Optimized Compilation of Multiset Rewriting with Comprehensions

Edmund S. L. Lam sllam@qatar.cmu.edu
Iliano Cervesato iliano@cmu.edu

Carnegie Mellon University

Supported by grant NPRP 09-667-1-100, *Effective Programming for Large Distributed Ensembles*

APLAS'14 Singapore, Nov 2014

Outline

Constraint Handling Rules (CHR)

- A rule-based programming language:
  - Pure committed choice forward chaining
  - Declarative
  - Concurrent
- CHR is a specific instance of
  - Multiset rewriting
  - Constraint logic programming

Constraint Handling Rules (CHR)

$$r \ @ \ \bar{H} \iff g \mid \bar{B}$$

- $\bar{H}$ and $\bar{B}$ are multisets of *atomic constraint patterns*: $p(\vec{t})$
  - r: Rule name
  - $\bar{H}$: Head constraints (LHS)
  - g: Guard conditions
  - $\bar{B}$: Body constraints (RHS)
- A *program* CHR $\mathcal{P}$ is a set of rules
- CHR programs $\mathcal{P}$ are executed on constraint stores:

$$\mathcal{P} \ \triangleright \ St \ \mapsto^*_\alpha \ St'$$

- The *stores* St and St' are multiset of constraints

Adding Comprehension Patterns to CHR (CHR^{cp})

$$r \ @ \ \bar{H} \iff g \mid \bar{B}$$

- Programming in CHR is great*!
  - declarative and concise
  - high-level
- but not perfect...
  - performing aggregated operation
  - rewrite dynamic numbers of facts
- This work:
  - CHR + Comprehension Patterns (CHR^{cp})
  - Optimized compilation scheme for CHR^{cp}
  - Implementation and preliminary experimental results

Outline

Example: Pivoted Swapping

- Entities X and Y want to swap data D w.r.t. pivot P
  - All X's data $\geq P$ to Y
  - All Y's data $\leq P$ to X
- Constraints:
  - $data(X, D)$ represents data D belonging to X
  - $swap(X, Y, P)$ represents an intend to swap data between X and Y w.r.t pivot P.

Example: Pivoted Swapping (in "Vanilla" CHR)

- LHS cannot match a dynamic numbers of constraints
- Standard CHR implementation:

$$
\begin{array}{rcl}
\mathit{init} \ @ \ \mathit{swap}(X, Y, P) & \iff & \mathit{grabGE}(X, P, Y, [\,]), \mathit{grabLE}(Y, P, X, [\,]) \\
\mathit{ge1} \ @ \ \mathit{grabGE}(X, P, Y, Ds), \mathit{data}(X, D) & \iff & D \geq P \mid \mathit{grabGE}(X, P, Y, [D \mid Ds]) \\
\mathit{ge2} \ @ \ \mathit{grabGE}(X, P, Y, Ds) & \iff & \mathit{unrollData}(Y, Ds) \\
\mathit{le1} \ @ \ \mathit{grabLE}(Y, P, X, Ds), \mathit{data}(Y, D) & \iff & D \leq P \mid \mathit{grabLE}(Y, P, X, [D \mid Ds]) \\
\mathit{le2} \ @ \ \mathit{grabLE}(Y, P, X, Ds) & \iff & \mathit{unrollData}(X, Ds) \\
\mathit{unroll1} \ @ \ \mathit{unrollData}(L, [D \mid Ds]) & \iff & \mathit{unrollData}(L, Ds), \mathit{data}(L, D) \\
\mathit{unroll2} \ @ \ \mathit{unrollData}(L, [\,]) & \iff & \mathit{true}
\end{array}
$$

- Verbose: 7 rules and 3 auxiliary constraints

Example: Pivoted Swapping (in "Vanilla" CHR)

- LHS cannot match a dynamic numbers of constraints
- Standard CHR implementation:

$$
\begin{array}{rcl}
\mathit{init} \mathbin{@} \mathit{swap}(X, Y, P) & \iff & \mathit{grabGE}(X, P, Y, [\,]), \mathit{grabLE}(Y, P, X, [\,]) \\
\mathit{ge1} \mathbin{@} \mathit{grabGE}(X, P, Y, \mathit{Ds}), \mathit{data}(X, D) & \iff & D \geq P \mid \mathit{grabGE}(X, P, Y, [D \mid \mathit{Ds}]) \\
\mathit{ge2} \mathbin{@} \mathit{grabGE}(X, P, Y, \mathit{Ds}) & \iff & \mathit{unrollData}(Y, \mathit{Ds}) \\
\mathit{le1} \mathbin{@} \mathit{grabLE}(Y, P, X, \mathit{Ds}), \mathit{data}(Y, D) & \iff & D \leq P \mid \mathit{grabLE}(Y, P, X, [D \mid \mathit{Ds}]) \\
\mathit{le2} \mathbin{@} \mathit{grabLE}(Y, P, X, \mathit{Ds}) & \iff & \mathit{unrollData}(X, \mathit{Ds}) \\
\mathit{unroll1} \mathbin{@} \mathit{unrollData}(L, [D \mid \mathit{Ds}]) & \iff & \mathit{unrollData}(L, \mathit{Ds}), \mathit{data}(L, D) \\
\mathit{unroll2} \mathbin{@} \mathit{unrollData}(L, [\,]) & \iff & \mathit{true}
\end{array}
$$

- Verbose: 7 rules and 3 auxiliary constraints
- Uses accumulators

Example: Pivoted Swapping (in "Vanilla" CHR)

- LHS cannot match a dynamic numbers of constraints
- Standard CHR implementation:

$$\begin{array}{rcl}
\mathit{init} \mathbin{@} \mathit{swap}(X, Y, P) & \iff & \mathit{grabGE}(X, P, Y, [\,]), \mathit{grabLE}(Y, P, X, [\,]) \\
\mathit{ge1} \mathbin{@} \mathit{grabGE}(X, P, Y, Ds), \mathit{data}(X, D) & \iff & D \geq P \mid \mathit{grabGE}(X, P, Y, [D \mid Ds]) \\
\mathit{ge2} \mathbin{@} \mathit{grabGE}(X, P, Y, Ds) & \iff & \mathit{unrollData}(Y, Ds) \\
\mathit{le1} \mathbin{@} \mathit{grabLE}(Y, P, X, Ds), \mathit{data}(Y, D) & \iff & D \leq P \mid \mathit{grabLE}(Y, P, X, [D \mid Ds]) \\
\mathit{le2} \mathbin{@} \mathit{grabLE}(Y, P, X, Ds) & \iff & \mathit{unrollData}(X, Ds) \\
\mathit{unroll1} \mathbin{@} \mathit{unrollData}(L, [D \mid Ds]) & \iff & \mathit{unrollData}(L, Ds), \mathit{data}(L, D) \\
\mathit{unroll2} \mathbin{@} \mathit{unrollData}(L, [\,]) & \iff & \mathit{true}
\end{array}$$

- Verbose: 7 rules and 3 auxiliary constraints
- Uses accumulators
- Relies on rule priorities: apply *ge1* before *ge2*, *le1* before *le2*

CHR with Comprehension Patterns

- Additional form of constraint patterns:
  - *Comprehension patterns m:* $\wr p(\vec{x}) \mid g \int_{\vec{x} \in t}$
  - Collects *all* $p(\vec{x})$ in the store that satisfy g
  - t is the multiset of all bindings to $\vec{x}$

CHR with Comprehension Patterns

- Additional form of constraint patterns:
  - *Comprehension patterns m:* $\wr p(\vec{x}) \mid g \int_{\vec{x} \in t}$
  - Collects *all* $p(\vec{x})$ in the store that satisfy g
  - t is the multiset of all bindings to $\vec{x}$
- Pivoted Swapping in CHR^{cp}:

$$\begin{array}{rll} & swap(X, Y, P) & \\ pivotSwap \ @ & \wr data(X, D) \mid D \geq P \int_{D \in Xs} & \Longleftrightarrow \quad \wr data(Y, D) \int_{D \in Xs} \\ & \wr data(Y, D) \mid D \leq P \int_{D \in Ys} & \qquad\quad \wr data(X, D) \int_{D \in Ys} \end{array}$$

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

CHR with Comprehension Patterns

- Pivoted Swapping in CHR^{cp}:

$$pivotSwap \ @ \begin{array}{l} swap(X, Y, P) \\ \wr data(X, D) \mid D \geq P \int_{D \in Xs} \\ \wr data(Y, D) \mid D \leq P \int_{D \in Ys} \end{array} \iff \begin{array}{l} \\ \wr data(Y, D) \int_{D \in Xs} \\ \wr data(X, D) \int_{D \in Ys} \end{array}$$

- An example of CHR^{cp} rule application:

$$\wr swap(a, b, 5), data(a, 1), data(a, 6), data(a, 7), data(b, 2), data(b, 8) \int$$

CHR with Comprehension Patterns

- Pivoted Swapping in CHR^{cp}:

$$pivotSwap \ @ \begin{array}{l} swap(X, Y, P) \\ \wr data(X, D) \mid D \geq P \int_{D \in Xs} \\ \wr data(Y, D) \mid D \leq P \int_{D \in Ys} \end{array} \iff \begin{array}{l} \\ \wr data(Y, D) \int_{D \in Xs} \\ \wr data(X, D) \int_{D \in Ys} \end{array}$$

- An example of CHR^{cp} rule application:

$$\begin{array}{ll} & \wr swap(a, b, 5), data(a, 1), data(a, 6), data(a, 7), data(b, 2), data(b, 8) \int \\ \mapsto_{\alpha} & \wr \quad data(a, 1), data(b, 6), data(b, 7), data(a, 2), data(b, 8) \int \end{array}$$

  – applying *pivotSwap* with $\{a/X \ , \ b/Y \ , \ 5/P \ , \ \wr 6, 7 \int /Xs \ , \ \wr 2 \int /Ys\}$

CHR with Comprehension Patterns

- Pivoted Swapping in CHR^{cp}:

$$\mathit{pivotSwap} \ @ \ \begin{array}{l} \mathit{swap}(X, Y, P) \\ \lbag \mathit{data}(X, D) \mid D \geq P \rbag_{D \in Xs} \\ \lbag \mathit{data}(Y, D) \mid D \leq P \rbag_{D \in Ys} \end{array} \iff \begin{array}{l} \lbag \mathit{data}(Y, D) \rbag_{D \in Xs} \\ \lbag \mathit{data}(X, D) \rbag_{D \in Ys} \end{array}$$

- An example of CHR^{cp} rule application:

$$\begin{array}{ll} & \lbag \mathit{swap}(a, b, 5), \mathit{data}(a, 1), \mathit{data}(a, 6), \mathit{data}(a, 7), \mathit{data}(b, 2), \mathit{data}(b, 8) \rbag \\ \mapsto_\alpha & \lbag \qquad \mathit{data}(a, 1), \mathit{data}(b, 6), \mathit{data}(b, 7), \mathit{data}(a, 2), \mathit{data}(b, 8) \rbag \end{array}$$

  – applying *pivotSwap* with $\{a/X \ , \ b/Y \ , \ 5/P \ , \ \lbag 6, 7 \rbag/Xs \ , \ \lbag 2 \rbag/Ys\}$

- Semantics of CHR^{cp} guarantees *maximality of comprehension*:

$$\lbag \mathit{swap}(a, b, 5), \mathit{data}(a, 1), \boxed{\mathit{data}(a, 6)}, \mathit{data}(a, 7), \mathit{data}(b, 2), \mathit{data}(b, 8) \rbag$$

CHR with Comprehension Patterns

- Pivoted Swapping in CHR^{cp}:

$$pivotSwap\ @\ \begin{array}{l} swap(X,Y,P) \\ \wr data(X,D) \mid D \geq P \int_{D \in Xs} \\ \wr data(Y,D) \mid D \leq P \int_{D \in Ys} \end{array} \iff \begin{array}{l} \\ \wr data(Y,D) \int_{D \in Xs} \\ \wr data(X,D) \int_{D \in Ys} \end{array}$$

- An example of CHR^{cp} rule application:

$$\begin{array}{ll} & \wr swap(a,b,5), data(a,1), data(a,6), data(a,7), data(b,2), data(b,8) \int \\ \mapsto_\alpha & \wr \quad\quad data(a,1), data(b,6), data(b,7), data(a,2), data(b,8) \int \end{array}$$

  – applying *pivotSwap* with $\{a/X\ ,\ b/Y\ ,\ 5/P\ ,\ \wr 6,7 \int /Xs\ ,\ \wr 2 \int /Ys\}$

- Semantics of CHR^{cp} guarantees *maximality of comprehension*:

$$\begin{array}{ll} & \wr swap(a,b,5), data(a,1), \boxed{data(a,6)}, data(a,7), data(b,2), data(b,8) \int \\ \not\mapsto_\alpha & \wr \quad\quad data(a,1), \boxed{data(a,6)}, data(b,7), data(a,2), data(b,8) \int \end{array}$$

  – not valid! $\boxed{data(a,6)}$ is left behind!

Semantics of CHR^{cp}

- Abstract semantics of CHR^{cp} ($\mapsto_\alpha$):
  - High level specification of CHR^{cp}
  - Formalizes "maximality of comprehensions"
- Operational semantics of CHR^{cp} ($\mapsto_\omega$):
  - Extends from [**?**]
  - Defines systematic execution scheme that avoids recomputation of matches
- $\mapsto_\omega$ is sound w.r.t $\mapsto_\alpha$
- See paper and technical report for details!

Outline

Highlights of CHR^{cp} optimized compilation

- Compile CHR^{cp} rules into procedural operations:
  - Implements the operational semantics ($\mapsto_\omega$)
  - with various optimizations
- Existing CHR optimizations are still applicable:
  - Optimal join ordering + index selection (**OJO**)
- Optimizations specific to comprehension patterns:
  - Incrementally store monotone constraints (**Mono**)
  - Selective enforcement of unique match (**Uniq**)
  - Bootstrapping active comprehension patterns (**Bt**)

Optimal Join-Ordering and Index Selection (OJO)

Given a rule: $p_1, ..., p_i, ..., p_n \iff ...$

- OJO in standard CHR:
  - Find candidates p_j in an optimal sequence
  - Compute optimal indexing structures on p_j
- We extend (**OJO**) to handle comprehension patterns:

$$swap(X, Y, P), \wr data(X, D) \mid D \geq P \int_{D \in Xs}, \wr data(Y, D) \mid D < P \int_{D \in Ys} \iff ...$$

  - Given $swap(john, jack, 40)$, how do we find all $data(john, D)$ such that $\wr data(john, D) \mid D \geq 40 \int_{D \in Xs}$?
  - Heuristics to compute optimal orderings
  - Compute indexing structures on comprehension patterns

Incrementally store monotone constraints (**Mono**) 1/3

- Standard CHR is monotonic:
  $\mathcal{P} \rhd St \mapsto_\alpha St'$ implies $\mathcal{P} \rhd \wr St, St'' \int \mapsto_\alpha \wr St', St'' \int$
  - We can *incrementally* store all constraints
  - Benefits? Performance improvements!

Incrementally store monotone constraints (**Mono**) 1/3

- Standard CHR is monotonic:
  $\mathcal{P} \rhd St \mapsto_\alpha St'$ implies $\mathcal{P} \rhd \wr St, St'' \int \mapsto_\alpha \wr St', St'' \int$
  - We can *incrementally* store all constraints
  - Benefits? Performance improvements!
- But CHR^{cp} is not monotonic! Recall:

$$\begin{array}{rl} & \wr swap(a,b,5), data(a,1), \boxed{data(a,6)}, data(a,7), data(b,2), data(b,8) \int \\ \not\mapsto_\alpha & \wr \quad\quad\quad\quad\quad\quad data(a,1), \boxed{data(a,6)}, data(b,7), data(a,2), data(b,8) \int \end{array}$$

  - Not valid! Because we want maximality of comprehensions!
  - Naive solution: Store ALL constraints IMMEDIATELY!

Incrementally store monotone constraints (**Mono**) 2/3

- Conditional monotonicity [**?**]:

$$\text{If} \quad St'' \text{ contains only constraints } \textit{monotone w.r.t. } \mathcal{P}$$
$$\text{then} \quad \mathcal{P} \rhd St \mapsto_\alpha St' \quad \text{implies} \quad \mathcal{P} \rhd \wr St, St'' \int \mapsto_\alpha \wr St', St'' \int$$

- A constraint c is *monotone* w.r.t. $\mathcal{P}$ (denoted $\mathcal{P} \stackrel{\triangle}{=}^{\neg}_{\textbf{unf}} c$), *iff* c cannot participate in any comprehension pattern m such that $(r@..., m, ... \Longleftrightarrow ...) \in \mathcal{P}$
- (Mono) optimization: Store ONLY non-monotone constraints IMMEDIATELY!

Incrementally store monotone constraints (**Mono**) 3/3

$$\begin{array}{lll} & \mathit{swap}(X,Y,P) & \\ \mathit{pivotSwap}\ @ & \wr \mathit{data}(X,D) \mid D \geq P \int_{D \in Xs} & \Longleftrightarrow \quad \wr \mathit{data}(Y,D) \int_{D \in Xs} \\ & \wr \mathit{data}(Y,D) \mid D < P \int_{D \in Ys} & \qquad\quad \wr \mathit{data}(X,D) \int_{D \in Ys} \end{array}$$

- *swap* is monotone, but not *data*
- "Glass is half-full!": incrementally store *swap* constraints!
- We statically compute $\mathcal{P} \stackrel{\triangle}{=}^{\neg}_{\mathbf{unf}} c$ for each RHS constraint
- In experiments, we compare performances of
  - Naive approach: store ALL IMMEDIATELY!
  - (Mono) optimization: ONLY store non-monotone constraints IMMEDIATELY!

Selective Enforcement of Unique Match (**Uniq**) 1/2

- Consider a rule with more than one comprehension pattern:

$$r@\ldots, m_i, \ldots, m_j, \ldots \Longleftrightarrow \ldots$$

- m_i and m_j potentially match with the same constraints.
  - E.g., $m_i = \wr p(X) \mid X < 4 \int$ and $m_j = \wr p(Y) \mid Y > 2 \int$
  - Notice that $p(3)$ can be matched to either m_i OR m_j
- In general, we need match m_i and m_j to non-overlapping multisets of constraints – a potentially expensive operation

Selective Enforcement of Unique Match (**Uniq**) 2/2

- But sometimes...

$$\begin{array}{rcl} & & swap(X, Y, P) \\ m_i & = & \wr data(X, D) \mid D \geq P \int \\ m_j & = & \wr data(Y, D) \mid D < P \int \end{array} \iff \ldots$$

  - Notice that $m_i \cap m_j \neq \emptyset$ is unsatisfiable!
- (Uniq) Optimization:
  - Statically test $m_i \cap m_j \neq \emptyset$ for each pair m_i and m_j
  - Avoid redundant runtime operations
- In experiments, we compare performances of:
  - Naive approach: Enforce uniqueness for ALL m_i, m_j pairs.
  - (Uniq) optimization: Only for m_i, m_j if $m_i \cap m_j \neq \emptyset$ is satisfiable

Reasoning about Comprehensions

- Wait! So, how do we statically compute $\mathcal{P} \stackrel{\triangleq}{=}^{\neg}_{\mathbf{unf}} c$ (from **Mono**)?
- We implemented a library extension to Microsoft's Z3 solver:
  - to reason about set comprehensions.
- Details in "Reasoning about Set Comprehensions", SMT'2014
- Download at: https://github.com/sllam/pysetcomp

Bootstrapping active comprehension patterns (**Bt**)

$$\begin{array}{lll} & swap(X,Y,P) & \\ pivotSwap\ @ & \wr data(X,D) \mid D \geq P \int_{D \in Xs} & \Longleftrightarrow \quad \wr data(Y,D) \int_{D \in Xs} \\ & \wr data(Y,D) \mid D \leq P \int_{D \in Ys} & \quad\quad\ \wr data(X,D) \int_{D \in Ys} \end{array}$$

- Suppose that we are processing a $data(john, 5)$ and attempting to match it to $\wr data(X,D) \mid D \geq P \int_{D \in Xs}$.
- Naive approach:
  1. Retrieve ALL $data(john, \_)$ (since P is unknown)
  2. Filter away bindings in Xs after we found a $swap(john, Y, P)$
- Bootstrapping active comprehension pattern (**Bt**):
  1. Match $data(john, 5)$ to $data(X, D)$ (treat as atomic pattern)
  2. Complete the comprehension pattern AFTER (and ONLY IF) we actually found $swap(john, Y, P)$

Outline

Implementation

- Prototype implementation
  - Available at:
    - https://github.com/sllam/msre for download
    - http://rise4fun.com/msre for online demo
  - Compiler and type-checker implemented in Python
  - Generates C++ codes that implements CHR^{cp} run-time
  - Utilizes Z3 SMT solver for type checking and other analysis [**?**]

Preliminary Experimental 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.
n vs m : *n* is execution time for standard rules, *m* is execution time for comprehensions
vs m : *m* is execution time for comprehensions

- Code reduction + performance improvement
- Its preliminary, but promising

Preliminary Experimental 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.
n vs m : *n* is execution time for standard rules, *m* is execution time for comprehensions
vs m : *m* is execution time for comprehensions

- Code reduction + performance improvement
- Its preliminary, but promising

Preliminary Experimental 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.
n vs m : *n* is execution time for standard rules, *m* is execution time for comprehensions
vs m : *m* is execution time for comprehensions

- Code reduction + performance improvement
- Its preliminary, but promising

Outline

Conclusion

- CHR^{cp}: Multiset rewriting + comprehension patterns
- Optimized compilation scheme:
  - Implements operational semantics of CHR^{cp}
  - Optimization schemes (OJO), (Bt), (Mono) and (Uniq)
  - Promising preliminary results (code reduction + improved performance)

Future Works

- Decentralized execution [**?**]:
  - over traditional computer networks
  - P2P applications on mobile devices (Android SDK)
- Logical interpretation of comprehensions:
  - Logical proof system for CHR^{cp}
  - Linear logic with appropriate extensions

Thank you for your attention!

Questions please?

Bibliography