XXII MFPS, 24-27 May 2006, Genova

Simplification and Computation

Eugenio Moggi

moggi@disi.unige.it

DISI, Univ. of Genova

Summary

- AIM: structuring operational semantics to separate computational effects from other (programming language) features
- Preliminary discussion
  - monadic approach to denotational semantics
  - what kind of operational semantics? TS vs LTS
- **General approach**
  - simplification: confluent term rewriting, referential transparency
  - computation: configurations, computational effects
- A concrete proposal
  - PMC [Kah03]: pattern matching calculus
  - CHAM [BB92] and Join calculus [FG96,FG02]: configurations as multi-sets of terms (and computation rules)
- Encodings: expected properties, some examples
- Conclusions and issues

Monadic approach in a nutshell

Traditional approach to denotational semantics

- $[\![-]\!]_o : PL \longrightarrow \mathcal{C}$ interpretation

  PL programming language

  $\mathcal{C}$ category (with suitable properties and additional structure)

Monadic approach to denotational semantics factors $[\![-]\!]_o$ into

- $(-) : PL \longrightarrow ML_M$ compositional translation

  ML_M monadic metalanguage – better **separation of mathematical concerns**

  - ML internal language for category $\mathcal{C}$
  - M computational types syntax for monad (or related notions)

- $[\![-]\!] : ML_M \longrightarrow (\mathcal{C}, M)$ **standard interpretation** parametric w.r.t. monad

  $\mathcal{C}$ category with universal properties, M additional structure (monad)

What kind of operational semantics?

- SOS [Plo81] based on inference rules for deriving operational judgments

  too general to suggest common patterns and points of variation

- Labeled Transition Systems (LTS) $s \overset{l}{\longmapsto} s'$

  describe potential interaction of open system with external environment

- Transition Systems (TS) $s \longmapsto s'$ describe potential evolution of closed system

  open system + environment = closed system

- TS vs LTS: TS are preferable to specify observational equivalence $\approx$ on program fragments (e.g. see work on HO π-calculus [San93])

  $\approx$ as congruence induced by basic observations on closed system

- Other (ignored) issues:
  - beyond non-determinism: probabilistic, stochastic and hybrid systems

    functorial operational semantics based on co-algebras [Tur96]
  - static guarantees: operational semantics specified independently

- REPLACE category (for denotational sem.) with TS (for operational sem.)

General approach: simplification and computation

- Distinguish atoms from variables – FreshML [GP99,SGP03]
- Terms e – $\mathrm{A}(e)$ and $\mathrm{FV}(e)$ denote set of atoms and free variables of e
- SIMPLIFICATION is a relation $e \longrightarrow e'$ on terms
  - preserving atoms and free variables, i.e. $\mathrm{A}(e') \subseteq \mathrm{A}(e)$ and $\mathrm{FV}(e') \subseteq \mathrm{FV}(e)$
  - confluent and compatible, i.e. can be applied in any order and any context
  - invariant w.r.t. permutations π of atoms and substitutions ρ of variables with terms, i.e. $\dfrac{e \longrightarrow e'}{e[\pi] \longrightarrow e'[\pi]}$ $\quad \dfrac{e \longrightarrow e'}{e[\rho] \longrightarrow e'[\rho]}$

General approach: simplification and computation

- Distinguish atoms from variables – FreshML [GP99,SGP03]
- Terms e – $\mathrm{A}(e)$ and $\mathrm{FV}(e)$ denote set of atoms and free variables of e
- SIMPLIFICATION is a relation $e \longrightarrow e'$ on terms
- Configurations s built from terms – $\mathrm{A}(s)$ set of atoms of s (no free variables)

  thus simplification extends to configurations $s_1 \longrightarrow s_2$
- COMPUTATION is a relation $s_1 \longmapsto s_2$ on configurations
  - invariant w.r.t. permutations π of atoms
  - preserved by simplification, i.e.

$$\begin{array}{ccc} s_1 & \longmapsto & s_2 \\ {*}\downarrow & & \downarrow{*} \\ s_1' & \longmapsto & s_2' \end{array}$$

General approach: simplification and computation

- Distinguish atoms from variables – FreshML [GP99,SGP03]
- Terms e – $\mathrm{A}(e)$ and $\mathrm{FV}(e)$ denote set of atoms and free variables of e
- SIMPLIFICATION is a relation $e \longrightarrow e'$ on terms
- Configurations s built from terms – $\mathrm{A}(s)$ set of atoms of s (no free variables)

  thus simplification extends to configurations $s_1 \longrightarrow s_2$
- COMPUTATION is a relation $s_1 \longmapsto s_2$ on configurations

Simplification supports referential transparency, thus

- **suitable for pure functional languages (PFL), typed calculi for proof assistants**
- **implementable using PFL techniques: lazy evaluation, graph reduction**

Computation induces TS on configurations modulo simplification – CHAM [BB92].
In *reduction semantics* configurations are terms.

A concrete proposal – terms and simplification

- atom $a \in \mathsf{A}$, name variable y, Name $u \in \mathsf{N} ::= a \mid y$
- term variable x, pattern p, Term $e \in \mathsf{E}$ – related to PMC [Kah03]

  $p \quad ::= \quad ?x \mid u\ \overline{p} \mid ?y\ \overline{p}$ $\quad u$ matches only itself, $?y$ matches any u

  atoms $\mathrm{A}(p)$, declared variables $\mathrm{DV}(p)$ and free (name) variables $\mathrm{FV}(p)$ of p

p	$\mathrm{A}(p)$	$\mathrm{DV}(p)$	$\mathrm{FV}(p)$
$?x$		x	
$u\ \overline{p}$	$\mathrm{A}(u, \overline{p})$	$\mathrm{DV}(\overline{p})$	$\mathrm{FV}(u, \overline{p})$
$?y\ \overline{p}$	$\mathrm{A}(\overline{p})$	$y, \mathrm{DV}(\overline{p})$	$\mathrm{FV}(\overline{p}) - y$
$p\ \overline{p}$	$\mathrm{A}(p, \overline{p})$	$\mathrm{DV}(p, \overline{p})$	$\mathrm{FV}(p), \mathrm{FV}(\overline{p}) - \mathrm{DV}(p)$

  - linearity: $?x$ and $?y$ can be declared at most once
  - binding: the occurrences of y on the left of $?y$ are bound

A concrete proposal – terms and simplification

- atom $a \in \mathsf{A}$, name variable y, Name $u \in \mathsf{N} ::= a \mid y$
- term variable x, pattern p, Term $e \in \mathsf{E}$ – related to PMC [Kah03]

$p \quad ::= \quad ?x \mid u\,\overline{p} \mid ?y\,\overline{p}$ $\quad u$ matches only itself, $?y$ matches any u

$e \quad ::= \quad x \mid u\,\overline{e} \mid ok\ e \mid fail$ $\quad$ constructor u applied to sequence of terms

$\quad\quad \mid \quad (p \Rightarrow e_1 | e_2) \mid e_1 @ e_2 \mid e_1 : p \Rightarrow e_2 \mid (e_1 ; e_2) \mid$ $\quad$ PMC [Kal03]

$\quad\quad \mid \quad$ let $\{x_i = e_i | i \in n\}$ in e $\quad$ binding for mutual recursive definitions

e	$\mathrm{FV}(e)$
$(p \Rightarrow e_1 \| e_2)$	$\mathrm{FV}(p), \mathrm{FV}(e_1) - \mathrm{DV}(p), \mathrm{FV}(e_2)$
$e_1 : p \Rightarrow e_2$	$\mathrm{FV}(e_1), \mathrm{FV}(p), \mathrm{FV}(e_2) - \mathrm{DV}(p)$
$\ldots$	$\ldots$

A concrete proposal – terms and simplification

- atom $a \in \mathsf{A}$, name variable y, Name $u \in \mathsf{N} ::= a \mid y$
- term variable x, pattern p, Term $e \in \mathsf{E}$ – related to PMC [Kah03]

$$p \quad ::= \quad ?x \mid u\,\overline{p} \mid ?y\,\overline{p} \qquad u \text{ matches only itself, } ?y \text{ matches any } u$$

$$\begin{array}{lll} e & ::= & x \mid u\,\overline{e} \mid ok\ e \mid fail \qquad \text{constructor } u \text{ applied to sequence of terms} \\ & \mid & (p{\Rightarrow}e_1|e_2) \mid e_1@e_2 \mid e_1\colon p{\Rightarrow}e_2 \mid (e_1; e_2) \mid \qquad \text{PMC [Kal03]} \\ & \mid & \text{let } \{x_i = e_i | i \in n\} \text{ in } e \qquad \text{binding for mutual recursive definitions} \end{array}$$

- Simplification induced by left-linear and non-overlapping rewrite rules

$$\begin{array}{rcl} (p{\Rightarrow}e_1|e_2)@e & \longrightarrow & (e\colon p{\Rightarrow}ok\ e_1; e_2@e) \\ (ok\ e; e') & \longrightarrow & e \\ (fail; e') & \longrightarrow & e' \\ e\colon ?x{\Rightarrow}e' & \longrightarrow & e'[x\colon e] \\ u\,\overline{e}\colon ?y\,\overline{p}{\Rightarrow}e' & \longrightarrow & \overline{e}\colon \overline{p}[y\colon u]{\Rightarrow}e'[y\colon u] \quad \text{when } |\overline{e}| = |\overline{p}| \\ a\,\overline{e}\colon a\,\overline{p}{\Rightarrow}e' & \longrightarrow & \overline{e}\colon \overline{p}{\Rightarrow}e' \quad \text{when } |\overline{e}| = |\overline{p}| \\ \text{let } \{x_i = e_i|i \in n\} \text{ in } e & \longrightarrow & e[x_i\colon \text{let } \{x_i = e_i|i \in n\} \text{ in } e_i|i \in n] \end{array}$$

A concrete proposal – terms and simplification

- atom $a \in \mathsf{A}$, name variable y, Name $u \in \mathsf{N} ::= a \mid y$
- term variable x, pattern p, Term $e \in \mathsf{E}$ – related to PMC [Kah03]

$$p ::= ?x \mid u\,\overline{p} \mid ?y\,\overline{p} \quad u \text{ matches only itself, } ?y \text{ matches any } u$$

$$\begin{array}{lll} e & ::= & x \mid u\,\overline{e} \mid ok\ e \mid fail \quad \text{constructor } u \text{ applied to sequence of terms} \\ & \mid & (p \Rightarrow e_1 | e_2) \mid e_1 @ e_2 \mid e_1 : p \Rightarrow e_2 \mid (e_1 ; e_2) \mid \quad \text{PMC [Kal03]} \\ & \mid & \text{let } \{x_i = e_i | i \in n\} \text{ in } e \quad \text{binding for mutual recursive definitions} \end{array}$$

- Simplification induced by left-linear and non-overlapping rewrite rules

$$\begin{array}{rcll} (v; e') & \longrightarrow & fail & \text{when } v \not\equiv ok\ e \mid fail \\ v @ e & \longrightarrow & fail & \text{when } v \not\equiv (p \Rightarrow e_1 | e_2) \\ v : ?y\,\overline{p} \Rightarrow e' & \longrightarrow & fail & \text{when } v \not\equiv u\,\overline{e} \text{ with } |\overline{e}| = |\overline{p}| \\ & \ldots & & \end{array}$$

$v ::= u\,\overline{e} \mid (p \Rightarrow e_1 | e_2)$ top-level unchanged by simplification or instantiation

A concrete proposal – terms and simplification

- atom $a \in \mathsf{A}$, name variable y, Name $u \in \mathsf{N} ::= a \mid y$
- term variable x, pattern p, Term $e \in \mathsf{E}$ – related to PMC [Kah03]

$$p ::= ?x \mid u\,\overline{p} \mid ?y\,\overline{p} \quad u \text{ matches only itself, } ?y \text{ matches any } u$$

$$\begin{array}{lll} e & ::= & x \mid u\,\overline{e} \mid ok\ e \mid fail \quad \text{constructor } u \text{ applied to sequence of terms} \\ & \mid & (p \Rightarrow e_1 | e_2) \mid e_1 @ e_2 \mid e_1 : p \Rightarrow e_2 \mid (e_1 ; e_2) \mid \quad \text{PMC [Kal03]} \\ & \mid & \text{let } \{x_i = e_i | i \in n\} \text{ in } e \quad \text{binding for mutual recursive definitions} \end{array}$$

- Simplification induced by left-linear and non-overlapping rewrite rules
- Examples of patterns
  - $p_0 \equiv c\ ?x$ (with $c \in \mathsf{A}$) matched by $c\ e$ for any $e \in \mathsf{E}$
  - $p_1 \equiv c\ ?y$ matched by $c\ a$ for any $a \in \mathsf{A}$
  - $p_2 \equiv ?y\ y$ matched by $a\ a$ for any $a \in \mathsf{A}$

A concrete proposal – terms and simplification

- atom $a \in \mathsf{A}$, name variable y, Name $u \in \mathsf{N} ::= a \mid y$
- term variable x, pattern p, Term $e \in \mathsf{E}$ – related to PMC [Kah03]

$$p \quad ::= \quad ?x \mid u\ \overline{p} \mid ?y\ \overline{p} \qquad u \text{ matches only itself, } ?y \text{ matches any } u$$

$$\begin{array}{lll} e & ::= & x \mid u\ \overline{e} \mid ok\ e \mid fail \qquad \text{constructor } u \text{ applied to sequence of terms} \\ & \mid & (p{\Rightarrow}e_1|e_2) \mid e_1@e_2 \mid e_1{:}\,p{\Rightarrow}e_2 \mid (e_1; e_2) \mid \qquad \text{PMC [Kal03]} \\ & \mid & \text{let } \{x_i = e_i | i \in n\} \text{ in } e \qquad \text{binding for mutual recursive definitions} \end{array}$$

- Simplification induced by left-linear and non-overlapping rewrite rules
- Examples of terms
  - test for equality of names $eq = (?y{\Rightarrow}(y{\Rightarrow}true|?y'{\Rightarrow}false|fail)|fail)$
  - term constructors as atoms, term destructors defined with let-binding, e.g. natural numbers: zero $z{:}\,N$ and successor $s{:}\,N \to N$ are atoms, iterator $it{:}\,X \to (X \to X) \to N \to X$ defined by recursion and pattern-matching let $it = (?x{\Rightarrow}?f{\Rightarrow}(z{\Rightarrow}x \mid s\ ?n{\Rightarrow}it@x@f@n \mid fail))$ in …

A concrete proposal – configurations and computation

- join pattern J – related to Join and Kell calculi [FG96,FG02,Ste03,BS03]

  $J \quad ::= \quad \{(u_i\, \overline{p}_i | i \in n)\}$ a multi-set of patterns $u\, \overline{p}$

  atoms, declared variables and free (name) variables of J define by union

  - weaken linearity: $?x$ can be declared in at most one $u\, \overline{p}$ of J
  - instantiation: $J\rho$ (J and ρ closed) is the *molecule* (multi-set of terms $u\, \overline{e}$) obtained by replacing the only occurrence of $?x$ in J with $\rho(x)$, and all occurrences of $?y$ and y in J with $\rho(y)$

  J in Join have restricted format $y\ ?y_1 \ldots ?y_n$: only linear name-matching

A concrete proposal – configurations and computation

- join pattern J – related to Join and Kell calculi [FG96,FG02,Ste03,BS03]

  $J \quad ::= \quad \{(u_i\ \overline{p}_i | i \in n)\}$ a multi-set of patterns $u\ \overline{p}$

- Computation rule $r \quad ::= \quad J > \nu\overline{y}.R|E \quad R|E$ multi-set of rules and terms

r	$\mathrm{FV}(r)$
$J > \nu\overline{y}.R\|E$	$\mathrm{FV}(J), \mathrm{FV}(R, E) - \overline{y} - \mathrm{DV}(J)$

A concrete proposal – configurations and computation

- join pattern J – related to Join and Kell calculi [FG96,FG02,Ste03,BS03]

  $J \quad ::= \quad \{(u_i\, \overline{p}_i | i \in n)\}$ a multi-set of patterns $u\, \overline{p}$
- Computation rule $r \quad ::= \quad J > \nu\overline{y}.R|E \quad R|E$ multi-set of rules and terms
- Configuration s = multi-set $R|E$ of closed terms and rules, i.e. $\mathrm{FV}(R,E) = \emptyset$
- Computation defined by name generation + multi-set rewriting

$$s \mid r \mid J\rho \quad \longmapsto \quad s \mid r \mid (R|E)[\overline{y}{:}\,\overline{a}, \rho] \quad \text{where } r \equiv J > \nu\overline{y}.R|E$$

ρ closed substitution for variables in $\mathrm{DV}(J)$ and $\overline{a}$ fresh for $s \mid r \mid J\rho$

  - Join/Kell rules are in BNF of terms: reflexive CHAM
  - r in Kell can be used only once: but replication allows indefinite reuse
  - Join has implicit set of locations (by partitioning A) and rules are located. One can impose syntactic restrictions on r to enforce this property.

A concrete proposal – configurations and computation

- join pattern J – related to Join and Kell calculi [FG96,FG02,Ste03,BS03]
  $J \quad ::= \quad \{(u_i\ \overline{p}_i | i \in n)\}$ a multi-set of patterns $u\ \overline{p}$
- Computation rule $r \quad ::= \quad J > \nu\overline{y}.R|E \quad R|E$ multi-set of rules and terms
- Configuration s = multi-set $R|E$ of closed terms and rules, i.e. $\mathrm{FV}(R,E) = \emptyset$
- Computation defined by name generation + multi-set rewriting
- Example of interpreter for imperative programs
  - interpreted atoms: . . .
    new: $X \to (RX \to MY) \to MY$ initialize new reference with a value
    get: $RX \to (X \to MY) \to MY$ get value store in reference
  - atoms *prg*: MY and *str*: RX, X for program and store
  - computation rules for imperative programs:
    prg $($*new* $?x\ ?k) > \nu y.$*prg* $(k@y) \mid$ *str* $y\ x$
    prg $($*get* $?y\ ?k) \mid$ *str* $?y\ ?x >$ *prg* $(k@x) \mid$ *str* $y\ x$

Encodings – general ideas

Direct approach to operational semantics

- PL programming language – syntax
- $(S_o, \vdash_o\!\!\rightarrow)$ transition system – small-step operational semantics

  configurations in S_o involve programs in PL and other stuff
- basic observations on configurations – ignored for simplicity

Operational semantics via encoding

- $(-): PL \longrightarrow \mathsf{E}$ compositional encoding of programs (and other syntactic categories) into terms
- $\mathbf{R} \subset S_o \times S$ surjective *bisimulation* between $(S_o, \vdash_o\!\!\rightarrow)$ and

  $(S, \xrightarrow{*} \longmapsto \xrightarrow{*})$

Encoding of monadic metalanguage ML_M (with references) [MF03]

- Syntax of monadic metalanguage – term $M \in \mathsf{E}_o$

  $M ::= x \mid \lambda x.M \mid M_1 M_2 \mid$ *ret* $M \mid$ *do* $M_1\ M_2 \mid$ *new* $M \mid$ *get* $M \mid$ *set* $M_1\ M_2 \mid a$

  $S ::=$ *none* $\mid$ *push* $M\ S$ – references a and control stacks S are instrumental
- Simplification for ML_M is β-reduction: $(\lambda x.M_1)M_2 \longrightarrow M_1[x\colon M_2]$
- Translation $(-)^*$ of ML_M is basically the identity, except
  - $\lambda x.M$ translates into $(?x \Rightarrow M^* | fail)$
  - $M_1 M_2$ translates into $M_1^* @ M_2^*$

Encoding of monadic metalanguage ML_M (with references) [MF03]

- Syntax of monadic metalanguage – term $M \in \mathsf{E}_o$

  $M ::= x \mid \lambda x.M \mid M_1 M_2 \mid \textit{ret}\ M \mid \textit{do}\ M_1\ M_2 \mid \textit{new}\ M \mid \textit{get}\ M \mid \textit{set}\ M_1\ M_2 \mid a$

  $S ::= \textit{none} \mid \textit{push}\ M\ S$ – references a and control stacks S are instrumental
- Simplification for ML_M is β-reduction: $(\lambda x.M_1)M_2 \longrightarrow M_1[x{:}\,M_2]$
- Configurations $(\mu|M, S)$ with $\mu{:}\ \mathsf{A} \stackrel{fin}{\to} \mathsf{E}_o$, and Computation rules for ML_M
  - $(\mu|\textit{do}\ M_1\ M_2, S) \longmapsto (\mu|M_1, \textit{push}\ M_2\ S)$
  - $(\mu|\textit{ret}\ M_1, \textit{push}\ M_2\ S) \longmapsto (\mu|M_2 M_1, S)$
  - $(\mu|\textit{new}\ M, S) \longmapsto (\mu, a{:}\ M|\textit{ret}\ a, S)$ with $a \in \mathsf{A}$ fresh
  - $(\mu|\textit{get}\ a, S) \longmapsto (\mu|\textit{ret}\ M, S)$ if $\mu(a) = M$
  - $(\mu, a{:}\ M'|\textit{set}\ a\ M, S) \longmapsto (\mu, a{:}\ M|\textit{ret}\ a, S)$

Encoding of monadic metalanguage ML_M (with references) [MF03]

- Syntax of monadic metalanguage – term $M \in \mathsf{E}_o$

  $M ::= x \mid \lambda x.M \mid M_1 M_2 \mid$ *ret* $M \mid$ *do* $M_1\ M_2 \mid$ *new* $M \mid$ *get* $M \mid$ *set* $M_1\ M_2 \mid a$

  $S ::=$ *none* $\mid$ *push* $M\ S$ – references a and control stacks S are instrumental
- Simplification for ML_M is β-reduction: $(\lambda x.M_1)M_2 \longrightarrow M_1[x{:}\,M_2]$
- Configurations $(\mu|M, S)$ with $\mu{:}\ \mathsf{A} \xrightarrow{fin} \mathsf{E}_o$, and Computation rules for ML_M
- *Strong bisimulation* relates $(\mu|M, S)$ to multi-set (modulo simplification) with
  - *prg* $M^*\ S^*$ represents program thread
  - *str* $a\ M^*$ whenever $\mu(a) = M$, represents the store μ

  and computation rules corresponding to those for ML_M, e.g.
  - *prg* $($*do* $?x_1\ ?x_2)\ ?S >$ *prg* $x_1\ ($*push* $x_2\ S)$
  - *prg* $($*new* $?x)\ ?S > \nu y.$*prg* $($*ret* $y)\ S \mid$ *str* $y\ x$
  - *prg* $($*get* $?y)\ ?S \mid$ *str* $?y\ ?x >$ *prg* $($*ret* $x)\ S \mid$ *str* $y\ x$

Encoding of Mobile Ambients [CG98]: *centralized* impl.

- Syntax of MA – processes $P \in \mathsf{E}_o$

  $P ::= 0 \mid (P_1|P_2) \mid !P \mid \nu y.P \mid y[P] \mid$ *in* $y.P \mid$ *out* $y.P \mid$ *open* $y.P$

- Translation $(-)^*$ of MA, sample of clauses
  - $(P_1|P_2)$ translates into *par* $P_1^*\ P_2^*$
  - $\nu y.P$ translates into *new* $(?y{\Rightarrow}P^*|fail)$
  - $y[P]$ translates into *box* $y\ P^*$
  - *in* $y.P$ translates into *in* $y\ P^*$

Encoding of Mobile Ambients [CG98]: *centralized* impl.

- Syntax of MA – processes $P \in \mathsf{E}_o$

  $P ::= 0 \mid (P_1|P_2) \mid !P \mid \nu y.P \mid y[P] \mid \textit{in}\ y.P \mid \textit{out}\ y.P \mid \textit{open}\ y.P$

- Configurations for MA = processes (modulo structural equivalence)

  Basic reduction rules for MA (there are other rules for propagation)

  - $n[\textit{in}\ m.P \mid Q] \mid m[R] \xmapsto{in} m[n[P \mid Q] \mid R]$
  - $m[n[\textit{out}\ m.P \mid Q] \mid R] \xmapsto{out} n[P \mid Q] \mid m[R]$
  - $\textit{open}\ m.P \mid m[Q] \xmapsto{open} P \mid Q$

Encoding of Mobile Ambients [CG98]: *centralized* impl.

- Syntax of MA – processes $P \in \mathsf{E}_o$

  $P ::= 0 \mid (P_1|P_2) \mid !P \mid \nu y.P \mid y[P] \mid$ *in* $y.P \mid$ *out* $y.P \mid$ *open* $y.P$
- Configurations for MA = processes (modulo structural equivalence)
- *Weak bisimulation* relates P to a multi-set (modulo simplification) with
  - *prg* $a\ e$ thread executing e in ambient a
  - *amb* $a\ n\ c$ ambient a has name n and parent ambient c
  - *opened* $a\ c$ ambient a has been opened in parent ambient c

  and rules located at the same place $\{$*prg, amb, opened*$\}$ –

Encoding of Mobile Ambients [CG98]: *centralized* impl.

- Syntax of MA – processes $P \in \mathsf{E}_o$

  $P ::= 0 \mid (P_1|P_2) \mid !P \mid \nu y.P \mid y[P] \mid$ *in* $y.P \mid$ *out* $y.P \mid$ *open* $y.P$
- Configurations for MA = processes (modulo structural equivalence)
- *Weak bisimulation* relates P to a multi-set (modulo simplification) with
  - *prg* $a\ e$ thread executing e in ambient a
  - *amb* $a\ n\ c$ ambient a has name n and parent ambient c
  - *opened* $a\ c$ ambient a has been opened in parent ambient c

  computation rules for heating (sample of rules)
  - *prg* $?y$ (*par* $?x_1\ ?x_2) >$ *prg* $y\ x_1 \mid$ *prg* $y\ x_2$
  - *prg* $?y$ (*new* $?x) > \nu n.$*prg* $y\ (x@n)$
  - *prg* $?y$ (*box* $?n\ ?x) > \nu y'.$*prg* $y\ x \mid$ *amb* $y'\ n\ y$

Encoding of Mobile Ambients [CG98]: *centralized* impl.

- Syntax of MA – processes $P \in \mathsf{E}_o$

  $P ::= 0 \mid (P_1|P_2) \mid !P \mid \nu y.P \mid y[P] \mid \textit{in}\ y.P \mid \textit{out}\ y.P \mid \textit{open}\ y.P$
- Configurations for MA = processes (modulo structural equivalence)
- *Weak bisimulation* relates P to a multi-set (modulo simplification) with
  - *prg* a e thread executing e in ambient a
  - *amb* a n c ambient a has name n and parent ambient c
  - *opened* a c ambient a has been opened in parent ambient c
- Computation rules for mobility
  - $\boxed{\textit{amb}\ ?y'\ ?m\ ?y''} \mid \textit{prg}\ ?y\ (\textit{in}\ ?m\ ?x) \mid \textit{amb}\ ?y\ ?n\ ?y'' > \textit{prg}\ y\ x \mid \textit{amb}\ y\ n\ y'$
  - $\boxed{\textit{amb}\ ?y'\ ?m\ ?y''} \mid \textit{prg}\ ?y\ (\textit{out}\ ?m\ ?x) \mid \textit{amb}\ ?y\ ?n\ ?y' > \textit{prg}\ y\ x \mid \textit{amb}\ y\ n\ y''$

Encoding of Mobile Ambients [CG98]: *centralized* impl.

- Syntax of MA – processes $P \in \mathsf{E}_o$

  $P ::= 0 \mid (P_1|P_2) \mid !P \mid \nu y.P \mid y[P] \mid \textit{in}\ y.P \mid \textit{out}\ y.P \mid \textit{open}\ y.P$
- Configurations for MA = processes (modulo structural equivalence)
- *Weak bisimulation* relates P to a multi-set (modulo simplification) with
  - *prg* $a\ e$ thread executing e in ambient a
  - *amb* $a\ n\ c$ ambient a has name n and parent ambient c
  - *opened* $a\ c$ ambient a has been opened in parent ambient c

  Computation rules for opening
  - *prg* $?y$ (*open* $?m\ ?x$) $\mid$ *amb* $?y'\ ?m\ ?y$ $>$ *prg* $y\ x$ $\mid$ *opened* $y'\ y$
  - *opened* $?y'\ ?y$ $\mid$ *prg* $?y'\ ?x$ $>$ *prg* $y\ x$
  - *opened* $?y'\ ?y$ $\mid$ *amb* $?y''\ ?n\ ?y'$ $>$ *amb* $y''\ n\ y$

Encodings of Mobile Ambients [CG98]

Alternative encoding for *distributed* impl. of MA [FLS00]

- process P of MA related to a multi-set (modulo simplification) with
  - a *prg* e thread executing e in ambient a
  - a *sup* c ambient a has parent ambient c
  - c *sub* a n ambient a has name n and is sub-ambient of c
- and computation rules for ambient a located at $\{a\}$ – for instance
  - a *prg* $($*box* $?n\ ?x) > \nu y.$*init* $y\ x \mid y$ *sup* $a \mid a$ *sub* $y\ n$
  - *init* $?y\ ?x > R[y] \mid y$ *prg* x – $R[y]$ set of computation rules for ambient y

Encodings of Mobile Ambients [CG98]

Alternative encoding for *distributed* impl. of MA [FLS00]

- process P of MA related to a multi-set (modulo simplification) with
  - a *prg* e thread executing e in ambient a
  - a *sup* c ambient a has parent ambient c
  - c *sub* a n ambient a has name n and is sub-ambient of c
- and computation rules for ambient a located at $\{a\}$
- weak bisimulation replaced by *weak coupled-simulation*:

  atomic steps of MA implemented with protocols with gradual commitment

Encodings of Mobile Ambients [CG98]

Alternative encoding for *distributed* impl. of MA [FLS00]

- process P of MA related to a multi-set (modulo simplification) with
  - a *prg* e thread executing e in ambient a
  - a *sup* c ambient a has parent ambient c
  - c *sub* a n ambient a has name n and is sub-ambient of c
- and computation rules for ambient a located at $\{a\}$

Extending encoding for centralized impl. of MA: adding HO communication

- $P ::= \ldots \mid x \mid \langle P \rangle \mid (x)P$ – extended syntax

  $\langle P \rangle \mid (x)Q \xmapsto{comm} Q[x{:}\,P]$ – reduction for local comm.
- extended translation $(-)^*$
  - $\langle P \rangle$ translates into *put* P^*
  - $(x)P$ translates into *get* $(?x \Rightarrow P^* | fail)$
- *prg* $?y$ (*get* $?x_1$) | *prg* $?y$ (*put* $?x_2$) $>$ *prg* y $(x_1 @ x_2)$ – rule for local comm.

Conclusions and Issues

- General approach for structuring operational semantics: simplification + computation

  Simplification capture things that one does not care to control/program, because they are *simple* and *semantics preserving* (referential transparency)

- Concrete proposal: based on ideas from FreshML, PMC and CHAM

  There is scope for variations and improvements, e.g.

  - first-class patterns – as in pure pattern calculus [JK06]
  - more refined computation rules – for probabilistic/stochastic systems
  - more elaborate configurations – to describe parts of a closed system, e.g. environment, that are loosely specified or not directly controlled/programed.