Overlap and Independence in Multiset Comprehension Patterns

Edmund S.L. Lam
sllam@qatar.cmu.edu

Iliano Cervesato
iliano@cmu.edu

Supported by QNRF grants NPRP 4-1593-1-260 and 4-341-1-059

الصندوق القطري لرعاية البحث العلمي
Qatar National Research Fund
Member of Qatar Foundation

Outline

Comingle

A programming language for distributed mobile apps

- Designed to implement mobile apps that run across Android devices
- Enables high-level system-centric abstraction
  - **specifies** distributed computations as *one* declarative program
  - **compiles** into node-centric fragments, executed by each node
- Typed multiset rewriting with
  - decentralization
  - comprehension patterns
  - time synchronization
  - modularity
- Declarative, concise, roots in linear logic

Comingle

A programming language for distributed mobile apps

- Designed to implement mobile apps that run across Android devices
- Enables high-level system-centric abstraction
  - **specifies** distributed computations as *one* declarative program
  - **compiles** into node-centric fragments, executed by each node
- Typed multiset rewriting with
  - decentralization
  - comprehension patterns
  - time synchronization
  - modularity
- Declarative, concise, roots in linear logic

Example: Swap Data between X and Y up to Threshold P

Example: Swap Data between X and Y up to Threshold P

- In math:

$$\begin{array}{rcccl} & & [X]swap(Y,P) & & \\ pivotSwap & : & \wr[X]item(D) \mid D \geq P\int_{D\to Xs} & \multimap & \wr[Y]item(D)\int_{D\leftarrow Xs} \\ & & \wr[Y]item(D) \mid D \leq P\int_{D\to Ys} & & \wr[X]item(D)\int_{D\leftarrow Ys} \end{array}$$

Example: Swap Data between X and Y up to Threshold P

- In math:

$$
\mathit{pivotSwap} : \begin{array}{l} [X]\mathit{swap}(Y,P) \\ \wr [X]\mathit{item}(D) \mid D \geq P \int_{D \rightarrow Xs} \\ \wr [Y]\mathit{item}(D) \mid D \leq P \int_{D \rightarrow Ys} \end{array} \multimap \begin{array}{l} \wr [Y]\mathit{item}(D) \int_{D \leftarrow Xs} \\ \wr [X]\mathit{item}(D) \int_{D \leftarrow Ys} \end{array}
$$

- In code:

```
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".
```

Example: pivotSwap Execution

```
[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: $n1$	Node: $n2$	Node: $n3$
$s(n2, 5), i(4), i(6), i(8)$	$i(3), i(20)$	$s(n2, 10), i(18)$

Example: pivotSwap Execution

```
[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: $n1$		Node: $n2$	Node: $n3$
$s(n2, 5)$, $i(4)$, $i(6)$, $i(8)$	⟶	$i(3)$, $i(20)$	$s(n2, 10)$, $i(18)$

Example: pivotSwap Execution

```
[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: $n3$
$s(n2, 10), i(18)$

Node: $n1$
$d(\text{"1 from n2"})$
$i(3), i(4)$

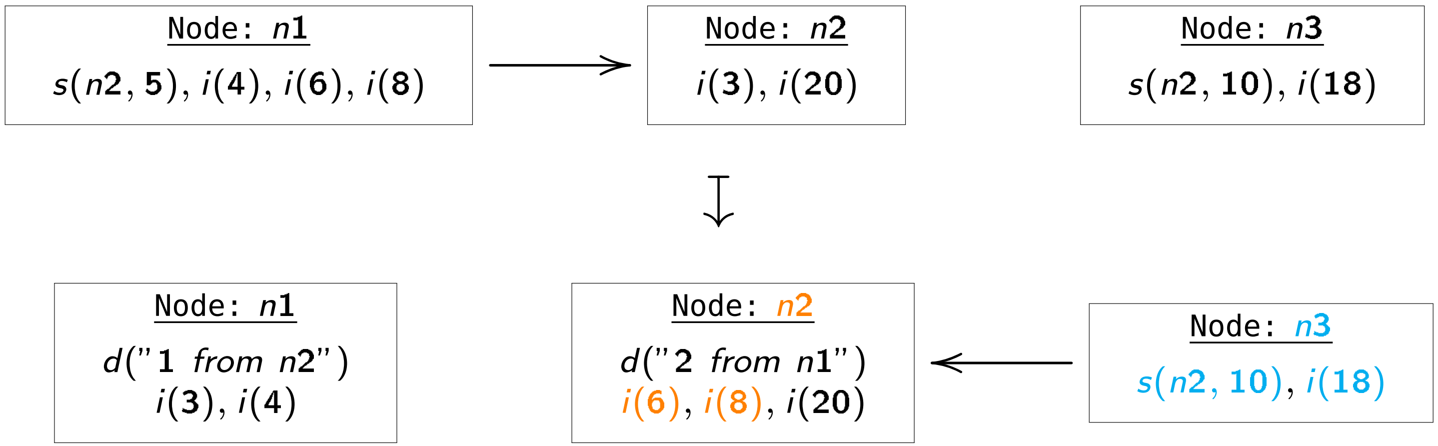
Node: $n2$
$d(\text{"2 from n1"})$
$i(6), i(8), i(20)$

Node: $n3$
$s(n2, 10), i(18)$

Example: pivotSwap Execution

```
[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".
```

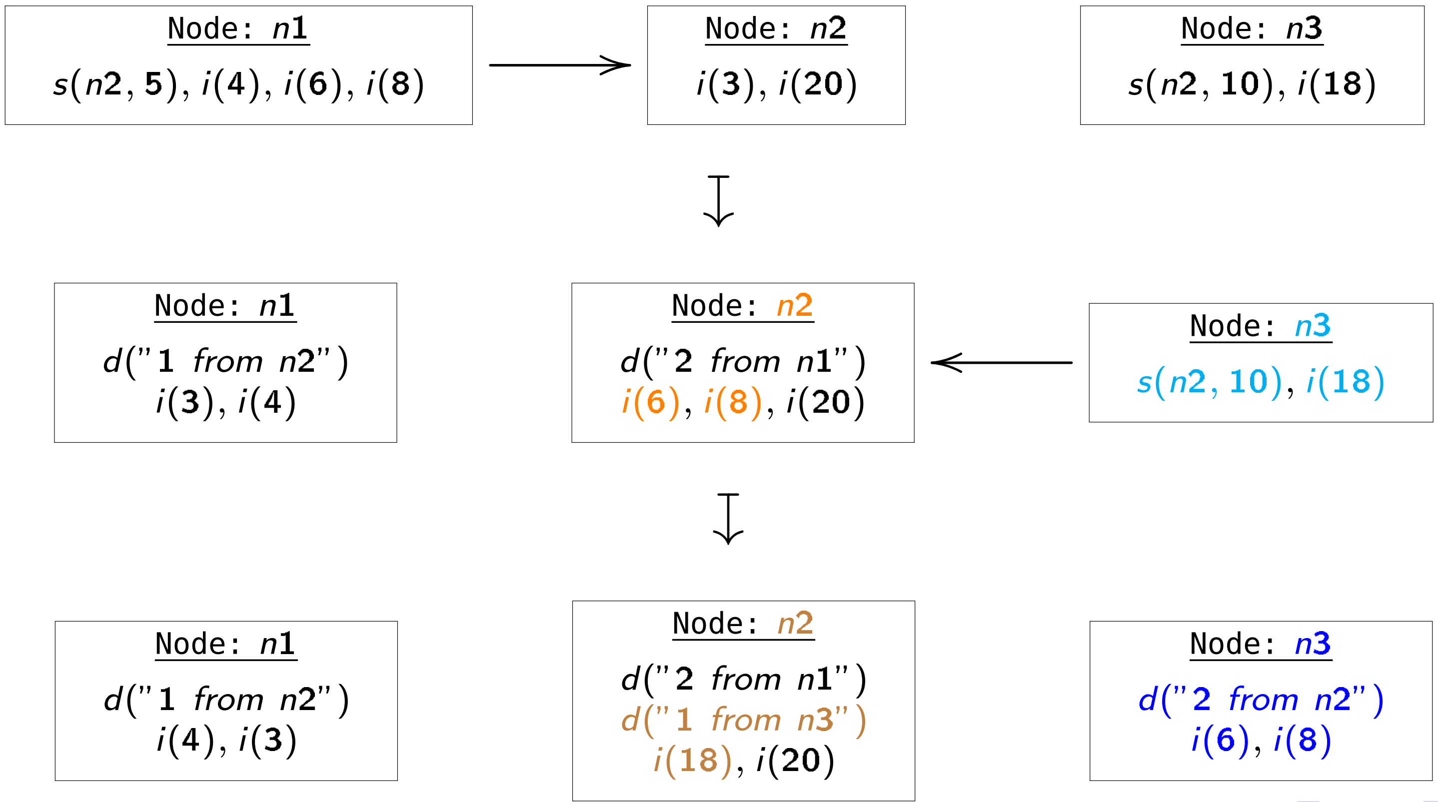
Let s = swap, i = item and d = display



Example: pivotSwap Execution

```
[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



Try it Yourself!

- Download from

  https://github.com/sllam/comingle

  Show your support, please STAR Comingle GitHub repository!

- Networking over Wifi-Direct, NFC, LAN and Bluetooth
  - support for drop-in/drop-out

- 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
  - *Swarbble* — Real-time team-based scrabble
  - *Mafia* — Traditional party game, with a mobile twist
  - *CoDoodle* — Interactive presentation tool

Outline

Syntax

(A Comingle program $\mathcal{P}$ is a set of rules $r : \bar{E} \mid g \multimap B$ where B is also a multisets of expressions; we are also ignoring locations and types)

- A ***head pattern*** $\bar{E} \mid g$ consists of
  - a multiset of expressions $\bar{E}$
  - a Boolean ***guard*** g
- An ***expression*** E is either
  - a fact: $p(\vec{t})$
  - a comprehension: $\wr p(\vec{t}) \mid g \int_{\vec{x} \to T}$
    - Multiset of *all* $p(\vec{t})$ in the state that satisfy g
    - $\vec{x}$ bound in g and $\vec{t}$
    - ***Comprehension range*** T is the multiset of all bindings $\vec{x}$

Matching Semantics

- A ***state*** St is a multiset of ground facts
- ***Matching*** a head pattern $H = \bar{E} \mid g$ against a state St with residual St^-:

$$St \xrightarrow{H} St^-$$

Holds if $St = St^+, St^-$ and there is a ground substitution θ such that

  - $\theta\bar{E}$ matches St^+
  - St^- does <u>not</u> match any comprehension in $\theta\bar{E}$
  - θg is valid

$$\frac{\theta\bar{E} \triangleq_{\text{head}} St^+ \quad \theta\bar{E} \triangleq^{\neg}_{\text{head}} St^- \quad \models \theta g}{St^+, St^- \xrightarrow{\bar{E}|g} St^-}$$

Comprehensions in $\bar{E} \mid g$ match *maximal* portions of St

Pattern Interactions

When do two head patterns interfere with each other?

Useful for

- debugging
- implementation
- reasoning
- cost analysis

Interference?

- One's consumed facts may prevent the other from being applicable
- ... possibly concurrently

Overlap and Independence

$H_1 = \bar{E}_1 \mid g_1$ and $H_2 = \bar{E}_2 \mid g_2$ without variables in common

- ***overlap*** if there is a state St such that
  - $St \overset{H_1}{\rightarrowtail} St_1$ for some St_1 and
  - $St \overset{H_2}{\rightarrowtail} St_2$ for some St_2,

  but there is no St' such that $St \overset{H_1 \| H_2}{\rightarrowtail} St'$.

  E.g., $H_1 = p(a, X), q(X)$ and $H_2 = p(Y, Y), r(Z)$

  Take $St = (a, a), q(a), r(b)$

- are ***independent*** if they don't overlap

  E.g., H_1 and $H_2' = p(b, Y), r(Z)$

Are there algorithmic criteria?

Outline

Case: Plain Multisets

$H = \bar{F}$: empty guard and no comprehensions

H_1 and H_2 ***overlap*** iff one contains a fact unifiable in the other:

- $H_1 = p(\vec{t_1}), \bar{F}'_1$
- $H_2 = p(\vec{t_2}), \bar{F}'_2$
- there is θ such that $\theta\vec{t_1} = \theta\vec{t_2}$

Notes:

- $p(\vec{t_1})$ and $p(\vec{t_2})$ may not be unique
- Polynomial complexity . . . for well-behaved term languages
- Implemented using term-language unification

Case: Guarded Multisets

$H = \bar{F} \mid g$: no comprehensions — *found in most rule-based languages*

H_1 and H_2 ***overlap*** iff

- $H_1 = p(\vec{t_1}), \bar{F}_1'$
- $H_2 = p(\vec{t_2}), \bar{F}_2'$
- there is θ such that $\theta\vec{t_1} = \theta\vec{t_2}$ and $\models \theta g_1$ and $\models \theta g_2$

Examples:

- $H_1 = p(X) \mid X > 3$ and $H_2 = p(Y) \mid Y < 10$ overlap
  E.g., in state $p(7)$
- H_1 and $H_2' = p(Y) \mid Y < 3$ are independent

Implementation: compute unifiers θ for $p(\vec{t_1})$ and $p(\vec{t_2})$, and then pass θg_1 and θg_2 to an SMT solver

Case: Open-ended Multisets

$H = \bar{E} \mid g$: comprehension ranges is never used

- $p(X), \wr p(x) \mid x > 0 \int_{x \to Xs}$
- but not $p(X), \wr p(x) \mid x > 0 \int_{x \to Xs} \mid size(Xs) = 0$

H_1 and H_2 ***overlap*** exactly as in last case!

- Open-ended comprehensions can never fail
- At most return the empty multiset

  Consider $H_1 = p(X)$ and $H_2 = \wr p(x) \int_{x \to Xs}$:

  - $p(a) \xrightarrow{H_1} \varnothing$
  - $p(a) \xrightarrow{H_2} \varnothing$
  - $p(a) \xrightarrow{H_1 \| H_2} \varnothing$ because $\varnothing \xrightarrow{H_2} \varnothing$

General Case

Unsolved!

General Case

Unsolved!

$$H_1 = \wr p(x) \int_{x \to Xs}, q(Y) \mid Y \in Xs \qquad \textit{and} \quad H_2 = p(Z)$$

are *overlapping*:

- Succeed separately on $St = p(a), q(a)$
- Composition fails as guard of H_2 fails

General Case

Unsolved!

$H_1 = \wr p(x) \int_{x \to Xs}, q(Y) \mid Y \in Xs$ *and* $H_2 = p(Z)$

are *overlapping*:

- Succeed separately on $St = p(a), q(a)$
- Composition fails as guard of H_2 fails

But

$H_1 = \wr p(x) \mid x < 3 \int_{x \to Xs}, q(Y) \mid Y \in Xs$ *and* $H_2 = p(Z) \mid Z > 5$

are *independent*:

- because no fact $p(n)$ can match both patterns

General Case

$$H_1 = \wr p(x) \int_{x \rightarrow Xs} \mid size(Xs) > 0 \qquad and \quad H_2 = p(Z)$$

General Case

$H_1 = \wr p(x) \int_{x \to Xs} \mid size(Xs) > 0$ *and* $H_2 = p(Z)$

are *overlapping*:

- Succeed separately on $St = p(a)$
- Composition fails as Xs set to $\varnothing$, violating guard

General Case

$$H_1 = \wr p(x) \int_{x \to Xs} \mid size(Xs) > 0 \qquad and \quad H_2 = p(Z)$$

are *overlapping*:

- Succeed separately on $St = p(a)$
- Composition fails as Xs set to $\varnothing$, violating guard

$$H_1 = \wr p(x) \int_{x \to Xs} \mid size(Xs) \leq 8 \qquad and \quad H_2 = p(Z)$$

General Case

$$H_1 = \wr p(x) \int_{x \to Xs} \mid size(Xs) > 0 \qquad and \quad H_2 = p(Z)$$

are *overlapping*:

- Succeed separately on $St = p(a)$
- Composition fails as Xs set to $\varnothing$, violating guard

But

$$H_1 = \wr p(x) \int_{x \to Xs} \mid size(Xs) \leq 8 \qquad and \quad H_2 = p(Z)$$

are *independent*:

- because it has an upper bound on the comprehension range, not a lower bound

General Case

$$H_1 = \wr p(x) \int_{x \to Xs} \mid size(Xs) > 0 \qquad \textit{and} \quad H_2 = p(Z)$$

are *overlapping*:

- Succeed separately on $St = p(a)$
- Composition fails as Xs set to $\varnothing$, violating guard

But

$$H_1 = \wr p(x) \int_{x \to Xs} \mid size(Xs) \leq 8 \qquad \textit{and} \quad H_2 = p(Z)$$

are *independent*:

- because it has an upper bound on the comprehension range, not a lower bound

Negation-as-absence:

$$H_1 = \wr p(x) \int_{x \to Xs} \mid size(Xs) = 0 \qquad \textit{and} \quad H_2 = p(Z)$$

General Case

$$H_1 = \wr p(x) \int_{x \to Xs}, \wr q(y) \int_{y \to Xs} \quad \textit{and} \quad H_2 = p(Z)$$

are *overlapping*:

- Succeed separately on $St = p(a), q(a)$
- Composition fails

General Case

$$H_1 = \wr p(x) \int_{x \to Xs}, \wr q(y) \int_{y \to Xs} \quad \textit{and} \quad H_2 = p(Z)$$

are *overlapping*:

- Succeed separately on $St = p(a), q(a)$
- Composition fails

$$H_1 = \wr p(x) \int_{x \to Xs}, \wr q(y) \mid y \in Xs \int_{y \to Ys} \quad \textit{and} \quad H_2 = p(Z)$$

are *independent*:

- because it filters out values for Ys rather than requiring that some terms be present

Outline

Future Work

Lots more work to be done!

Questions?

Comingle Example: Drag Racing

- Inspired by Chrome Racer (www.chrome.com/racer)
- Race across a group of mobile devices
- Purely local communications

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 run-time