Simplification and Computation

Eugenio Moggi

moggi@disi.unige.it

DISI, Univ. of Genova

Overall Goal

A framework for operational semantics based on

- ideas from monadic metalanguages [MF03]
  - simplification $e \longrightarrow e'$ compatible&confluent relation on terms
  - computation $Cfg_1 \longmapsto Cfg_2$ relation on configurations
- ideas from CHAM [BB92] and related calculi (Join [FG96], Kell [Ste03,BS03])
  - configurations as multisets of terms (and reaction rules)
  - computation as chemical reaction (and heating)
- expressive patterns
  - patterns p for simplification subsume ML& PMC [Kah03]
    Do we need more? LINDA/KLAIM, XML
  - join patterns J with weaken linearity assumptions
    Kell patterns $a[!x]$ address a different issue: match for active kells.
  - distinguish atoms a from varaibles y (as in FreshML [GP99,SGP03])

Use framework for multi-lingual extensions and for defining monadic interpreters.

Overview of Key Properties

- we distinguish atoms a, name variables y and term variables x
- a term e may have free occurrences of atoms $\mathrm{FN}(e)$ and variables $\mathrm{FV}(e)$
- a configuration *Cfg* is a finite multiset of closed terms (i.e. no free variables)

Overview of Key Properties

- we distinguish atoms a, name variables y and term variables x
- a term e may have free occurrences of atoms $\mathrm{FN}(e)$ and variables $\mathrm{FV}(e)$
- a configuration *Cfg* is a finite multiset of closed terms (i.e. no free variables)

The SIMPLIFICATION relation $e \longrightarrow e'$ is

- name/variable preserving, i.e. $\mathrm{FN}(e') \subseteq \mathrm{FN}(e)$ and $\mathrm{FV}(e') \subseteq \mathrm{FV}(e)$
- compatible, i.e. can be performed in any context
- confluent, i.e. can be performed in any order
- invariant w.r.t. permutations π of atoms and substitutions ρ of name/term variables with names/terms, i.e. $\dfrac{e \longrightarrow e'}{e[\pi] \longrightarrow e'[\pi]}$ $\dfrac{e \longrightarrow e'}{e[\rho] \longrightarrow e'[\rho]}$

Overview of Key Properties

- we distinguish atoms a, name variables y and term variables x
- a term e may have free occurrences of atoms $\mathrm{FN}(e)$ and variables $\mathrm{FV}(e)$
- a configuration *Cfg* is a finite multiset of closed terms (i.e. no free variables)

The COMPUTATION relation $\mathit{Cfg}_1 \longmapsto \mathit{Cfg}_2$ is

- invariant w.r.t. permutations π of atoms
- preserved by simplification, i.e.

$$\begin{tikzcd}
\mathit{Cfg}_1 \arrow[r, mapsto] \arrow[d, "*"'] & \mathit{Cfg}_2 \arrow[d, dashed, "*"] \\
\mathit{Cfg}_1' \arrow[r, dashed, mapsto] & \mathit{Cfg}_2'
\end{tikzcd}$$

- preserved by extension, i.e. $\mathit{Cfg} \uplus \mathit{Cfg}_1 \longmapsto \mathit{Cfg} \uplus \mathit{Cfg}_2$ when $\mathit{Cfg}_1 \longmapsto \mathit{Cfg}_2$ and $\mathrm{FN}(\mathit{Cfg})\#(\mathrm{FN}(\mathit{Cfg}_2) - \mathrm{FN}(\mathit{Cfg}_1))$. Thus atomic broadcast not allowed.

Syntax

- atom $a \in \mathsf{A}$, name variable y, term variable x, Name $u \in \mathsf{N} ::= a \mid y$
- Term $e \in \mathsf{E}$, Pattern p, Join pattern J, Match m

e	$::=$	$x \mid u\,\overline{e}$	constructor (name) u applied to sequence $\overline{e}$ of terms
	$\mid$	$fail \mid (p \Rightarrow e_1 \mid e_2) \mid e_1 @ e_2 \mid (m; e')$	analogies with PMC [Kal03]
	$\mid$	let $\{x_i = e_i \mid i \in n\}$ in e	binding for mutual recursive definitions
	$\mid$	$\nu y.e \mid \{\!\{e_i \mid i \in n\}\!\} \mid J > e$	freshness, multiset, and reaction rule
p	$::=$	$!x \mid !y\,\overline{p} \mid u\,\overline{p}$	u matches only itself, $!y$ matches any constructor u
J	$::=$	$\{\!\{u_i\,\overline{p}_i \mid i \in n\}\!\}$	generalizes the Join-calculus
m	$::=$	$ok\ e \mid e\!: p \Rightarrow m$	analogies with PMC [Kal03]

Meaning of matches and related constructs:

- $ok\ e$ succeeds and returns e
- $e\!: p \Rightarrow m$ if e matches p try instance of m, otherwise fail
- $(m; e')$ returns e when m succeeds and returns e, and e' when m fails
- $(p \Rightarrow e_1 \mid e_2) @ e \longrightarrow (e\!: p \Rightarrow e_1 ; e_2 @ e)$ is β-reduction

Syntax

- atom $a \in \mathsf{A}$, name variable y, term variable x, Name $u \in \mathsf{N} ::= a \mid y$
- Term $e \in \mathsf{E}$, Pattern p, Join pattern J, Match m

e	$::=$	$x \mid u\,\overline{e}$	constructor (name) u applied to sequence $\overline{e}$ of terms
	$\mid$	$fail \mid (p{\Rightarrow}e_1 \vert e_2) \mid e_1@e_2 \mid (m; e')$	analogies with PMC [Kal03]
	$\mid$	let $\{x_i = e_i \vert i \in n\}$ in e	binding for mutual recursive definitions
	$\mid$	$\nu y.e \mid \{(e_i \vert i \in n)\} \mid J > e$	freshness, multiset, and reaction rule
p	$::=$	$!x \mid\, !y\,\overline{p} \mid u\,\overline{p}$	u matches only itself, $!y$ matches any constructor u
J	$::=$	$\{(u_i\,\overline{p}_i \vert i \in n)\}$	generalizes the Join-calculus
m	$::=$	$ok\ e \mid e{:}\,p{\Rightarrow}m$	analogies with PMC [Kal03]

Linearity constrains and binding in patterns:

- in p a variable $!x$ or $!y$ can be declared at most once
  * the occurrences of y after $!y$ are bound
- in J a term variable $!x$ can be declared at most one $u_i\,\overline{p}_i$
  * while a name variable $!y$ can be declared in several $u_i\,\overline{p}_i$.

Examples of patterns p and J without free variables

- $p_0 \equiv c\ (c\ !x)$ matches $c\ (c\ e)$, where $c \in$ A
- $p_1 \equiv c\ !y\ y$ matches $c\ a\ a$ for any $a \in$ A
- $p_2 \equiv !y\ (y\ !x)$ matches $a\ (a\ e)$ for any $a \in$ A

How to express function eq to test equality of names (e.g. references)

$$eq = (!y \Rightarrow (y \Rightarrow true | !y' \Rightarrow false | fail) | fail)$$

eq fails when an argument is not an atom (does not simplify to an atom)

Examples of patterns p and J without free variables

We define reaction rules for the operation on ML-references, represented by atoms $newR$, $getR$ and $setR$ (term constructor names). Program represented by molecule named $prog$, store represented by molecules named ref.

- $\{(prog\ (newR\ !x\ !x'))\} > \nu y.\ \{(prog\ (x'@y) \mid ref\ y\ x)\}$
  semantics of `let val` y`=ref` x `in` $x'y$
- $\{(prog\ (getR\ !y\ !x') \mid ref\ !y\ !x)\} > \{(prog\ (x'@x) \mid ref\ y\ x)\}$
  semantics of `let val` x`=!`y `in` $x'x$
- $\{(prog\ (setfR\ !y\, x_1\ !x') \mid ref\ !y\ !x_2)\} > \{(prog\ x' \mid ref\ y\ x_1)\}$
  semantics of y`:=`x`;` x'

Simplification rules: left-linear and non-overlapping

$v ::= u\ \overline{e} \mid fail \mid (p \Rightarrow e_1 \mid e_2) \mid \nu y.e \mid \{\!\!\{(e_i \mid i \in n)\}\!\!\} \mid J > e$	$p ::= !x \mid !y\ \overline{p} \mid u\ \overline{p}$
$e ::= x \mid v \mid e_1 @ e_2 \mid (m; e) \mid \text{let } \{x_i = e_i \mid i \in n\} \text{ in } e$	$m ::= ok\ e \mid e\colon p \Rightarrow m$

Unfolding of recursive definitions

$$\text{let } \{x_i = e_i \mid i \in n\} \text{ in } e \quad \longrightarrow \quad e[x_i\colon \text{let } \{x_i = e_i \mid i \in n\} \text{ in } e_i \mid i \in n]$$

Application

$$\begin{array}{rcl} fail @ e & \longrightarrow & fail \\ (p \Rightarrow e_1 \mid e_2) @ e & \longrightarrow & (e\colon p \Rightarrow ok\ e_1; e_2 @ e) \end{array}$$

Simplification rules: left-linear and non-overlapping

$v ::= u\ \overline{e} \mid fail \mid (p \Rightarrow e_1 \mid e_2) \mid \nu y.e \mid \{(e_i \mid i \in n)\} \mid J > e$	$p ::= !x \mid !y\ \overline{p} \mid u\ \overline{p}$
$e ::= x \mid v \mid e_1 @ e_2 \mid (m; e) \mid$ let $\{x_i = e_i \mid i \in n\}$ in e	$m ::= ok\ e \mid e{:}\, p \Rightarrow m$

Simplification of matching

$$
\begin{array}{rcl}
(ok\ e; e') & \longrightarrow & e \\
(e{:}\,!x \Rightarrow m; e') & \longrightarrow & (m[x{:}\,e]; e') \\
(u\ \overline{e}{:}\,!y\ \overline{p} \Rightarrow m; e') & \longrightarrow & (\overline{e}{:}\,\overline{p}[y{:}\,u] \Rightarrow m[y{:}\,u]; e') \quad \text{when } |\overline{e}| = |\overline{p}| \\
(v{:}\,!y\ \overline{p} \Rightarrow m; e') & \longrightarrow & e' \quad \text{when } v \not\equiv u\ \overline{e} \text{ with } |\overline{e}| = |\overline{p}| \\
(a_1\ \overline{e}{:}\, a_2\ \overline{p} \Rightarrow m; e') & \longrightarrow & \left\{ \begin{array}{ll} (\overline{e}{:}\,\overline{p} \Rightarrow m; e') & \text{if } a_1 = a_2 \\ e' & \text{if } a_1 \neq a_2 \end{array} \right. \quad \text{when } |\overline{e}| = |\overline{p}| \\
(v{:}\, u\ \overline{p} \Rightarrow m; e') & \longrightarrow & e' \quad \text{when } v \not\equiv u\ \overline{e} \text{ with } |\overline{e}| = |\overline{p}|
\end{array}
$$

Computation rules: heating and chemical reaction

$$\boxed{\begin{array}{ll} v ::= u\,\overline{e} \mid fail \mid (p \Rightarrow e_1 | e_2) \mid \nu y.e \mid \{(e_i | i \in n)\} \mid J > e & \\ p ::= !x \mid !y\,\overline{p} \mid u\,\overline{p} & J ::= \{(u_i\,\overline{p}_i | i \in n)\} \end{array}}$$

Heating

$$\begin{array}{rcll} \textit{Cfg},\ \{(e_i | i \in n)\} & \longmapsto & \textit{Cfg},\ \{e_i | i \in n\} & \\ \textit{Cfg},\ \nu y.e & \longmapsto & \textit{Cfg},\ e[y\colon a] & \text{with } a \notin \mathrm{FN}(\textit{Cfg}, \nu y.e) \end{array}$$

Reaction a la Join-calculus

$$\textit{Cfg},\ J > e,\ J\rho \quad \longmapsto \quad \textit{Cfg},\ e[\rho],\ J > e \qquad \rho \text{ closed substitution}$$

$J\rho$ is the multiset obtained by replacing the only occurrence of $!x$ in J with $\rho(x)$, and occurrences of $!y$ and y in J with $\rho(y)$ (each occurrence of y in $u_i\,\overline{p}_i$ is bound by a $!y$)

Conclusion: multi-lingual extensions and interpreters

- new term constructors encoded as fresh atoms

  new term destructors (and their simplification rules) defined using let-binding

  encoding of natural numbers: zero z and successor s are atoms,
  iterator $it\colon X \to (X \to X) \to N \to X$ is a variable defined recursively

$$\nu z.\ \nu s.\ \text{let } it = (!x \Rightarrow !f \Rightarrow (z \Rightarrow x \mid s\ !n \Rightarrow it@x@f@n \mid fail)) \text{ in } \ldots$$

- interpreter for existing term constructors as reaction rules for new molecules

  interpreter for operation on references $newR$, $getR$ and more

$$\nu p.\ \nu r.\quad \text{molecule names for interpreted programs and local store}$$
$$\left\{\left(\begin{array}{l} \{(p\ (newR\ !x\ !x'))\} > \nu y.\ \{(p\ (x'@y) \mid r\ y\ x)\}\ , \\ \{(p\ (getR\ {\color{red}!y}\ !x') \mid r\ {\color{red}!y}\ !x)\} > \{(p\ (x'@x) \mid r\ y\ x)\}\ , \\ \ldots \end{array}\right)\right\}$$

  restrict visibility of r to ensure that store is manipulated only by the interpreter

Conclusion: multi-lingual extensions and interpreters

- new term constructors encoded as fresh atoms

  new term destructors (and their simplification rules) defined using let-binding
- interpreter for existing term constructors as reaction rules for new molecules

  interpreter for operation on references $newR$, $getR$ and more

$$\nu p.\, \nu r. \quad \text{molecule names for interpreted programs and local store}$$

$$\left\{ \left(\begin{array}{l} \{(p\ (newR\ !x\ !x'))\} > \nu y.\, \{(p\ (x'@y) \mid r\ y\ x)\}\,, \\ \{(p\ (getR\ !y\ !x') \mid r\ !y\ !x)\} > \{(p\ (x'@x) \mid r\ y\ x)\}\,, \\ \ldots \end{array} \right) \right\}$$

restrict visibility of r to ensure that store is manipulated only by the interpreter