On connections between CHR and LCC

Semantics-preserving program transformations from CHR to LCC and back

Thierry Martinez

INRIA Paris–Rocquencourt

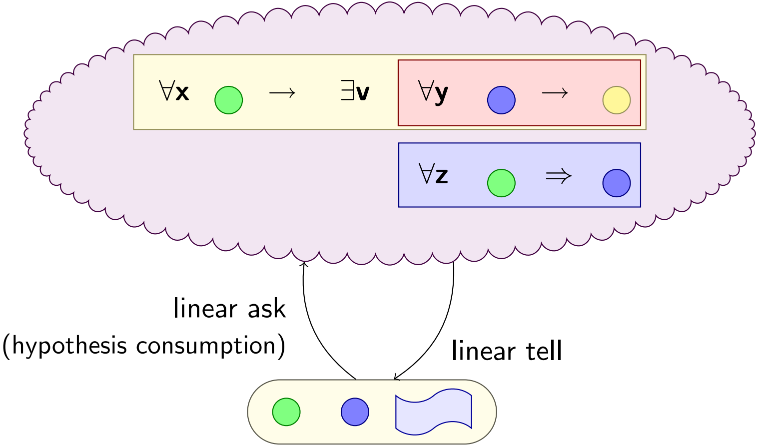
CHR'09, 15 July 2009

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 !)

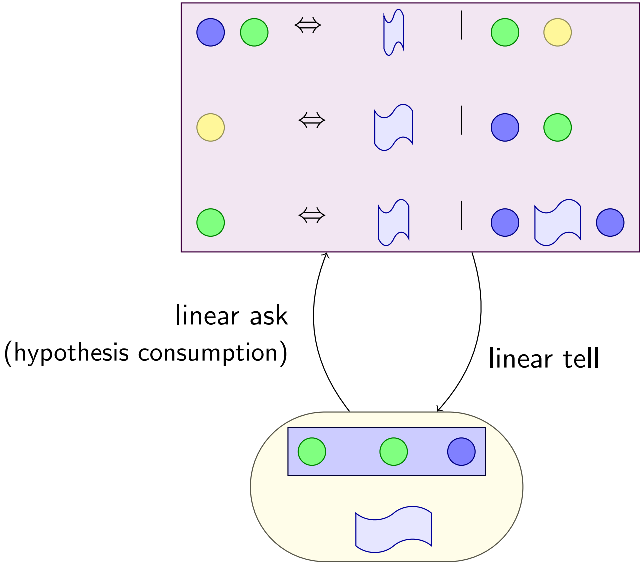
LCC with declaration as agents

- Simple arrows denote transient asks.
  Linear-logic semantics: $\forall \mathbf{x}(c \multimap \ldots)$.
- Double arrows denote persistent asks.
  Linear-logic semantics: $!\forall \mathbf{x}(c \multimap \ldots)$.



CHR as a Concurrent Constraint language

The program is a fixed set of rules.



Linear logic and CHR

In the literature

- Linear semantics [Betz Frühwirth 05]
  - Rules $\Leftrightarrow$ (Banged) linear implication
  - Built-in constraints $\Leftrightarrow$ Girard's translation of classical formulas
  - User-defined constraint $\Leftrightarrow$ 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:

$$\underbrace{B_1, \dots B_p,}_{\text{built-ins}} \qquad \underbrace{C_1, \dots C_q}_{\text{user-defined}}.$$

$$\rightsquigarrow$$

$$!B_1 \otimes \cdots \otimes !B_n \quad \otimes \quad C_1 \otimes \cdots \otimes C_n$$

Rules

Program translated to a parallel composition of persistent asks:

$$H_1, \dots, H_n \quad \Longleftrightarrow \quad G \quad | \quad \underbrace{B_1, \dots B_p,}_{\text{built-ins}} \qquad \underbrace{C_1, \dots C_q}_{\text{user-defined}}.$$

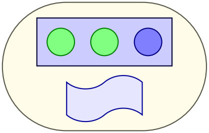
$$\rightsquigarrow$$

$$\forall \mathbf{x}(\quad H_1 \otimes \cdots \otimes H_n \quad \otimes \quad !G \quad \Rightarrow \exists \mathbf{y} \quad !B_1 \otimes \cdots \otimes !B_p \quad \otimes \quad C_1 \otimes \cdots \otimes C_q)$$

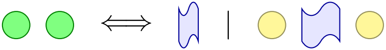
Constraint Theory / Linear Constraint System

In CHR: two kinds of constraints

- Store:

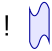


- Rules:



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: $\exists \mathbf{V}(!B \otimes U)$

Axioms:

$$!B \Vdash !C$$

if and only if

$$CT \vDash B \rightarrow C$$

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:

$$\mathcal{C} \quad \| \quad \forall \mathbf{x}_1(\mathcal{C}_1 \Rightarrow \mathcal{C}'_1) \quad \| \cdots \| \quad \forall \mathbf{x}_n(\mathcal{C}_n \Rightarrow \mathcal{C}'_n)$$

Translation for asks

$$C_1 \equiv \exists \mathbf{V}_1(!B_1 \otimes U_1) \qquad C_n \equiv \exists \mathbf{V}_n(!B_n \otimes U_n)$$

$$U_1 \Leftrightarrow B_1 \parallel B'_1, U'_1. \qquad U_n \Leftrightarrow B_n \parallel B'_n, U'_n.$$

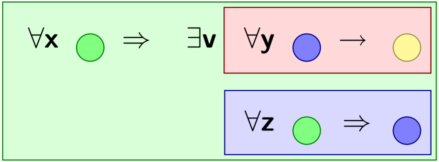
Variable hiding in query

In the initial constraint $\mathcal{C} \equiv \exists \mathbf{V}(!B \otimes U)$, variables $\mathbf{V}$ are hidden.
The initial constraint is translated to the rule: start($\mathbf{G}$) $\Leftrightarrow B, U$.
and the query: start($\mathbf{G}$), where $\mathbf{G} = \mathrm{fv}(\mathcal{C}) \setminus \mathbf{V}$.

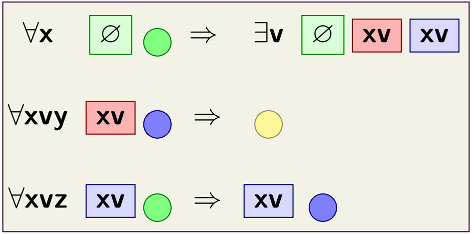
Ask-lifting: translation from LCC to flat-LCC

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

Weakening elimination

LCC transition and weakening

Given the store c_0 and the agent $\forall \mathbf{x}(d \rightarrow a)$, if c_0 linearly implies $d \otimes c_1$, transition to the store c_1 and the agent a.
Classical constraints weakening: $x \leqslant 2 \Rightarrow x \leqslant 3$.

In CHR, no weakening in the semantics

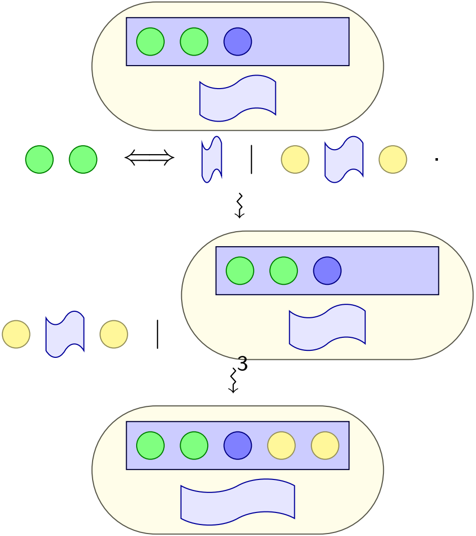
- User-constraints are counted in multi-sets.
- Built-in constraints always grow by conjunctions.

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
$\forall c$, if c_0 implies $d \otimes c$ then c_1 implies c.

Steps collapsing



$\Rightarrow$: 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.

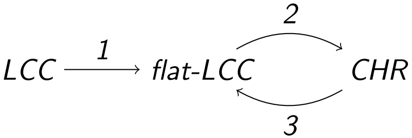
$\sim$ is a bisimulation if and only if::

$$\begin{array}{ccc} s & \Longrightarrow & s' \\ \Big|\sim & & \Big|\sim \\ \kappa & \Longrightarrow & \kappa' \end{array}$$

Operational Semantics preservation

Theorem

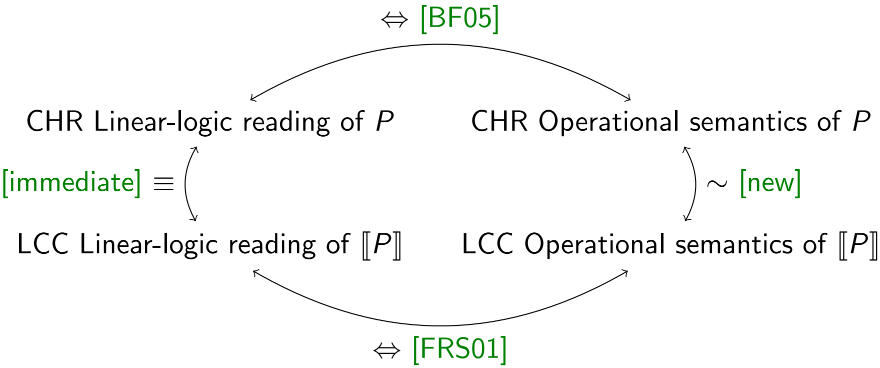
The three following transformations:



transform configurations(LCC)/states(CHR) to bisimilar configurations/states with respect to $\Rightarrow$*.*

Linear Logic Semantics correction

Let P be a CHR program and $[\![P]\!]$ its translation as LCC agent.



Encoding the λ-calculus in LCC

The λ-calculus is a functional language $\Rightarrow$ each expression computes a value, designated by a distinguished variable V.

- $[\![x]\!] = (V = x)$
- $[\![\lambda x.e]\!] = \forall x E(\text{apply}(V, x, E) \Rightarrow \exists V([\![e]\!] \parallel E = V))$
- $[\![f\ \ e]\!] = \exists FE(\ \exists V([\![f]\!] \parallel F = V) \parallel$
  $\exists V([\![e]\!] \parallel E = V) \parallel$
  $\text{apply}(F, E, V))$

Encoding the λ-calculus in CHR

λ-calculus
↓
LCC
↓
flat-LCC
↓
CHR

Direct translation in CHR:

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

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

$start(R, A, B) \Longleftrightarrow$
$\quad p1(F1), apply(F1, A, F2), apply(F2, B, R)$
$p1(F1) \setminus apply(F1, X, F2) \Longleftrightarrow$
$\quad p2(F2, X).$
$p2(F2, X) \setminus apply(F2, Y, R) \Longleftrightarrow$
$\quad R = X.$

? $start(R, A, B).$
$R = A$

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.