SiLCC a Concurrent Constraint Programming Language based on Linear Logic

François Fages
INRIA Rocquencourt,
Francois.Fages@inria.fr

FJCP, Tokyo, October 25th 2004.

Models of Execution of Constraint Programming

1. CLP = LP + Constraints [Jaffar-Lassez 86, Colmerauer 82]

   built-in constraints, inductive definition of relations.

2. CC = CLP + Concurrency [Saraswat 89, Ueda 85]

   simple constraint propagation programming, co-routines.

3. LCC = CC + Constraints consumption [Fages Ruet Soliman 98-01, Lincoln Saraswat 91]

   logical semantics in Linear Logic,

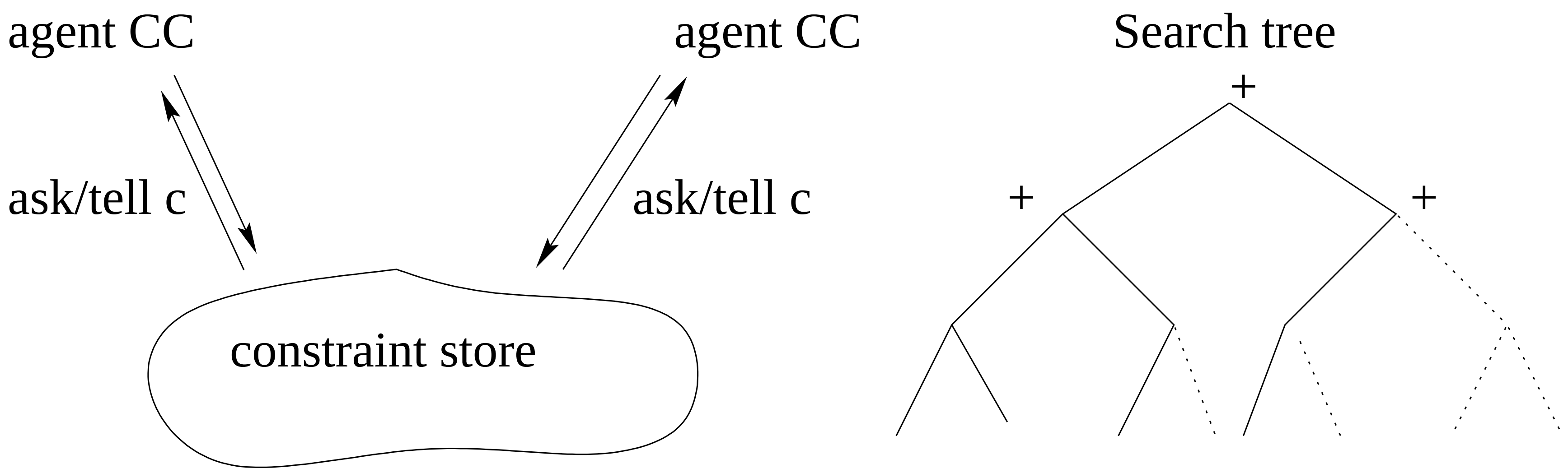
   non-monotonic store, backtrackable assignment,

   global constraints programming,

   bootstrap of SiLCC from a small kernel.

(Linear) Concurrent Constraint Programs

Processes	P ::= D.A
Declarations	D ::= $p(\vec{x}) :- A$ | D,D | ϵ
Agents	A ::= tell(c) | $\forall X$ ask(c) $\rightarrow$ A | (A $\parallel$ A) | A + A | $\exists$X A | $p(\vec{x})$



States: couple constraint store - agents $< c; A >$. Non-deterministic transitions

LCC computations

Concurrency	$=$	communication (through shared variables)
	$+$	synchronization (through ask)
Constraint consumption	$=$	linear tokens (through ask)

- Communication channels (variables) are transmissible by agents (like in π-calculus, unlike CSP, CCS)
- Private communication: only the agents sharing a variable see a constraint posted on that variable,
- Apart from linear tokens which can be removed (like Linda messages) communication is additive (like in CLP, CC)
- *Not a parallel implementation model* (global store).

Logical Semantics of (L)CC

- Interpreting $ask(c \rightarrow A)$ as logical implication leads to identify **CC transitions** with **logical deductions**:

$$\frac{c \vdash_{\mathcal{X}} d}{< c; (d \rightarrow A) >^{\dagger} \quad \vdash \quad < c; A >^{\dagger}}$$

- In order to characterize success stores, agents should not disappear by the **weakening** nor **contraction** rules: $\dfrac{\Gamma \vdash c}{\Gamma, A \vdash c} \qquad \dfrac{\Gamma, A, A \vdash c}{\Gamma, A \vdash c}$

- Linear Logic [Girard 86] appears as the logic of CC agents.

  Logic of resources: $A \Rightarrow B = !A \multimap B$

  $A \otimes A \not\vdash_{LL} A \qquad A \otimes (A \multimap B) \vdash_{LL} B \qquad A \otimes (A \multimap B) \not\vdash_{LL} A \otimes B$

  The bang operator ! provides arbitrary duplication

  $!A \otimes (A \multimap B) \vdash_{LL} !A \otimes A \otimes (A \multimap B) \vdash_{LL} !A \otimes B \vdash_{LL} B$

Intuitionistic Linear Logic

Axiom $A \vdash A$

Exchange $\dfrac{\Gamma, A, B, \Delta \vdash C}{\Gamma, B, A, \Delta \vdash C}$

Multiplicative conjunction

$$\frac{\Gamma \vdash A \quad \Delta \vdash B}{\Gamma, \Delta \vdash A \otimes B} \qquad \frac{\Gamma, A, B \vdash C}{\Gamma, A \otimes B \vdash C}$$

Implication

$$\frac{\Gamma, A \vdash B}{\Gamma \vdash A \multimap B} \qquad \frac{\Gamma \vdash A \quad \Delta, B \vdash C}{\Delta, \Gamma, A \multimap B \vdash C}$$

Additive conjunction

$$\frac{\Gamma \vdash A \quad \Gamma \vdash B}{\Gamma \vdash A \,\&\, B}$$

$$\frac{\Gamma, A \vdash C}{\Gamma, A \,\&\, B \vdash C} \qquad \frac{\Gamma, B \vdash C}{\Gamma, A \,\&\, B \vdash C}$$

Additive disjunction

$$\frac{\Gamma \vdash B}{\Gamma \vdash A \oplus B}$$

$$\frac{\Gamma, A \vdash C \quad \Gamma, B \vdash C}{\Gamma, A \oplus B \vdash C}$$

Intuitionistic Linear Logic (continued)

Constants

$$\frac{\Gamma \vdash A}{\Gamma, \mathbf{1} \vdash A} \qquad \vdash \mathbf{1} \qquad \Gamma \vdash \top \qquad \Gamma, \mathbf{0} \vdash A$$

Bang

$$\frac{\Gamma, !A, !A \vdash B}{\Gamma, !A \vdash B} \qquad \frac{\Gamma \vdash B}{\Gamma, !A \vdash B} \qquad \frac{\Gamma, A \vdash B}{\Gamma, !A \vdash B} \qquad \frac{!\Gamma \vdash A}{!\Gamma \vdash !A}$$

Quantifiers

$$\frac{\Gamma, A[t/x] \vdash B}{\Gamma, \forall x A \vdash B} \qquad \frac{\Gamma \vdash A}{\Gamma \vdash \forall x A}\ x \notin fv(\Gamma)$$

$$\frac{\Gamma, A \vdash B}{\Gamma, \exists x A \vdash B}\ x \notin fv(\Gamma, B) \qquad \frac{\Gamma \vdash A[t/x]}{\Gamma \vdash \exists x A}$$

SiLCC Constraint Systems $(\mathcal{C}, \vdash_{\mathcal{C}})$

$\mathcal{C}$ is a set of logical formulas containing 1 and = and closed by conjunction $\otimes$, $\exists$ and !.

- Classical constraints are banged !c(x)
- Linear tokens are not banged t(x)

  Equality axiom schema:

$$t(x) \otimes !(x = y) \vdash t(y)$$

- Other axioms on linear tokens of the form $t_1 \otimes ... \otimes t_n \vdash t$ would make entailment NP-complete [Palamidessi-Best 92]
- *No complex logical dependency maintenance* for linear token removals

LCC($\mathcal{X}$) Operational Semantics

Configurations: triples variables-store-agents $(X; c; \Gamma)$ modulo $\equiv$

$\mathcal{X}$-equiv. $\dfrac{c \dashv\vdash_{\mathcal{X}} d}{c \equiv d}$

α-conv. $\dfrac{z \notin fv(A)}{\exists y A \equiv \exists z A[z/y]}$

Parallel $(X; c; A \parallel B, \Gamma) \equiv (X; c; A, B, \Gamma)$

Hiding $\dfrac{y \notin fv(c, \Gamma)}{(X; c; \exists y A, \Gamma) \equiv (X \cup \{y\}; c; A, \Gamma)}$ $\dfrac{y \notin fv(c, \Gamma)}{(X \cup \{y\}; c; \Gamma) \equiv (X; c; \Gamma)}$

LCC($\mathcal{X}$) Operational Semantics (continued)

Tell $$(X; c; tell(d), \Gamma) \longrightarrow (X; c \otimes d; \Gamma)$$

Ask $$\frac{c \vdash_{\mathcal{C}} d \otimes e}{(X; c; (ask\ d \rightarrow A), \Gamma) \longrightarrow (X; e; A, \Gamma)}$$

Procedure call $$\frac{(p(\vec{y}) : -A) \in \mathcal{D}}{(X; c; p(\vec{y}), \Gamma) \longrightarrow (X; c; A, \Gamma)}$$

Blind choice $$(X; c; A + B, \Gamma) \longrightarrow (X; c; A, \Gamma)$$

$$(X; c; A + B, \Gamma) \longrightarrow (X; c; B, \Gamma)$$

Non-monotonic store (token constaints consumed by ask)

Monotonicity of transitions (*ask without default case*, distribution principle)

Logical Semantics of (L)CC in LL

Axioms of $(\mathcal{C}, \vdash_{\mathcal{C}})$ and $p(\vec{x}) \vdash A^{\dagger}$ for every declaration $p(\vec{x}) : -A \in \mathcal{D}$

$$
\begin{array}{lll}
tell(c)^{\dagger} = c^{\dagger} & (ask\ c \to A)^{\dagger} = c \multimap A^{\dagger} & (A \parallel B)^{\dagger} = A^{\dagger} \otimes B^{\dagger} \\
p(\vec{x})^{\dagger} = p(\vec{x}) & (A + B)^{\dagger} = A^{\dagger} \,\&\, B^{\dagger} & (\exists x A)^{\dagger} = \exists x A^{\dagger} \\
 & (X; c; A)^{\dagger} = \exists X (c^{\dagger} \otimes A^{\dagger}) &
\end{array}
$$

Soundness If $(X; c; \Gamma) \longrightarrow_{CC} (Y; d; \Delta)$ then $(X; c; \Gamma)^{\dagger} \vdash_{LL(\mathcal{X},\mathcal{D})} (Y; d; \Delta)^{\dagger}$.

Completeness on successes If $A^{\dagger} \vdash_{LL(\mathcal{C},\mathcal{D})} c$, then there exists a constraint d s.t. $(\emptyset; 1; A) \longrightarrow (X; d; \emptyset)$ and $\exists X d \vdash_{\mathcal{C}} c$.

Completeness on accessible stores If $A^{\dagger} \vdash_{LL(\mathcal{C},\mathcal{D})} c \otimes \top$ then there exist a constraint d and multiset Γ of agents s.t. $(\emptyset; 1; A) \longrightarrow (\emptyset; d; \Gamma)$ and $d \vdash_{\mathcal{C}} c \otimes \top$.

On-going SiLCC implementation

- Small **Kernel** module implementing LCC on labeled graphs constraints
- **Bootstrapping** Libraries :
  - Herbrand term constraints,
  - lexing (dynamic), parsing (grammatical constraints), typing (TCLP)
  - extended WAM and native code compiling (GNU-prolog technology),
  - C and OS interface,
  - search procedures, ...
- **Constraint systems** Libraries:
  - lists, graphs, quasi-lattices, ...
  - FD, global constraints,
  - R simplex, F(R) differential equations...
- **Language** Libraries: CHR, TCLP, Biocham, ...

(Logical) Imperative Features of LCC

- **Mutable Terms** [SICStus]

  linear token *value/2*

  ```
  create_mutable(D, M) :- value(M,D).
  update_mutable(D, M) :- ask(value(M,_), value(M,D)).
  get_mutable(D, M) :- ask(value(M,D), value(M,D)).
  ```

- **Global Variables and Arrays** [GNU Prolog]

  ```
  glink(V, D) :- ask(value(V,_), value(V,D)).
  gread(V, D) :- ask(value(V,D), value(V,D)).
  ```

- **Attributed Variables** [Holzbaur 92]

  linear token *attribute/2*

- **Finite domains** [Dincbas&al. 85, Tokyo, first French-Japanese Conf.]

  linear tokens *min/2 max/2 dom/2*

Conclusion

- The execution model is often hidden in C++ or Java CP libraries
  - Constraint programming is not standard C++/Java programming, nor pure modeling (no automatic heuristics yet)
  - The search tree should be explicit and central.
- LCC provides a powerful model of execution for Constraint Programming:
  - Imperative features are first-class citizens (linear tokens),
  - High-level programming of global constraints,
  - SiLCC bootstrapped implementation from a small kernel language LCC(labelled graphs) and libraries (in the spirit of Java implementation).
- Bootstrapping not only eases the development and makes it *uniform* and *robust* but is pedagogical and useful to the programmer (capability to make extensions)

  Was not achieved previously for a CP language (since Prolog): modules and imperative features are necessary

Conclusion (continued)

- Linear Logic is *the logic* of (L)CC agents.

  (Hence of constraint propagation distinguishing *alldiff* from all *diff*s)

  Refinements:

  - Characterization of suspensions in Non-Commutative NL [Ruet-Fages'97]

  - Precise semantics [Soliman'01]

- Proof methods based on :

  - Sequent calculi for LL [Miller&al.91] [Andreoli 00]

  - Proof nets [Notin'04]

  - Phase model checking [Fages-Ruet-Soliman'98, Okada'99, Soliman'02]