A taste of soft-linear logic for self-modification

Hubert Godfroy

joint work with Jean-Yves Marion
Loria Nancy

14 octobre 2015

Self-modification

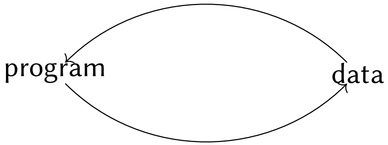
Self-modifying program have the ability to

- generate others programs at runtime
  - Ex. just-in-time compilers
- execute data
  - Ex. `exec` function in python
- match data against patterns (as any normal program)
  - Ex. parser
- inspect its own code
  - Ex. integrated integrity checkers

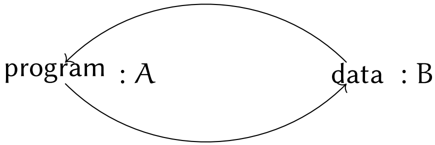
The goal

program data

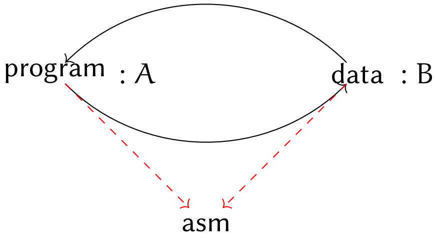
The goal



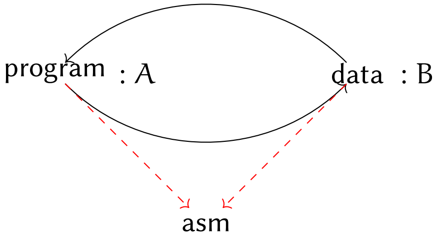
The goal



The goal



The goal



The problem

- Imperative languages: logical aspects are unnatural
- λ-calculus: no data!

Plan

Functional language

Low level correspondence

Functional language with self-modifying abilities

Idea

- Syntactically mark frozen terms: $\langle t \rangle$.
- These terms are in NF ($\forall t$).
- These terms are the data of the language.

Problem with free variables

- Confluence may be lost : $(\lambda x. \langle x \rangle)\ t$

$\Rightarrow$ Some free variables must be rejected

Staged computation

- Paradigm where computation is split in stages
- Partial evaluation
- Run-time code generation
- Community uses tools like modal or temporal logic (MetaML)

R. Davies and F. Pfenning.
A modal analysis of staged computation.
Principles of programming languages, 1996.

R. Davies.
A temporal-logic approach to binding-time analysis.
Logic in Computer Science, pages 184–195, Jul 1996.

Operational semantics

- $\forall t, \langle t \rangle$ is in NF (data).
- Meta-binder: $\text{let}\langle x \rangle = t'$ in t
- Meta-redex: $(\text{let}\langle x \rangle = \langle t' \rangle \text{ in } t) \rightarrow t[x/t']$

Operational semantics

- $\forall t, \langle t \rangle$ is in NF (data).
- Meta-binder: $\text{let}\langle x \rangle = t'$ in t
- Meta-redex: $(\text{let}\langle x \rangle = \langle t' \rangle \text{ in } t) \to t[x/t']$

Example

Notice the (intensional) difference between

$$\text{let}\langle x \rangle = \langle t \rangle \text{ in } \langle x \rangle \quad \to \quad \langle t \rangle$$

Operational semantics

- $\forall t, \langle t \rangle$ is in NF (data).
- Meta-binder: $\mathsf{let}\langle x \rangle = t'$ in t
- Meta-redex: $(\mathsf{let}\langle x \rangle = \langle t' \rangle \text{ in } t) \to t[x/t']$

Example

Notice the (intensional) difference between

$$\mathsf{let}\langle x \rangle = \langle t \rangle \text{ in } \langle x \rangle \quad \to \quad \langle t \rangle$$

and

$$(\lambda x.x)\langle t \rangle \quad \to \quad \langle t \rangle$$

Operational semantics

- $\forall t, \langle t \rangle$ is in NF (data).
- Meta-binder: $\mathsf{let}\langle x \rangle = t'$ in t
- Meta-redex: $(\mathsf{let}\langle x \rangle = \langle t' \rangle \text{ in } t) \rightarrow t[x/t']$

Example

Notice the (intensional) difference between

$$\mathsf{let}\langle x \rangle = \langle t \rangle \text{ in } \langle x \rangle \quad \rightarrow \quad \langle t \rangle$$

and

$$(\lambda x.x)\langle t \rangle \quad \rightarrow \quad \langle t \rangle$$

$\Rightarrow$ Behaviors are the same extensionally but not intensionally.

$\Rightarrow$ This difference will appear clearly latter

Typing rules

- Structural rule:

$$\frac{\Gamma \vdash t' : !A \qquad \langle x \rangle : A, \Gamma \vdash t : B}{\Gamma \vdash \mathrm{let} \langle x \rangle = t' \ \mathrm{in}\ t : B}\ \textsc{Let}$$

- Linear Logic:

$$\frac{\Delta \vdash t : A}{\langle \Delta \rangle, \Gamma \vdash \langle t \rangle : !A}\ \textsc{SoftPromotion} \qquad \frac{\langle x \rangle : A \in \Gamma}{\Gamma \vdash x : A}\ \textsc{Dereliction}$$

where $\Gamma = x_i : A_i$, then $\langle \Gamma \rangle \stackrel{\text{def}}{=} \langle x_i \rangle : !A_i$.

Language behavior

$$\frac{}{\Gamma, x : A \vdash x : A}\ \textsc{Var} \qquad \frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x.t : A \to B}\ \textsc{Abstraction} \qquad \frac{\Gamma \vdash t : A \to B \qquad \Gamma \vdash t' : A}{\Gamma \vdash t\, t' : B}\ \textsc{Application}$$

$$\frac{\Gamma \vdash t' : !A \qquad \Gamma, \langle x \rangle : !A \vdash t : B}{\Gamma \vdash \mathsf{let}\langle x \rangle = t' \text{ in } t : B}\ \textsc{Let} \qquad \frac{x_i : A_i \vdash t : B}{\Gamma, \langle x_i \rangle : !A_i \vdash \langle t \rangle : !B}\ \textsc{SoftPromotion} \qquad \frac{}{\Gamma, \langle x \rangle : !A \vdash x : A}\ \textsc{Dereliction}$$

Abilities

Properties inherited from languages inspired by modal logic.

- program generation (plug data into another)
- execution

Properties

- Language is confluent
- Type system has subject reduction property

Language behavior

$$\textsc{Var}\ \frac{}{\Gamma, x : A \vdash x : A} \qquad \textsc{Abstraction}\ \frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x.t : A \to B} \qquad \textsc{Application}\ \frac{\Gamma \vdash t : A \to B \qquad \Gamma \vdash t' : A}{\Gamma \vdash t\, t' : B}$$

$$\textsc{Let}\ \frac{\Gamma \vdash t' : !A \qquad \Gamma, \langle x \rangle : !A \vdash t : B}{\Gamma \vdash \mathsf{let}\langle x \rangle = t'\ \mathsf{in}\ t : B} \qquad \textsc{SoftPromotion}\ \frac{x_i : A_i \vdash t : B}{\Gamma, \langle x_i \rangle : !A_i \vdash \langle t \rangle : !B} \qquad \textsc{Dereliction}\ \frac{}{\Gamma, \langle x \rangle : !A \vdash x : A}$$

Abilities

Properties inherited from languages inspired by modal logic.

- program generation (plug data into another)
- execution

Properties

- Language is confluent
- Type system has subject reduction property

Missing

- No reflexion possibilities
- No syntactic analysis of data

Example

- Writing:

$$(\text{let}\langle x\rangle = \langle t'\rangle \text{ in } \langle t\rangle) \rightarrow \langle t[x/t']\rangle$$

Example

- Writing:

$$(\mathsf{let}\langle x\rangle = \langle t'\rangle \text{ in } \langle t\rangle) \longrightarrow \langle t[x/t']\rangle$$

Use this derivation (for $t = x$):

$$\dfrac{\dfrac{\cdots}{\Gamma \vdash \langle t'\rangle : !A} \qquad \dfrac{\dfrac{}{\Gamma', x : A \vdash x : A}\,\textsc{Var}}{\Gamma, \langle x\rangle : !A \vdash \langle x\rangle : !A}\,\textsc{SoftPromotion}}{\Gamma \vdash \mathsf{let}\langle x\rangle = \langle t'\rangle \text{ in } \langle x\rangle : !A}\,\textsc{Abstraction}$$

Example

- Writing:

$$(\mathsf{let}\langle x\rangle = \langle t'\rangle \text{ in } \langle t\rangle) \longrightarrow \langle t[x/t']\rangle$$

Use this derivation (for $t = x$):

$$\textsc{Abstraction}\ \dfrac{\dfrac{\cdots}{\Gamma \vdash \langle t'\rangle : !A} \qquad \dfrac{\dfrac{}{\Gamma', x : A \vdash x : A}\ \textsc{Var}}{\Gamma, \langle x\rangle : !A \vdash \langle x\rangle : !A}\ \textsc{SoftPromotion}}{\Gamma \vdash \mathsf{let}\langle x\rangle = \langle t'\rangle \text{ in } \langle x\rangle : !A}$$

- Execution:

$$\mathsf{run} \stackrel{\text{def}}{=} \lambda x.\, \mathsf{let}\langle y\rangle = x \text{ in } y$$

Example

- Writing:

$$(\mathsf{let}\langle x\rangle = \langle t'\rangle \text{ in } \langle t\rangle) \longrightarrow \langle t[x/t']\rangle$$

Use this derivation (for $t = x$):

$$\dfrac{\dfrac{\cdots}{\Gamma \vdash \langle t'\rangle : !A} \qquad \dfrac{\dfrac{}{\Gamma', x : A \vdash x : A}\,\textsc{Var}}{\Gamma, \langle x\rangle : !A \vdash \langle x\rangle : !A}\,\textsc{SoftPromotion}}{\Gamma \vdash \mathsf{let}\langle x\rangle = \langle t'\rangle \text{ in } \langle x\rangle : !A}\,\textsc{Abstraction}$$

- Execution:

$$\mathsf{run} \stackrel{\text{def}}{=} \lambda x.\, \mathsf{let}\langle y\rangle = x \text{ in } y$$

Use this derivation:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\Gamma, y : A \vdash y : A}\,\textsc{Var}}{\Gamma, \langle y\rangle : !A \vdash y : A}\,\textsc{Dereliction}}{\Gamma, x : !A \vdash \mathsf{let}\langle y\rangle = x \text{ in } y : A}\,\textsc{Let}}{\Gamma \vdash \lambda x.\, \mathsf{let}\langle y\rangle = x \text{ in } y : !A \to A}\,\textsc{Abstraction}$$

Example

- Writing:

$$(\mathsf{let}\langle x\rangle = \langle t'\rangle \text{ in } \langle t\rangle) \to \langle t[x/t']\rangle$$

Use this derivation (for $t = x$):

$$\textsc{Abstraction}\ \frac{\dfrac{\cdots}{\Gamma \vdash \langle t'\rangle : !A} \qquad \dfrac{\dfrac{}{\Gamma', x : A \vdash x : A}\ \textsc{Var}}{\Gamma, \langle x\rangle : !A \vdash \langle x\rangle : !A}\ \textsc{SoftPromotion}}{\Gamma \vdash \mathsf{let}\langle x\rangle = \langle t'\rangle \text{ in } \langle x\rangle : !A}$$

- Execution:

$$\mathsf{run} \stackrel{\text{def}}{=} \lambda x.\, \mathsf{let}\langle y\rangle = x \text{ in } y$$

Use this derivation:

$$\textsc{Abstraction}\ \frac{\textsc{Let}\ \dfrac{\textsc{Dereliction}\ \dfrac{\textsc{Var}\ \dfrac{}{\Gamma, y : A \vdash y : A}}{\Gamma, \langle y\rangle : !A \vdash y : A}}{\Gamma, x : !A \vdash \mathsf{let}\langle y\rangle = x \text{ in } y : A}}{\Gamma \vdash \lambda x.\, \mathsf{let}\langle y\rangle = x \text{ in } y : !A \to A}$$

$\Rightarrow$ Dereliction is used as a run

Example

- Writing:

$$(\mathsf{let}\langle x\rangle = \langle t'\rangle \text{ in } \langle t\rangle) \to \langle t[x/t']\rangle$$

Use this derivation (for $t = x$):

$$\dfrac{\dfrac{\cdots}{\Gamma \vdash \langle t'\rangle : !A} \quad \dfrac{\dfrac{}{\Gamma', x : A \vdash x : A}\,\textsc{Var}}{\Gamma, \langle x\rangle : !A \vdash \langle x\rangle : !A}\,\textsc{SoftPromotion}}{\Gamma \vdash \mathsf{let}\langle x\rangle = \langle t'\rangle \text{ in } \langle x\rangle : !A}\,\textsc{Abstraction}$$

- Execution:

$$\mathsf{run} \stackrel{\text{def}}{=} \lambda x.\,\mathsf{let}\langle y\rangle = x \text{ in } y$$

Use this derivation:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\Gamma, y : A \vdash y : A}\,\textsc{Var}}{\Gamma, \langle y\rangle : !A \vdash y : A}\,\textsc{Dereliction}}{\Gamma, x : !A \vdash \mathsf{let}\langle y\rangle = x \text{ in } y : A}\,\textsc{Let}}{\Gamma \vdash \lambda x.\,\mathsf{let}\langle y\rangle = x \text{ in } y : !A \to A}\,\textsc{Abstraction}$$

$\Rightarrow$ Dereliction is used as a run

Question

- Is dereliction necessary to run data?
- Is dereliction used for another behavior?

Dereliction as launch control

Term typed without DERELICTION cannot run data.

Corollary (non interference)

If $\langle x \rangle : !B, \Gamma \vdash C : A$ is derivable without DERELICTION, then it exists C' such that

$$\forall t, C[x/t] \xrightarrow{*} C'[x/t] \not\to .$$

Dereliction as launch control

Term typed without DERELICTION cannot run data.

Corollary (non interference)

If $\langle x \rangle : !B, \Gamma \vdash C : A$ is derivable without DERELICTION, then it exists C' such that

$$\forall t, C[x/t] \xrightarrow{*} C'[x/t] \not\rightarrow .$$

Reciprocal

If $t \rightarrow t'$ and $\langle x \rangle : !B, \Gamma \vdash C : A$ is derivable with DERELICTION on x, then

$$C[x/t] \xrightarrow{*} C[x/t'].$$

Plan

Functional language

Low level correspondence

Evaluation strategy

- Give low level interpretation of the language
- Simulate CBV strategy:

$$\frac{}{(\lambda x.t)\ v \overset{\text{CBV}}{\rightarrow} t[x/v]} \qquad \frac{}{\text{let}\langle x\rangle = \langle t\rangle \text{ in } t_1 \overset{\text{CBV}}{\rightarrow} t_1[x/t]}$$

$$\frac{t_1 \overset{\text{CBV}}{\rightarrow} t_1'}{t_1\ t_2 \overset{\text{CBV}}{\rightarrow} t_1'\ t_2} \qquad \frac{t_2 \overset{\text{CBV}}{\rightarrow} t_2'}{v\ t_2 \overset{\text{CBV}}{\rightarrow} v\ t_2'}$$

$$\frac{t_2 \overset{\text{CBV}}{\rightarrow} t_2'}{\text{let}\langle x\rangle = t_2 \text{ in } t_1 \overset{\text{CBV}}{\rightarrow} \text{let}\langle x\rangle = t_2' \text{ in } t_1}$$

Evaluation strategy

- Give low level interpretation of the language
- Simulate CBV strategy:

$$\frac{}{(\lambda x.t)\ v \overset{\text{CBV}}{\rightarrow} t[x/v]} \qquad \frac{}{\mathsf{let}\langle x\rangle = \langle t\rangle \ \mathsf{in}\ t_1 \overset{\text{CBV}}{\rightarrow} t_1[x/t]}$$

$$\frac{t_1 \overset{\text{CBV}}{\rightarrow} t_1'}{t_1\ t_2 \overset{\text{CBV}}{\rightarrow} t_1'\ t_2} \qquad \frac{t_2 \overset{\text{CBV}}{\rightarrow} t_2'}{v\ t_2 \overset{\text{CBV}}{\rightarrow} v\ t_2'}$$

$$\frac{t_2 \overset{\text{CBV}}{\rightarrow} t_2'}{\mathsf{let}\langle x\rangle = t_2\ \mathsf{in}\ t_1 \overset{\text{CBV}}{\rightarrow} \mathsf{let}\langle x\rangle = t_2'\ \mathsf{in}\ t_1}$$

Why CBV?

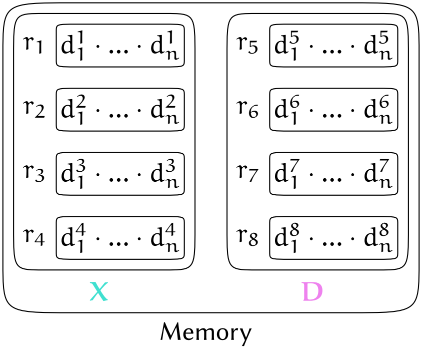
- Simple way to solve problem of redex $\mathsf{let}\langle x\rangle = \langle t'\rangle\ \mathsf{in}\ t$
- Already well studied abstract CBV machine (SECD)

ASM_2 Machine: principle

RAM-like machine for structure and SECD-like for instructions. A state of the machine: $\langle \mathrm{d} \mid \mathrm{k} \mid \mathrm{e} \mid \mathbf{X} \mid \mathbf{D} \rangle$

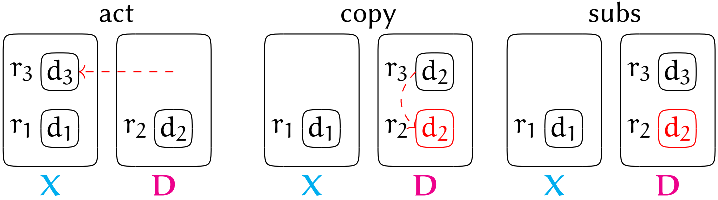
ASM_2 Machine: principle

RAM-like machine for structure and SECD-like for instructions. A state of the machine: $\langle d \mid k \mid e \mid \mathbf{X} \mid \mathbf{D} \rangle$



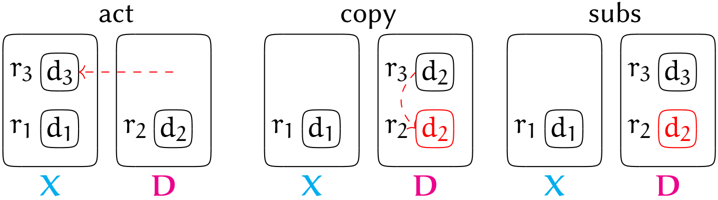
Abilities

SECD machine with...



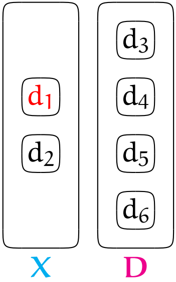
Abilities

SECD machine with...



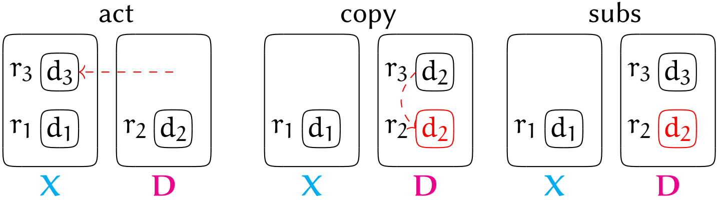
Example

If $d_1 =$ subs d_4; subs d_6; act d_6; run d_6



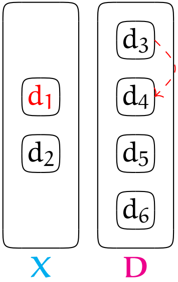
Abilities

SECD machine with…



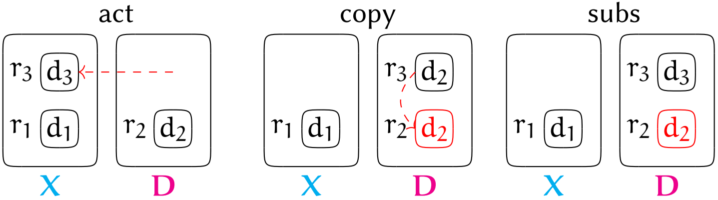
Example

If $d_1 =$ subs d_4; subs d_6; act d_6; run d_6



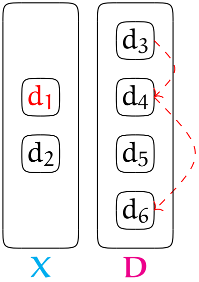
Abilities

SECD machine with…



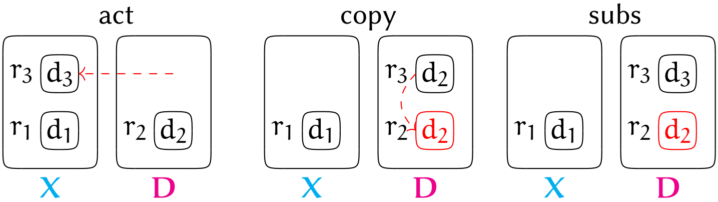
Example

If $d_1 = $ subs d_4; subs d_6; act d_6; run d_6



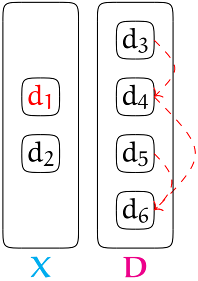
Abilities

SECD machine with…



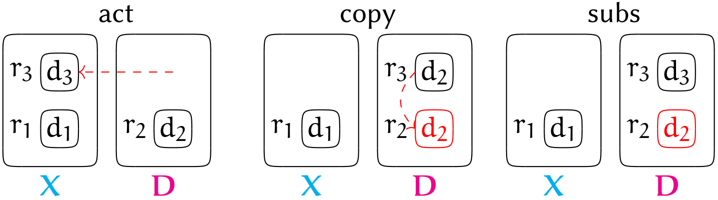
Example

If $d_1 =$ subs d_4; subs d_6; act d_6; run d_6



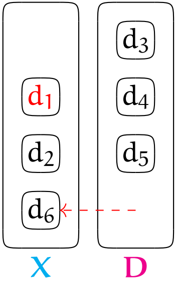
Abilities

SECD machine with...



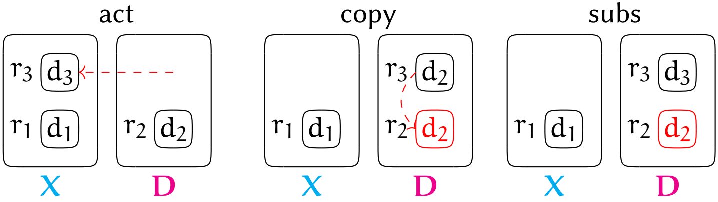
Example

If $d_1 =$ subs d_4; subs d_6; act d_6; run d_6



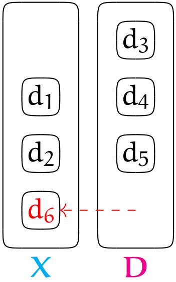
Abilities

SECD machine with...



Example

If $d_1 =$ subs d_4; subs d_6; act d_6; run d_6



Abilities

- Program and data live in the same world (program are data)
- Clear distinction between program and data
- Execution of data are made explicit

Abilities

- Program and data live in the same world (program are data)
- Clear distinction between program and data
- Execution of data are made explicit

Lacks

- Pattern matching on data
- Reflexivity

⇒ Same lacks than the high level language.

Compilation

Our will

- Compilation of t is given by a set of data $\mathbf{T}$ and an executable data d:

$$\mathbf{T} \vdash t \sim d$$

Compilation

Our will

- Compilation of t is given by a set of data $\mathbf{T}$ and an executable data d:

$$\Gamma; \mathbf{T} \vdash t \sim d : A$$

Compilation

Our will

- Compilation of t is given by a set of data $\mathbf{T}$ and an executable data d:

$$\Gamma; \mathbf{T} \vdash t \sim d : A$$

- If $t \overset{CBv*}{\rightarrow} v$ and $\Gamma; \mathbf{T} \vdash t \sim d : A$, then it exists $\mathbf{X}', \mathbf{D}', v'$ such that

$$\langle d \cdot d' \mid k \mid e \mid \mathbf{X} \mid \mathbf{D} \rangle \overset{*}{\rightarrow} \langle d' \mid v'.k \mid e \mid \mathbf{X}' \mid \mathbf{D}' \rangle$$

  with $\mathbf{X} \subset \mathbf{X}'$, $\mathbf{D} \subset \mathbf{D}'$ and $\mathbf{T} \in \mathbf{D}$

- If $t \overset{CBv*}{\rightarrow} \infty$ and $\Gamma; \mathbf{T} \vdash t \sim d : A$, then

$$\langle d \cdot d' \mid k \mid e \mid \mathbf{X} \mid \mathbf{D} \rangle \overset{*}{\rightarrow} \infty$$

  with $\mathbf{T} \in \mathbf{D}$.

ASM_2 Machine: Semantics

- Usual SECD rules:

$$
\begin{array}{rrrrrr}
 & push(d').d & k & e & X & D \\
\to & d & (d', e).k & e & X & D \\
\\
 & call\,.d & v.(d', e').k & e & X & D \\
\to & d' & (d, e).k & v.e' & X & D \\
\\
 & ret\,.d & v.(d', e').k & e & X & D \\
\to & d' & v.k & e' & X & D
\end{array}
$$

- Usual compilation rules:

CApp

$$\frac{\Gamma; \mathbf{T} \vdash t \sim d : A \to B \qquad \Gamma; \mathbf{T} \vdash t' \sim d' : A}{\Gamma; \mathbf{T} \vdash t\ t' \sim d.d'.[call] : B}$$

CAbs

$$\frac{A, \Gamma; \mathbf{T} \vdash t \sim d : B}{\Gamma; \mathbf{T} \vdash \lambda.t \sim [push(d.[ret])] : A \to B}$$

Box

- Rule:

$$\begin{array}{rrrrrl} & (\mathsf{copy}\,r').d & k & e & \mathbf{X} & \mathbf{D} \\ \overset{\text{if } r'' \text{ is fresh}}{\longrightarrow} & d & \langle r''\rangle.k & e & \mathbf{X} & \mathbf{D}[r'' \mapsto \mathbf{D}(r')] \end{array}$$

$$\begin{array}{rrrrrl} & \mathsf{subs}\,.d & \langle r'\rangle.k & e & \mathbf{X} & \mathbf{D} \\ \longrightarrow & \langle d & \langle r'\rangle.k & e & \mathbf{X} & \mathbf{D}[r' \mapsto \mathbf{D}(r')[e, \mathbf{D}]] \end{array}$$

- Compilation rule:

$$\frac{\Delta; \mathbf{T} \vdash t \sim d : A \qquad \mathbf{T}(r) = d.[\mathsf{ret}]}{\Gamma, \langle\Delta\rangle; \mathbf{T} \vdash \langle t\rangle \sim [\mathsf{copy}\ r].[\mathsf{subs}] : !A}\ \text{CBox}$$

Let

- Rule:

$$\begin{array}{lllll} & \mathrm{act}\,n.d & k & e & \mathbf{X} & \mathbf{D} \\ \xrightarrow{\text{if } e(n)=\langle r'\rangle} & d & k & e[n \to r'] & \mathbf{X}[r' \mapsto \mathbf{D}(r')] & \mathbf{D} \end{array}$$

- Compilation rule:

$$\text{CLet}\ \frac{\langle x\rangle : !A, \Gamma; \mathbf{T} \vdash t \sim d : B \qquad \Gamma; \mathbf{T} \vdash t' \sim d' : !A}{\Gamma; \mathbf{T} \vdash \mathrm{let}\ t'\ \mathrm{in}\ t \sim [\mathrm{push}([\mathrm{act}\ 0].d.[\mathrm{ret}])].d'.[\mathrm{call}] : B}$$

Remark

- By default, a let activate the given data.
- Ex: $\mathrm{let}\langle x\rangle = t$ in $f\ x\ \langle x\rangle$

Variables

- Variables have two roles
  - reference:
  $$\lambda x.x$$
  - execution:
  $$\text{let}\langle x\rangle = t \text{ in } x$$
- Instructions:

$$\begin{array}{rrrrrr}
 & (\text{fetch}\, n).d & k & e & X & D \\
\to & d & e(n).k & e & X & D \\
 & (\text{run}\, n).d & k & e & X & D \\
\xrightarrow{\text{if } e(n) = r'} & X(r') & (d, e).k & \cdot & X & D
\end{array}$$

- Compilation of variables:

CVar
$$\frac{\Gamma(n) = A}{\Gamma; \mathbf{T} \vdash n \sim [\text{fetch}\, n] : A}$$

CRun
$$\frac{\Gamma(\langle n\rangle) = !A}{\Gamma; \mathbf{T} \vdash n \sim [\text{run}\, n] : A}$$

Variables

- Variables have two roles
  - reference:
$$\lambda x.x$$
  - execution:
$$\text{let}\langle x\rangle = t \text{ in } x$$
- Instructions:
$$\begin{array}{rrrccc}
 & (\text{fetch}\, n).d & k & e & X & D \\
\to & d & e(n).k & e & X & D \\
 & (\text{run}\, n).d & k & e & X & D \\
\xrightarrow{\text{if } e(n) = r'} & X(r') & (d, e).k & \cdot & X & D
\end{array}$$
- Compilation of variables:

CVar
$$\frac{\Gamma(n) = A}{\Gamma; \mathbf{T} \vdash n \sim [\text{fetch}\, n] : A}$$

Dereliction
$$\frac{\Gamma(\langle n\rangle) = !A}{\Gamma \vdash n : A}$$

Examples (1/2)

Writing: $\text{let}\langle x\rangle = \langle t\rangle \text{ in } \langle x\rangle$

$$
\begin{array}{rrrll}
 & k & e & \mathbf{X} & \mathbf{D} \\
\xrightarrow{\langle t\rangle} & \langle r\rangle.k & e & \mathbf{X} & \mathbf{D}[r \to t] \\
\xrightarrow{\text{let}} & k & r.e & \mathbf{X}[r \to t] & \mathbf{D}[r \to t] \\
\xrightarrow{\langle x\rangle} & \langle r'\rangle.k & r.e & \mathbf{X}[r \to t] & \mathbf{D}[r \to t][r' \to t]
\end{array}
$$

Examples (1/2)

Writing: let$\langle x\rangle = \langle t\rangle$ in $\langle x\rangle$

$$
\begin{array}{rrrll}
 & k & e & \mathbf{X} & \mathbf{D} \\
\xrightarrow{\langle t\rangle} & \langle r\rangle.k & e & \mathbf{X} & \mathbf{D}[r \to t] \\
\xrightarrow{\text{let}} & k & r.e & \mathbf{X}[r \to t] & \mathbf{D}[r \to t] \\
\xrightarrow{\langle x\rangle} & \langle r'\rangle.k & r.e & \mathbf{X}[r \to t] & \mathbf{D}[r \to t][r' \to t]
\end{array}
$$

Execution: let$\langle x\rangle = \langle t\rangle$ in x

$$
\begin{array}{rrrll}
 & \vdots & \vdots & \vdots & \vdots \\
 & k & r.e & \mathbf{X}[r \to t] & \mathbf{D}[r \to t] \\
\xrightarrow{x} & (\cdot, e).k & \cdot & \mathbf{X}[r \to t] & \mathbf{D}[r \to t] \\
\xrightarrow{\mathbf{X}(r)} & \dots & & &
\end{array}
$$

Examples (2/2)

Writing: let$\langle x \rangle = \langle t \rangle$ in $\langle x \rangle$

$$
\begin{array}{rrrll}
 & k & e & \mathbf{X} & \mathbf{D} \\
\xrightarrow{\langle t \rangle} & \langle r \rangle.k & e & \mathbf{X} & \mathbf{D}[r \to t] \\
\xrightarrow{\text{let}} & k & r.e & \mathbf{X}[r \to t] & \mathbf{D}[r \to t] \\
\xrightarrow{\langle x \rangle} & \langle r' \rangle.k & r.e & \mathbf{X}[r \to t] & \mathbf{D}[r \to t][r' \to t]
\end{array}
$$

Examples (2/2)

Writing: $\mathrm{let}\langle x\rangle = \langle t\rangle \text{ in } \langle x\rangle$

$$
\begin{array}{lrlll}
 & k & e & X & D \\
\xrightarrow{\langle t\rangle} & \langle r\rangle.k & e & X & D[r \to t] \\
\xrightarrow{\text{let}} & k & r.e & X[r \to t] & D[r \to t] \\
\xrightarrow{\langle x\rangle} & \langle r'\rangle.k & r.e & X[r \to t] & D[r \to t][r' \to t]
\end{array}
$$

Identity: $(\lambda x.x)\langle t\rangle$

$$
\begin{array}{lrlll}
 & k & e & X & D \\
\xrightarrow{\lambda x.x} & (x, e).k & e & X & D \\
\xrightarrow{\langle t\rangle} & \langle r\rangle.(\lambda x.x, e).k & e & X & D[r \to t] \\
\xrightarrow{@} & (\cdot, e).k & \langle r\rangle.e & X & D[r \to t] \\
\xrightarrow{x} & \langle r\rangle.(\cdot, e).k & e & X & D[r \to t] \\
\xrightarrow{\text{ret}} & \langle r\rangle.k & e & X & D[r \to t]
\end{array}
$$

Conclusion

- High level language with self-modifying behaviors (writing & executing data)
- Dereliction is a data execution
- Meaningful compilation in low-level machine designed for self-modification
- Dereliction still corresponds to data execution

Still missing

- Reflexion & pattern matching
- A more deterministic partial evaluation