapsto^{r\_4bj(E)}_{\omega^e_0} \Omega'''$ and $\mathcal{P}_0 \rhd \Omega''' \mapsto^*_{\omega^e_0} \Omega$*

Proof: The proof for this is similar to Lemma 11's. The main observation is that each item of the lemma essentially states a specific 'input' 'output' dependency between the encoding rules. This dependency is defined by the synchronizing constraints that appear in the encoding rules. For instance, encoding rule *r_1* adds a unique synchronizing constraint (r_req_i) for each of n neighbors. Each *r_2i* encoding rule has synchronizing constraint r_req_i in its left-hand side, and given that these synchronizing constraints appear

no where else in the program $\mathcal{P}_0$, we can infer (1). For the rest of the lemma, we can repeat this same argument about the definition of the encoding rules. □

Lemma 21 (n-Neighbor Commit-Free Reachability) *Given a locally quiescent n-neighbor restricted program $\mathcal{P}_n$ and a 0-neighbor restricted program $\mathcal{P}_0$ such that $\mathcal{P}_n \rightsquigarrow_{\text{nNb}} \mathcal{P}_0$ and states Ω reachable by $\mathcal{P}_0$, if Ω is not commit-free, then there exists some commit-free state Ω' such that $\mathcal{P}_0 \rhd \Omega \mapsto^*_{\omega_0^e} \Omega'$.*

Proof: The proof for this is similar to Lemma 13's. For the case of n-neighbor restricted rule encoding, the proof of this lemma can be substantiated by the guarantee that the encoding rules have the following form and property: Encoding rule *r_3* and *r_4ai* for each $i \in \mathcal{I}_n$ are the rules ($n+1$ in total for each n-neighbor restricted encoding) in which any form of commit synchronizing constraint will be added to the state. Hence a state will be non-commit free after the application of any of those rules. If each of the n-neighbors applies *r_4ai* we have $n+1$ matching commit synchronizing constraint of the same consensus destination E. With Lemma 19, we have a progress guarantee that each commit synchronizing constraint will eventually be delivered to a location and processed by the location. Hence, these $n+1$ commit synchronizing constraints can be consumed by a matching application of *5_a*. If not all of the n-neighbors applies *r_4ai*, by definition of the *r_4bi* rule, a matching *r_4bi* rule will be applied instead, producing an abort synchronizing constraint. This instead of the abort synchronizing constraint will initiate the application of the *r_5bx* and *r_5bi* encoding rules, that will remove each of the commit synchronizing constraints of consensus destination E. The derivation dependencies of these encoding rules are guarantee by Lemma 20. Hence we have proven this Lemma. □

Lemma 22 (n-Neighbor Encoding Rule Serializability) *Given an n-neighbor restricted and locally quiescent CHR^e program $\mathcal{P}_n$ and a 0-neighbor restricted CHR^e program $\mathcal{P}_0$ such that $\mathcal{P}_n \rightsquigarrow_{\text{nNb}} \mathcal{P}_0$ and commit free abstract states $\mathcal{A}$,$\mathcal{A}'$ such that $\mathcal{P}_0 \rhd \mathcal{A} \mapsto^*_{\omega_\alpha^e} \mathcal{A}'$ labeled as derivation $\mathcal{I}$, if $\mathcal{I}$ contain derivations of a n-neighbor restricted encoding rule instance r of consensus destination E such that:*

- *We have a successful application of rule instance r of E, then there exists a valid derivation $\mathcal{I}' = \mathcal{P}_0 \rhd \mathcal{A} \mapsto^*_{\omega_\alpha^e} \mathcal{A}_E \mapsto^*_{\omega_\alpha^e} \mathcal{A}'_E \mapsto^*_{\omega_\alpha^e} \mathcal{A}'$ such that $\mathcal{P}_0 \rhd \mathcal{A}_E \mapsto^*_{\omega_\alpha^e} \mathcal{A}'_E$ consists the following non-interleaving derivation sequence:*
  1. *The derivation step r_3(E).*
  2. *All n derivation steps of r_4ai(E) each of Y_i for $i \in \mathcal{I}_n$.*
  3. *The derivation step r_5a(E).*
- *We have an aborted attempt to apply rule instance r of E, then there exists a valid derivation $\mathcal{I}' = \mathcal{P}_0 \rhd \mathcal{A} \mapsto^*_{\omega_\alpha^e} \mathcal{A}_E \mapsto^*_{\omega_\alpha^e} \mathcal{A}'_E \mapsto^*_{\omega_\alpha^e} \mathcal{A}'$ such that $\mathcal{P}_0 \rhd \mathcal{A}_E \mapsto^*_{\omega_\alpha^e} \mathcal{A}'_E$ consists a non-interleaving derivation sequences of derivation step r_3(E) followed by all n derivation steps of r_4ai(E) or r_4bi each of Y_i for $i \in \mathcal{I}_n$, followed by*
  1. *The derivation step r_3(E).*
  2. *All m derivation steps of either r_4ai(E) each of Y_i for $i \in \mathcal{I}_n$ where $m \leq n$.*
  3. *All $n-m$ derivation steps of either r_4bi(E) each of Y_i for $i \in \mathcal{I}_n$.*
  4. *The derivation step r_5b(E).*
  5. *All $n-m$ derivation steps of r_5bi(E) of Y_i for $i \in \mathcal{I}_n$.*

Proof: The proof of this Lemma is similar to Lemma 14's. Firstly, it relies on Lemma 20 for the guarantee that all the necessary encoding rules exists (For instance, if we have a derivation step of the *r_4ai* encoding rule, then *r_3* must have been applied as some preceding derivation step). The next part of the proof is to show that we can safely push all interleaving derivations between the consensus E derivation

steps away (either before or after all E derivation steps). By Lemma 9, all we need to prove that all such interleaving derivations are non-overlapping with the E derivation steps in question.

For the case that E derivation steps model the successful application of a rule instance r, since the encoding rules *r_3* and each *r_4ai* are defined such that their propagated rule heads are persistent (*r_3* propagated heads consist of only P_x while each *r_4ai* consist of only P_i, all of which are persistent), derivations interleaving between any of these encoding rules must be non-overlapping with all E derivation steps. As such we can push all interleaving derivations before all *r_3* and *r_4ai* E derivation steps. For *r_5a*, its rule head contains entirely of only commit synchronizing constraints produced by *r_3* and *r_4ai*, hence all other interleaving derivations can be push after it. Hence we have proven this Lemma for the success case.

For the case that E derivation steps model the abortion of an attempt to apply a rule instance of r, encoding rules *r_3* and *r_4ai* are the same as the previous case, while we now also have *r_4bi* rules. These encoding rules are single headed rules with only one synchronizing constraint r_vote_i as its only rule head. Since this constraint is produced by *r_3*, we can push all other non consensus E derivation steps interleaving between them, before the derivation step of *r_3*. For encoding rules *5_bx* and *5_bi*, all their rule heads are synchronizing constraints produced by *r_4ai* and *r_4bi* rules. Hence we can push all other interleaving derivations downwards after all E derivation steps. Hence we have proven this Lemma for the abort case. □

Theorem 23 (Soundness of n-Neighbor Restricted Encoding) *Given an n-neighbor restricted and locally quiescent CHR^e program $\mathcal{P}_n$ and a 0-neighbor restricted CHR^e program $\mathcal{P}_0$ such that $\mathcal{P}_n \rightsquigarrow_{\text{nNb}} \mathcal{P}_0$, for any reachable states Ω and Ω', if $\mathcal{P}_0 \rhd \Omega \mapsto^*_{\omega^e_0} \Omega'$, then we have either Ω' is not commit free or $\mathcal{P}_n \rhd \text{DropSyncs}(\lceil\Omega\rceil; \mathcal{P}_n) \mapsto^*_{\omega^e_\alpha} \text{DropSyncs}(\lceil\Omega''\rceil; \mathcal{P}_n)$.*

Proof: The prove of this theorem is similar to Theorem 15's. Firstly, by Theorem 6, we are guaranteed that from $\mathcal{P}_0 \rhd \Omega \mapsto^*_{\omega^e_0} \Omega'$, we have $\mathcal{P}_0 \rhd \lceil\Omega\rceil \mapsto^*_{\omega^e_\alpha} \lceil\Omega'\rceil$. By Lemma 22, we can safely permutate any such derivations to an equivalent one, such that all consensus E derivations are compressed together (no concurrent and interleaving non E derivations between each E derivation step). Hence, we can focus our attention only to such orderly derivations. From here, we can prove this Theorem is a manner similar to Theorem 15: by induction on ω^e_α derivation steps to show that each derivation step of $\mathcal{P}_0$ to a commit free state has a corresponding derivation of $\mathcal{P}_1$ with all synchronizing constraints removed. □