On connections between CHR and LCC

Semantics-preserving program transformations from CHR to LCC and back

Thierry Martinez

INRIA Paris–Rocquencourt

CHR’09, 15 July 2009
The Linear Concurrent Constraint (LCC) language

- CC [Saraswat 91]: agents add constraints (tell) and wait for entailment (ask)
- LCC [Saraswat 93]: asks consume linear constraints
- Semantics formalized in [Fages Ruet Soliman 01]: asks are resources consumed by firing, recursion via declarations
- Declaration as agents [Haemmerlé Fages Soliman 07]: persistent asks (semantics via the linear-logic bang !)
LCC with declaration as agents

- Simple arrows denote transient asks.
  Linear-logic semantics: $\forall x (c \rightarrow \ldots)$.
- Double arrows denote persistent asks.
  Linear-logic semantics: $!\forall x (c \rightarrow \ldots)$.
CHR as a Concurrent Constraint language

The program is a fixed set of rules.

linear ask
(hypothesis consumption)

linear tell
Linear logic and CHR

In the literature

- Linear semantics [Betz Frühwirth 05]
  - Rules ⇔ (Banged) linear implication
  - Built-in constraints ⇔ Girard’s translation of classical formulas
  - User-defined constraint ⇔ Linear-logic predicates

- Phase semantics [Haemmerlé Betz 08]
  - Safety properties (unreachability of bad stores)

In this paper

- Translations from LCC to CHR and back.
- Operational semantics preservation.
- Linear semantics and phase semantics for free!
- Encoding the λ-calculus.
Translation from CHR to LCC

Queries
Goal translated into a single linear-logic constraint:

\[
\begin{align*}
&\underbrace{B_1, \ldots, B_p,}_{\text{built-ins}} \quad \underbrace{C_1, \ldots, C_q}_{\text{user-defined}} \\
\downarrow
\end{align*}
\]

\[!B_1 \otimes \cdots \otimes !B_n \otimes C_1 \otimes \cdots \otimes C_n\]

Rules
Program translated to a parallel composition of persistent asks:

\[
\begin{align*}
&H_1, \ldots, H_n \iff G \mid \underbrace{B_1, \ldots, B_p,}_{\text{built-ins}} \quad \underbrace{C_1, \ldots, C_q}_{\text{user-defined}} \\
\downarrow
\end{align*}
\]

\[\forall x(\ H_1 \otimes \cdots \otimes H_n \otimes !G \Rightarrow \exists y \ !B_1 \otimes \cdots \otimes !B_p \otimes C_1 \otimes \cdots \otimes C_q)\]
In CHR: two kinds of constraints

- **Store:**

- **Rules:**

In LCC: linear-logic constraints

Translation from a CHR constraint theory $CT$:

- are constraints;
- all are constraints;
- constraints closed by $\otimes$ and $\exists$.

Constraints have form: $\exists V(!B \otimes U)$

Axioms:

\[
!B \vdash !C
\]

if and only if

\[
CT \models B \rightarrow C
\]

Linear-logic predicates without axioms (linear tokens) for user-defined constraints.
Translation from flat-LCC to CHR

Flat-LCC
LCC restricted to top-level persistent asks (neither nested asks, nor transient asks)
General form of flat-LCC program:

\[ C \parallel \forall x_1(C_1 \Rightarrow C'_1) \parallel \cdots \parallel \forall x_n(C_n \Rightarrow C'_n) \]

Translation for asks
\[ C_1 \equiv \exists V_1(\exists B_1 \otimes U_1) \quad C_n \equiv \exists V_n(\exists B_n \otimes U_n) \]
\[ U_1 \Leftrightarrow B_1 \parallel B'_1, U'_1. \quad U_n \Leftrightarrow B_n \parallel B'_n, U'_n. \]

Variable hiding in query
In the initial constraint \( C \equiv \exists V(\exists B \otimes U) \), variables \( V \) are hidden. The initial constraint is translated to the rule: \( \text{start}(G) \Leftrightarrow B, U. \) and the query: \( \text{start}(G) \), where \( G = \text{fv}(C) \setminus V. \)
Ask-lifting: translation from LCC to flat-LCC

To carve asks in stone: identify them with linear tokens.

From nested asks...  ...to flat programs

Flat programs only contain persistent asks.

Tokens encode:

- ask persistence (tokens representing persistent asks are re-added to the store, the others are consumed)
- nested variable scopes

\[
\forall x \Rightarrow \exists v \\
\forall y \Rightarrow \ \\
\forall z \Rightarrow \\
\forall x \text{ xv} \Rightarrow \\
\forall x \text{ xvz} \Rightarrow \\
\forall x \text{ xv} \Rightarrow \\
\forall x \text{ xvz} \Rightarrow \text{ xv}
\]
Weakening elimination

LCC transition and weakening

Given the store $c_0$ and the agent $\forall x (d \rightarrow a)$, if $c_0$ linearly implies $d \otimes c_1$, transition to the store $c_1$ and the agent $a$.

Classical constraints weakening: $x \leq 2 \Rightarrow x \leq 3$.

In CHR, no weakening in the semantics

- User-constraints are counted in multi-sets.
- Built-in constraints always grow by conjunctions.

Weakening elimination in LCC

Disallowing weakening do not cut derivations.

Only accept transition to a store $c_1$ if there is no more general $c$ such that $c_0$ implies $d \otimes c$ (valid for principal constraint system).

Transition from $c_0$ to $c_1$ with guard $d$ only if

\[ \forall c, \text{if } c_0 \text{ implies } d \otimes c \text{ then } c_1 \text{ implies } c. \]
Steps collapsing

\[ \Rightarrow: \text{one firing per transition} \]
Strong Bisimulations

Strong comparison of processes between transition systems. Here:

- CHR transition system over states.
- LCC transition system over configurations.

**Similarity relations** $\sim$. Here:

- LCC configurations and configurations induced by ask-lifting;
- flat-LCC configurations and their translated states;
- CHR states and their translated configurations.

$\sim$ is a bisimulation if and only if:

$$
\begin{align*}
  &s \quad \xrightarrow{\sim} \quad s' \\
  &\downarrow \quad \sim \quad \downarrow \sim \\
  &\kappa \quad \xrightarrow{\sim} \quad \kappa'
\end{align*}
$$
Operational Semantics preservation

Theorem

The three following transformations:

\[
\begin{align*}
\text{LCC} & \xrightarrow{1} \text{flat-LCC} \\
\text{flat-LCC} & \xrightarrow{2} \text{CHR} \\
\text{CHR} & \xrightarrow{3} \text{LCC}
\end{align*}
\]

transform configurations(LCC)/states(CHR) to bisimilar configurations/states with respect to \(\Rightarrow\).
Let $P$ be a CHR program and $\llbracket P \rrbracket$ its translation as LCC agent.

$\iff$ [BF05]

CHR Linear-logic reading of $P$ \quad CHR Operational semantics of $P$

$\llbracket\text{immediate}\rrbracket \equiv$

LCC Linear-logic reading of $\llbracket P \rrbracket$ \quad LCC Operational semantics of $\llbracket P \rrbracket$

$\iff$ [FRS01]

$\sim$ [new]
The λ-calculus is a functional language ⇒ each expression computes a value, designated by a distinguished variable \( V \).

- \([x] = (V = x)\)
- \([\lambda x. e] = \forall x E(apply(V, x, E) ⇒ \exists V([e] || E = V))\)
- \([f \ e] = \exists F E (\exists V([f] || F = V)||\exists V([e] || E = V)||apply(F, E, V))\)
Encoding the λ-calculus in CHR

Direct translation in CHR:

$$\lambda X. \lambda Y. X) A B$$

λ-labeling: $$(\lambda_1^() X. \lambda_2^{(X)} Y. X) A B$$

$$\text{start}(R, A, B) \iff$$

$$p_1(F_1), \text{apply}(F_1, A, F_2), \text{apply}(F_2, B, R)$$

$$p_1(F_1) \backslash \text{apply}(F_1, X, F_2) \iff$$

$$p_2(F_2, X) \backslash \text{apply}(F_2, Y, R) \iff$$

$$R = X.$$
Conclusion

- Compilation scheme for LCC with committed-choice semantics
  \[ \text{LCC} \rightarrow \text{CHR} \rightarrow \ldots \]

- Proof for free for CHR linear-logic and phase semantics relying on the existing results for LCC.
- Explanation of the linear-logic reading of a CHR rule.
- Encoding of functional language with closures in CHR.
- Partially compositional (the preprocessing phase of ask-lifting, ask-labeling, is not compositional)
- Independent from the choice of Constraint Theory
Perspectives

Refined semantics for a committed-choice LCC

- From a CHR programmer point-of-view:
  - a CHR-like language with more structure constructs (nested rules & variable hiding)
  - still with a clean semantics in linear logic,
  - benefits from works on modular programming in LCC [Haemmerlé Fages Soliman 07].

- From an LCC programmer point-of-view:
  - a refined semantics,
  - with syntactic variations on asks to distinguish propagations and simplifications,
  - depending of the order agents are written.