CoMingle: Distributed Declarative Programming for Decentralized Mobile Ensembles

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

Supported by QNRF grants JSREP 4-003-2-001 and NPRP 09-667-1-100
Outline

1. Introduction
2. Example
3. Semantics
4. Compilation
5. Status
6. Conclusions & Future Work
Distributed Programming

- Computations that run at more than one place at once
  - A 40 year old paradigm, now more popular than ever
    - Cloud computing
    - Modern webapps
    - Mobile device applications

- Hard to get right
  - Concurrency bugs (race conditions, deadlocks, . . .)
  - Communication bugs
  - “Normal” bugs

- Two views
  - Node-centric — program each node separately
  - System-centric — program the distributed system as a whole
    - Compiled to node-centric code
    - Used in limited settings (Google Web Toolkit, MapReduce)
    - Related to choreographic programming (Jolie, DIOC)
What is CoMingle?

A programming language for distributed mobile apps

- Declarative, concise, based on linear logic
- Enables high-level system-centric abstraction
  - specifies distributed computations as ONE declarative program
  - compiles into node-centric fragments, executed by each node
- Designed to implement mobile apps that run across Android devices
- Inspired by CHR [Frühwirth and Raiser, 2011], extended with
  - Decentralization [Lam and Cervesato, 2013]
  - Comprehension patterns [Lam and Cervesato, 2014]
  - Time synchronization [Lam et al., 2015]
  - Modularity [Cervesato and Lam, 2015]
- Also inspired by Linear Meld [Cruz et al., 2014]
Outline

1. Introduction
2. Example
3. Semantics
4. Compilation
5. Status
6. Conclusions & Future Work
module comingle.lib.ExtLib import {
    size :: A -> int.
}

predicate swap :: (loc,int) -> trigger.
predicate item :: int -> fact.
predicate display :: (string,A) -> actuator.

rule pivotSwap :: [X]swap(Y,P),
    {[X]item(D)|D->Xs. D >= P},
    {[Y]item(D)|D->Ys. D <= P}
    --o [X]display(Msg,size(Ys),Y), {[X]item(D)|D<-Ys},
    [Y]display(Msg,size(Xs),X), {[Y]item(D)|D<-Xs}
    where Msg = "Received %s items from %s".
CoMingle by Example: Decentralized Multiset Rewriting

[X]swap(Y,P) 
{[X]item(D)|D->Xs.D>=P} --- o [X]display(Msg,size(Ys),Y), {[X]item(D)|D<=Ys} 
{[Y]item(D)|D->Ys.D<=P} [Y]display(Msg,size(Xs),X), {[Y]item(D)|D<=Xs} 

where Msg = "Received %s items from %s".

Let s = swap, i = item and d = display

Node: n1  
|s(n2, 5), i(4), i(6), i(8)|

Node: n2  
i(3), i(20)

Node: n3  
s(n2, 10), i(18)
CoMingle by Example: Decentralized Multiset Rewriting

\[ [X]\text{swap}(Y,P) \]
\{ [X]\text{item}(D)|D\rightarrow Xs.D \geq P \} \rightarrow_o \{ [X]\text{display}(\text{Msg},\text{size}(Ys),Y), \{ [X]\text{item}(D)|D \leftarrow Ys \} \}
\{ [Y]\text{item}(D)|D \rightarrow Ys.D \leq P \} \rightarrow_o \{ [Y]\text{display}(\text{Msg},\text{size}(Xs),X), \{ [Y]\text{item}(D)|D \leftarrow Xs \} \}

\text{where} \quad \text{Msg} = "\text{Received } \%s \text{ items from } \%s".

Let \( s = \text{swap}, i = \text{item} \) and \( d = \text{display} \)

\[
\begin{array}{c|c|c}
\text{Node: } n_1 & \text{Node: } n_2 & \text{Node: } n_3 \\
\hline
s(n_2, 5), i(4), i(6), i(8) & i(3), i(20) & s(n_2, 10), i(18) \\
\end{array}
\]
CoMingle by Example: Decentralized Multiset Rewriting

\[ X \text{swap}(Y,P) \]
\{[X]\text{item}(D)|D->Xs.D>=P\} \rightarrow \text{\ o } \{[X]\text{display}(Msg,\text{size}(Ys),Y), \{[X]\text{item}(D)|D<-Ys\}\}
\{[Y]\text{item}(D)|D->Ys.D<=P\} \rightarrow \{[Y]\text{display}(Msg,\text{size}(Xs),X), \{[Y]\text{item}(D)|D<-Xs\}\}

where \( Msg = "Received \%s \ items from \%s". \)

Let \( s = \text{swap} \), \( i = \text{item} \) and \( d = \text{display} \)

Let \( s = \text{swap} \), \( i = \text{item} \) and \( d = \text{display} \)

\begin{align*}
\text{Node: } n_1 & \quad \text{Node: } n_2 \quad \text{Node: } n_3 \\
&s(n_2,5),i(4),i(6),i(8) & i(3),i(20) & s(n_2,10),i(18) \\
\downarrow & \quad \downarrow & \quad \downarrow \\
\text{Node: } n_1 & \quad \text{Node: } n_2 \quad \text{Node: } n_3 \\
&d("1 from n_2") & d("2 from n_1") & s(n_2,10),i(18) \\
&i(3),i(4) & i(6),i(8),i(20) & \end{align*}
CoMingle by Example: Decentralized Multiset Rewriting

\[
[X]_{\text{swap}}(Y,P) \\
\{[X]_{\text{item}}(D) | D \rightarrow Xs. D \geq P \} \longrightarrow [X]_{\text{display}}(\text{Msg}, \text{size}(Ys), Y), \{[X]_{\text{item}}(D) | D \leftarrow Ys \} \\
\{[Y]_{\text{item}}(D) | D \rightarrow Ys. D \leq P \} \quad [Y]_{\text{display}}(\text{Msg}, \text{size}(Xs), X), \{[Y]_{\text{item}}(D) | D \leftarrow Xs \}
\]

where \( \text{Msg} = \text{"Received } \%s \text{ items from } \%s\).

Let \( s = \text{swap}, i = \text{item} \) and \( d = \text{display} \)

\[
\begin{array}{c}
\text{Node: } n_1 \\
s(n_2, 5), i(4), i(6), i(8)
\end{array}
\rightarrow
\begin{array}{c}
\text{Node: } n_2 \\
i(3), i(20)
\end{array}
\rightarrow
\begin{array}{c}
\text{Node: } n_3 \\
s(n_2, 10), i(18)
\end{array}
\]

\[
\begin{array}{c}
\text{Node: } n_1 \\
d("1 \text{ from } n_2") \\
i(3), i(4)
\end{array}
\downarrow
\begin{array}{c}
\text{Node: } n_2 \\
d("2 \text{ from } n_1") \\
i(6), i(8), i(20)
\end{array}
\rightarrow
\begin{array}{c}
\text{Node: } n_3 \\
s(n_2, 10), i(18)
\end{array}
\]
CoMingle by Example: Decentralized Multiset Rewriting

\[
[X]\text{swap}(Y,P) \\
\{[X]\text{item}(D)|D\to Xs.D\geq P\} \rightarrow o \quad [X]\text{display}(\text{Msg},\text{size}(Ys),Y), \{[X]\text{item}(D)|D\leftarrow Ys\}\ \\
\{[Y]\text{item}(D)|D\to Ys.D\leq P\} \quad [Y]\text{display}(\text{Msg},\text{size}(Xs),X), \{[Y]\text{item}(D)|D\leftarrow Xs\}\ \\
\text{where } \text{Msg} = "\text{Received } \%s \text{ items from } \%s".
\]

Let \(s = \text{swap}, i = \text{item}\) and \(d = \text{display}\)

```
Node: \(n_1\)  
\(s(n_2, 5), i(4), i(6), i(8)\) 

Node: \(n_2\)  
i(3), i(20)

Node: \(n_3\)  
\(s(n_2, 10), i(18)\)
```

```
Node: \(n_1\)  
\(d("1 \text{ from } n_2")\)  
i(3), i(4)

Node: \(n_2\)  
\(d("2 \text{ from } n_1")\)  
i(6), i(8), i(20)

Node: \(n_3\)  
\(s(n_2, 10), i(18)\)
```

```
Node: \(n_1\)  
\(d("1 \text{ from } n_2")\)  
i(4), i(3)

Node: \(n_2\)  
\(d("2 \text{ from } n_1")\)  
\(d("1 \text{ from } n_3")\)  
i(18), i(20)

Node: \(n_3\)  
\(d("2 \text{ from } n_2")\)  
i(6), i(8)
```
CoMingle Architecture
CoMingle by Example: Triggers and Actuators

**predicate** swap :: (loc,int) -> trigger.
**predicate** item :: int -> fact.
**predicate** display :: (string,A) -> actuator.

**rule** pivotSwap :: [X]swap(Y,P),
{[X]item(D)|D->Xs. D >= P},
{[Y]item(D)|D->Ys. D <= P}
  --o [X]display(Msg,size(Ys),Y), {[X]item(D)|D<-Ys},
  [Y]display(Msg,size(Xs),X), {[Y]item(D)|D<-Xs}
  **where** Msg = "Received %s items from %s".

- **Abstracts** communications between node (i.e., X, Y)
- Executed by a **rewriting runtime** on each node
- Interacts with a local **application runtime** on each node
- **Triggers**: inputs from the application runtime
- **Actuators**: outputs into the application runtime
CoMingle by Example: Triggers and Actuators

**Predicate**: `swap` is a **trigger**

- An input interface into the rewriting runtime
- Only in rule heads
- `swap(Y,P)` is added to rewriting state when button on device X is pressed
CoMingle by Example: Triggers and Actuators

**Predicate** `swap` :: (loc, int) -> `trigger`.
**Predicate** `item` :: int -> `fact`.
**Predicate** `display` :: (string, A) -> `actuator`.

**Rule** `pivotSwap` :: `[X]swap(Y,P),`  
{[X]item(D)|D->Xs. D >= P},  
{[Y]item(D)|D->Ys. D <= P}  
--o [X]display(Msg, size(Ys), Y), {[X]item(D)|D<-Ys},  
[Y]display(Msg, size(Xs), X), {[Y]item(D)|D<-Xs}  
where Msg = "Received %s items from %s".

- **Predicate** `display` is an **actuator**
  - An output interface from the rewriting runtime
  - Only in rule body
  - `display("2 from n1")` executes a screen display callback function
Predicate *swap* is a standard *fact*

*Can appear in rule head or body*

*Atoms of the rewriting state*
CoMingle by Example

[X]\text{swap}(Y,P)
\{[X]\text{item}(D)|D->Xs.D\geq P\} \quad - \circ \quad [X]\text{display}(\text{\textit{Msg}}, \text{size}(Ys), Y), \{[X]\text{item}(D)|D<-Ys\}
\{[Y]\text{item}(D)|D->Ys.D\leq P\} \quad [Y]\text{display}(\text{\textit{Msg}}, \text{size}(Xs), X), \{[Y]\text{item}(D)|D<-Xs\}

\text{where } \text{Msg} = \text{"Received } \%s \text{ items from } \%s\text{"}.

- High-level specification of distributed triggers/actuators

\begin{align*}
\text{\textit{Distributed Triggers}} && \text{\textit{Distributed Actuators}} \\
\text{\textit{Distributed State Patterns}} + \circ \quad \text{\textit{Distributed State Patterns}}
\end{align*}

- Declarative, concise and executable!
- Abstracts away
  - Low-level message passing
  - Synchronization
- Ensures atomicity and isolation
Outline

1. Introduction
2. Example
3. Semantics
4. Compilation
5. Status
6. Conclusions & Future Work
Abstract Syntax

- A CoMingle program $\mathcal{P}$ is a set of rules of the form
  \[ r : H_p \setminus H_s \mid g \rightarrow B \]
  - $H_p$, $H_s$ and $B$: Multisets of patterns
  - $g$: Guard conditions

- A pattern is either
  - a fact: $[\ell]p(\bar{t})$
  - a comprehension: $\bigcup [\ell]p(\bar{t}) \mid g \bigcap \bar{x} \in t$

- Three kinds of facts
  - **Triggers** (only in $H_p$ or $H_s$): Inputs from the “Android world”
  - **Actuators** (only in $B$): Outputs to the “Android world”
  - Standard facts: Atoms of rewriting state
Semantics of CoMingle: Abstract State Transitions

- **CoMingle state** \( \langle St; \Psi \rangle \) represents the mobile ensemble
  - \( St \) is the *rewriting state*, a multiset of ground facts \([\ell]f\)
  - \( \Psi \) is the *application state*, a set of local states \([\ell]\psi\)
  - A location \( \ell \) is a computing node
Semantics of CoMingle: Abstract State Transitions

- **CoMingle state** \( \langle St; \Psi \rangle \) represents the mobile ensemble
  - \( St \) is the *rewriting state*, a multiset of ground facts \([\ell]f\)
  - \( \Psi \) is the *application state*, a set of local states \([\ell]\psi\)
  - A location \( \ell \) is a computing node

- The rewrite runtime: \( \mathcal{P} \triangleright \langle St; \Psi \rangle \mapsto \langle St'; \Psi \rangle \)
  - Applies a rule in \( \mathcal{P} \)
  - Several locations may participate
  - Decentralized multiset rewriting
**Semantics of CoMingle: Abstract State Transitions**

- **CoMingle state** $\langle St; \Psi \rangle$ represents the mobile ensemble
  - $St$ is the *rewriting state*, a multiset of ground facts $[\ell]f$
  - $\Psi$ is the *application state*, a set of local states $[\ell]\psi$
  - A location $\ell$ is a computing node

- **The rewrite runtime**: $\mathcal{P} \models \langle St; \Psi \rangle \rightarrow \langle St'; \Psi \rangle$
  - Applies a rule in $\mathcal{P}$
  - Several locations may participate
  - Decentralized multiset rewriting

- **The application runtime**: $\langle A; \psi \rangle \mapsto_{\ell} \langle T; \psi' \rangle$
  - Models local computation within a node
  - All within location $\ell$
Rewriting Runtime: Overview

- **Decentralized semantics** [Lam and Cervesato, 2013]
  - Facts are explicitly annotated with locations, $[\ell]p(t)$
  - System-centric decentralized multiset rewriting
  - Compiled into node-centric specifications
Rewriting Runtime: Overview

- **Decentralized semantics** [Lam and Cervesato, 2013]
  - Facts are explicitly annotated with locations, $[\ell]p(\vec{t})$
  - System-centric decentralized multiset rewriting
  - Compiled into node-centric specifications

- **Comprehension patterns** [Lam and Cervesato, 2014]
  \[ \exists [\ell]p(\vec{t}) \mid g \bigcap_{\vec{x} \in T} \]
  - Multiset of all $[\ell]p(\vec{t})$ in the state that satisfy $g$
  - $\vec{x}$ bound in $g$ and $\vec{t}$
  - $T$ is the multiset of all bindings $\vec{x}$
  - Semantics enforces maximality of $T$
Comprehension Example: Pivoted Swapping

\[
pivotSwap : \begin{align*}
[X]_{\text{swap}}(Y, P) & \quad \text{where} \quad [X]_{\text{item}}(D) \mid D \geq P \models D \in X_s \\
[Y]_{\text{item}}(D) \mid D \leq P \models D \in Y_s
\end{align*}
\]

- \( X_s \) and \( Y_s \) built from the rewriting state — \textit{output}
- \( X_s \) and \( Y_s \) used to unfold the comprehensions — \textit{input}
- Atomic
Rewriting Runtime: Rewriting Semantics

- Rewriting runtime transition: \( \mathcal{P} \triangleright \langle St; \Psi \rangle \mapsto \langle St'; \Psi \rangle \)
  - Applies a rule in \( \mathcal{P} \) to transform \( St \) into \( St' \)

\[
\begin{align*}
(\overline{H_p} \setminus \overline{H_s} \mid g \rightarrow \overline{B}) & \in \mathcal{P} \quad \models \theta g \\
\theta \overline{H_p} & \triangleq_{\text{lhs}} St_p \\
\theta \overline{H_s} & \triangleq_{\text{lhs}} St_s \\
\theta(\overline{H_p}, \overline{H_s}) & \triangleq_{\text{lhs}} St \\
\theta \overline{B} & \gg \text{rhs} St_b
\end{align*}
\]

\[
\mathcal{P} \triangleright \langle St_p, St_s, St; \Psi \rangle \mapsto \langle St_p, St_b, St; \Psi \rangle
\]
A local computation at location $\ell$:

\[ \langle A; \psi \rangle \mapsto_\ell \langle T; \psi' \rangle \]

- $A$ is a set of actuator facts, introduced by the rewrite state $St$
- $T$ is a set of trigger facts, produced by the above local computation

\[ \mathcal{P} \triangleright \langle St, [\ell]A; \psi, [\ell]\psi \rangle \mapsto \langle St, [\ell]T; \psi, [\ell]\psi' \rangle \]

- Entire computation must be happen at $\ell$
Outline

1. Introduction
2. Example
3. Semantics
4. Compilation
5. Status
6. Conclusions & Future Work
Compilation of CoMingle Programs

System-centric specification
- High-level, concise
- Allows distributed events

rule pSwap :: [X]\text{swap}(Y,Z),
|   [[X]\text{item}(I)] \text{I} \rightarrow \text{Is}, |
|   [[Y]\text{item}(J)] \text{J} \rightarrow \text{Js}, |
|   [[Z]\text{item}(K)] \text{K} \rightarrow \text{Ks} |
| \rightarrow [X]\text{display}(\text{Msg},\text{size}(\text{Js}),Y), [[X]\text{item}(J)] \rightarrow \text{Js}, |
| [Y]\text{display}(\text{Msg},\text{size}(\text{Ks}),Z), [[Y]\text{item}(K)] \rightarrow \text{Ks}, |
| [Z]\text{display}(\text{Msg},\text{size}(\text{Is}),X), [[Z]\text{item}(I)] \rightarrow \text{Is} |

where \( \text{Msg} = \text{"%s from %s"} \).
### Compilation of CoMingle Programs

<table>
<thead>
<tr>
<th>System-centric specification</th>
<th>Node-centric specification</th>
</tr>
</thead>
<tbody>
<tr>
<td>- High-level, concise</td>
<td>- Match facts within a node</td>
</tr>
<tr>
<td>- Allows distributed events</td>
<td>- Handles lower-level concurrency</td>
</tr>
<tr>
<td></td>
<td>- Synchronization</td>
</tr>
<tr>
<td></td>
<td>- Progress</td>
</tr>
<tr>
<td></td>
<td>- Atomicity and Isolation</td>
</tr>
</tbody>
</table>

Choreographic Transformation ↓ [Lam and Cervesato, 2013]
Compilation of CoMingle Programs

System-centric specification
- High-level, concise
- Allows distributed events

**Choreographic Transformation**  
[Lam and Cervesato, 2013]

Node-centric specification
- Match facts within a node
- Handles lower-level concurrency
  - Synchronization
  - Progress
  - Atomicity and Isolation

**Imp erative Compilation**  
[Lam and Cervesato, 2014]

Low-level imperative compilation
- Java code
- Low-level network calls
- Operationalize multiset rewriting
- Trigger and actuator interfaces
Outline

1. Introduction
2. Example
3. Semantics
4. Compilation
5. Status
6. Conclusions & Future Work
Implementation

- Prototype Available at
  
  https://github.com/sllam/comingle

- Networking over Wifi-Direct, NFC and LAN
  - Bluetooth (LE) coming soon

- Proof-of-concept apps
  - Drag Racing — Racing cars across mobile devices
  - Battleship — Traditional maritime war game, multi-party
  - Wifi-Direct directory — Maintaining IP table for Wifi-Direct
  - Musical shares — Bounce a musical piece between devices
  - Swarble — Real-time team-based scrabble
  - Mafia — Traditional party game, with a mobile twist

- See tech.report [Lam and Cervesato, 2015] for details!
Drag Racing

- Inspired by Chrome Racer ([www.chrome.com/racer](http://www.chrome.com/racer))
- Race across a group of mobile devices
- Decentralized communication (over Wifi-Direct)
Implementing Drag Racing in CoMingle

```
rule init :: [I]initRace(Ls)  
    --o {[A]next(B)|(A,B)<-Cs}, [E]last(),  
        {[I]has(P), [P]all(Ps), [P]at(I), [P]rendTrack(Ls) | P<-Ps}  
where (Cs,E) = makeChain(I,Ls), Ps = list2mset(Ls).

rule start :: [X]all(Ps) \ [X]startRace() --o {[P].release()}|P<-Ps}.

rule tap :: [X]at(Y) \ [X]sendTap() --o [Y].recvTap(X).

rule trans :: [X]next(Z) \ [X].exiting(Y), [Y]at(X) --o [Z].has(Y), [Y].at(Z).

rule win :: [X]last() \ [X]all(Ps), [X].exiting(Y) --o {[P].decWinner(Y) | P <- Ps}.
```

- + 862 lines of properly indented Java code
- 700++ lines of local operations (e.g., display and UI operations)
- < 100 lines for initializing CoMingle runtime
DEMO
Outline

1. Introduction
2. Example
3. Semantics
4. Compilation
5. Status
6. Conclusions & Future Work
Conclusion

- **CoMingle**: Distributed logic programming language
  - For programming distributed mobile applications
  - Based on decentralized multiset rewriting with comprehension patterns

- **Prototype implementation**
  - Available at [https://github.com/sllam/comingle](https://github.com/sllam/comingle)
  - Example apps available for download as well
  - *Show your support, please STAR CoMingle GitHub repository!*
Future Work

- Front end refinements
  - Additional primitive types
  - More syntactic sugar
  - Refine Java interfaces

- Incremental extensions
  - Additional networking middlewares (Bluetooth, Bluetooth LE)
  - Sensor abstraction in CoMingle (GPS, speedometer, etc)
  - More platforms (iOS, Raspberry Pi, Arduino, backend servers)

- Beyond toy applications
  - Augmenting event/conference applications
  - Social interactive mobile applications
  - Home automation
  - Suggestions? *We are interested in hearing from you!*
Questions?
(Backup Slides)
Operational Semantics
Matching Judgment: \( \overline{H} \triangleq_{\text{lhs}} St \)
- Matches rule left-hand side \( \overline{H} \) against rewriting state \( St \)

\[
\begin{align*}
\frac{\overline{H} \triangleq_{\text{lhs}} St \quad H \triangleq_{\text{lhs}} St'}{\overline{H}, H \triangleq_{\text{lhs}} St, St'} & \quad \frac{\emptyset \triangleq_{\text{lhs}} \emptyset}{F \triangleq_{\text{lhs}} F} \\
\frac{\overline{t}/\overline{x}f \triangleq_{\text{lhs}} F \quad \models \overline{t}/\overline{x}g \quad \ell f \mid g \bigcap_{x \in \ell ts} \triangleq_{\text{lhs}} St}{\ell f \mid g \bigcap_{x \in \emptyset, ts} \triangleq_{\text{lhs}} St, F} & \quad \frac{\ell f \mid g \bigcap_{x \in \emptyset} \triangleq_{\text{lhs}} \emptyset}{\ell f \mid g \bigcap_{x \in \emptyset} \triangleq_{\text{lhs}} \emptyset}
\end{align*}
\]
Rewriting Runtime: Semantics of Matching

- Residual Non-matching: $\overline{H} \triangleleft_{\text{lhs}} St$
  - Checks that $\overline{H}$ matches nothing (else) in $St$
  - Ensures maximality

\[
\begin{align*}
\overline{H} \triangleleft_{\text{lhs}} St & \quad H \triangleleft_{\text{lhs}} St \\
\hline
\overline{H}, H \triangleleft_{\text{lhs}} St & \quad \emptyset \triangleleft_{\text{lhs}} St & \quad F \triangleleft_{\text{lhs}} St
\end{align*}
\]

\[
\begin{align*}
F \not\sqsubseteq_{\text{lhs}} \not\exists f \mid g \forall x \in ts \quad & \not\exists f \mid g \forall x \in ts \triangleleft_{\text{lhs}} St \\
\hline
\not\exists f \mid g \forall x \in ts \triangleleft_{\text{lhs}} St, F & \quad \not\exists f \mid g \forall x \in ts \triangleleft_{\text{lhs}} \emptyset
\end{align*}
\]

Subsumption: $F \sqsubseteq_{\text{lhs}} \not\exists f \mid g \forall x \in ts$ iff $F = \theta f$ and $\models \theta g$ for some $\theta = [t/x]$
Rewriting Runtime: Rewriting Semantics

- Unfolding rule body: $\overline{B} \gg_{\text{rhs}} St$
  - Expands $\overline{B}$ into $St$

$$
\overline{B} \gg_{\text{rhs}} St \quad B \gg_{\text{rhs}} St' \\
\overline{B}, B \gg_{\text{rhs}} St, St' \\
\emptyset \gg_{\text{rhs}} \emptyset \\
F \gg_{\text{rhs}} F
$$

$$
\models [\overline{t/\overline{x}}]g \quad [t/\overline{x}]b \gg_{\text{rhs}} F \\
\models b \mid g\sum\overline{x} \in \overline{t},ts \gg_{\text{rhs}} St \\
\models b \mid g\sum\overline{x} \in \overline{t},ts \gg_{\text{rhs}} F, St
$$

$$
\not\models [\overline{t/\overline{x}}]g \\
\not\models b \mid g\sum\overline{x} \in \overline{t},ts \gg_{\text{rhs}} St \\
\models b \mid g\sum\overline{x} \in \emptyset \gg_{\text{rhs}} \emptyset
$$
## Introduction

### Example

### Semantics

### Compilation

### Status

### Conclusions

---

**Wifi-Direct P2P Directory**
Wifi P2P Directory

- Wifi Direct APIs in the Android SDK
  - Enable “easy” setup of a mobile ad-hoc network
  - One device act as the owner \((O)\)
  - Others are members \((M)\)
  - But only takes you half-way: Each \(M\) has IP of \(O\) only
Wifi P2P Directory

- Wifi P2P Directory program
  - Maintains a dynamic IP directory on each node
  - Implements a daemon on each $M$ to receive updates from $O$
  - Implements a daemon on $O$ that broadcasts updates to each $M$
Implemented in CoMingle within each CoMingle App
- P2P Directory bootstrapped into CoMingle initialization
- Runs in the background as a separate CoMingle runtime instance
Implementing P2P Directory in CoMingle

```
rule owner :: [O]startOwner(C) --o [O]owner(C), [O]joined(O).
rule member :: [M]startMember(C) --o [M]member(C).

rule connect :: [M]member(C) \ [M]connect(N) --o [O]joinRequest(C,N,M)
    where O = ownerLoc().

rule join :: [O]owner(C), {[O]joined(M')|M'->Ms} \ [O]joinRequest(C,N,M) | notIn(M,Ms)
    --o {[M']_added(D)|M'<-Ms}, {[M]_added(D')|D'<-Ds},
        [M]_added(D), [O]joined(M), [M]_connected()
    where IP = lookupIP(M), D = (M,IP,N), Ds = retrieveDir().


rule quitM :: {[O]joined(M')|M'->Ms.not(M' = M)} \ [M]member(C), [M]quit(), [O]joined(M)
    --o {[M']_removed(M)|M'<-Ms}, [M]_deleteDir().
```

- Two implementations of P2P Directory
  - “Vanilla” Java + Android SDK: 694 lines of Java code
  - CoMingle + Java + Android SDK: 53 lines of CoMingle code + 154 lines of Java code

- All code is properly indented

- Omitting common libraries used by both implementations
Bibliography

Cervesato, I. and Lam, E. S. (2015).
Modular Multiset Rewriting.
In *LPAR 2015*, Suva, Fiji.

A linear logic programming language for concurrent programming over graph structures.
*CoRR*, abs/1405.3556.

*Constraint Handling Rules: Compilation, Execution and Analysis.*
ISBN 9783839115916. BoD.

Decentralized Execution of Constraint Handling Rules for Ensembles.
In *PPDP’13*, pages 205–216, Madrid, Spain.

Optimized Compilation of Multiset Rewriting with Comprehensions.

Comingle: Distributed Logic Programming for Decentralized Android Applications.

Programmable Orchestration of Time-Synchronized Events Across Decentralized Android Ensembles.
In *WiMob’15*, Abu Dhabi, UAE.