On connections between CHR and LCC

Thierry Martinez

Contraintes Project–Team, INRIA Paris–Rocquencourt, France

Abstract. Both CHR and LCC languages are based on the same model of concurrent computation, where agents communicate through a shared constraint store, with a synchronization mechanism based on constraint entailment. The *Constraint Simplification Rules* (CSR) subset of CHR and the flat subset of LCC, where agent nesting is restricted, are very close syntactically and semantically. The first contribution of this paper is to provide translations between CSR and flat-LCC and back. The second contribution is a transformation from the full LCC language to flat-LCC which preserves semantics. This transformation is similar to λ-lifting in functional languages. In conjunction with the equivalence between CHR and CSR with respect to naive operational semantics, these results lead to semantics-preserving translations from full LCC to CHR and conversely. Immediate consequences of this work include new proofs for CHR linear logic and phase semantics, relying on corresponding results for LCC, plus an encoding of the λ-calculus in CHR.

1 Introduction

Constraint Handling Rules (CHR) [1] is a rule-based declarative programming language. Programs are sets of transformation rules on constraint stores. Some constraints are built-ins and can only be accumulated into the store. Other constraints are user-defined and can be added or deleted. Although initial motivations were the definition of constraint solvers and propagators, nowadays applications include typing [2,3], software testing [4], scheduling [5] and so on.

Foundations of the class CC of *Concurrent Constraint* programming languages [6] are very close to CHR: both are based on a model of concurrent computation, where agents communicate through a shared constraint store, with a synchronization mechanism based on constraint entailment. In classical constraint settings, the store evolves monotonically, similarly to the built-in constraint store of CHR. The LCC languages [7,8] introduce linear constraint systems, based on Girard's intuitionistic linear logic (ILL) [9]. A remarkable kind of linear constraints are linear tokens [8], which can be freely added or consumed, comparably to CHR constraints. Linear logic leads to a natural semantics for classical CC languages as well [8]. More recently, a precise declarative semantics for CHR has been described in linear logic [10].

This paper formalizes connections between CHR with naive operational semantics and LCC. Two translations from CHR to LCC and back are proposed, both preserving the semantics. Strong bisimilarity results are formulated. As direct corollary, we obtain a natural encoding of the λ-calculus in CHR. While existence of low-level translations is guaranteed by Turing-completeness *via* a compilation process, there are more fine-grained criteria to compare expressiveness [11]. In particular, translations presented here are natural and (relatively) agnostic with respect to the constraint theory.

Section 2 presents CHR and LCC in full generality and recalls some already published and well-known results. Section 3 focuses on distinguished subsets *Constraint Simplification Rules* (CSR) and flat-LCC, provides translations from flat-LCC to CHR and back. Linear logic semantics [10] and phase semantics [12] of CHR are recovered as corollary. Section 4 introduces the *ask-lifting* transformation from full LCC to flat-LCC.

Related work

The adaptations of functional concepts in LCC languages have been initiated in Rémy Haemmerlé's PhD thesis [13] with the embedding of closures and modules, leading to an encoding of λ-calculus in LCC. This paper pursues the effort of transposing results in functional languages to concurrent constraint systems.

The translation from full LCC to CHR relies on *ask-lifting*. This is a transformation comparable to the λ-lifting [14] for functional languages: the common idea is the materialization of the environment in data structures, *i.e.* values in functional languages or tokens in LCC.

Flattening nested programming structures to CHR programs was suggested in [15] for connecting the Celf system [16] to CHR but, to our knowledge, no formal description of the transformation has been published.

2 Syntax and Semantics of CHR and LCC

We will denote by $\mathcal{V}$ a set of variables, and by Σ a signature for constant, function and predicate symbols. The set of free variables of a formula e is denoted $\mathrm{fv}(e)$, a sequence of variables is denoted by $\boldsymbol{x}$. $e[\boldsymbol{t}/\boldsymbol{x}]$ denotes the formula e in which free occurrences of variables $\boldsymbol{x}$ are substituted by terms $\boldsymbol{t}$ (with the usual renaming of bound variables to avoid variable clashes).

For a set S, $S^\star$ denotes the set of finite sequences of elements of S and $\mathcal{M}(S)$ denotes the set of finite multi-sets of elements of S. More formally, $(S^\star; \cdot; \varepsilon)$ denotes the free monoid and $(\mathcal{M}(S); ,; \varnothing)$ the free commutative monoid over the elements of S. For relations $\mathcal{R}$ and $\mathcal{R}'$, $a\,\mathcal{R} \cdot \mathcal{R}'\,c$ if there exists b such that $a\,\mathcal{R}\,b\,\mathcal{R}'\,c$. For a relation $\rightarrow$, $\xrightarrow{\star}$ is the reflexive and transitive closure of $\rightarrow$.

2.1 Syntax and Semantics of CHR

Let $\mathcal{P}_b$ and $\mathcal{P}_c$ be two disjoint subsets of predicate symbols in Σ. Predicates built from Σ with predicate symbols in $\mathcal{P}_b$ are *atomic built-in constraints*, their set is denoted $\mathcal{B}_0$. *Built-in constraints* are conjunctions of atomic built-in constraints, their set is denoted $\mathcal{B}$. Predicates built from Σ with predicate symbols in $\mathcal{P}_c$ are *atomic* CHR *constraints*, their set is denoted $\mathcal{U}_0$. CHR *constraints* are (finite) multi-sets of atomic CHR constraints, their set is denoted $\mathcal{U}$. A *goal* is a multi-set of built-in constraints and CHR constraints.

Definition 1 (Syntax). *A* CHR program *is a set of* rules*, each rule being denoted* $\langle H \backslash H' \Leftrightarrow G \mid B \rangle$ *where* heads H *and* H' *are* CHR *constraints such that* $\langle H, H' \rangle \neq \varnothing$*, the* guard G *is a built-in constraint, and the* body B *is a goal.*

Example 1. The CHR program below, adapted from [17], describes the *dining philosophers* protocol [18], where N philosophers are sitting around a table and alternate thinking and eating. N forks are dispatched between them. Each philosopher is in competition with her neighbors to take her two adjacent forks and eat.

```
diner(N) ⇔ recphilo(0, N).
recphilo(I, N) ⇔
    J is (I + 1) mod N, philo(I, J), fork(I), nextphilo(I, N).
nextphilo(I, N) ⇔ I < N − 1 | J is I + 1, recphilo(J, N).
philo(I, J) \ fork(I), fork(J) ⇔ eat(I, J).
eat(I, J) ⇔ fork(I), fork(J).
```

Built-in constraints are supposed to include the syntactic equality $=$. There is a *constraint theory* CT over the built-in constraints: CT is supposed to be a non-empty, consistent and decidable first-order theory. For two multi-sets $H = (H_1, \ldots, H_m)$ and $H' = (H'_1, \ldots, H'_n)$, $H \doteq H'$ denotes the formula $H_1 = H'_1 \wedge \cdots \wedge H_n = H'_n$ if $m = n$, and false if $m \neq n$ [19].

A *state* is a tuple denoted $\langle g; b; c \rangle_V$ where g is a goal, b is a built-in constraint, c is a CHR constraint and V is a set of variables. The relation $\equiv_{\mathrm{C}}$ over states is the smallest equivalence relation such that:

– $\langle g; b; c \rangle_V \equiv_{\mathrm{C}} \langle g; b'; c \rangle_V$ for $CT \models b \leftrightarrow b'$;
– $\langle g; b; c \rangle_V \equiv_{\mathrm{C}} \langle g; b; c \rangle_V [y/x]$ for variables $x \notin V$ and $y \notin V \cup \mathrm{fv}(g, b, c)$.

Let $\mathcal{P}$ be the set of pairs of CHR programs and states.

Definition 2 (Naive Operational Semantics [1]). *A* CHR *program* P *is executed along a* transition relation $\rightarrow_P$ *over states:*

Firing Rule

Apply

$$\frac{\langle H \backslash H' \Leftrightarrow G \mid B \rangle \text{ is a fresh variant of a rule in } P \text{ with variables } \boldsymbol{x} \qquad CT \models \forall (b \rightarrow \exists \boldsymbol{x} (H \doteq h \wedge H' \doteq h' \wedge G))}{\langle g; b; h, h', c \rangle_V \rightarrow_P \langle B, g; H \doteq h \wedge H' \doteq h' \wedge G \wedge b; h, c \rangle_V}$$

Solving Rules

Solve

$$\frac{B \in \mathcal{B} \qquad CT \models B \wedge b \leftrightarrow b'}{\langle B, g; b; c \rangle_V \rightarrow_P \langle g; b'; c \rangle_V}$$

Introduce

$$\frac{C \in \mathcal{U}}{\langle C, g; b; c \rangle_V \rightarrow_P \langle g; b; C, c \rangle_V}$$

Let q *be an initial goal, the* query*.* V *is defined as* $\mathrm{fv}(q)$ *and, from the* initial state $s_0 = \langle q; \top; \varnothing \rangle_V$*, a* derivation *is a sequence* $s_0 \rightarrow_P s_1 \rightarrow_P \cdots \rightarrow_P s_n$*. Such a state* s_n *is an* accessible state*.*

Definition 3 (Linear Logic Semantics [10]).

For built-in constraint $B = \langle B_1 \wedge \cdots \wedge B_n \rangle$, *let* $B^\dagger = \langle !B_1 \otimes \cdots \otimes !B_n \rangle$.
For CHR *constraint* $C = (C_1, \ldots, C_n)$, *let* $C^\dagger = \langle C_1 \otimes \cdots \otimes C_n \rangle$.
For goal $G = (G_1, \ldots, G_n)$, *let* $G^\dagger = \langle G_1^\dagger \otimes \cdots \otimes G_n^\dagger \rangle$.
For state $S = \langle g; b; c \rangle_V$, *let* $S^\dagger = \exists \boldsymbol{x}(g^\dagger \otimes b^\dagger \otimes c^\dagger)$, *where* $\boldsymbol{x} = \mathrm{fv}(G, B, C) \setminus V$.

The semantics of a rule $r = \langle H \backslash H' \Leftrightarrow G \mid B \rangle$ *is, with* $\boldsymbol{x} = \mathrm{fv}(B) \setminus \mathrm{fv}(H, H', G)$*:*
$r^\dagger = \langle !\forall (G^\dagger \otimes H^\dagger \otimes H'^\dagger \multimap \exists \boldsymbol{x}(H^\dagger \otimes B^\dagger)) \rangle$.
The linear logic semantics of a program $P = \{r_1, \ldots, r_n\}$ *is* $P^\dagger = \langle r_1^\dagger \otimes \cdots \otimes r_n^\dagger \rangle$.

Theorem 1 (Soundness & Completeness [10]).
Let $CT^\dagger$ *be the Girard translation of* CT *[9],* P *a* CHR *program and* q *a query.*

– *(Sound) If* s *is an accessible state from* q *in* P*, then* $P^\dagger, CT^\dagger \models \forall (q^\dagger \multimap s^\dagger)$.
– *(Complete) For every formula* c *such that* $P^\dagger, CT^\dagger \models \forall (q^\dagger \multimap c)$*, there is an accessible state* s *from* q *in* P *such that* $CT^\dagger \models \forall (s^\dagger \multimap c)$.

2.2 Syntax and Semantics of LCC

Definition 4 (Linear Constraint System [8]). *A* linear constraint system *is a pair* $(\mathcal{C}, \vdash_{\mathcal{C}})$*, where:*

– $\mathcal{C}$ *is a set of formulas (the* linear constraints*) built from variables* $\mathcal{V}$ *and the signature* Σ*, with logical operators: multiplicative conjunction* $\otimes$*, its neutral* 1*, existential* $\exists$*, exponential* $!$ *and constant* $\top$*;* $\mathcal{C}$ *is assumed to be closed by renaming, multiplicative conjunction and existential quantification;*
– $\Vdash_{\mathcal{C}}$ *is a binary relation over* $\mathcal{C}$*, which defines the non-logical axioms.*
– $\vdash_{\mathcal{C}}$ *is the least subset of* $\mathcal{C}^\star \times \mathcal{C}$ *containing* $\Vdash_{\mathcal{C}}$ *and closed by the rules of intuitionistic multiplicative exponential linear logic for* 1, $\top$, $\otimes$, $!$ *and* $\exists$.

Definition 5 (Syntax with Persistent Asks [20]). *The syntax for building* LCC agents *follows the grammar:* $A ::= \forall \mathcal{V}^\star (\mathcal{C} \rightarrow A) \big| \forall \mathcal{V}^\star (\mathcal{C} \Rightarrow A) \big| \exists \mathcal{V}.A \big| \mathcal{C} \big| A \| A$ *where* $\|$ *stands for parallel composition,* $\exists$ *for variable hiding,* $\rightarrow$ *for (transient) ask and* $\Rightarrow$ *for persistent ask. In the particular case where there are no universally quantified variables in an ask, the notation* $(c \rightarrow a)$ *is preferred to* $\forall \varepsilon (c \rightarrow a)$.

Agent $\forall \boldsymbol{x}(c \rightarrow a)$ suspends until c is entailed then wakes up and does a. Transient asks wake up at most one time. Persistent asks are introduced [20] to replace declarations by agents. The agent $\forall \boldsymbol{x}(c \Rightarrow a)$ can wake up as many times as c is entailed. This behavior makes sense as entailment consumes resources.

Example 2. Here is the LCC version for dining philosophers [8,13].

$$
\begin{array}{l}
\forall N (\,\mathrm{diner}(N) \Rightarrow \\
\quad \exists K (\forall I (\,\mathrm{recphilo}(K, I) \Rightarrow \\
\qquad \mathrm{fork}(K, I) \parallel \\
\qquad \exists J. (\,J\ \mathbf{is}\ (I+1) \bmod N \parallel \\
\qquad\quad (\,\mathrm{fork}(K, I) \otimes \mathrm{fork}(K, J) \Rightarrow \\
\qquad\quad\ \ \mathrm{eat}(K, I) \parallel (\mathrm{eat}(K, I) \rightarrow \mathrm{fork}(K, I) \otimes \mathrm{fork}(K, J)) \parallel \\
\qquad\quad (I < N - 1 \rightarrow \mathrm{recphilo}(K, J)) \parallel \\
\qquad \mathrm{recphilo}(\mathrm{K}, 0))))))
\end{array}
$$

This example makes use of non-trivial scopes: variables N, K, I and J are in turn introduced and shared by subsequent asks. The recursive loop $\langle$recphilo$\rangle$ installs N forks and composes N agents (the philosophers) in parallel. The variable K identifies tokens and let several diners to be run in parallel (a *banquet* [13]) while preventing tables from stealing cutlery from each other. The philosopher between forks I and J is an agent in LCC, whereas she is materialized in example 1 by the CHR constraint philo(I, J) in order to carry the environment $\{\mathrm{I}, \mathrm{J}\}$.

A *configuration* is a triple $(X; c; \Gamma)$ where c is a constraint (the *store*), Γ is a multi-set of agents and X is a set of variables (the *hidden variables*). The relation $\equiv_{\mathrm{L}}$ over configurations is the smallest equivalence relation such that:

- $(X; c; a \parallel b, \Gamma) \equiv_{\mathrm{L}} (X; c; a, b, \Gamma)$ for all agents a and b;
- $(X; c; 1, \Gamma) \equiv_{\mathrm{L}} (X; c; \Gamma)$;
- $(X; c; \Gamma) \equiv_{\mathrm{L}} (X; c'; \Gamma)$ for all constraints c, c' such that $c \dashv\vdash_{\mathcal{C}} c'$;
- $(X; c; \Gamma) \equiv_{\mathrm{L}} (X; c; \Gamma)[y/x]$ for all variables $x \in X$ and $y \notin \mathrm{fv}(X, c, \Gamma)$

Let $\mathcal{K}$ be the set of configurations.

Definition 6 (Operational Semantics [8,20]). *The* transition relation $\rightarrow_{\mathrm{L}}$ *is the least relation on configurations satisfying the following rules:*

FIRING RULES

TRANSIENT ASK

$$\frac{c \vdash_{\mathcal{C}} \exists Y (d \otimes e[\boldsymbol{t}/\boldsymbol{x}]) \qquad Y \cap \mathrm{fv}(X, c, \Gamma) = \varnothing \\ \forall d' ((c \vdash_{\mathcal{C}} \exists Y (d' \otimes e[\boldsymbol{t}/\boldsymbol{x}])) \wedge (d' \vdash_{\mathcal{C}} d) \Rightarrow d \dashv\vdash_{\mathcal{C}} d')}{(X; c; \forall \boldsymbol{x}(e \rightarrow a), \Gamma) \rightarrow_{\mathrm{L}} (X \cup Y; d; a[\boldsymbol{t}/\boldsymbol{x}], \Gamma)}$$

PERSISTENT ASK

$$\frac{c \vdash_{\mathcal{C}} \exists Y (d \otimes e[\boldsymbol{t}/\boldsymbol{x}]) \qquad Y \cap \mathrm{fv}(X, c, \Gamma) = \varnothing \\ \forall d' ((c \vdash_{\mathcal{C}} \exists Y (d' \otimes e[\boldsymbol{t}/\boldsymbol{x}])) \wedge (d' \vdash_{\mathcal{C}} d) \Rightarrow d \dashv\vdash_{\mathcal{C}} d')}{(X; c; \forall \boldsymbol{x}(e \Rightarrow a), \Gamma) \rightarrow_{\mathrm{L}} (X \cup Y; d; a[\boldsymbol{t}/\boldsymbol{x}], \forall \boldsymbol{x}(e \Rightarrow a), \Gamma)}$$

SOLVING RULES

HIDING

$$\frac{y \notin X \cup \mathrm{fv}(c, \Gamma)}{(X; c; \exists x.a, \Gamma) \rightarrow_{\mathrm{L}} (X \cup \{y\}; c \otimes d; a[y/x], \Gamma)}$$

TELL

$$\frac{}{(X; c; d, \Gamma) \rightarrow_{\mathrm{L}} (X; c \otimes d; \Gamma)}$$

EQUIVALENCE

$$\frac{\kappa_0 \equiv_{\mathrm{L}} \kappa_0' \rightarrow_{\mathrm{L}} \kappa_1' \equiv_{\mathrm{L}} \kappa_1}{\kappa_0 \rightarrow_{\mathrm{L}} \kappa_1}$$

An agent a is associated with the initial configuration $(\varnothing; \top; a)$. Accessible observables *from a configuration κ are the configurations κ' such that $\kappa \xrightarrow{\star}_{\mathrm{L}} \kappa'$.*

Definition 7 (Linear Logic Semantics [8,20]). *The translation $(\cdot)^{\ddagger}$ of* LCC *agents into their* linear logic semantics *is defined inductively as follows:*

$$(\forall \boldsymbol{x}(c \rightarrow a))^{\ddagger} = \forall \boldsymbol{x}(c \multimap a^{\ddagger}) \qquad (\forall \boldsymbol{x}(c \Rightarrow a))^{\ddagger} = !\forall \boldsymbol{x}(c \multimap a^{\ddagger})$$

$$(\exists x.a)^{\ddagger} = \exists x(a^{\ddagger}) \qquad c^{\ddagger} = c \qquad (a \parallel b)^{\ddagger} = a^{\ddagger} \otimes b^{\ddagger}$$

If Γ is a multi-set of agents $(a_1, \ldots, a_n)$, we define $\Gamma^\ddagger = \langle a_1^\ddagger \otimes \cdots \otimes a_n^\ddagger \rangle$. Configurations are translated to $(X; c; \Gamma)^\ddagger = \langle \exists X (c \otimes \Gamma^\ddagger) \rangle$.

Theorem 2 (Soundness & Completeness [8,20,13]). *For all agents a:*

- *(Sound) If κ is an accessible observable from $(\varnothing; \top; a)$, then $a^\ddagger \vdash_{\mathcal{C}} \kappa^\ddagger$.*
- *(Complete) If c is such that $a^\ddagger \vdash_{\mathcal{C}} c$, then there is an accessible observable $(X; d; \Gamma)$ from $(\varnothing; \top; a)$ with $\exists X(d) \vdash_{\mathcal{C}} c$ and agents in Γ are persistent asks.*

2.3 Circumscribing non-determinism in CHR and LCC operational semantics

Whereas non-determinism in firing rules seems to be inherent to the computation model (and is tackled in CHR by the committed-choice strategy and by the refined semantics), the non-determinism in sequencing solving rules can be completely eliminated. This is a classical result for constraint logic programming [21] and it was proved for LCC in [13]. We formalize such a result for CHR and LCC since the precise bisimulation results presented in next sections rely on it.

Let $\rightarrow_P^s$ and $\rightarrow_P^f$ be the restrictions of $\rightarrow_P$ to solving and firing rules respectively. Let $\rightarrow_{\mathrm{L}}^s$ and $\rightarrow_{\mathrm{L}}^f$ be the similar restrictions for $\rightarrow_{\mathrm{L}}$.

We define $\Rightarrow_P^s$ such that $s \Rightarrow_P^s s'$ if and only if $s \xrightarrow{\star}{}_P^s s' \not\rightarrow_P^s$. Similarly, $\Rightarrow_{\mathrm{L}}^s$ is such that $\kappa \Rightarrow_{\mathrm{L}}^s \kappa'$ if and only if $\kappa \xrightarrow{\star}{}_{\mathrm{L}}^s \kappa' \not\rightarrow_{\mathrm{L}}^s$.

Lemma 1 (Solving rules terminate and are confluent modulo $\equiv$).
For every CHR *program P, for all state s, there exists s' such that $s \Rightarrow_P^s s'$ and for all s', s'', if $s \Rightarrow_P^s s'$ and $s \Rightarrow_P^s s''$, then $s' \equiv_{\mathrm{C}} s''$.*
For every configuration κ, there exists κ' such that $\kappa \Rightarrow_{\mathrm{L}}^s \kappa'$ and for all κ', κ'', if $\kappa \Rightarrow_{\mathrm{L}}^s \kappa'$ and $\kappa \Rightarrow_{\mathrm{L}}^s \kappa''$ then $\kappa' \equiv_{\mathrm{L}} \kappa''$.

Thus, observed configurations can be restricted to be final for $\rightarrow^s$ (or, equivalently, normalized by $\Rightarrow^s$) without losing derivations. The following lemma is a specialization of the "*Andorra*" principle [22] to the rule selection strategy:

Lemma 2 (*Full solving before firing*). *For every* CHR *program P,*

$$\left(\xrightarrow{\star}_P \cdot \Rightarrow_P^s\right) = \left((\Rightarrow_P^s \cdot \rightarrow_P^f)^\star \cdot \Rightarrow_P^s\right)$$

and, similarly,

$$\left(\xrightarrow{\star}_{\mathrm{L}} \cdot \Rightarrow_{\mathrm{L}}^s\right) = \left((\Rightarrow_{\mathrm{L}}^s \cdot \rightarrow_{\mathrm{L}}^f)^\star \cdot \Rightarrow_{\mathrm{L}}^s\right)$$

The lemma 2 is a corollary of the monotonous selection strategy [13]: intuitively, $\rightarrow^s$ can always be exhausted before applying $\rightarrow^f$.

Lemma 3 (Solving rules preserve declarative semantics). *For every* CHR *program P, if $s \rightarrow_P^s s'$, then $s^\dagger \equiv s'^\dagger$. Similarly, if $\kappa \rightarrow_{\mathrm{L}}^s \kappa'$, then $\kappa^\ddagger \equiv \kappa'^\ddagger$.*

Therefore, next sections focus on $\Rightarrow$-transitions where $\Rightarrow_P = (\Rightarrow_P^s \cdot \rightarrow_P^f \cdot \Rightarrow_P^s)$, and $\Rightarrow_{\mathrm{L}} = (\Rightarrow_{\mathrm{L}}^s \cdot \rightarrow_{\mathrm{L}}^f \cdot \Rightarrow_{\mathrm{L}}^s)$: a $\Rightarrow_P$-*accessible state* from s is a state s' such that $s \stackrel{\star}{\Rightarrow}_P s'$ and a $\Rightarrow_{\mathrm{L}}$-*accessible observable* from κ is a configuration κ' such that $\kappa \stackrel{\star}{\Rightarrow}_{\mathrm{L}} \kappa'$. It is worth noticing that a firing occurs at each $\Rightarrow$-transition.

3 Translations between sub-languages CSR and flat-LCC

From now on, we consider the linear constraint system $(\mathcal{C}, \vdash_{\mathcal{C}})$ induced by the constraint theory CT and with atomic CHR constraints as linear tokens. More precisely, $\mathcal{C}$ is the least set of formulas which contains $\top$ and $!B$ for all $B \in \mathcal{B}_0$ and C for all $C \in \mathcal{U}_0$, closed by renaming, multiplicative conjunction and existential quantification. We suppose that $c \Vdash_{\mathcal{C}} d$ if and only if $CT^{\dagger} \models \forall(c \multimap d)$. The result is a particular form of linear constraint system where non-logical axioms follow from the translation of a classical theory.

Bisimulation is the most popular method for comparing concurrent processes [23], characterizing a notion of strong equivalence between processes. A *transition system* is a tuple $(S, \rightarrow)$ with S a set of states and $\rightarrow$ a binary relation over S. We define the CHR transition system as $(\mathcal{P}, \Rightarrow_{\mathrm{C}})$ where $(P, s) \Rightarrow_{\mathrm{C}} (P', s')$ when $P = P'$ and $s \Rightarrow_P s'$, and the LCC transition system as $(\mathcal{K}, \Rightarrow_{\mathrm{L}})$.

Definition 8 (Bisimulation). *Let $(S_1, \xrightarrow{1})$ and $(S_2, \xrightarrow{2})$ be two transition systems. A* bisimulation *is a relation $\sim \subseteq S_1 \times S_2$ such that for all $s_1 \sim s_2$:*

- *for all s_1' such that $s_1 \xrightarrow{1} s_1'$, there exists s_2' such that $s_2 \xrightarrow{2} s_2'$ and $s_1' \sim s_2'$;*
- *for all s_2' such that $s_2 \xrightarrow{2} s_2'$, there exists s_1' such that $s_1 \xrightarrow{1} s_1'$ and $s_1' \sim s_2'$.*

3.1 From Constraint Simplification Rules (CSR) to flat-LCC

Resulting configurations of LCC Firing Rules enjoy a new store where guards have been consumed. This behavior corresponds to simplification rules in CHR.

Definition 9 (CSR **programs[19]).** *A* CHR *program P is a* CSR program *when all rules of P are simplifications (*i.e. *rules are of the form $\langle H \Leftrightarrow G \,|\, B.\rangle$).*

As far as naive operational semantics and linear-logic semantics are concerned, expressiveness of CHR and CSR is identical. For a rule $r = \langle H \backslash H' \Leftrightarrow G \,|\, B\rangle$, let $r^{\times} = \langle H, H' \Leftrightarrow G \,|\, H, B.\rangle$ and for $P = \{r_1, \ldots, r_n\}$, let $P^{\times} = \{r_1^{\times}, \ldots, r_n^{\times}\}$.

Example 3. Here is leq$^{\times}$ translated from a version of the leq program [24]:

```
leq(X, X) ⇔ true.
leq(X, Y) ⇔ number(X), number(Y) | X ⩽Y.
leq(X, Y), leq(Y, X) ⇔ X = Y.
leq(X, Y), leq(Y, Z) ⇔ leq(X, Y), leq(Y, Z), leq(X, Z).
leq(X, Y), leq(X, Y) ⇔ leq(X, Y).
```

Proposition 1 (CHR **and** CSR **equivalence).** *For every* CHR *program P, we have $\rightarrow_P = \rightarrow_{P^{\times}}$ and $P^{\dagger} \equiv (P^{\times})^{\dagger}$.*

This equivalence only holds for naive CHR semantics. There is probably no natural encoding of the traditional semantics for propagation [25] in LCC, at least without *ad-hoc* support hard-wired in the constraint system.

Let $r = \langle H' \Leftrightarrow G | B.\rangle$ be a simplification rule. $G^\dagger \otimes H'^\dagger$ and $B^\dagger$ are in $\mathcal{C}$, thus the following agent is well-formed: $r^{\multimap} = \langle \forall \boldsymbol{y}(G^\dagger \otimes H'^\dagger \Rightarrow \exists \boldsymbol{x}.B^\dagger)\rangle$, where $\boldsymbol{x} = \mathrm{fv}(B) \setminus \mathrm{fv}(H', G)$ and $\boldsymbol{y} = \mathrm{fv}(H', G)$. For every CSR program $P = \{r_1, \ldots, r_n\}$, the *translation of P in* LCC is: $P^{\multimap} = \langle r_1^{\multimap} \parallel \ldots \parallel r_n^{\multimap}\rangle$. States $\langle g; b; c\rangle_V$ are translated in $\mathcal{C}$ as well: $\langle g; b; c\rangle_V{}^{\multimap} = g^\dagger \otimes b^\dagger \otimes c^\dagger$.

Example 4. The $\mathrm{leq}^\times$ program (Example 3) is translated to the agent $\mathrm{leq}^{\multimap}$:

$$
\begin{aligned}
\mathrm{leq}^{\multimap} = {} & \forall X(\mathrm{leq}(X, X) \Rightarrow 1) \parallel \\
& \forall XY(\mathrm{number}(X) \otimes \mathrm{number}(Y) \otimes \mathrm{leq}(X, Y) \Rightarrow X \leqslant Y) \parallel \\
& \forall XY(\mathrm{leq}(X, Y) \otimes \mathrm{leq}(Y, X) \Rightarrow X = Y) \parallel \\
& \forall XYZ(\mathrm{leq}(X, Y) \otimes \mathrm{leq}(Y, Z) \Rightarrow \mathrm{leq}(X, Y) \otimes \mathrm{leq}(Y, Z) \otimes \mathrm{leq}(X, Z)) \parallel \\
& \forall XY(\mathrm{leq}(X, Y) \otimes \mathrm{leq}(X, Y) \Rightarrow \mathrm{leq}(X, Y))
\end{aligned}
$$

Since there is no possible confusion between linear tokens and classical constraints, then, by abuse of notations, we omit the ! operator on $\mathcal{U}_0$ constraints.

Definition 10 (CSR **to** LCC **translation).** *A* CSR *program P and a query q are translated to the agent $a(P, q) = \langle P^{\multimap} \| q^\dagger\rangle$.*

Main Result 1 (Bisimilarity) *Let $\sim \subseteq \mathcal{P} \times \mathcal{K}$ be the relation where $(P, s) \sim \kappa$ if and only if $\kappa \equiv_{\mathrm{L}} (X; s^{\multimap}; P^{\multimap})$ with $X = \mathrm{fv}(s) \setminus V$. Then, $\sim$ is a bisimulation.*

Corollary 1 (Semantics preservation). *For* CSR *program P, query q:*

- *if κ is a $\Rightarrow_{\mathrm{L}}$-accessible observable of $a(P, q)$, then $\kappa \equiv (X; c; P^{\multimap})$ and there is a $\Rightarrow_P$-accessible state s from q with $\exists \boldsymbol{x}(s^{\multimap}) \dashv\vdash_{\mathcal{C}} \exists X(c)$, $\boldsymbol{x} = \mathrm{fv}(s) \setminus \mathrm{fv}(q)$;*
- *if s is a $\Rightarrow_P$-accessible state from q, then there is a $\Rightarrow_{\mathrm{L}}$-accessible observable $(X; c; P^{\multimap})$ from $a(P, q)$ such that $\exists \boldsymbol{x}(s^{\multimap}) \dashv\vdash_{\mathcal{C}} \exists X(c)$, where $\boldsymbol{x} = \mathrm{fv}(s) \setminus \mathrm{fv}(q)$.*

3.2 From flat-LCC to CSR

The translation of CSR into LCC generates agents of the particular form $p \parallel q$, where the sub-agent p is the translation of a CSR program and is therefore a parallel composition of persistent asks without any nested asks, and the sub-agent q is a translation of a query and is therefore reduced to a constraint. Moreover, every ask guard consumes at least a linear token (since CHR heads are non-empty) and asks are closed term (*i.e.* without free variables). Such agents are characterized by the following definition:

Definition 11 (flat-LCC**).** Flat-LCC agents *are restricted to the grammar: $A^\uparrow ::= A^\forall \parallel \mathcal{C}$ where $A^\forall ::= \forall \mathcal{V}^\star(\mathcal{C} \Rightarrow \mathcal{C}) \mid A^\forall \parallel A^\forall \mid 1$ with the following side condition for every ask $\forall \boldsymbol{x}(g \Rightarrow c)$: $g \nvdash_{\mathcal{C}} g \otimes g$ (consumption) and $\mathrm{fv}(g, c) \subseteq \boldsymbol{x}$.*

This subsection is dedicated to establishing the reverse translation, from $A^\uparrow$ to CSR. It is worth noticing first that, like a CSR program, an $A^\uparrow$-agent essentially transforms constraint stores without introducing new suspensions:

Lemma 4 (Configurations form). *Non-initial $\Rightarrow_{\mathrm{L}}$-accessible configurations from an $A^\uparrow$-agent a are $\equiv_{\mathrm{L}}$-equivalent to configurations of the form $(\_; \_; a^\forall)$.*

The translation from flat-LCC to CSR is a bit more intricate than the other direction on account of, firstly, splitting between built-in constraints and CHR constraints, and secondly, possible introductions of local variables by $\exists$. Fresh variables should be introduced to translate constraints such as $a(X,Y) \otimes \exists X(b(X,Y))$ into $\langle a(X,Y), b(K,Y)\rangle$ where K is a new local variable. The function $f^{\mathcal{C}}$ translates every constraint in $\mathcal{C}$ to a tuple $(X;B;C)$ where B is a built-in constraint, C a CHR constraint and X a set of variables local to B and C:

$$
\begin{aligned}
f^{\mathcal{C}}(\top) &= (\varnothing; \mathrm{true}; \varnothing) \\
f^{\mathcal{C}}(!B) &= (\varnothing; B; \varnothing) \qquad\qquad \text{for all } B \in \mathcal{B}_0 \\
f^{\mathcal{C}}(C) &= (\varnothing; \mathrm{true}; C) \qquad\qquad \text{for all } C \in \mathcal{U}_0 \\
f^{\mathcal{C}}(c \otimes d) &= (\sigma_c(X_c) \cup \sigma_d(X_d); \sigma_c(B_c) \wedge \sigma_d(B_d); \sigma_c(C_c), \sigma_d(C_d)) \\
&\quad \text{if } f^{\mathcal{C}}(c) = (X_c; B_c; C_c) \text{ and } f^{\mathcal{C}}(d) = (X_d; B_d; C_d) \\
&\quad \text{with } \sigma_c \text{ and } \sigma_d \text{ renaming of } X_c \text{ and } X_d \text{ respectively} \\
&\quad \text{such that } \sigma_c(X_c) \cap \mathrm{fv}(\sigma_d(B_d, C_d)) = \sigma_d(X_d) \cap \mathrm{fv}(\sigma_c(B_c, C_c)) = \varnothing \\
f^{\mathcal{C}}(\exists x(c)) &= (X_c \cup \{x\}; B_c; C_c) \ \text{ if } f^{\mathcal{C}}(c) = (X_c; B_c; C_c)
\end{aligned}
$$

$A^{\forall}$-agents are translated to CSR programs through the function $f^{\forall}$. Translation of asks should take care of clashes with similar renaming as for $\otimes$ in $f^{\mathcal{C}}$:

$$
\begin{aligned}
f^{\forall}(\forall \boldsymbol{x}(g \Rightarrow c)) &= \{\langle \sigma_g(C_g) \Leftrightarrow \sigma_g(B_g) \mid \sigma_c(B_c), \sigma_c(C_c).\rangle\} \\
&\quad \text{where } f^{\mathcal{C}}(g) = (X_g; B_g; C_g) \text{ and } f^{\mathcal{C}}(c) = (X_c; B_c; C_c) \\
&\quad \text{and } \sigma_g \text{ and } \sigma_c \text{ renaming of } X_g \text{ and } X_c \text{ respectively} \\
&\quad \text{with } \sigma_g(X_g) \cap \mathrm{fv}(\sigma_c(B_c, C_c)) = \sigma_c(X_c) \cap \mathrm{fv}(\sigma_g(B_g, C_g)) = \varnothing \\
f^{\forall}(a \parallel b) &= f^{\forall}(a) \cup f^{\forall}(b) \\
f^{\forall}(1) &= \varnothing
\end{aligned}
$$

For every ask $\forall \boldsymbol{x}(g \Rightarrow c)$, $f^{\forall}(\forall \boldsymbol{x}(g \Rightarrow c))$ is a well-formed CHR rule. In particular, the side condition on g ensures that $\sigma_g(C_g) \neq \varnothing$.

$f^s_V : c \mapsto \langle \varnothing; b; c\rangle_V$ maps constraints to states with $(\_; b; c) = f^{\mathcal{C}}(c)$.

Note that all variables in CSR queries are global. The CHR program initialization should hide existentially quantified variables in the top-level constraint c_0 of the agent. We suppose a fresh symbol $\langle \mathrm{start}/n\rangle \in \mathcal{U}_0$ where $n = \#\mathrm{fv}(c_0)$.

Definition 12 (Flat-LCC to CSR translation). *A flat-*LCC *agent* $\langle a^{\forall} \parallel c_0\rangle$ *is translated to the* CHR *program* $P(a^{\forall} \parallel c_0) = f^{\forall}(a^{\forall}) \cup \{\mathrm{start}(\boldsymbol{v}) \Leftrightarrow B_0, C_0.\}$ *and the query* $q(a) = (\mathrm{start}(\boldsymbol{v}))$ *where* $(\_; B_0; C_0) = f^{\mathcal{C}}(c_0)$ *and* $\boldsymbol{v} = \mathrm{fv}(c_0)$.

Main Result 2 (Bisimilarity) *Let* $\sim \subseteq \mathcal{K} \times \mathcal{P}$ *be the relation where* $\kappa \sim (P, s)$ *if and only if there exists a flat-*LCC *agent* $\langle a^{\forall} \parallel c_0\rangle$ *where* $\kappa \equiv_{\mathrm{L}} (X; c; a^{\forall})$ *and* $P = P(a)$ *and* $s \equiv_{\mathrm{C}} f^s_V(c)$*, with* $V = \mathrm{fv}(c_0)$*. Then,* $\sim$ *is a bisimulation.*

Corollary 2 (Semantics preservation). *For every flat-*LCC *agent* $a = \langle a^{\forall} \parallel c_0\rangle$*, let* $s_0 = \langle q(a); \top; \varnothing\rangle_V$*,* $V = \mathrm{fv}(c_0)$*, then:*

- *for all* $\Rightarrow_{\mathrm{L}}$*-accessible configuration* $(X; c; a^{\forall})$ *from* a*, there exists a* $\Rightarrow_{P(a)}$*-accessible state* s *from* s_0 *such that* $\exists \boldsymbol{x}(s^{-\circ}) \dashv\vdash_{\mathcal{C}} \exists X(c)$*;*
- *for all* $\Rightarrow_{P(a)}$*-accessible state* s *from* s_0*, if* $s \neq s0$*, there exists a* $\Rightarrow_{\mathrm{L}}$*-accessible configuration* $(X; c; a^{\forall})$ *from* a*, such that* $\exists \boldsymbol{x}(s^{-\circ}) \dashv\vdash_{\mathcal{C}} \exists X(c)$*;*

where, in both cases, $\boldsymbol{x} = \mathrm{fv}(s) \setminus V$*.*

3.3 CHR linear-logic and phase semantics revisited

Lemma 5 (Identical Semantics). *For every* CSR *program* P *and query* q, $(P^{\multimap})^{\ddagger} \equiv P^{\dagger}$ *and we have* $P^{\dagger}, CT^{\dagger} \models \forall(q^{\dagger} \multimap c)$ *if and only if* $(P^{\multimap})^{\ddagger}, q^{\dagger} \vdash_{\mathcal{C}} c$.

Relating theorem 1 and theorem 2 supposes to prove that accessible constraints are included in provable constraints (correctness), and conversely (completeness). Thus, the correctness and completeness result amounts to equality between sets, which we make explicit here to prove both ways at the same time.

$$\mathcal{O}_{\mathrm{C}}(P,q) = \left\{(P,s) \in \mathcal{P} \,\middle|\, (q;\top;\varnothing) \overset{\star}{\rightarrow}_P s\right\} \qquad \mathcal{O}_{\mathrm{L}}(a) = \left\{\kappa \in \mathcal{K} \,\middle|\, (\varnothing;\top;a) \overset{\star}{\rightarrow}_{\mathrm{L}} \kappa\right\}$$

$$\mathcal{O}_{\mathrm{C}}^{\Rightarrow}(P,q) = \left\{(P,s) \in \mathcal{P} \,\middle|\, (q;\top;\varnothing) \overset{\star}{\Rightarrow}_P s\right\} \qquad \mathcal{O}_{\mathrm{L}}^{\Rightarrow}(a) = \left\{\kappa \in \mathcal{K} \,\middle|\, (\varnothing;\top;a) \overset{\star}{\Rightarrow}_{\mathrm{L}} \kappa\right\}$$

$$\mathcal{LL}_{\mathrm{C}}(P,q) = \left\{c \in \mathcal{C} \,\middle|\, P^{\dagger}, CT^{\dagger} \models \forall(q^{\dagger} \multimap c)\right\} \qquad \Downarrow_{\mathrm{C}}^{s} S = \left\{s \in \mathcal{P} \,\middle|\, \exists s' \in S \Rightarrow_{\mathrm{C}}^{s} s\right\}$$

$$\mathcal{LL}_{\mathrm{L}}(a) = \left\{c \in \mathcal{C} \,\middle|\, a^{\ddagger} \vdash_{\mathcal{C}} c\right\} \qquad \Downarrow_{\mathrm{L}}^{s} S = \left\{\kappa \in \mathcal{K} \,\middle|\, \exists \kappa' \in S \Rightarrow_{\mathrm{L}}^{s} \kappa\right\}$$

$$\text{⫰}\, S = \left\{c \in \mathcal{C} \,\middle|\, \exists c' \in S, c' \vdash_{\mathcal{C}} c\right\}$$

Some results mentioned up to now are summarized in the following table:

For every CSR program P, query q, and flat-LCC agent a,
- Theorem 1: $\text{⫰}(\mathcal{O}_{\mathrm{C}}(P,q))^{\dagger} = \mathcal{LL}_{\mathrm{C}}(P,q)$;
- Theorem 2: $\text{⫰}(\mathcal{O}_{\mathrm{L}}(a))^{\ddagger} = \mathcal{LL}_{\mathrm{L}}(a)$;
- Lemma 2: $\Downarrow_{\mathrm{C}}^{s}\mathcal{O}_{\mathrm{C}}(P,q) = \mathcal{O}_{\mathrm{C}}^{\Rightarrow}(P,q)$ and $\Downarrow_{\mathrm{L}}^{s}\mathcal{O}_{\mathrm{L}}(a) = \mathcal{O}_{\mathrm{L}}^{\Rightarrow}(a)$;
- Lemma 3: $(\Downarrow_{\mathrm{C}}^{s}\mathcal{O}_{\mathrm{C}}(P,q))^{\dagger} = (\mathcal{O}_{\mathrm{C}}(P,q))^{\dagger}$ and $(\Downarrow_{\mathrm{L}}^{s}\mathcal{O}_{\mathrm{L}}(a))^{\ddagger} = (\mathcal{O}_{\mathrm{L}}(a))^{\ddagger}$;
- Proposition 1: $\mathcal{O}_{\mathrm{C}}(P,q) = \mathcal{O}_{\mathrm{C}}(P^{\times},q)$ and $\mathcal{LL}_{\mathrm{C}}(P,q) = \mathcal{LL}_{\mathrm{C}}(P^{\times},q)$;
- Corollary 1: $(\mathcal{O}_{\mathrm{C}}^{\Rightarrow}(P^{\times},q))^{\dagger} = (\mathcal{O}_{\mathrm{L}}^{\Rightarrow}(a(P^{\times},q)))^{\ddagger}$;
- Lemma 5: $\mathcal{LL}_{\mathrm{C}}(P^{\times},q) = \mathcal{LL}_{\mathrm{L}}(a(P^{\times},q))$

We are now ready to prove theorem 1 again from the other results.

Proof of theorem 1. For every CHR program P and query q:

$$\begin{aligned}
\text{⫰}(\mathcal{O}_{\mathrm{C}}(P,q))^{\dagger} &\overset{\text{proposition 1}}{=} \text{⫰}(\mathcal{O}_{\mathrm{C}}(P^{\times},q))^{\dagger} &&\overset{\text{lemma 3}}{=} \text{⫰}(\Downarrow_{\mathrm{C}}^{s}\mathcal{O}_{\mathrm{C}}(P^{\times},q))^{\dagger}\\
&\overset{\text{lemma 2}}{=} \text{⫰}(\mathcal{O}_{\mathrm{C}}^{\Rightarrow}(P^{\times},q))^{\dagger} &&\overset{\textbf{corollary 1}}{=} \text{⫰}(\mathcal{O}_{\mathrm{L}}^{\Rightarrow}(a(P^{\times},q)))^{\ddagger}\\
&\overset{\text{lemma 2}}{=} \text{⫰}(\Downarrow_{\mathrm{L}}^{s}\mathcal{O}_{\mathrm{L}}(a(P^{\times},q)))^{\ddagger} &&\overset{\text{lemma 3}}{=} \text{⫰}(\mathcal{O}_{\mathrm{L}}(a(P^{\times},q)))^{\ddagger}\\
&\overset{\textbf{theorem 2}}{=} \mathcal{LL}_{\mathrm{L}}(a(P^{\times},q)) &&\overset{\text{lemma 5}}{=} \mathcal{LL}_{\mathrm{C}}(P^{\times},q)\\
&\overset{\text{proposition 1}}{=} \mathcal{LL}_{\mathrm{C}}(P,q)
\end{aligned}$$
□

The following proposition describes a method to prove unreachability property in CHR using phase semantics, adapted from similar result in LCC [8].

Proposition 2 (Safety through Phase Semantics [12]). *To prove a safety property of the kind* $s \not\rightarrow_P s'$ *for a given* CHR *program* P, *it is enough to prove that for a well-chosen phase space* $\mathbf{P}$ *and valuation* η *compatible with* CT *and* P, *there exists an element* $a \in \eta(s^{\dagger})$ *such that* $a \notin \eta(t^{\dagger})$.

Such a valuation η is compatible with $\vdash_{\mathcal{C}}$ and $(P^{\times})^{\multimap}$ (note that it is immediate to see $(P^{\times})^{\multimap}$ as declarations in the sense of the original presentation of LCC [8]). Let κ and κ' be the respective images of s and s' by the transformation. Then the element a is such that $a \in \eta(\kappa^{\ddagger})$ and $a \notin \eta(\kappa'^{\ddagger})$. Thus $\kappa \not\rightarrow_{\mathrm{L}} \kappa'$ comes from the phase semantics of LCC. Therefore the property $s \not\rightarrow_{P^{\times}} s'$ follows from corollary 1, and is generalizable to P with proposition 1. That proves proposition 2. □

4 Ask-lifting: encoding LCC into CSR

The main result of this section is a translation from LCC to flat-LCC which preserves the semantics. Consequently, thanks to corollary 2, we can deduce a semantics-preserving translation from LCC to CSR. This section begins with a preliminary step introducing an intermediary language LCC$^\ell$ where asks are labeled with linear tokens: these tokens do not change the operational semantics and there is a trivial labeling to transform LCC programs to LCC$^\ell$ programs. These linear tokens are introduced in order to follow asks through the transitions of the operational semantics, which is used to prove the semantics preservation.

4.1 Preliminary step: labeling LCC-agents

Labeled LCC agents A^ℓ differ from agents A by labels inserted on each ask. In the following definition, labels are arbitrary linear tokens.

Definition 13 (LCC$^\ell$ **agents).** *The syntax of* LCC$^\ell$ agents *is given by the following grammar:* $A^\ell ::= \forall \mathcal{V}^\star(\mathcal{C} \xrightarrow{\mathcal{U}_0} A^\ell) \mid \forall \mathcal{V}^\star(\mathcal{C} \overset{\mathcal{U}_0}{\Rightarrow} A^\ell) \mid \exists \mathcal{V}.A^\ell \mid \mathcal{C} \mid A^\ell \parallel A^\ell$.

The transition relation $\rightarrow_{\mathrm{L}}$ is lifted to the transition $\rightarrow_{\mathrm{LCC}^\ell}$ for LCC$^\ell$.

Transient Ask (with labeling)

$$\frac{c \vdash_{\mathcal{C}} \exists Y(d \otimes e[\boldsymbol{t}/\boldsymbol{x}]) \qquad Y \cap \mathrm{fv}(X, c, \Gamma) = \varnothing \quad \forall d'((c \vdash_{\mathcal{C}} \exists Y(d' \otimes e[\boldsymbol{t}/\boldsymbol{x}])) \wedge (d' \vdash_{\mathcal{C}} d) \Rightarrow d \dashv\vdash d')}{\left(X; c; \forall \boldsymbol{x}(e \xrightarrow{l} a), \Gamma\right) \rightarrow_{\mathrm{LCC}^\ell} (X \cup Y; d; a[\boldsymbol{t}/\boldsymbol{x}], \Gamma)}$$

Persistent Ask (with labeling)

$$\frac{c \vdash_{\mathcal{C}} \exists Y(d \otimes e[\boldsymbol{t}/\boldsymbol{x}]) \qquad Y \cap \mathrm{fv}(X, c, \Gamma) = \varnothing \quad \forall d'((c \vdash_{\mathcal{C}} \exists Y(d' \otimes e[\boldsymbol{t}/\boldsymbol{x}])) \wedge (d' \vdash_{\mathcal{C}} d) \Rightarrow d \dashv\vdash d')}{\left(X; c; \forall \boldsymbol{x}(e \overset{l}{\Rightarrow} a), \Gamma\right) \rightarrow_{\mathrm{LCC}^\ell} \left(X \cup Y; d; a[\boldsymbol{t}/\boldsymbol{x}], \forall \boldsymbol{x}(e \overset{l}{\Rightarrow} a), \Gamma\right)}$$

Agents A are translated to a particular family of labeled agents denoted A^{ℓ_0} with the labeling transformation, which ensures the following conditions: each label carries a distinct symbol taken from a set $\mathcal{P}$ of fresh predicate symbols and the arguments enumerate exactly the free variables of the ask. Such a labeling is simple to obtain as soon as $\mathcal{P}$ is large enough to label each ask of a.

Example 5. The dining philosophers (example 2) can be labeled as follows:

$$\begin{array}{l}
\forall N(\,\mathrm{diner}(N) \overset{p_1}{\Rightarrow} \\
\quad \exists K(\forall I(\,\mathrm{recphilo}(K, I) \xRightarrow{p_2(K,N)} \\
\qquad \mathrm{fork}(K, I) \parallel \\
\qquad \exists J.(\,J\ \mathbf{is}\ (I+1) \bmod N \parallel \\
\qquad\quad (\,\mathrm{fork}(K, I) \otimes \mathrm{fork}(K, J) \xRightarrow{p_3(I,J,K)} \\
\qquad\quad\ \ \mathrm{eat}(K, I) \parallel (\mathrm{eat}(K, I) \xrightarrow{p_4(I,J,K)} \mathrm{fork}(K, I) \otimes \mathrm{fork}(K, J)) \parallel \\
\qquad\quad (I < N - 1 \xrightarrow{p_5(I,J,K,N)} \mathrm{recphilo}(K, J)) \parallel \\
\qquad \mathrm{recphilo}(\mathrm{K}, 0))))))
\end{array}$$

4.2 The ask-lifting transformation

The ask-lifting transformation is defined with two helper functions. $\langle a\rangle^{\mathcal{C}}$ transforms the agent a to constraints where asks become linear tokens. $\langle a\rangle^{\forall}$ puts in parallel every ask occurring in a and the representing token is added to the guard. A persistent ask restores the token, a transient ask consumes it.

The function $\langle\cdot\rangle^{\mathcal{C}} : A^{\ell} \to \mathcal{C}$ is defined inductively as follows:

$$\left\langle \forall \boldsymbol{x}(c \xrightarrow{f(\boldsymbol{t})} a)\right\rangle^{\mathcal{C}} = f(\boldsymbol{t}) \qquad \left\langle \forall \boldsymbol{x}(c \xRightarrow{f(\boldsymbol{t})} a)\right\rangle^{\mathcal{C}} = f(\boldsymbol{t}) \qquad \left\langle \exists x.a\right\rangle^{\mathcal{C}} = \exists x.\left\langle a\right\rangle^{\mathcal{C}}$$

$$\left\langle a \parallel b\right\rangle^{\mathcal{C}} = \left\langle a\right\rangle^{\mathcal{C}} \otimes \left\langle b\right\rangle^{\mathcal{C}} \qquad \left\langle c\right\rangle^{\mathcal{C}} = c$$

The function $\langle\cdot\rangle^{\forall} : A^{\ell_0} \to A^{\forall}$ is defined inductively as follows:

$$\left\langle \forall \boldsymbol{x}(c \xrightarrow{f(\boldsymbol{v})} a)\right\rangle^{\forall} = \forall \boldsymbol{v}\boldsymbol{x}(f(\boldsymbol{v}) \otimes c \xRightarrow{f(\boldsymbol{v})} \left\langle a\right\rangle^{\mathcal{C}}) \parallel \left\langle a\right\rangle^{\forall}$$

$$\left\langle \forall \boldsymbol{x}(c \xRightarrow{f(\boldsymbol{v})} a)\right\rangle^{\forall} = \forall \boldsymbol{v}\boldsymbol{x}(f(\boldsymbol{v}) \otimes c \xRightarrow{f(\boldsymbol{v})} f(\boldsymbol{v}) \otimes \left\langle a\right\rangle^{\mathcal{C}}) \parallel \left\langle a\right\rangle^{\forall}$$

$$\left\langle \exists x.a\right\rangle^{\forall} = \left\langle a\right\rangle^{\forall} \qquad \left\langle a \parallel b\right\rangle^{\forall} = \left\langle a\right\rangle^{\forall} \parallel \left\langle b\right\rangle^{\forall} \qquad \left\langle c\right\rangle^{\forall} = 1$$

The function $\langle\cdot\rangle^{\forall}$ is well-defined: every ask satisfies the side-condition for $A^{\forall}$.

Definition 14 (Ask-lifting). *The* agent ask-lifting function $[\![\cdot]\!]^{\Lsh} : A \to A^{\uparrow}$ *transforms the agent a to the agent $[\![a]\!]^{\Lsh} = \langle a^{\ell}\rangle^{\forall} \parallel \langle a^{\ell}\rangle^{\mathcal{C}}$ where a is translated to a^{ℓ} by the labeling defined in 4.1 with symbol predicates from a subset $\mathcal{P}$ of $\mathcal{P}_c$ whose predicates do not appear in a. $[\![\cdot]\!]^{\Lsh}$ is well-defined as soon as the set $\mathcal{P}$ is large enough to label agent a.*

Main Result 3 (Bisimilarity) *Let a be a labeled* LCC *agent. Let $\sim \subseteq \mathcal{K} \times \mathcal{K}$ be the relation such that $\kappa \sim \kappa'$ if and only if $\kappa \equiv_{\mathrm{L}} (X; c; \Gamma)$ is $\Rightarrow$-accessible from a and $\kappa' \equiv_{\mathrm{L}} (X; c \otimes \langle\Gamma\rangle^{\mathcal{C}}; \langle a\rangle^{\forall})$. Then, $\sim$ is a bisimilarity.*

Corollary 3 (Semantics preservation). *For every* LCC *agent a:*

- *for all $\Rightarrow_{\mathrm{L}}$-accessible configuration $(X; c; \Gamma)$ from a, there is a $\Rightarrow_{\mathrm{L}}$-accessible configuration $(X; c'; \langle a\rangle^{\forall})$ from $[\![a]\!]^{\Lsh}$ such that $\exists X(c \otimes \langle\Gamma\rangle^{\mathcal{C}}) \dashv\vdash_{\mathcal{C}} \exists X'(c')$;*
- *for all $\Rightarrow_{\mathrm{L}}$-accessible configuration $(X; c'; \langle a\rangle^{\forall})$ from $[\![a]\!]^{\Lsh}$, there exists a $\Rightarrow_{\mathrm{L}}$-accessible configuration $(X; c; \Gamma)$ from a and $\exists X(c \otimes \langle\Gamma\rangle^{\mathcal{C}}) \dashv\vdash_{\mathcal{C}} \exists X'(c')$.*

Example 6. The labeled diner (example 5) can be lifted as follows:

$$\begin{array}{ll}
\forall N(& p_1 \otimes \mathrm{diner}(N) \Rightarrow p_1 \otimes p_2(K,N) \otimes \mathrm{recphilo}(K,0)) \parallel \\
\forall IKN(& p_2(K,N) \otimes \mathrm{recphilo}(K,I) \Rightarrow \\
 & p_2(K,N) \otimes \mathrm{fork}(K,I) \otimes \\
 & \exists J(J \ \mathbf{is}\ (I+1) \bmod N \otimes p_3(I,J,K) \otimes p_5(I,J,K,N))) \parallel \\
\forall IJK(& p_3(I,J,K) \otimes \mathrm{fork}(K,I) \otimes \mathrm{fork}(K,J) \Rightarrow \\
 & p_3(I,J,K) \otimes \mathrm{eat}(K,I) \otimes p_4(I,J,K)) \parallel \\
\forall IJK(& p_4(I,J,K) \otimes \mathrm{eat}(K,I) \Rightarrow \mathrm{fork}(K,I) \otimes \mathrm{fork}(K,J)) \parallel \\
\forall IJKN(& p_5(I,J,K,N) \otimes I < N-1 \Rightarrow \mathrm{recphilo(K,J)}) \parallel p_1
\end{array}$$

4.3 Encoding the λ-calculus in CHR

The following transformation from pure λ-terms to LCC is proved correct [13]. Every function, *aka* λ-value, is represented by a variable K. The constraint $\mathrm{apply}(K, X, V)$ represents that V should code the result of the application of the function (coded by) K to the λ-term (coded by) X. Therefore, the transformation of a λ-abstraction $\lambda x.e$ coded by K should be a persistent ask which transforms, for all X and V, the constraint $\mathrm{apply}(K, X, V)$ to the equality constraint between V and the evaluation of $e[t/x]$, where t is the λ-term coded by X. The equality constraint is put at the level of λ-variables. The constraint $\mathrm{value}(K)$ indicates that the λ-term K has been reduced to a value so as to encode the particular call-by-value strategy [26].

Definition 15 (Call-by-value λ-calculus in LCC [13]). *For every λ-term e, $[\![e]\!]$ is a function from variables to* LCC *agents. $[\![e]\!]$ is described inductively on the structure of e:*

- $[\![X]\!](K) = \langle X = K \otimes \mathrm{value}(K)\rangle$
- $[\![\lambda X.e]\!](K) = \forall XV(\mathrm{apply}(K, X, V) \otimes \mathrm{value}(X) \Rightarrow [\![e]\!](V) \parallel \mathrm{value}(X))$
- $[\![f\ e]\!](K) = \exists XY(\mathrm{apply}(X, Y, K) \parallel [\![f]\!](X) \parallel [\![e]\!](Y))$

Each ask introduced by this transformation corresponds to a λ-abstraction and this property is preserved by ask-lifting. Therefore, the CSR program obtained by translation has one rule for each λ-abstraction.

We explicit below the direct transformation from λ-terms to CSR. We suppose that the labeling has been prepared directly in λ-terms: λ-abstractions are of the form $\lambda_i X.e$ where i is a unique index.

Definition 16 (Call-by-value λ-calculus in CHR). *For every λ-term e, $[e]$ is a function from variables to pairs* CHR *programs and queries, each component being denoted $[e]^p$ and $[e]^g$. $[e]$ is described inductively on the structure of e as follows.*

– $[X](K) \quad = (\ \varnothing \qquad\qquad ; (X = K, \mathit{value}(K)))$

– $[\lambda X_i.e](K) = (\ [e]^p(V) \cup \{$
$\langle p_i(K, \boldsymbol{v}), \mathrm{value}(X), \mathrm{apply}(K, X, V) \Leftrightarrow p_i(K, \boldsymbol{v}), \mathrm{value}(X), [e]^g(V).\rangle$
$\} \qquad ; p_i(K, \boldsymbol{v}))$
where $\boldsymbol{v} = \mathrm{fv}(\lambda X.e)$ *and* X *and* V *fresh variables*

– $[f\ e](K) \quad = (\ [f]^p(X) \cup [f]^p(Y)\ ; ([f]^g(X), [e]^g(Y), \mathrm{apply}(X, Y, K)))$
where X *and* Y *fresh variables*

$p_i(\boldsymbol{v})$ CHR *constraints are supposed to be fresh. Then, the* CSR *program associated to e is $P[e] = [e]^p(R) \cup \{\langle start(R, \boldsymbol{v}) \Leftrightarrow [e]^g(R).\rangle\}$ and the query is $q[e] = \mathrm{start}(R, \boldsymbol{v})$ with $\boldsymbol{v} = \mathrm{fv}(e)$.*

It is immediate that the program and the goal produced by the transformation above correspond syntactically to the composition of the three transformations: λ-terms to LCC (definition 15) to flat-LCC (definition 14) to CSR (definition 12). Therefore, the transformation preserves the semantics as composition of semantics preserving transformations.

In the case of a CHR encoding, the rule associated to each λ-abstraction can be denoted as a simpagation: $\langle p_i(K, \boldsymbol{v}), \mathrm{value}(X)\backslash\mathrm{apply}(K, X, V) \Leftrightarrow [e]^g(V).\rangle$.

Example 7. The λ-term $(\lambda_1 X.\lambda_2 Y.X)\ A\ B$ is transformed to the rules:

```
start(R,A,B) ⇔
    p1(F1), apply(F1,A0,F2), apply(F2,B0,R),
    A=A0, value(A0), B=B0, value(B0).
p1(F1), value(X) \ apply(F1,X,F2) ⇔ p2(F2,X).
p2(F2,X), value(Y) \ apply(F2,Y,R) ⇔ X = R, value(R).
```

and the following goal, where the variable R codes the result:

```
| ?- start(R, A, B).
p1(_) value(_X) value(_) value(_X) p2(_, _X)
R = A
```

5 Conclusion

The translations presented in this paper generalize previous links between CHR and linear logic. As the work for modules in LCC suggest[20], variables and CHR constraints are expressive enough to embed a form of closures, and thus leads to a simple encoding for the λ-calculus.

Whereas the state during a CHR derivation is entirely determined by the contents of constraint stores, an LCC configuration contains suspended agents as well. The *ask-lifting* transformation reveals that suspensions can be reified to linear tokens, which in turns become CHR constraints: transient asks are consumed whereas persistent asks are propagated.

Behaviors of programs or agents obtained by translation are precisely related to their antecedents by (strong) bisimulation. To our knowledge, only weak bisimulation results [27] were formulated in the literature for CHR before. To achieve strong bisimulation in our case, we have managed to circumscribe collaterally the non-determinism in the naive operational semantics of CHR and in the operational semantics of LCC.

Future work

Suggested transformations are straightforward enough to be implemented. However, the moot point is to understand the relevance of CHR refined semantics for the translated LCC agents: the question of control in LCC is still open.

Interpreting operational semantics (indifferently CHR or LCC) as a proof search method in linear logic reveals a parallel between the elimination of solving non-determinism and focalization theory [28] which remains to explore.

Transition systems considered here are non-labeled: this was sufficient for semantics preservation and there are good intuitions about the pair of involved firing rules at each step. Formalizing these intuitions by labeling with rule names seems feasible but with low interest. However, labels usually serve to follow messages that an agent either sends or receives. A challenge would be to label

$\Rightarrow$-transitions by constraints whereas each single transition consumes some while adding others.

The closure encoding may suggest a new programming style, complementary to the imperative RAM-based style recently described [29]. Optimization of the CHR constraints which reify closures could be explored.

Acknowledgments. I would like to thank François Fages, Rémy Haemmerlé, Julien Martin and Sylvain Soliman for all the useful discussions and comments since the very beginning of this work.

References

1. Frühwirth, T.: Theory and practice of Constraint Handling Rules. Journal of Logic Programming, Special Issue on Constraint Logic Programming **37** (1998) 95–138
2. Coquery, E., Fages, F.: From typing constraints to typed constraint systems in CHR. In: Proceedings of Third workshop on Rule-based Constraint Reasoning and Programming, associated to CP'01. (2001)
3. Sulzmann, M., Wazny, J., Stuckey, P.J.: A framework for extended algebraic data types. In: In Proc. of FLOPS'06, volume 3945 of LNCS, Springer-Verlag (2006) 47–64
4. Otzbeyer, H., , Pretschner, E.: Testing concurrent reactive systems with constraint logic programming. In: In Proc. 2nd workshop on Rule-Based Constraint Reasoning and Programming. (2000)
5. Abdennadher, S., Marte, M.: University course timetabling using constraint handling rules. Journal of Applied Artificial Intelligence **14** (2000)
6. Saraswat, V.A.: Concurrent constraint programming. ACM Doctoral Dissertation Awards. MIT Press (1993)
7. Saraswat, V.: A brief introduction to linear concurrent constraint programming. Xerox PARC (1993)
8. Fages, F., Ruet, P., Soliman, S.: Linear concurrent constraint programming: operational and phase semantics. Information and Computation **165** (2001) 14–41
9. Girard, J.Y.: Linear logic. Theoretical Computer Science **50(1)** (1987)
10. Betz, H., Frühwirth, T.W.: A linear-logic semantics for constraint handling rules. In: Proceeding of CP 2005, 11th. (2005) 137–151
11. Giusto, C., Gabbrielli, M., Meo, M.C.: Expressiveness of multiple heads in CHR. In: SOFSEM '09: Proceedings of the 35th Conference on Current Trends in Theory and Practice of Computer Science, Berlin, Heidelberg, Springer-Verlag (2009) 205–216
12. Haemmerlé, R., Betz, H.: Verification of constraint handling rules using linear logic phase semantics. In: Proceeding of CHR 2008, the fifth Constraint Handling Rules Workshop. Report Series 08-10, RICS-Linz (2008)
13. Haemmerlé, R.: Fermetures et Modules dans les Langages Concurrents avec Contraintes fondés sur la Logique Linéaire. PhD thesis, Univ. Paris 7. Soutenance le 17 janvier 2008 (2007)
14. Johnsson, T.: Lambda lifting: transforming programs to recursive equations. In: Proc. of a conference on Functional programming languages and computer architecture, New York, NY, USA, Springer-Verlag New York, Inc. (1985) 190–203

15. Anders Schack-Nielsen, C.S.: Invited talk: The CHR-Celf connection. In Frühwirth, T., Schrijvers, T., eds.: Proceedings of the fifth Constraint Handling Rules Workshop CHR'08. (2008)
16. Schack-Nielsen, A., Schürmann, C.: Celf — a logical framework for deductive and concurrent systems (system description). In: IJCAR '08: Proceedings of the 4th international joint conference on Automated Reasoning, Berlin, Heidelberg, Springer-Verlag (2008) 320–326
17. Best, E., de Boer, F.S., Palamidessi, C.: Concurrent constraint programming with information removal. In: Proceedings of Coordination. Lecture Notes in Computer Science, Springer-Verlag (1997)
18. Dijkstra, E.: Hierarchical ordering of sequential processes. Acta Informatica **1** (1971) 115–138
19. Abdennadher, S., Frühwirth, T.W., Meuss, H.: Confluence and semantics of constraint simplification rules. Constraints **4** (1999) 133–165
20. Haemmerlé, R., Fages, F., Soliman, S.: Closures and modules within linear logic concurrent constraint programming. In Arvind, V., Prasad, S., eds.: Proceedings of FSTTCS 2007, IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science. Volume 4855 of Lecture Notes in Computer Science., Springer-Verlag (2007) 544–556
21. Jaffar, J., .Maher, M., .Marriott, K., Stuckey, P.: The semantics of constraint logic programs. Journal of Logic Programming **37** (1998) 1–46
22. Warren, D.S.: The Andorra principle. In: Presented at the Gigalips Workshop, Swedish Institute of Computer Science (SICS), Stockholm, Sweden. (1988)
23. Sangiorgi, D.: On the bisimulation proof method. Mathematical Structures in Computer Science **8** (1998) 447–479
24. Schrijvers, T., Stuckey, P.J., Duck, G.J.: Abstract interpretation for constraint handling rules. In: PPDP '05: Proceedings of the 7th ACM SIGPLAN international conference on Principles and practice of declarative programming, New York, NY, USA, ACM (2005) 218–229
25. Duck, G.J., Stuckey, P.J., Banda, M.G.D.L., Holzbaur, C.: The refined operational semantics of constraint handling rules. In: In 20th International Conference on Logic Programming (ICLP'04, Springer (2004) 90–104
26. Plotkin, G.: Call-by-name, call-by-value and the lambda calculus. Theoretical Computer Science **1** (1975) 125–159
27. Koninck, L.D.: Logical algorithms meets CHR: A meta-complexity theorem for Constraint Handling Rules with rule priorities. Theory and Practice of Logic Programming (to appear) (2009)
28. Andreoli, J.M.: Logic programming with focusing proofs in linear logic. Journal of Logic and Computation **2** (1992)
29. Sneyers, J., Schrijvers, T., Demoen, B.: The computational power and complexity of Constraint Handling Rules. In: Proceedings of the second Constraint Handling Rules Workshop, at ICLP'05. (2005) 3–17