MSRE: Distributed Logic Programming for Decentralized Ensembles

Edmund S. L. Lam and Iliano Cervesato   Carnegie Mellon University, Qatar

1. Challenges of Parallel and Distributed Programming

- A notoriously laborious and difficult endeavor
  - Wide range of technical difficulties (e.g. deadlock, atomicity, fault-tolerance).
  - Traditional computational problems (e.g. correctness, completeness, termination).
  - While ensuring scalability and performance effectiveness.
- Open research problem:
  - Distributed programming frameworks (e.g. Map reduce [DG08], Graph Lab [LGK+10], Pregel [MAB+10], Mizan [KKAJ10])
  - Distributed programming languages (e.g. Erlang [AV90], X10 [SSvP07], NetLog [GW10], Meld [CARG+12])
  - High-level programming abstractions (e.g. Join Patterns [TR11], Parallel CHR [LS11])
- We seek an approach that is *declarative*, based on *logical foundations*, *expressive and concise.*
- Motivated by chemical reaction equations:

$$6CO_2 + 6H_2O \rightarrow C_6H_{12}O_6 + 6O_2$$

2. Introducing Rule-Based Multiset Rewriting

- Constraint Handling Rules (CHR) [Frü98]
  - Rule-based constraint logic programming language.
  - Based on multiset rewriting over first order predicate terms.
  - Concurrent, committed choice and declarative.
- CHR programs consist of a set of CHR rules of the following form:

$$r : P \setminus S \Longleftrightarrow G \mid B$$

  - Informally means: If we have **P** and **S** such that **G** is satisfiable, replace **S** with **B**.
- Example: Greatest common divisor (GCD)

$$\begin{array}{ll} base & : gcd(0) \Longleftrightarrow \text{true} \\ reduce & : gcd(N) \setminus gcd(M) \Longleftrightarrow 0 < N \wedge N \leq M \mid gcd(M{-}N) \end{array}$$

$$\begin{array}{lll}
 & \{\!\{gcd(9), gcd(6), gcd(3)\}\!\} & reduce : gcd(6) \setminus gcd(9) \Longleftrightarrow 0 < 6 \wedge 6 \leq 9 \mid \underline{gcd(3)} \\
\rightarrowtail & \{\!\{\underline{gcd(3)}, gcd(6), gcd(3)\}\!\} & reduce : gcd(3) \setminus gcd(6) \Longleftrightarrow 0 < 3 \wedge 6 \leq 9 \mid \underline{gcd(3)} \\
\rightarrowtail & \{\!\{gcd(3), \underline{gcd(3)}, gcd(3)\}\!\} & reduce : gcd(3) \setminus gcd(3) \Longleftrightarrow 0 < 3 \wedge 6 \leq 9 \mid \underline{gcd(0)} \\
\rightarrowtail & \{\!\{\underline{gcd(0)}, gcd(3), gcd(3)\}\!\} & base : gcd(0) \Longleftrightarrow \mathtt{true} \\
\rightarrowtail & \{\!\{gcd(3), gcd(3)\}\!\} & reduce : gcd(3) \setminus gcd(3) \Longleftrightarrow 0 < 3 \wedge 6 \leq 9 \mid \underline{gcd(0)} \\
\rightarrowtail & \{\!\{\underline{gcd(0)}, gcd(3)\}\!\} & base : gcd(0) \Longleftrightarrow \mathtt{true} \\
\rightarrowtail & \{\!\{gcd(3)\}\!\} &
\end{array}$$

3. MSRE, Distributed Multiset Rewriting for Ensembles

- Elements are *distributed* across distinct locations (k_1, k_2, etc..), each possessing its own multiset of elements.

$$\{\!\{edge(k_2, 1), ..\}\!\}@k_1 \longleftrightarrow \{\!\{edge(k_1, 2), edge(k_3, 8), ..\}\!\}@k_2$$
$$\nwarrow \quad \downarrow$$
$$\{\!\{edge(k_1, 10)\}\!\}@k_3$$

- Rewrite rules explicitly reference the *relative location* of constraints:

$$\begin{array}{l}
base_rule : [X]\, edge(Y, D) \setminus . \Longleftrightarrow [X]\, path(Y, D). \\
elim_rule : [X]\, path(Y, D1) \setminus [X]\, path(Y, D2) \Longleftrightarrow D1 < D2 \mid \text{true}. \\
trans_rule : [X]\, edge(Y, D), [Y]\, path(Z, D') \Longleftrightarrow X != Z \mid [X]\, path(Z, D + D').
\end{array}$$

$[l]\, c$ specifies that matching c is located at l.

- Rewrite rules can specify "local" rewriting:

$$\{\!\{edge(k_2, 1), path(k_2, 1), path(k_2, 10)\}\!\}@k_1 \ldots$$
$$\rightarrowtail \{\!\{edge(k_2, 1), path(k_2, 1)\}\!\}@k_1 \ldots \qquad [k_1]\, path(k_2, 1) \setminus [k_1]\, path(k_2, 10) \Longleftrightarrow 1 < 10 \mid \text{true}.$$

- Rewrite rules can specify link-restricted rewriting:

$$\{\!\{edge(k_2, 1), ..\}\!\}@k_1 \longleftrightarrow \{\!\{path(k_3, 8), edge(k_1, 2), edge(k_3, 8), ..\}\!\}@k_2$$
$$\nwarrow \quad \downarrow$$
$$\{\!\{edge(k_1, 10)\}\!\}@k_3$$
$$\rightarrowtail$$
$$\{\!\{edge(k_2, 1), \underline{path(k_3, 9)}, ..\}\!\}@k_1 \longleftrightarrow \{\!\{path(k_3, 8), edge(k_1, 2), edge(k_3, 8), ..\}\!\}@k_2$$
$$\nwarrow \quad \downarrow$$
$$\{\!\{edge(k_1, 10)\}\!\}@k_3$$

$$[k_1]\, edge(k_2, 1), [k_2]\, path(k_3, 8) \Longleftrightarrow k_1 != k_3 \mid \underline{[k_1]\, path(k_3, 9)}$$

4. Example: Parallel Mergesort

Parallel mergesort: Assumes tightly coupled ensembles (multicore, shared memory, etc..)

$$\begin{array}{l}
[X]\, unsorted([I]) \Longleftrightarrow [X]\, sorted([I]). \\
[X]\, unsorted(Xs) \Longleftrightarrow len(Xs) > 2 \mid \text{exists } Y.\ \text{exists } Z.\ \text{let } (Ys, Zs) = split(Xs). \\
\qquad [Y]\, parent(X), [Y]\, unsorted(Ys), [Z]\, parent(X), [Z]\, unsorted(Zs). \\
[X]\, sorted(Xs), [X]\, parent(Y) \Longleftrightarrow [Y]\, unmerged(Xs). \\
[X]\, unmerged(Xs1), [X]\, unmerged(Xs2) \Longleftrightarrow [X]\, sorted(merge(Xs1, Xs2))
\end{array}$$

- New locations "dynamically" created to solve sub-problems.
- completed sub-problems are transmitted to the "parent" location.

5. Example: Distributed Hyper-Quicksort

Distributed Hyper-Quicksort: Assumes loosely coupled ensembles (network, message passing interface, etc..)

- - "Local" sorting algorithm Parallel merge sort rules

...

- - Distributed Hyper quicksort rules

$$\begin{array}{l}
[X]\, sorted(Xs), [X]\, leader() \setminus [X]\, leaderLinks(G) \Longleftrightarrow len(G) > 1 \mid \\
\qquad \text{let } LG, GG{=}split(G).\ [X]\, leaderLinks(LG), \\
\qquad [head(GG)]\, leader(), [head(GG)]\, leaderLinks(GG), \\
\qquad \{[Y]\, median(Xs[len(Xs)/2]) \mid Y \text{ in } G\} \\
\qquad \{[Y]\, partnerLink(Z) \mid Y, Z \text{ in } zip(LG, GG)\} \\
[X]\, median(M), [X]\, sorted(Xs) \Longleftrightarrow \text{let } Ls, Gs{=}partition(Xs, M).\ [X]\, leqM(Ls), [X]\, grM(Gs) \\
[X]\, partnerLink(Y), [X]\, grM(Xs), [Y]\, leqM(Ys) \Longleftrightarrow [X]\, leqM(Ys), [Y]\, grM(Xs) \\
[X]\, leqM(Ls1), [X]\, leqM(Ls2) \Longleftrightarrow [X]\, sorted(merge(Ls1, Ls2)) \\
[X]\, grM(Gs1), [X]\, grM(Gs2) \Longleftrightarrow [X]\, sorted(merge(Gs1, Gs2))
\end{array}$$

- Data (unsorted numbers) initially distributed across 2^n locations.
- In termination, 2^n locations are in total order.

6. Multiset Comprehensions

- MSRE language includes *multiset comprehension patterns*
- Additional form of constraint patterns: $\{\!\{p(\vec{t}) \mid g\}\!\}_{\vec{x} \in t}$
  - Collects *all* $p(\vec{t})$ in the store that satisfy g
  - t is the multiset of all bindings to $\vec{x}$
- Example: pivoted swapping via comprehensions

$$pivotSwap\ @ \begin{array}{l} swap(X, Y, P) \\ \{\!\{data(X, D) \mid D \geq P\}\!\}_{D \in Xs} \\ \{\!\{data(Y, D) \mid D < P\}\!\}_{D \in Ys} \end{array} \Longleftrightarrow \begin{array}{l} \{\!\{data(Y, D)\}\!\}_{D \in Xs} \\ \{\!\{data(X, D)\}\!\}_{D \in Ys} \end{array}$$

  - Xs and Ys built from the store — *output*
  - Xs and Ys used to unfold the comprehensions — *input*

7. Preliminary Experiment Results

Program	Standard rules only			With comprehensions			Code reduction (lines)
Swap	5 preds	7 rules	21 lines	2 preds	1 rule	10 lines	110%
GHS	13 preds	13 rules	47 lines	8 preds	5 rules	35 lines	34%
HQSort	10 preds	15 rules	53 lines	7 preds	5 rules	38 lines	39%

Program	Input Size	Orig	+OJO	+OJO +Bt	+OJO +Mono	+OJO +Uniq	All	Speedup
Swap	(40, 100)	241 vs 290	121 vs 104	vs 104	vs 103	vs 92	vs 91	33%
	(200, 500)	1813 vs 2451	714 vs 681	vs 670	vs 685	vs 621	vs 597	20%
	(1000, 2500)	8921 vs 10731	3272 vs 2810	vs 2651	vs 2789	vs 2554	vs 2502	31%
GHS	(100, 200)	814 vs 1124	452 vs 461	vs 443	vs 458	vs 437	vs 432	5%
	(500, 1000)	7725 vs 8122	3188 vs 3391	vs 3061	vs 3290	vs 3109	vs 3005	6%
	(2500, 5000)	54763 vs 71650	15528 vs 16202	vs 15433	vs 16097	vs 15835	vs 15214	2%
HQSort	(8, 50)	1275 vs 1332	1117 vs 1151	vs 1099	vs 1151	vs 1081	vs 1013	10%
	(16, 100)	5783 vs 6211	3054 vs 2980	vs 2877	vs 2916	vs 2702	vs 2661	15%
	(32, 150)	13579 vs 14228	9218 vs 8745	vs 8256	vs 8617	vs 8107	vs 8013	15%

- Execution times (ms) for various optimizations on programs with increasing input size.
- Preliminary, but promising: we get code reduction plus some performance improvement.

8. Research Outcome, So Far...

- Prototype Implementation:
  - Download: https://github.com/sllam/chrcp
  - Online Demo: http://rise4fun.com/msre
- Publications:
  - "Decentralized Execution of Constraint Handling Rules for Ensembles", PPDP'13
  - "Constraint Handling Rules with Multiset Comprehension Patterns", CHR'14
  - "Reasoning About Set Comprehensions", SMT'14
  - "Optimized Compilation of Multiset Rewriting with Comprehensions", APLAS'14

9. What's Next?

- Practical applications for MSRE:
  - cloud computing applications
  - P2P apps on mobile devices, Android SDK (Funded by the QNRF, JSREP 4-003-2-001)
- Logical interpretation of comprehensions:
  - Logical proof system for MSRE
  - Linear logic + Fixed Points + Subexponentials

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

Carnegie Mellon University Qatar

مؤسسة قطر Qatar Foundation

http://www.qatar.cmu.edu/~sllam/   sllam@qatar.cmu.edu