Decentralized Execution of Multiset Rewriting Rules for 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, called CHR constraints.
  - 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)

$$base \; : gcd(0) \Longleftrightarrow \text{true}$$
$$reduce : gcd(N) \setminus gcd(M) \Longleftrightarrow 0 < N \wedge N \leq M \mid gcd(M{-}N)$$

$\wr gcd(9), gcd(6), gcd(3) \int$	$reduce : gcd(6) \setminus gcd(9) \Longleftrightarrow 0 < 6 \wedge 6 \leq 9 \mid gcd(3)$
$\rightarrowtail \wr gcd(3), gcd(6), gcd(3) \int$	$reduce : gcd(3) \setminus gcd(6) \Longleftrightarrow 0 < 3 \wedge 6 \leq 9 \mid gcd(3)$
$\rightarrowtail \wr gcd(3), gcd(3), gcd(3) \int$	$reduce : gcd(3) \setminus gcd(3) \Longleftrightarrow 0 < 3 \wedge 6 \leq 9 \mid gcd(0)$
$\rightarrowtail \wr gcd(0), gcd(3), gcd(3) \int$	$base : gcd(0) \Longleftrightarrow \text{true}$
$\rightarrowtail \wr gcd(3), gcd(3) \int$	$reduce : gcd(3) \setminus gcd(3) \Longleftrightarrow 0 < 3 \wedge 6 \leq 9 \mid gcd(0)$
$\rightarrowtail \wr gcd(0), gcd(3) \int$	$base : gcd(0) \Longleftrightarrow \text{true}$
$\rightarrowtail \wr gcd(3) \int$	

3. *CHR^e*, Distributed Multiset Rewriting for Ensembles

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

$$\wr edge(k_2, 1), .. \int @k_1 \longleftrightarrow \wr edge(k_1, 2), edge(k_3, 8), .. \int @k_2$$
$$\nwarrow \qquad \downarrow$$
$$\wr edge(k_1, 10) \int @k_3$$

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

$$base\_rule : [X]\, edge(Y, D) \setminus . \Longleftrightarrow [X]\, path(Y, D).$$
$$elim\_rule : [X]\, path(Y, D1) \setminus [X]\, path(Y, D2) \Longleftrightarrow D1 < D2 \mid \text{true}.$$
$$trans\_rule : [X]\, edge(Y, D), [Y]\, path(Z, D') \Longleftrightarrow X\, !{=} Z \mid [X]\, path(Z, D + D').$$

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

- Rewrite rules can specify "local" rewriting:

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

- Rewrite rules can specify link-restricted rewriting:

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

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

4. Example: Parallel Mergesort

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

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

- 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

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

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

6. Main Challenges

- Effective execution of multiset rewriting in decentralized context:
  - Incremental matching
  - Termination on *quiescence*
  - Interrupt (event) driven matching
- Execution of link-restricted rewrite rules is non-trivial:
  - $[X]\, partnerLink(Y), [X]\, grM(Xs), [Y]\, leqM(Ys) \Longleftrightarrow [X]\, leqM(Ys), [Y]\, grM(Xs)$
  - Requires that locations X and Y rewrites respective multisets *atomicity* .
  - In general (n locations involved), its essentially n-consensus problem.
- Designing effective mappings from *locations* to *computation resources*
  - Initialization: How are "locations" distributed across actual distributed system?
  - Load-balancing: How are dynamically created "locations" distributed?
- Designing the Language:
  - What are the minimal core language features?
  - What extended language features do we need?
  - What kind of type safety guarantees can we provide?
- Existing woes and challenges of distributed programming:
  - Fault tolerance and recovery.
  - Serializability of distributed execution.

7. Current Contributions and Results

- Developed an operational semantics for 0-link restricted rewriting
  - Based on CHR refined operational semantics [DSdlBH04].
  - Decentralized, Incremental, interrupt driven execution.
  - Proven soundness and completeness (exhaustiveness) of rewriting
- Formalized encoding of n-link restricted rewriting into 0-link restricted rewriting
  - Based on 2 Phase commit n-consensus protocol [ML85].
  - Optimized encoding for 1-link restricted rewriting
  - General encoding for n-link restricted rewriting
- Prototype implementation
  - Implemented in Python, decentralized execution via OpenMPI bindings and thread scheduling via multi-threading libraries.
  - CHR based optimization of multiset matching (e.g. optimal join ordering, indexing for non-linear patterns, early guard scheduling)
  - Basic resource mapping: Initial locations mapped to OpenMPI nodes, dynamically created locations mapped to threaded computation at source of creation.

8. Future Works

- Finalizing language design and high performance implementation
  - C, C++ or Haskell(GHC) as source language
  - Improving high-level feature encodings
  - Explore implementation via Pregel [MAB+10] or Mizan [KKAJ10].
- Improve language design
  - Aggregates, linear comprehensions, Datalog style retraction
  - Extending core language
  - New features via encoding in core language
- Dealing with unreliable communications and faulty computation resources
  - Fault tolerance backends and fault recovery interfaces
  - Improved n-link restriction encodings (via 3 Phase commit [KD95] or Paxos Algorithm [Lam98])

∗ 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