An Imperative Rho

(iRho)

Luigi Liquori

joint work with

Bernard (Big) Paul Serpette

INRIA

iRho

Motivations

Multiparadigm

- *Rewriting-based* languages (like, $e.g.$,) try to unify the logic paradigm with the functional paradigm.

Multiparadigm

- *Rewriting-based* languages (like, $e.g.$,) try to unify the logic paradigm with the functional paradigm.

- Although these languages are less used in common practice than $e.g.$ object-oriented languages (like, $e.g.$, Java, C#, OCaml, . . .), they can be used also as (formal) common intermediate languages for implementing compilers for rewriting-based, functional, object-oriented, logic, and others "high-level" modern languages and meta languages.

Main *Atout*: *Pattern Matching*.

- *Pattern-matching* allows to discriminate between alternatives. Once a pattern is recognized, a pattern is associated to an action. The corresponding pattern is thus rewritten in an appropriate instance of a new one

- Ability to handle a collection of results: pattern matching need not to be exclusive, $i.e.$ multiple branches can be "fired" simultaneously

- An empty collection of results represents an application failure, a singleton represents a deterministic result, and a collection with more than one element represents a non-deterministic choice between the elements of the collection

Main *Atout*: *Pattern Matching*.

- *Pattern-matching* allows to discriminate between alternatives. Once a pattern is recognized, a pattern is associated to an action. The corresponding pattern is thus rewritten in an appropriate instance of a new one
- Ability to handle a collection of results: pattern matching need not to be exclusive, $i.e.$ multiple branches can be "fired" simultaneously
- An empty collection of results represents an application failure, a singleton represents a deterministic result, and a collection with more than one element represents a non-deterministic choice between the elements of the collection
- Applications: pattern recognition, strings or trees manipulation, etc
- Pattern-matching present in ML, Haskell, Scheme, or Prolog; considered a convenient mechanism for expressing complex requirements about the function's argument, rather than a basis for an $ad\ hoc$ paradigm of computation

Rho's Goodies

- One of the main features of the Rewriting-calculus is its capacity to deal with (de)structuring structures like $e.g.$ lists: : we record only the names of the constructor and we discard those of the accessors

- Since structures are built-in the calculus, it follows that the encoding of constructor/accessors is simpler $\mathrm{w.r.t.}$ the standard encoding in the Lambda-calculus. The table below (informally) compares the (untyped) encoding of accessors in both formalisms

ops/form	Rewriting-calculus	Lambda-calculus
cons	$X \twoheadrightarrow Y \twoheadrightarrow (\text{cons } X\ Y)$	$\lambda XYZ.\ ZXY$
car	$(\text{cons } X\ Y) \twoheadrightarrow X$	$\lambda Z.\ Z(\lambda XY.X)$
cdr	$(\text{cons } X\ Y) \twoheadrightarrow Y$	$\lambda Z.\ Z(\lambda XY.Y)$

iRho *vs.* Programming

- iRho is an extension of Rho with references, memory allocation, and assignment. It features, all the "idiosyncrasies" of functional/rewriting-based languages with imperative features and modern pattern matching facilities

iRho *vs.* Programming

- iRho is an extension of Rho with references, memory allocation, and assignment. It features, all the "idiosyncrasies" of functional/rewriting-based languages with imperative features and modern pattern matching facilities

- The controlled and conscious use of references, gives to the user the programming comfort and a good expressiveness which we could not a priori expect from a so simple calculus

iRho *vs.* Programming

- iRho is an extension of Rho with references, memory allocation, and assignment. It features, all the "idiosyncrasies" of functional/rewriting-based languages with imperative features and modern pattern matching facilities

- The controlled and conscious use of references, gives to the user the programming comfort and a good expressiveness which we could not a priori expect from a so simple calculus

- The "magic ingredients" of iRho are the combination of modern and safe imperative features (full control over internal data-structure reprs), and of the "matching power" (full Lisp-like operations, like cons/car/cdr)

iRho *vs.* Programming

- iRho is an extension of Rho with references, memory allocation, and assignment. It features, all the "idiosyncrasies" of functional/rewriting-based languages with imperative features and modern pattern matching facilities

- The controlled and conscious use of references, gives to the user the programming comfort and a good expressiveness which we could not a priori expect from a so simple calculus

- The "magic ingredients" of iRho are the combination of modern and safe imperative features (full control over internal data-structure reprs), and of the "matching power" (full Lisp-like operations, like cons/car/cdr)

- $insumma$, iRho as theoretical engine for an family of $ad\ hoc$ languages combining functions, patterns, objects with semi-structured XML-data (XDUCE, CDUCE, HYDROJ, TOM) (*"...O-O pattern matching focuses on the essential information in a msg and is insensitive to inessential information..."*)

A "Fresh" Approach to Rewriting

- We present an imperative fully typed ($e.g.$ $\grave{a}$ la Church) version of the Rho, a pattern-matching based calculus with side-effects, which we call iRho
- We formulate the static and dynamic semantics of iRho
- A call-by-value deterministic Natural Semantics $\grave{a}$ la Kahn: it immediately suggests how to build an interpreter for the calculus
- The static semantics is given via a first-order type system based on a form of product-types reflecting the (non-commutative) structure of the term

A "Fresh" Approach to Rewriting

- We present an imperative fully typed ($e.g.$ $\grave{a}$ la Church) version of the Rho, a pattern-matching based calculus with side-effects, which we call iRho
- We formulate the static and dynamic semantics of iRho
- A call-by-value deterministic Natural Semantics $\grave{a}$ la Kahn: it immediately suggests how to build an interpreter for the calculus
- The static semantics is given via a first-order type system based on a form of product-types reflecting the (non-commutative) structure of the term
- Access and modify a (monomorphic) typed store, and define fixpoints, control structures, ...
- iRho enjoys determinism of the interpreter, subject reduction, and decidability of type-checking (completely checked by a machine assisted approach, using the Coq proof assistant). Progress and decidability of type-checking are proved by pen and paper

Special Emphasis (iRho)

- A special emphasis has been put in our work on the design of the static/dynamic semantics. We always had in mind the main objectives of a conscious implementor

Special Emphasis (iRho)

- A special emphasis has been put in our work on the design of the static/dynamic semantics. We always had in mind the main objectives of a conscious implementor
  - ⋆ a "sound" machine (the interpreter)
  - ⋆ a "sound" type system (the type checker), such that
    "well typed programs do not go wrong" [Milner],
  - ⋆ both semantics being suitable to be specified with nice mathematics,
  - ⋆ to be implemented with high-level programming languages,
  - ⋆ and to be certified with modern and semi-automatic theorem provers, like Coq

Special Emphasis (iRho)

- A special emphasis has been put in our work on the design of the static/dynamic semantics. We always had in mind the main objectives of a conscious implementor
  - ⋆ a "sound" machine (the interpreter)
  - ⋆ a "sound" type system (the type checker), such that
    "well typed programs do not go wrong" [Milner],
  - ⋆ both semantics being suitable to be specified with nice mathematics,
  - ⋆ to be implemented with high-level programming languages,
  - ⋆ and to be certified with modern and semi-automatic theorem provers, like Coq

- Thus, we have encoded in Coq the static and dynamic semantics of iRho

Special Emphasis (iRho)

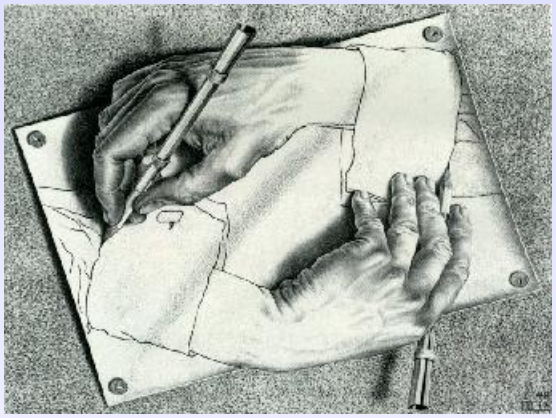
- A special emphasis has been put in our work on the design of the static/dynamic semantics. We always had in mind the main objectives of a conscious implementor
  - ⋆ a "sound" machine (the interpreter)
  - ⋆ a "sound" type system (the type checker), such that
    "well typed programs do not go wrong" [Milner],
  - ⋆ both semantics being suitable to be specified with nice mathematics,
  - ⋆ to be implemented with high-level programming languages,
  - ⋆ and to be certified with modern and semi-automatic theorem provers, like Coq

- Thus, we have encoded in Coq the static and dynamic semantics of iRho
- All subtle aspects, (usually "swept under the rug") on the paper, are here enlightened by the rigid discipline imposed by the Logical Framework of Coq

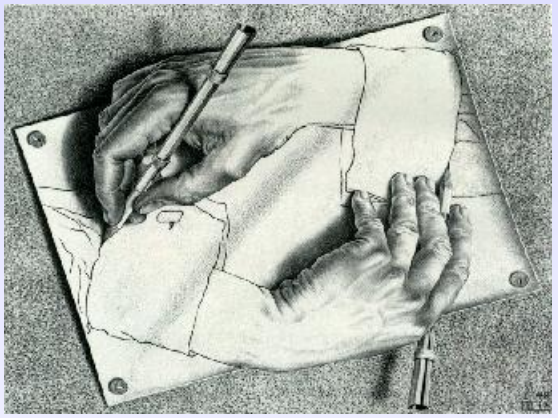
- Often, this process may had a bearing on the design of the static and dynamic semantics

- Often, this process may had a bearing on the design of the static and dynamic semantics
- This (positive) continuous cycle ↻ ↻ between mathematics, ↻ ↻ manual ($i.e.$ pen-and-paper) and ↻ ↻ mechanical proofs, and ↻ ↻ "toy" implementations using high-level languages such Scheme (and back) has been fruitful since the very beginning of our project

- Often, this process may had a bearing on the design of the static and dynamic semantics
- This (positive) continuous cycle ↻ ↻ between mathematics, ↻ ↻ manual ($i.e.$ pen-and-paper) and ↻ ↻ mechanical proofs, and ↻ ↻ "toy" implementations using high-level languages such Scheme (and back) has been fruitful since the very beginning of our project



- Often, this process may had a bearing on the design of the static and dynamic semantics
- This (positive) continuous cycle ↻ ↻ between mathematics, ↻ ↻ manual ($i.e.$ pen-and-paper) and ↻ ↻ mechanical proofs, and ↻ ↻ "toy" implementations using high-level languages such Scheme (and back) has been fruitful since the very beginning of our project



- Although our calculus is rather simple, it is not impossible, in a near future, to scale-up to larger projects, such as the certified implementation of compilers for a "real" programming language of the C family
(Action Recherche Coordonnée INRIA, Concert)

Remark: Linearity is allowed in iRho

This choice induces also a modification in the classical syntactic pattern matching algorithm, since we "hide" the first binding in favor of the second one. The classical syntactic pattern matching algorithm forces both occurrences to be matchable with the same value. As a comparison, both solutions are presented in the table below:

patt $\ll$ term	hide	force
$f(X;X) \ll f(3;4)$	$\theta \triangleq \{4/X\}$	fail
$f(X;X) \ll f(4;4)$	$\theta \triangleq \{4/X\}$	$\theta \triangleq \{4/X\}$

iRho

The Syntax

First step: Functional fRho

The symbol P ranges over the set $\mathcal{P}$ of pseudo-patterns, ($\mathcal{V} \subseteq \mathcal{P} \subseteq \mathcal{T}$)

$$\tau ::= \mathsf{b} \mid \tau \twoheadrightarrow \tau \mid \tau \wedge \tau \qquad Types$$

$$\Delta ::= \emptyset \mid \Delta, X{:}\tau \mid \Delta, f{:}\tau \qquad Contexts$$

$$P ::= f \mid X \mid f\,\overline{P} \mid P; P \qquad Patterns$$

$$A ::= f \mid X \mid P \twoheadrightarrow_{\Delta} A \mid A\,A \mid A; A \qquad Terms$$

First step: Functional fRho

The symbol P ranges over the set $\mathcal{P}$ of pseudo-patterns, $(\mathcal{V} \subseteq \mathcal{P} \subseteq \mathcal{T})$

$$\tau ::= \mathsf{b} \mid \tau \to \tau \mid \tau \wedge \tau \qquad \mathit{Types}$$

$$\Delta ::= \emptyset \mid \Delta, X{:}\tau \mid \Delta, f{:}\tau \qquad \mathit{Contexts}$$

$$P ::= f \mid X \mid f\,\overline{P} \mid P; P \qquad \mathit{Patterns}$$

$$A ::= f \mid X \mid P \to_{\Delta} A \mid A\,A \mid A; A \qquad \mathit{Terms}$$

First step: Functional fRho

The symbol P ranges over the set $\mathcal{P}$ of pseudo-patterns, $(\mathcal{V} \subseteq \mathcal{P} \subseteq \mathcal{T})$

$$\tau ::= \mathsf{b} \mid \tau \to \tau \mid \tau \wedge \tau \qquad \textit{Types}$$

$$\Delta ::= \emptyset \mid \Delta, X{:}\tau \mid \Delta, f{:}\tau \qquad \textit{Contexts}$$

$$P ::= f \mid X \mid f\,\overline{P} \mid P; P \qquad \textit{Patterns}$$

$$A ::= f \mid X \mid P \to_{\Delta} A \mid A\,A \mid A; A \qquad \textit{Terms}$$

First step: Functional fRho

The symbol P ranges over the set $\mathcal{P}$ of pseudo-patterns, $(\mathcal{V} \subseteq \mathcal{P} \subseteq \mathcal{T})$

$$\tau ::= \mathsf{b} \mid \tau \to \tau \mid \tau \wedge \tau \qquad \mathit{Types}$$

$$\Delta ::= \emptyset \mid \Delta, X{:}\tau \mid \Delta, f{:}\tau \qquad \mathit{Contexts}$$

$$P ::= f \mid X \mid f\,\overline{P} \mid P; P \qquad \mathit{Patterns}$$

$$A ::= f \mid X \mid P \to_{\Delta} A \mid A\,A \mid A; A \qquad \mathit{Terms}$$

First step: Functional fRho

The symbol P ranges over the set $\mathcal{P}$ of pseudo-patterns, $(\mathcal{V} \subseteq \mathcal{P} \subseteq \mathcal{T})$

$$\tau ::= \mathsf{b} \mid \tau \to \tau \mid \tau \wedge \tau \qquad \mathit{Types}$$

$$\Delta ::= \emptyset \mid \Delta, X{:}\tau \mid \Delta, f{:}\tau \qquad \mathit{Contexts}$$

$$P ::= f \mid X \mid f\,\overline{P} \mid P; P \qquad \mathit{Patterns}$$

$$A ::= f \mid X \mid P \to_{\Delta} A \mid A\,A \mid A; A \qquad \mathit{Terms}$$

Second step: Imperative iRho

The symbol P ranges over the set $\mathcal{P}$ of pseudo-patterns, ($\mathcal{V} \subseteq \mathcal{P} \subseteq \mathcal{T}$).

τ ::= . . . as in Rho . . . | τ ref — *Types*

Δ ::= . . . as in Rho . . . | $\iota{:}\tau$ — *Contexts*

P ::= . . . as in Rho . . . | ref P — *Patterns*

A ::= . . . as in Rho . . . | ref A | $!A$ | $A := A$ — *Terms*

Second step: Imperative iRho

The symbol P ranges over the set $\mathcal{P}$ of pseudo-patterns, ($\mathcal{V} \subseteq \mathcal{P} \subseteq \mathcal{T}$).

τ ::= . . . as in Rho . . . | τ ref — *Types*

Δ ::= . . . as in Rho . . . | $\iota{:}\tau$ — *Contexts*

P ::= . . . as in Rho . . . | ref P — *Patterns*

A ::= . . . as in Rho . . . | ref A | $!A$ | $A := A$ — *Terms*

Second step: Imperative iRho

The symbol P ranges over the set $\mathcal{P}$ of pseudo-patterns, ($\mathcal{V} \subseteq \mathcal{P} \subseteq \mathcal{T}$).

τ ::= . . . as in Rho . . . | τ ref — *Types*

Δ ::= . . . as in Rho . . . | $\iota{:}\tau$ — *Contexts*

P ::= . . . as in Rho . . . | ref P — *Patterns*

A ::= . . . as in Rho . . . | ref A | $!A$ | $A := A$ — *Terms*

Second step: Imperative iRho

The symbol P ranges over the set $\mathcal{P}$ of pseudo-patterns, ($\mathcal{V} \subseteq \mathcal{P} \subseteq \mathcal{T}$).

τ ::= . . . as in Rho . . . | τ ref $\quad Types$

Δ ::= . . . as in Rho . . . | $\iota{:}\tau$ $\quad Contexts$

P ::= . . . as in Rho . . . | ref P $\quad Patterns$

A ::= . . . as in Rho . . . | ref A | $!A$ | $A := A$ $\quad Terms$

Second step: Imperative iRho

The symbol P ranges over the set $\mathcal{P}$ of pseudo-patterns, ($\mathcal{V} \subseteq \mathcal{P} \subseteq \mathcal{T}$).

τ ::= . . . as in Rho . . . | τ ref — *Types*

Δ ::= . . . as in Rho . . . | $\iota{:}\tau$ — *Contexts*

P ::= . . . as in Rho . . . | ref P — *Patterns*

A ::= . . . as in Rho . . . | ref A | $!A$ | $A := A$ — *Terms*

iRho in a glance

Intuitively iRho deals with references à la Caml $i.e.$:

- **(Deref-types)** The type τ ref is the type of "refs" containing a value of type τ;
- **(Deref-op)** The operator ! is a "dereferencing" operator (goto memory);
- **(Ref-op)** The operator ref is a "referencing" operator (`malloc`);
- **(Assign)** The term $A := B$ is an "assignment" operator, which returns as result the value obtained by evaluating B.

We have not (yet) modeled garbage collection: new locations created during reduction will remain in the store forever

Hint: Talcott, Mason, Morrissett, *et al.*

```
for i < n,
 letrec x_1=ref V_1 and...and x_n=ref V_n in M
 -->
 letrec x_1=ref V_1 and...and x_i=ref V_i in M
 if FV(V_1, ..., V_i)/\{x_(i+1), ..., x_n} = 0.
```

Values and Environments in fRho

- We introduce the set $\mathcal{V}al$ of values and the set of environments $\mathcal{E}nv$

$$A_{\mathsf{v}} \ ::= \ f \mid f\,\overline{A}_{\mathsf{v}} \mid A_{\mathsf{v}}; A_{\mathsf{v}} \mid \langle P \rightarrowtriangle_{\Delta} A \cdot \rho \rangle \mid \langle [P \ll_{\Delta} A_{\mathsf{v}}].B \cdot \rho \rangle$$

- Environments are partial functions from the set of variables to the set of values

$$\rho[X \mapsto A_{\mathsf{v}}](Y) \ \triangleq \ \begin{cases} A_{\mathsf{v}} & \text{if } X \equiv Y \\ \rho(Y) & \text{otherwise} \end{cases}$$

- "Failure-values" $\langle [P \ll_{\Delta} A_{\mathsf{v}}].B \cdot \rho \rangle$ denote failure occurring when we cannot found a correct substitution θ on the free variables of P such that $\theta(P) \equiv A_{\mathsf{v}}$; Failure-values are obtained during the computation when a matching failure occurs. It could in principle be caught by a suitable exception handler

Values and Environments in fRho

- We introduce the set $\mathcal{V}al$ of values and the set of environments $\mathcal{E}nv$

$$A_{\mathsf{v}} \ ::= \ f \mid f\,\overline{A}_{\mathsf{v}} \mid A_{\mathsf{v}}; A_{\mathsf{v}} \mid \langle P \rightarrowtriangle_{\Delta} A \cdot \rho \rangle \mid \langle [P \ll_{\Delta} A_{\mathsf{v}}].B \cdot \rho \rangle$$

- Environments are partial functions from the set of variables to the set of values

$$\rho[X \mapsto A_{\mathsf{v}}](Y) \ \triangleq \ \begin{cases} A_{\mathsf{v}} & \text{if } X \equiv Y \\ \rho(Y) & \text{otherwise} \end{cases}$$

- "Failure-values" $\langle [P \ll_{\Delta} A_{\mathsf{v}}].B \cdot \rho \rangle$ denote failure occurring when we cannot found a correct substitution θ on the free variables of P such that $\theta(P) \equiv A_{\mathsf{v}}$; Failure-values are obtained during the computation when a matching failure occurs. It could in principle be caught by a suitable exception handler

Values and Environments in fRho

- We introduce the set $\mathcal{V}al$ of values and the set of environments $\mathcal{E}nv$

$$A_{\mathsf{v}} \;::=\; f \mid f\,\overline{A}_{\mathsf{v}} \mid A_{\mathsf{v}}; A_{\mathsf{v}} \mid \langle P \rightarrowtriangle_{\Delta} A \cdot \rho \rangle \mid \langle [P \ll_{\Delta} A_{\mathsf{v}}].B \cdot \rho \rangle$$

- Environments are partial functions from the set of variables to the set of values

$$\rho[X \mapsto A_{\mathsf{v}}](Y) \;\triangleq\; \begin{cases} A_{\mathsf{v}} & \text{if } X \equiv Y \\ \rho(Y) & \text{otherwise} \end{cases}$$

- "Failure-values" $\langle [P \ll_{\Delta} A_{\mathsf{v}}].B \cdot \rho \rangle$ denote failure occurring when we cannot found a correct substitution θ on the free variables of P such that $\theta(P) \equiv A_{\mathsf{v}}$; Failure-values are obtained during the computation when a matching failure occurs. It could in principle be caught by a suitable exception handler

Values and Environments and Stores in iRho

- The new set of values is enriched by locations. Moreover we define the set of global stores $\mathcal{S}tore$ (the symbol σ ranges over stores). They are defined below:

$$A_{\mathsf{v}} \;::=\; f \mid f\,\overline{A}_{\mathsf{v}} \mid A_{\mathsf{v}}; A_{\mathsf{v}} \mid \langle P \rightarrowtriangle_{\Delta} A \cdot \rho \rangle \mid \langle [P \ll_{\Delta} A_{\mathsf{v}}] B \cdot \rho \rangle$$

$$\mid \iota \qquad\qquad \text{Imperative Values}$$

- Stores are partial functions from the set $\mathcal{L}$ of locations to the set of values

$$\sigma[\iota \mapsto A_{\mathsf{v}}](\iota') \;\triangleq\; \begin{cases} A_{\mathsf{v}} & \text{if } \iota \equiv \iota' \\ \sigma(\iota') & \text{otherwise} \end{cases}$$

Values and Environments and Stores in iRho

- The new set of values is enriched by locations. Moreover we define the set of global stores $\mathcal{S}tore$ (the symbol σ ranges over stores). They are defined below:

$$A_{\mathsf{v}} \;::=\; f \mid f\,\overline{A}_{\mathsf{v}} \mid A_{\mathsf{v}}; A_{\mathsf{v}} \mid \langle P \twoheadrightarrow_{\Delta} A \cdot \rho \rangle \mid \langle [P \ll_{\Delta} A_{\mathsf{v}}] B \cdot \rho \rangle$$

$$\mid \iota$$

Imperative Values

- Stores are partial functions from the set $\mathcal{L}$ of locations to the set of values

$$\sigma[\iota \mapsto A_{\mathsf{v}}](\iota') \;\triangleq\; \begin{cases} A_{\mathsf{v}} & \text{if } \iota \equiv \iota' \\ \sigma(\iota') & \text{otherwise} \end{cases}$$

Values and Environments and Stores in iRho

- The new set of values is enriched by locations. Moreover we define the set of global stores $\mathcal{S}tore$ (the symbol σ ranges over stores). They are defined below:

$$A_{\mathsf{v}} \;::=\; f \mid f\,\overline{A}_{\mathsf{v}} \mid A_{\mathsf{v}}; A_{\mathsf{v}} \mid \langle P \twoheadrightarrow_{\Delta} A \cdot \rho \rangle \mid \langle [P \ll_{\Delta} A_{\mathsf{v}}] B \cdot \rho \rangle$$

$$\mid \iota \qquad\qquad \text{Imperative Values}$$

- Stores are partial functions from the set $\mathcal{L}$ of locations to the set of values

$$\sigma[\iota \mapsto A_{\mathsf{v}}](\iota') \;\triangleq\; \begin{cases} A_{\mathsf{v}} & \text{if } \iota \equiv \iota' \\ \sigma(\iota') & \text{otherwise} \end{cases}$$

Natural Semantics for fRho

- We define a call-by-value optimistic operational semantics via a natural proof deduction system
- The present interpreter machine is optimistic since it gives a result if at least one computation does not produce a failure-value: of course other choices are possible, like $e.g.$ a "pessimistic" machine which stops if at least one failure-value occurs
- The purpose of the deduction system is to map every expression into a normal form, $i.e.$ an irreducible term in weak head normal form. The present strategy is call-by-value since it does not work under plain abstractions ($i.e.$ $P \rightarrow\!\!\triangleright_{\Delta} A$).

Optimistic- *vs.* Pessimistic-machines

Our machine is optimistic, in the sense that it does not abort computations if a matching failure occurs, keeping in the final result the fact that one computation goes wrong, and hoping that at least one computation succeeds. This is clearly visible when the semantics deals with structures. The choice of killing the computation once a failure-value is produced leads to a pessimistic machine

$$\frac{\vdots}{\emptyset \vdash (3 \rightarrowtriangle 3; 4 \rightarrowtriangle 4)\, 4 \Downarrow^{\mathsf{opt}}_{\mathsf{val}} \langle [3 \ll 4].3 \cdot \emptyset \rangle; 4}$$

$$\frac{\vdots}{\emptyset \vdash (3 \rightarrowtriangle 3; 4 \rightarrowtriangle 4)\, 4 \Downarrow^{\mathsf{pex}}_{\mathsf{val}} \langle [3 \ll 4].3 \cdot \emptyset \rangle}$$

A pessimistic-machine can be quite easily produced by an optimistic one dealing with propagation of failure-values

iRho:

Reductions in a glance

An Imperative Term in iRho

- Take the imperative term (types omitted)

$$A \triangleq (f(X,Y) \rightarrowtriangle (3 \rightarrowtriangle X = !Y)\, !X)\, f(\mathsf{ref}\, 3; \mathsf{ref}\, 4)$$

- We evaluate this term in the empty store $\emptyset$ and the empty environment $\emptyset$

- The result would be?

An Imperative Term in iRho

- Take the imperative term (types omitted)

$$A \triangleq (f(X,Y) \twoheadrightarrow (3 \twoheadrightarrow X = !Y)\, !X)\; f(\mathsf{ref}\, 3; \mathsf{ref}\, 4)$$

- We evaluate this term in the empty store $\emptyset$ and the empty environment $\emptyset$

- The result would be?

$$\emptyset \cdot \emptyset \;\vdash\; A \Downarrow_{\mathsf{val}} 4 \cdot [\iota_0 \mapsto 4, \iota_1 \mapsto 4]$$

iRho

Natural Semantics

Natural Judgments in iRho

- The semantics is given in terms of three judgments

$$\sigma \cdot \rho \ \vdash \ A \Downarrow_{\mathsf{val}} A_{\mathsf{v}} \cdot \sigma'$$

$$\sigma \ \vdash \ \langle A_{\mathsf{v}} \cdot B_{\mathsf{v}} \rangle \Downarrow_{\mathsf{call}} C_{\mathsf{v}} \cdot \sigma'$$

$$\sigma \cdot \rho \ \vdash \ \langle A \cdot A_{\mathsf{v}} \rangle \Downarrow_{\mathsf{match}} \rho'$$

- The first judgment evaluates a term in iRho, the second apply one value to another producing a result value and a new store, and the last judgment updates a correct environment obtained by matching a term against a value

ALL CALL & MATCH SEXY

Value Reduction $\Downarrow_{\mathsf{val}}$ for Rho *vs.* iRho

$$\frac{\rho \vdash A \Downarrow_{\mathsf{val}} A_{\mathsf{v}} \quad \rho \vdash B \Downarrow_{\mathsf{val}} B_{\mathsf{v}} \quad \vdash \langle A_{\mathsf{v}} \cdot B_{\mathsf{v}} \rangle \Downarrow_{\mathsf{call}} C_{\mathsf{v}}}{\rho \vdash A\, B \Downarrow_{\mathsf{val}} C_{\mathsf{v}}}\ (Red{-}\rho_{\mathsf{v}})$$

$$\frac{\begin{array}{l}\sigma_0 \cdot \rho \vdash A \Downarrow_{\mathsf{val}} A_{\mathsf{v}} \cdot \sigma_1 \\ \sigma_1 \cdot \rho \vdash B \Downarrow_{\mathsf{val}} B_{\mathsf{v}} \cdot \sigma_2 \quad \sigma_2 \vdash \langle A_{\mathsf{v}} \cdot B_{\mathsf{v}} \rangle \Downarrow_{\mathsf{call}} C_{\mathsf{v}} \cdot \sigma_3\end{array}}{\sigma_0 \cdot \rho \vdash A\, B \Downarrow_{\mathsf{val}} C_{\mathsf{v}} \cdot \sigma_3}\ (Red{-}\rho_{\mathsf{v}})$$

Value Reduction $\Downarrow_{\mathsf{val}}$ for iRho II

$$\frac{\iota \in \mathsf{Dom}(\sigma_1) \qquad \sigma_0 \cdot \rho \vdash A \Downarrow_{\mathsf{val}} \iota \cdot \sigma_1}{\sigma_0 \cdot \rho \vdash\, !A \Downarrow_{\mathsf{val}} \sigma_1(\iota) \cdot \sigma_1}\ (Red{-}!)$$

$$\frac{\begin{array}{l}\iota \in \mathsf{Dom}(\sigma_1) \\ \sigma_0 \cdot \rho \vdash A \Downarrow_{\mathsf{val}} \iota \cdot \sigma_1 \qquad \sigma_1 \cdot \rho \vdash B \Downarrow_{\mathsf{val}} B_{\mathsf{v}} \cdot \sigma_2\end{array}}{\sigma_0 \cdot \rho \vdash A := B \Downarrow_{\mathsf{val}} B_{\mathsf{v}} \cdot \sigma_2[\iota \mapsto B_{\mathsf{v}}]}\ (Red{-}:=)$$

$$\frac{\iota \notin \mathsf{Dom}(\sigma_1) \qquad \sigma_0 \cdot \rho \vdash A \Downarrow_{\mathsf{val}} A_{\mathsf{v}} \cdot \sigma_1}{\sigma_0 \cdot \rho \vdash \mathsf{ref}\, A \Downarrow_{\mathsf{val}} \iota \cdot \sigma_1[\iota \mapsto A_{\mathsf{v}}]}\ (Red{-}\mathsf{ref}\,)$$

$\Downarrow_{\text{call}}$ and $\Downarrow_{\text{match}}$ for iRho:

Straightforward extension

for the imperative case

Call Reduction $\Downarrow_{\mathsf{call}}$ for Rho I

$$\frac{\rho \vdash \langle P \cdot B_{\mathsf{v}} \rangle \Downarrow_{\mathsf{match}} \rho' \qquad \rho' \vdash A \Downarrow_{\mathsf{val}} A_{\mathsf{v}}}{\vdash \langle \langle P{:}\Delta \twoheadrightarrow A \cdot \rho \rangle \cdot B_{\mathsf{v}} \rangle \Downarrow_{\mathsf{call}} A_{\mathsf{v}}} \; (Call{-}FunOk)$$

$$\frac{\nexists \rho'.\ \rho \vdash \langle P \cdot B_{\mathsf{v}} \rangle \Downarrow_{\mathsf{match}} \rho'}{\vdash \langle \langle P{:}\Delta \twoheadrightarrow A \cdot \rho \rangle \cdot B_{\mathsf{v}} \rangle \Downarrow_{\mathsf{call}} \langle [P \ll_{\Delta} B_{\mathsf{v}}].A \cdot \rho \rangle} \; (Call{-}FunK$$

Call Reduction $\Downarrow_{\mathsf{call}}$ for Rho II

$$\frac{\vdash \langle A_{\mathsf{v}} \cdot C_{\mathsf{v}} \rangle \Downarrow_{\mathsf{call}} D_{\mathsf{v}} \qquad \vdash \langle B_{\mathsf{v}} \cdot C_{\mathsf{v}} \rangle \Downarrow_{\mathsf{call}} E_{\mathsf{v}}}{\vdash \langle (A_{\mathsf{v}}; B_{\mathsf{v}}) \cdot C_{\mathsf{v}} \rangle \Downarrow_{\mathsf{call}} D_{\mathsf{v}}; E_{\mathsf{v}}} \; (Call\text{-}Struct)$$

$$\frac{}{\vdash \langle a\, \overline{A}_{\mathsf{v}} \cdot B_{\mathsf{v}} \rangle \Downarrow_{\mathsf{call}} a\, \overline{A}_{\mathsf{v}}\, B_{\mathsf{v}}} \; (Call\text{-}Algbr)$$

Matching Reduction $\Downarrow_{\mathsf{match}}$ for Rho

$$\frac{}{\rho \vdash \langle a \cdot a \rangle \Downarrow_{\mathsf{match}} \rho} \ (Match{-}Const)$$

$$\frac{}{\rho \vdash \langle X \cdot A_{\mathsf{v}} \rangle \Downarrow_{\mathsf{match}} \rho[X \mapsto A_{\mathsf{v}}]} \ (Match{-}Var)$$

$$\frac{\rho_0 \vdash \langle A \cdot A_{\mathsf{v}} \rangle \Downarrow_{\mathsf{match}} \rho_1 \quad \rho_1 \vdash \langle B \cdot B_{\mathsf{v}} \rangle \Downarrow_{\mathsf{match}} \rho_2 \qquad ; \in \{'';'' \ \ ''@''\}}{\rho_0 \vdash \langle A; B \cdot A_{\mathsf{v}}; B_{\mathsf{v}} \rangle \Downarrow_{\mathsf{match}} \rho_2} \ (Match{-}Pair)$$

SEXY

$$\frac{\iota \in \mathsf{Dom}(\sigma_1) \qquad \sigma_0 \cdot \rho \vdash A \Downarrow_{\mathsf{val}} \iota \cdot \sigma_1}{\sigma_0 \cdot \rho \vdash !A \Downarrow_{\mathsf{val}} \sigma_1(\iota) \cdot \sigma_1} \; (Red{-}!)$$

SEXY

$$\frac{\iota \in \mathsf{Dom}(\sigma_1) \qquad \sigma_0 \cdot \rho \vdash A \Downarrow_{\mathsf{val}} \iota \cdot \sigma_1}{\sigma_0 \cdot \rho \vdash\, !A \Downarrow_{\mathsf{val}} \sigma_1(\iota) \cdot \sigma_1} \; (Red{-}!)$$

$$\frac{\iota \notin \mathsf{Dom}(\sigma_1) \qquad \sigma_0 \cdot \rho \vdash A \Downarrow_{\mathsf{val}} A_{\mathsf{v}} \cdot \sigma_1}{\sigma_0 \cdot \rho \vdash \mathsf{ref}\, A \Downarrow_{\mathsf{val}} \iota \cdot \sigma_1[\iota \mapsto A_{\mathsf{v}}]} \; (Red{-}\mathsf{ref}\,)$$

SEXY

$$\frac{\iota \in \mathsf{Dom}(\sigma_1) \qquad \sigma_0 \cdot \rho \vdash A \Downarrow_{\mathsf{val}} \iota \cdot \sigma_1}{\sigma_0 \cdot \rho \vdash\, !A \Downarrow_{\mathsf{val}} \sigma_1(\iota) \cdot \sigma_1} \; (Red{-}!)$$

$$\frac{\iota \notin \mathsf{Dom}(\sigma_1) \qquad \sigma_0 \cdot \rho \vdash A \Downarrow_{\mathsf{val}} A_{\mathsf{v}} \cdot \sigma_1}{\sigma_0 \cdot \rho \vdash \mathsf{ref}\, A \Downarrow_{\mathsf{val}} \iota \cdot \sigma_1[\iota \mapsto A_{\mathsf{v}}]} \; (Red{-}\mathsf{ref}\,)$$

SEXY

$$\frac{\iota \in \mathsf{Dom}(\sigma_1) \qquad \sigma_0 \cdot \rho \vdash \mathbf{A} \Downarrow_{\mathsf{val}} \iota \cdot \sigma_1}{\sigma_0 \cdot \rho \vdash !A \Downarrow_{\mathsf{val}} \sigma_1(\iota) \cdot \sigma_1} \; (Red{-}!)$$

$$\frac{\iota \notin \mathsf{Dom}(\sigma_1) \qquad \sigma_0 \cdot \rho \vdash A \Downarrow_{\mathsf{val}} A_{\mathsf{v}} \cdot \sigma_1}{\sigma_0 \cdot \rho \vdash \mathsf{ref}\, A \Downarrow_{\mathsf{val}} \iota \cdot \sigma_1[\iota \mapsto A_{\mathsf{v}}]} \; (Red{-}\mathsf{ref}\,)$$

SEXY

$$\frac{\iota \in \mathsf{Dom}(\sigma_1) \qquad \sigma_0 \cdot \rho \vdash \mathbf{A} \Downarrow_{\mathsf{val}} \iota \cdot \sigma_1}{\sigma_0 \cdot \rho \vdash \,!\mathbf{A} \Downarrow_{\mathsf{val}} \sigma_1(\iota) \cdot \sigma_1} \; (Red{-}!)$$

$$\frac{\iota \notin \mathsf{Dom}(\sigma_1) \qquad \sigma_0 \cdot \rho \vdash A \Downarrow_{\mathsf{val}} A_{\mathsf{v}} \cdot \sigma_1}{\sigma_0 \cdot \rho \vdash \mathsf{ref}\, A \Downarrow_{\mathsf{val}} \iota \cdot \sigma_1[\iota \mapsto A_{\mathsf{v}}]} \; (Red{-}\mathsf{ref}\,)$$

SEXY

$$\frac{\iota \in \mathsf{Dom}(\sigma_1) \qquad \sigma_0 \cdot \rho \vdash \mathbf{A} \Downarrow_{\mathsf{val}} \iota \cdot \sigma_1}{\sigma_0 \cdot \rho \vdash !\mathbf{A} \Downarrow_{\mathsf{val}} \sigma_1(\iota) \cdot \sigma_1} \; (Red{-}!)$$

$$\frac{\iota \notin \mathsf{Dom}(\sigma_1) \qquad \sigma_0 \cdot \rho \vdash A \Downarrow_{\mathsf{val}} A_{\mathsf{v}} \cdot \sigma_1}{\sigma_0 \cdot \rho \vdash \mathsf{ref}\, A \Downarrow_{\mathsf{val}} \iota \cdot \sigma_1[\iota \mapsto A_{\mathsf{v}}]} \; (Red{-}\mathsf{ref}\,)$$

SEXY

$$\frac{\iota \in \mathsf{Dom}(\sigma_1) \qquad \sigma_0 \cdot \rho \vdash \mathbf{A} \Downarrow_{\mathsf{val}} \iota \cdot \sigma_1}{\sigma_0 \cdot \rho \vdash !\mathbf{A} \Downarrow_{\mathsf{val}} \sigma_1(\iota) \cdot \sigma_1} \; (Red{-}!)$$

$$\frac{\iota \notin \mathsf{Dom}(\sigma_1) \qquad \sigma_0 \cdot \rho \vdash \mathbf{A} \Downarrow_{\mathsf{val}} \mathbf{A}_{\mathsf{v}} \cdot \sigma_1}{\sigma_0 \cdot \rho \vdash \mathsf{ref}\, A \Downarrow_{\mathsf{val}} \iota \cdot \sigma_1[\iota \mapsto A_{\mathsf{v}}]} \; (Red{-}\mathsf{ref}\,)$$

SEXY

$$\frac{\iota \in \mathsf{Dom}(\sigma_1) \qquad \sigma_0 \cdot \rho \vdash \mathbf{A} \Downarrow_{\mathsf{val}} \iota \cdot \sigma_1}{\sigma_0 \cdot \rho \vdash !\mathbf{A} \Downarrow_{\mathsf{val}} \sigma_1(\iota) \cdot \sigma_1} \; (Red{-}!)$$

$$\frac{\iota \notin \mathsf{Dom}(\sigma_1) \qquad \sigma_0 \cdot \rho \vdash \mathbf{A} \Downarrow_{\mathsf{val}} \mathbf{A}_{\mathsf{v}} \cdot \sigma_1}{\sigma_0 \cdot \rho \vdash \mathsf{ref}\ \mathbf{A} \Downarrow_{\mathsf{val}} \iota \cdot \sigma_1[\iota \mapsto \mathbf{A}_{\mathsf{v}}]} \; (Red{-}\mathsf{ref}\ )$$

SEXY

$$\frac{\iota \in \mathsf{Dom}(\sigma_1) \qquad \sigma_0 \cdot \rho \vdash A \Downarrow_{\mathsf{val}} \iota \cdot \sigma_1 \qquad \sigma_1 \cdot \rho \vdash B \Downarrow_{\mathsf{val}} B_{\mathsf{v}} \cdot \sigma_2}{\sigma_0 \cdot \rho \vdash A := B \Downarrow_{\mathsf{val}} B_{\mathsf{v}} \cdot \sigma_2[\iota \mapsto B_{\mathsf{v}}]}\ (Red{-}{:=})$$

SEXY

$$\frac{\iota \in \mathsf{Dom}(\sigma_1) \qquad \sigma_0 \cdot \rho \vdash A \Downarrow_{\mathsf{val}} \iota \cdot \sigma_1 \qquad \sigma_1 \cdot \rho \vdash B \Downarrow_{\mathsf{val}} B_{\mathsf{v}} \cdot \sigma_2}{\sigma_0 \cdot \rho \vdash A := B \Downarrow_{\mathsf{val}} B_{\mathsf{v}} \cdot \sigma_2[\iota \mapsto B_{\mathsf{v}}]} \; (Red{-}{:=})$$

$$\frac{\iota \in \mathsf{Dom}(\sigma) \qquad \sigma(\iota) \equiv A_{\mathsf{v}} \qquad \sigma \cdot \rho \vdash \langle P \cdot A_{\mathsf{v}} \rangle \Downarrow_{\mathsf{match}} \rho'}{\sigma \cdot \rho \vdash \langle \mathsf{ref}\, P \cdot \iota \rangle \Downarrow_{\mathsf{match}} \rho'} \; (Match{-}Ref)$$

SEXY

$$\frac{\iota \in \mathsf{Dom}(\sigma_1) \qquad \sigma_0 \cdot \rho \vdash \mathbf{A} \Downarrow_{\mathsf{val}} \iota \cdot \sigma_1 \qquad \sigma_1 \cdot \rho \vdash B \Downarrow_{\mathsf{val}} B_{\mathsf{v}} \cdot \sigma_2}{\sigma_0 \cdot \rho \vdash A := B \Downarrow_{\mathsf{val}} B_{\mathsf{v}} \cdot \sigma_2[\iota \mapsto B_{\mathsf{v}}]} \; (Red{-}{:=})$$

$$\frac{\iota \in \mathsf{Dom}(\sigma) \qquad \sigma(\iota) \equiv A_{\mathsf{v}} \qquad \sigma \cdot \rho \vdash \langle P \cdot A_{\mathsf{v}} \rangle \Downarrow_{\mathsf{match}} \rho'}{\sigma \cdot \rho \vdash \langle \mathsf{ref}\, P \cdot \iota \rangle \Downarrow_{\mathsf{match}} \rho'} \; (Match{-}Ref)$$

SEXY

$$\frac{\iota \in \mathsf{Dom}(\sigma_1) \quad \sigma_0 \cdot \rho \vdash \mathbf{A} \Downarrow_{\mathsf{val}} \iota \cdot \sigma_1 \qquad \sigma_1 \cdot \rho \vdash B \Downarrow_{\mathsf{val}} B_{\mathsf{v}} \cdot \sigma_2}{\sigma_0 \cdot \rho \vdash A := B \Downarrow_{\mathsf{val}} B_{\mathsf{v}} \cdot \sigma_2[\iota \mapsto B_{\mathsf{v}}]}\ (Red{-}:=)$$

$$\frac{\iota \in \mathsf{Dom}(\sigma) \quad \sigma(\iota) \equiv A_{\mathsf{v}} \quad \sigma \cdot \rho \vdash \langle P \cdot A_{\mathsf{v}} \rangle \Downarrow_{\mathsf{match}} \rho'}{\sigma \cdot \rho \vdash \langle \mathsf{ref}\, P \cdot \iota \rangle \Downarrow_{\mathsf{match}} \rho'}\ (Match{-}Ref)$$

SEXY

$$\frac{\iota \in \mathsf{Dom}(\sigma_1) \qquad \sigma_0 \cdot \rho \vdash \mathbf{A} \Downarrow_{\mathsf{val}} \iota \cdot \sigma_1 \qquad \sigma_1 \cdot \rho \vdash \mathbf{B} \Downarrow_{\mathsf{val}} \mathbf{B_v} \cdot \sigma_2}{\sigma_0 \cdot \rho \vdash A := B \Downarrow_{\mathsf{val}} B_{\mathsf{v}} \cdot \sigma_2[\iota \mapsto B_{\mathsf{v}}]} \; (Red{-}{:=})$$

$$\frac{\iota \in \mathsf{Dom}(\sigma) \qquad \sigma(\iota) \equiv A_{\mathsf{v}} \qquad \sigma \cdot \rho \vdash \langle P \cdot A_{\mathsf{v}} \rangle \Downarrow_{\mathsf{match}} \rho'}{\sigma \cdot \rho \vdash \langle \mathsf{ref}\, P \cdot \iota \rangle \Downarrow_{\mathsf{match}} \rho'} \; (Match{-}Ref)$$

SEXY

$$\frac{\iota \in \mathsf{Dom}(\sigma_1) \qquad \sigma_0 \cdot \rho \vdash \mathbf{A} \Downarrow_{\mathsf{val}} \iota \cdot \sigma_1 \qquad \sigma_1 \cdot \rho \vdash \mathbf{B} \Downarrow_{\mathsf{val}} \mathbf{B}_{\mathsf{v}} \cdot \sigma_2}{\sigma_0 \cdot \rho \vdash \mathbf{A} := \mathbf{B} \Downarrow_{\mathsf{val}} \mathbf{B}_{\mathsf{v}} \cdot \sigma_2[\iota \mapsto \mathbf{B}_{\mathsf{v}}]} \; (Red{-}{:=})$$

$$\frac{\iota \in \mathsf{Dom}(\sigma) \qquad \sigma(\iota) \equiv A_{\mathsf{v}} \qquad \sigma \cdot \rho \vdash \langle P \cdot A_{\mathsf{v}} \rangle \Downarrow_{\mathsf{match}} \rho'}{\sigma \cdot \rho \vdash \langle \mathsf{ref}\, P \cdot \iota \rangle \Downarrow_{\mathsf{match}} \rho'} \; (Match{-}Ref)$$

SEXY

$$\frac{\iota \in \mathsf{Dom}(\sigma_1) \qquad \sigma_0 \cdot \rho \vdash \mathbf{A} \Downarrow_{\mathsf{val}} \iota \cdot \sigma_1 \qquad \sigma_1 \cdot \rho \vdash \mathbf{B} \Downarrow_{\mathsf{val}} \mathbf{B}_{\mathsf{v}} \cdot \sigma_2}{\sigma_0 \cdot \rho \vdash \mathbf{A} := \mathbf{B} \Downarrow_{\mathsf{val}} \mathbf{B}_{\mathsf{v}} \cdot \sigma_2[\iota \mapsto \mathbf{B}_{\mathsf{v}}]} \; (Red{-}{:=})$$

$$\frac{\iota \in \mathsf{Dom}(\sigma) \qquad \sigma(\iota) \equiv \mathbf{A}_{\mathsf{v}} \qquad \sigma \cdot \rho \vdash \langle P \cdot A_{\mathsf{v}} \rangle \Downarrow_{\mathsf{match}} \rho'}{\sigma \cdot \rho \vdash \langle \mathsf{ref}\, P \cdot \iota \rangle \Downarrow_{\mathsf{match}} \rho'} \; (Match{-}Ref)$$

SEXY

$$\frac{\iota \in \mathsf{Dom}(\sigma_1) \qquad \sigma_0 \cdot \rho \vdash \mathbf{A} \Downarrow_{\mathsf{val}} \iota \cdot \sigma_1 \qquad \sigma_1 \cdot \rho \vdash \mathbf{B} \Downarrow_{\mathsf{val}} \mathbf{B}_{\mathsf{v}} \cdot \sigma_2}{\sigma_0 \cdot \rho \vdash \mathbf{A} := \mathbf{B} \Downarrow_{\mathsf{val}} \mathbf{B}_{\mathsf{v}} \cdot \sigma_2[\iota \mapsto \mathbf{B}_{\mathsf{v}}]} \; (Red{-}{:=})$$

$$\frac{\iota \in \mathsf{Dom}(\sigma) \qquad \sigma(\iota) \equiv \mathbf{A}_{\mathsf{v}} \qquad \sigma \cdot \rho \vdash \langle \mathbf{P} \cdot \mathbf{A}_{\mathsf{v}} \rangle \Downarrow_{\mathsf{match}} \rho'}{\sigma \cdot \rho \vdash \langle \mathsf{ref}\, P \cdot \iota \rangle \Downarrow_{\mathsf{match}} \rho'} \; (Match{-}Ref)$$

SEXY

$$\frac{\iota \in \mathsf{Dom}(\sigma_1) \qquad \sigma_0 \cdot \rho \vdash \mathbf{A} \Downarrow_{\mathsf{val}} \iota \cdot \sigma_1 \qquad \sigma_1 \cdot \rho \vdash \mathbf{B} \Downarrow_{\mathsf{val}} \mathbf{B}_{\mathsf{v}} \cdot \sigma_2}{\sigma_0 \cdot \rho \vdash \mathbf{A} := \mathbf{B} \Downarrow_{\mathsf{val}} \mathbf{B}_{\mathsf{v}} \cdot \sigma_2[\iota \mapsto \mathbf{B}_{\mathsf{v}}]} \; (Red{-}{:=})$$

$$\frac{\iota \in \mathsf{Dom}(\sigma) \qquad \sigma(\iota) \equiv \mathbf{A}_{\mathsf{v}} \qquad \sigma \cdot \rho \vdash \langle \mathbf{P} \cdot \mathbf{A}_{\mathsf{v}} \rangle \Downarrow_{\mathsf{match}} \rho'}{\sigma \cdot \rho \vdash \langle \mathsf{ref}\, \mathbf{P} \cdot \iota \rangle \Downarrow_{\mathsf{match}} \rho'} \; (Match{-}Ref)$$

The Uncle Pat and the Lady Match Still at Work...

An Imperative Natural Derivation

Take the imperative term

$$(f(X,Y) \twoheadrightarrow (3 \twoheadrightarrow X := !Y)\, !X)\; f(\mathsf{ref}\, 3; \mathsf{ref}\, 4)$$

with $\sigma_0 \triangleq [\iota_0 \mapsto 3][\iota_1 \mapsto 4]$, and $\sigma_1 \triangleq \sigma_0[\iota_0 \mapsto 4]$, and $\rho_0 \triangleq [X \mapsto \iota_0][Y \mapsto \iota_1]$.

$$\dfrac{\dfrac{\iota_0 \in \mathsf{Dom}(\sigma_0)}{\sigma_0 \cdot \rho_0 \vdash X \Downarrow_{\mathsf{val}} \iota_0 \cdot \sigma_0} \quad \dfrac{\iota_1 \in \mathsf{Dom}(\sigma_0)}{\sigma_0 \cdot \rho_0 \vdash Y \Downarrow_{\mathsf{val}} \iota_1 \cdot \sigma_0}}{\sigma_0 \cdot \rho_0 \vdash !Y \Downarrow_{\mathsf{val}} 4 \cdot \sigma_0}$$

$$\dfrac{\sigma_0 \cdot \rho_0 \vdash X := !Y \Downarrow_{\mathsf{val}} 4 \cdot \sigma_1 \quad \sigma_0 \cdot \rho_0 \vdash \langle 3 \cdot 3 \rangle \Downarrow_{\mathsf{match}} \rho_0}{\sigma_0 \vdash \langle \langle 3 \rightarrowtriangle X := !Y \cdot \rho_0 \rangle \cdot 3 \rangle \Downarrow_{\mathsf{call}} 4 \cdot \sigma_1}$$

$$\dfrac{\sigma_0 \cdot \rho_0 \vdash !X \Downarrow_{\mathsf{val}} 3 \cdot \sigma_0 \quad \sigma_0 \cdot \rho_0 \vdash 3 \rightarrowtriangle X := !Y \Downarrow_{\mathsf{val}} \langle 3 \rightarrowtriangle X := !Y \cdot \rho_0 \rangle \cdot \sigma_0}{\sigma_0, \rho_0 \vdash (3 \rightarrowtriangle X := !Y)\, !X \Downarrow_{\mathsf{call}} 4 \cdot \sigma_1}$$

$$\dfrac{\sigma_0 \cdot \emptyset \vdash \langle f(X,Y) \cdot f(\iota_0; \iota_1) \rangle \Downarrow_{\mathsf{match}} \rho_0}{\sigma_0 \cdot \emptyset \vdash \langle \langle (f(X,Y) \rightarrowtriangle (3 \rightarrowtriangle X := !Y)\, !X) \cdot \emptyset \rangle \cdot f(\iota_0; \iota_1) \rangle \Downarrow_{\mathsf{call}} 4 \cdot \sigma_1}$$

$$\dfrac{\emptyset \cdot \emptyset \vdash f(\mathsf{ref}\, 3; \mathsf{ref}\, 4) \Downarrow_{\mathsf{val}} f(\iota_0; \iota_1) \cdot \sigma_0 \quad \emptyset \cdot \emptyset \vdash (f(X,Y) \rightarrowtriangle (3 \rightarrowtriangle X := !Y)\, !X) \Downarrow_{\mathsf{val}} \langle (f(X,Y) \rightarrowtriangle (3 \rightarrowtriangle X := !Y)\, !X) \cdot \emptyset \rangle \cdot \emptyset}{\emptyset \cdot \emptyset \vdash (f(X,Y) \rightarrowtriangle (3 \rightarrowtriangle X := !Y)\, !X)\, f(\mathsf{ref}\, 3; \mathsf{ref}\, 4) \Downarrow_{\mathsf{val}} 4 \cdot \sigma_1}$$

Aliases

$$\begin{array}{lcl}
\mathsf{let}\ \ P \ll A\ \ \mathsf{in}\ \ B & \triangleq & (P \rightarrowtriangle B)\, A \\
\mathsf{if}\ \ A\ \ \mathsf{then}\, B\ \ \mathsf{else}\ \ C & \triangleq & (\mathsf{true} \rightarrowtriangle B; \mathsf{false} \rightarrowtriangle C)\, A \\
\mathsf{neg} & \triangleq & (\mathsf{true} \rightarrowtriangle \mathsf{false}; \mathsf{false} \rightarrowtriangle \mathsf{true}) \\
A\,; B & \triangleq & (X \rightarrowtriangle B)\, A \qquad X \notin \mathsf{Fv}(B) \\
(X_1; \ldots X_n) := (A_1; \ldots A_n) & \triangleq & X_1 := A_1\,; \ldots\,; X_n := A_n \\
!A & \triangleq & (\mathsf{ref}\, X \rightarrowtriangle X)\, A
\end{array}$$

EX

Aliases

$$\begin{array}{lcl}
\mathsf{let}\ \ P \ll A\ \ \mathsf{in}\ \ B & \triangleq & (P \twoheadrightarrow B)\, A \\
\mathsf{if}\ \ A\ \ \mathsf{then}\, B\ \ \mathsf{else}\ \ C & \triangleq & (\mathsf{true} \twoheadrightarrow B; \mathsf{false} \twoheadrightarrow C)\, A \\
\mathsf{neg} & \triangleq & (\mathsf{true} \twoheadrightarrow \mathsf{false}; \mathsf{false} \twoheadrightarrow \mathsf{true}) \\
A\,; B & \triangleq & (X \twoheadrightarrow B)\, A \qquad X \notin \mathsf{Fv}(B) \\
(X_1; \ldots X_n) := (A_1; \ldots A_n) & \triangleq & X_1 := A_1\,; \ldots\,; X_n := A_n \\
!A & \triangleq & (\mathsf{ref}\, X \twoheadrightarrow X)\, A
\end{array}$$

EX

Computing a Negation Normal Form

This function is used in implementing *decision procedures*, present in almost all model checkers. The processed input is a implication-free languages of formulas with generating grammar:

$$\phi ::= \mathsf{p} \mid \mathsf{and}(\phi;\phi) \mid \mathsf{or}(\phi;\phi) \mid \mathsf{not}(\phi)$$

We present three encodings: the first is purely functional, since it just makes use of recursion via self-application. The second is imperative: the function is shared via a pointer and recursion is achieved via dereferencing. The third encoding manipulates formulas with sharing (back-pointers to shared-subtrees). Type decorations are omitted.

Functional, $\mathbb{F}$

- This encoding uses the Samuel Kamin's *self application* mechanism to implement the recursion via the *object itself*. This technique is well-understood in object-oriented languages, where message passing is formalized via the "dot notation". More precisely, we let the following notation as syntactic sugar in iRho, $i.e.$ obj.meth $\triangleq$ obj(meth(obj)).

- The function nnf is defined as follows, and called as in, $e.g.$ (nnf.fix) ϕ.

$$\mathsf{nnf} \triangleq \mathsf{fix}(\mathsf{Self}) \rightarrowtriangle \left(\begin{array}{lcl} \mathsf{p} & \rightarrowtriangle & \mathsf{p}; \\ \mathsf{not}(\mathsf{not}(X)) & \rightarrowtriangle & \mathsf{Self.fix}(X); \\ \mathsf{not}(\mathsf{or}(X;Y)) & \rightarrowtriangle & \mathsf{and}(\mathsf{Self.fix}(\mathsf{not}(X)); \mathsf{Self.fix}(\mathsf{not}(Y))); \\ \mathsf{not}(\mathsf{and}(X;Y)) & \rightarrowtriangle & \mathsf{or}(\mathsf{Self.fix}(\mathsf{not}(X)); \mathsf{Self.fix}(\mathsf{not}(Y))); \\ \mathsf{and}(X;Y) & \rightarrowtriangle & \mathsf{and}(\mathsf{Self.fix}(X); \mathsf{Self.fix}(Y)); \\ \mathsf{or}(X;Y) & \rightarrowtriangle & \mathsf{or}(\mathsf{Self.fix}(X); \mathsf{Self.fix}(Y)) \end{array} \right)$$

Imperative, I

This imperative encoding uses a variable Self which contains a pointer to the recursive code: here the recursion is achieved directly via pointer dereferencing, assignment and classical imperative fixed-point in order to implement recursion. Given the constant dummy, the function nnf is defined as follows:

$$\mathsf{nnf} \quad \triangleq \quad \mathsf{SELF} \rightarrowtriangle \left(\begin{array}{lcl} \mathsf{p} & \rightarrowtriangle & \mathsf{p}; \\ \mathsf{not}(\mathsf{not}(X)) & \rightarrowtriangle & \mathsf{SELF}(X); \\ \mathsf{not}(\mathsf{or}(X;Y)) & \rightarrowtriangle & \mathsf{and}(\mathsf{SELF}(\mathsf{not}(X)); \mathsf{SELF}(\mathsf{not}(Y))); \\ \mathsf{not}(\mathsf{and}(X;Y)) & \rightarrowtriangle & \mathsf{or}(\mathsf{SELF}(\mathsf{not}(X)); \mathsf{SELF}(\mathsf{not}(Y))); \\ \mathsf{and}(X;Y) & \rightarrowtriangle & \mathsf{and}(\mathsf{SELF}(X); \mathsf{SELF}(Y)); \\ \mathsf{or}(X;Y) & \rightarrowtriangle & \mathsf{or}(\mathsf{SELF}(X); \mathsf{SELF}(Y)) \end{array} \right)$$

and the imperative encoding is:

$$\mathsf{let\,Self} \ll \mathsf{ref\ \ dummy\ \ in\ \ Self} := (\mathsf{nnf\ \ !Self}); \mathsf{let\ \ Nnf} \ll \mathsf{!Self\ \ in\ \ Nnf}(\phi)$$

Computing a Negation Normal Form

This function is used in implementing *decision procedures*, present in almost all model checkers. The processed input is a implication-free languages of formulas with generating grammar:

$$\phi ::= \mathsf{p} \mid \mathsf{and}(\phi;\phi) \mid \mathsf{or}(\phi;\phi) \mid \mathsf{not}(\phi)$$

We present three encodings: the first is purely functional, since it just makes use of recursion via self-application. The second is imperative: the function is shared via a pointer and recursion is achieved via dereferencing. The third encoding manipulates formulas with sharing (back-pointers to shared-subtrees). Type decorations are omitted.

Imperative-with-Sharing, IS

This encoding uses a variable Self which contains a pointer to the recursive code and a flag-pointer to a boolean value associated to each node: all flag-pointers are initially set to false; each time we scan a (possibly) shared-formulas, we set the corresponding flag-pointer to true. The grammar of shared-formulas is as follows:

$$
\begin{array}{lcl}
\mathsf{bool} & ::= & \mathsf{true} \mid \mathsf{false} \\
\mathsf{flag} & ::= & \mathsf{bool\ ref} \\
\psi & ::= & \mathsf{ref}\ \phi \\
\phi & ::= & \mathsf{p} \mid \mathsf{and}(\mathsf{flag}; \psi; \psi) \mid \mathsf{or}(\mathsf{flag}; \psi; \psi) \mid \mathsf{not}(\mathsf{flag}; \psi)
\end{array}
$$

Imperative-with-Sharing, IS

Given the constant dummy, the function nnf is defined as follows:

$$\mathsf{nnf} \triangleq \mathsf{SELF} \rightarrowtriangle$$

$$\left(\begin{array}{lcl}
\mathsf{p} & \rightarrowtriangle & \mathsf{p}; \\
\mathsf{not}(B_1; \mathsf{ref\ not}(B_2; X)) & \rightarrowtriangle & \mathsf{SELF}(!X); \\
\mathsf{not}(B_1; \mathsf{ref\ or}(B_2; X; Y)) & \rightarrowtriangle & \mathsf{and}(\mathsf{ref\ false}; \mathsf{SELF}(\mathsf{ref\ not}(\mathsf{ref\ false}; X)); \\
 & & \quad \mathsf{SELF}(\mathsf{ref\ not}(\mathsf{ref\ false}; Y))); \\
\mathsf{not}(B_1; \mathsf{ref\ and}(B_2; X; Y)) & \rightarrowtriangle & \mathsf{or}(\mathsf{ref\ false}; \mathsf{SELF}(\mathsf{ref\ not}(\mathsf{ref\ false}; X)); \\
 & & \quad \mathsf{SELF}(\mathsf{ref\ not}(\mathsf{ref\ false}; Y))); \\
\mathsf{and}(B; X; Y) & \rightarrowtriangle & \mathsf{if\ (neg\ ref}\ B) \\
 & & \mathsf{then}\ \ (B; X; Y) := (\mathsf{true}; \mathsf{SELF}(!X); \mathsf{SELF}(!Y)) \\
 & & \mathsf{else\ and}(B; X; Y); \\
\mathsf{or}(B; X; Y) & \rightarrowtriangle & \mathsf{if\ (neg\ ref}\ B) \\
 & & \mathsf{then}\ \ (B; X; Y) := (\mathsf{true}; \mathsf{SELF}(!X); \mathsf{SELF}(!Y)) \\
 & & \mathsf{else\ or}(B; X; Y)
\end{array}\right)$$

$$\mathsf{let\ Self} \ll \mathsf{ref\ \ dummy\ \ in\ \ Self} := (\mathsf{nnf\ \ !Self}); \mathsf{let\ \ Nnf} \ll \mathsf{!Self\ \ in\ \ Nnf}(\psi)$$

iRho

Types

Simply Types + Product-Types for Structures

Little flavor ...

- The static semantics is given via a first-order type system based on a form of product-types reflecting the (non-commutative) structure of the term

$$\frac{\Gamma \vdash_{\mathsf{A}} A : \tau_1 \quad \Gamma \vdash_{\mathsf{A}} B : \tau_2}{\Gamma \vdash_{\mathsf{A}} A; B : \tau_1 \wedge \tau_2}\ (Term{-}Struct)$$

- Capability to access and modify a (monomorphic) typed store

$$\frac{\Gamma \vdash_{\mathsf{A}} A : \tau\ \mathsf{ref} \qquad \Gamma \vdash_{\mathsf{A}} B : \tau}{\Gamma \vdash_{\mathsf{A}} A := B : \tau}\ (Term{-}Assign)$$

Little flavor ...

- The static semantics is given via a first-order type system based on a form of product-types reflecting the (non-commutative) structure of the term

$$\frac{\Gamma \vdash_{\mathsf{A}} A : \tau_1 \quad \Gamma \vdash_{\mathsf{A}} B : \tau_2}{\Gamma \vdash_{\mathsf{A}} A; B : \tau_1 \wedge \tau_2}\ (Term{-}Struct)$$

- Capability to access and modify a (monomorphic) typed store

$$\frac{\Gamma \vdash_{\mathsf{A}} A : \tau\ \mathsf{ref} \qquad \Gamma \vdash_{\mathsf{A}} B : \tau}{\Gamma \vdash_{\mathsf{A}} A := B : \tau}\ (Term{-}Assign)$$

Little flavor ...

- The static semantics is given via a first-order type system based on a form of product-types reflecting the (non-commutative) structure of the term

$$\frac{\Gamma \vdash_{\mathsf{A}} A : \tau_1 \quad \Gamma \vdash_{\mathsf{A}} B : \tau_2}{\Gamma \vdash_{\mathsf{A}} A; B : \tau_1 \wedge \tau_2}\ (Term{-}Struct)$$

- Capability to access and modify a (monomorphic) typed store

$$\frac{\Gamma \vdash_{\mathsf{A}} A : \tau\ \mathsf{ref} \qquad \Gamma \vdash_{\mathsf{A}} B : \tau}{\Gamma \vdash_{\mathsf{A}} A := B : \tau}\ (Term{-}Assign)$$

Typing the Two Imperative Encodings

If b is the type of formulas ϕ, and b ref is the type of the shared-formulas ψ, and $\tau^{\wedge^n} \triangleq \overbrace{\tau \wedge \cdots \wedge \tau}^{n}$, and $\tau_1 \triangleq$ b $\rightarrowtriangle$ b, and $\tau_2 \triangleq$ b ref $\rightarrowtriangle$ b ref, then the reader can verify that the following judgments are derivable

(recall $A\,;B \triangleq (X \rightarrowtriangle B)\,A$, if $X \notin \mathsf{Fv}(B)$):

(I)

$$\mathsf{dummy}{:}\tau_1^{\wedge^6}, \mathsf{Self/SELF}{:}\tau_1^{\wedge^6}\mathsf{ref} \;\vdash\; \mathsf{nnf} : \tau_1^{\wedge^6} \rightarrowtriangle \tau_1^{\wedge^6}$$

$$\mathsf{dummy}{:}\tau_1^{\wedge^6}, \mathsf{Self/SELF}{:}\tau_1^{\wedge^6}\mathsf{ref}\,, X{:}\tau_1^{\wedge^6}, \mathsf{Nnf}{:}\tau_1^{\wedge^6} \;\vdash\; \mathsf{Nnf}(\phi) : \mathsf{b}^{\wedge^6}$$

(IS)

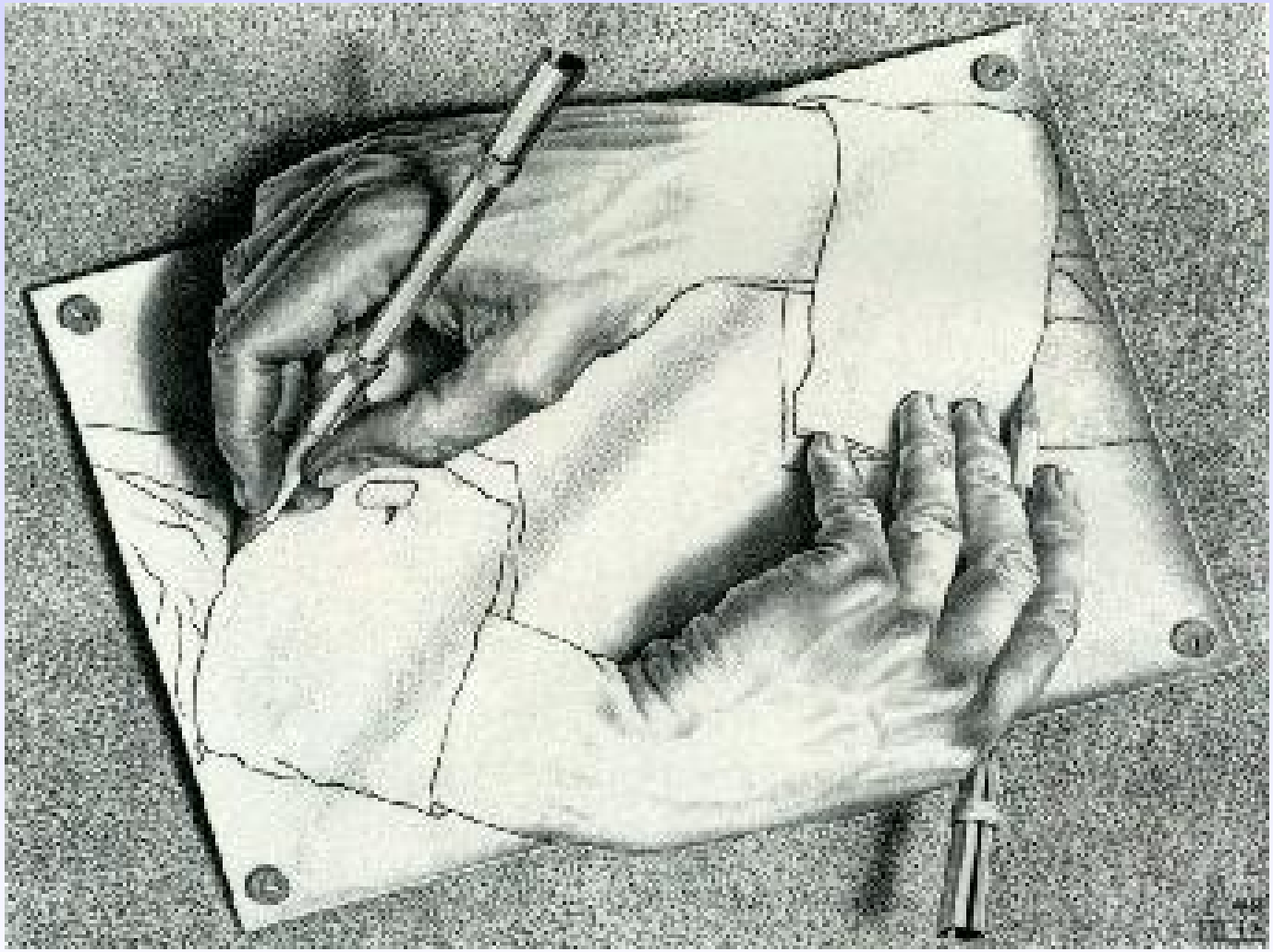
$$\mathsf{dummy}{:}\tau_2^{\wedge^6}, \mathsf{Self/SELF}{:}\tau_2^{\wedge^6}\mathsf{ref} \;\vdash\; \mathsf{nnf} : \tau_2^{\wedge^6} \rightarrowtriangle \tau_2^{\wedge^6}$$

$$\mathsf{dummy}{:}\tau_2^{\wedge^6}, \mathsf{Self/SELF}{:}\tau_2^{\wedge^6}\mathsf{ref}\,, X{:}\tau_2^{\wedge^6}, \mathsf{Nnf}{:}\tau_2^{\wedge^6} \;\vdash\; \mathsf{Nnf}(\psi) : \mathsf{b\ ref}^{\wedge^6}$$

Conclusion

- A special emphasis has been put in our work on the design of the static/dynamic semantics. We always had in mind the main objectives of a conscious implementor
  - ⋆ a "sound" machine (the interpreter)
  - ⋆ a "sound" type system (the type checker), such that
    "well typed programs do not go wrong" [Milner],
  - ⋆ both semantics being suitable to be specified with nice mathematics,
  - ⋆ to be implemented with high-level programming languages,
  - ⋆ and to be certified with modern and semi-automatic theorem provers, like Coq
- The paper is quasi a "technical manual" of the softwares (2)
- See `imprho.ps`, and `imprho.scm`, and `TypedImpRho.v`

DIMPRO



P^2TS

ALL THE ZOOMS BELOW!

Simple Functional Fixpoint

Let $\Gamma \equiv \mathsf{b}{:}ok$, $f{:}(\mathsf{b} \multimap \mathsf{b}) \multimap \mathsf{b}$, and $\Delta \equiv X{:}\mathsf{b} \multimap \mathsf{b}$. Below a derivation for

$$\Omega \triangleq f(X){:}\Delta \multimap X\ f(X)$$

$$\dfrac{\dfrac{\Gamma, \Delta \vdash_{\mathsf{A}} f : (\mathsf{b} \multimap \mathsf{b}) \multimap \mathsf{b} \quad \Gamma, \Delta \vdash_{\mathsf{A}} X : \mathsf{b} \multimap \mathsf{b}}{\Gamma, \Delta \vdash_{\mathsf{A}} f(X) : \mathsf{b}} \quad \dfrac{\Gamma, \Delta \vdash_{\mathsf{A}} X : \mathsf{b} \multimap \mathsf{b} \quad \Gamma, \Delta \vdash_{\mathsf{A}} f(X) : \mathsf{b}}{\Gamma, \Delta \vdash_{\mathsf{A}} X\ f(X) : \mathsf{b}}}{\Gamma \vdash_{\mathsf{A}} \Omega : \mathsf{b} \multimap \mathsf{b}} \quad \dfrac{\Gamma, \Delta \vdash_{\mathsf{A}} f : (\mathsf{b} \multimap \mathsf{b}) \multimap \mathsf{b} \quad \Gamma, \Delta \vdash_{\mathsf{A}} \Omega : \mathsf{b} \multimap \mathsf{b}}{\Gamma, \Delta \vdash_{\mathsf{A}} f(\Omega) : \mathsf{b}}}{\Gamma \vdash_{\mathsf{A}} \Omega\ f(\Omega) : \mathsf{b}}$$

The reader can verify that our interpreter diverges, $i.e.$ [BACK]

$$\dfrac{\begin{array}{c}\infty \\ \vdots\end{array}}{\emptyset \vdash \Omega\ f(\Omega) \Downarrow_{\mathsf{val}} \text{segmentation fault}}$$

Encoding Term Rewrite Systems

- However, a suitable recursion operator in the style of the object-oriented encoding allows us to simulate the *global* behavior of a TRS Let the constants rec and add, and let

$$obj.meth \triangleq ((obj\, meth)\, obj)$$

$$plus \triangleq \left(\begin{array}{l} rec(S) \rightarrowtriangle add(0; Y) \rightarrowtriangle Y; \\ rec(S) \rightarrowtriangle add(suc(X), Y) \rightarrowtriangle suc\,(S.rec\ \ add(X, Y)) \end{array} \right)$$

A Nice Circle ... and 

- $$\frac{\sigma_0 \cdot \rho \vdash F \Downarrow_{\mathsf{val}} f \cdot \sigma_1 \quad \sigma_1 \cdot \rho \vdash A \Downarrow_{\mathsf{val}} a \cdot \sigma_2 \quad \sigma_2 \vdash \langle f \cdot a \rangle \Downarrow_{\mathsf{call}} v \cdot \sigma_3}{\sigma_0 \cdot \rho \vdash F\, A \Downarrow_{\mathsf{val}} v \cdot \sigma_3}\ (Red{-}\rho_{\mathsf{v}})$$

- ```
  (define-method (Eval::value t::App env) ;()
     (with-access::App t (F A)
        (let ( (f (Eval F env)) )
           (let ( (a (Eval A env)) )
              (call f a) ))))
  ```

- ```
  Mutual Inductive eval : expr->env->store->value->store->Prop :=
  .....
   | evalApp: (F:expr)(e:env)(s:store)(f:value)(s1:store)
         (eval F e s f s1) -> (A:expr)(a:value)(s2:store)
         (eval A e s1 a s2) -> (v:value)(s3:store)
         (call f a s2 v s3) ->
         (eval (App F A) e s v s3)
  .....
  ```

Properties of the Imperative Calculus

The main objectives of this section is to prove the main properties of iRho, namely that:

1. Natural Semantics for $\Downarrow_{\mathsf{val}}$ is deterministic;

2. Type-checking with $\vdash_{\mathsf{A}}$ is unique;

3. Subject Reduction holds ($i.e.$ types are preserved under reduction);

4. Type Soundness holds ($i.e.$ the type system preserves the evaluator from "stuck" states);

5. Both Type-checking and Type Reconstruction are decidable.

We start with a natural definition of free variables. Free variables Fv

$$
\begin{array}{lcl@{\qquad}lcl}
\mathsf{Fv}(a) & \triangleq & \emptyset & \mathsf{Fv}(P{:}\Delta \twoheadrightarrow A) & \triangleq & \mathsf{Fv}(A) \setminus \mathsf{Fv}(P) \\
\mathsf{Fv}(X) & \triangleq & \{X\} & \mathsf{Fv}(A\,B) & \triangleq & \mathsf{Fv}(A) \cup \mathsf{Fv}(B) \\
\mathsf{Fv}(*A) & \triangleq & \mathsf{Fv}(A) & \mathsf{Fv}(A;B) & \triangleq & \mathsf{Fv}(A) \cup \mathsf{Fv}(B) \\
\mathsf{Fv}(\&A) & \triangleq & \mathsf{Fv}(A) & \mathsf{Fv}(A=B) & \triangleq & \mathsf{Fv}(A) \cup \mathsf{Fv}(B)
\end{array}
$$

Then, we prove that our natural semantics is deterministic: this result shows that our semantics is an algorithm indeed. A software prototype for the untyped iRho can be found in our web appendix; Deterministic Semantics

1. If $\sigma \cdot \rho \vdash A \Downarrow_{\mathsf{val}} A'_{\mathsf{v}} \cdot \sigma'$, and $\sigma \cdot \rho \vdash A \Downarrow_{\mathsf{val}} A''_{\mathsf{v}} \cdot \sigma''$, then $A'_{\mathsf{v}} \equiv A''_{\mathsf{v}}$, and $\sigma' \equiv \sigma''$;
2. If $\sigma \vdash \langle A_{\mathsf{v}} \cdot B_{\mathsf{v}} \rangle \Downarrow_{\mathsf{call}} A_{\mathsf{v}} \cdot \sigma'$, and $\rho \vdash \langle A_{\mathsf{v}} \cdot B_{\mathsf{v}} \rangle \Downarrow_{\mathsf{call}} A_{\mathsf{v}} \cdot \sigma'$, then $\sigma \equiv \sigma'$;
3. If $\sigma \cdot \rho \vdash \langle P \cdot A_{\mathsf{v}} \rangle \Downarrow_{\mathsf{match}} \rho'$, and $\sigma \vdash \langle P \cdot A_{\mathsf{v}} \rangle \Downarrow_{\mathsf{call}} \rho''$, then $\rho' \equiv \rho''$.

The type system is algorithmic, hence it enjoy the nice property of uniqueness of typing; a software prototype of a simple type checker is on the way Uniqueness of Typing If $\Gamma \vdash_{\mathsf{A}} A : \tau$, then τ is unique. This definition will be useful to state

subject reduction. Coherence The context Γ is coherent with a store σ, and an environment ρ, denoted by

$$\sigma \cdot \rho \vdash_{\mathsf{coh}} \Gamma$$

if there exist two sub-contexts Γ_1, and Γ_2, such that $\Gamma_1, \Gamma_2 \equiv \Gamma$, and $\Gamma \vdash_\sigma \sigma : \Gamma_1$, and $\Gamma \vdash_\rho \rho : \Gamma_2$. This result is crucial to prove subject reduction for closed terms. TH: Subject Reduction for Open Terms

1. If $\sigma_1 \cdot \rho_1 \vdash A \Downarrow_{\mathsf{val}} A_{\mathsf{v}} \cdot \sigma_2$, and $\sigma_1 \cdot \rho_1 \vdash_{\mathsf{coh}} \Gamma$, and $\Gamma \vdash_{\mathsf{A}} A : \tau$, then there exists Γ', which extend Γ, and τ', such that $\sigma_2 \cdot \rho_1 \vdash_{\mathsf{coh}} \Gamma'$, and $\Gamma' \vdash_{\mathsf{v}} A_{\mathsf{v}} : \tau'$, and $\Gamma' \vdash_{<:} \tau' <: \tau$;

2. If $\sigma_1 \vdash \langle A_{\mathsf{v}} \cdot B_{\mathsf{v}} \rangle \Downarrow_{\mathsf{call}} C_{\mathsf{v}} \cdot \sigma_2$, and $\sigma_1 \cdot \rho_1 \vdash_{\mathsf{coh}} \Gamma$, and $\Gamma \vdash_{\mathsf{v}} A_{\mathsf{v}} : \tau_1$, and $\Gamma \vdash_{\mathsf{v}} B_{\mathsf{v}} : \tau_2$, and $\mathsf{arr}(\tau_1) \equiv \tau_3 \rightarrowtriangle \tau_4$, and $\Gamma \vdash_{<:} \tau_2 <: \tau_3$, then there exists Γ', which extend Γ, and τ_5, such that $\sigma_2 \cdot \rho_1 \vdash_{\mathsf{coh}} \Gamma'$, and $\Gamma' \vdash_{\mathsf{v}} C_{\mathsf{v}} : \tau_5$, and $\Gamma' \vdash_{<:} \tau_5 <: \tau_4$;

3. If $\sigma_1 \cdot \rho_1 \vdash \langle P \cdot A_{\mathsf{v}} \rangle \Downarrow_{\mathsf{match}} \rho_2$, and $\sigma_1 \cdot \rho_1 \vdash_{\mathsf{coh}} \Gamma$, and $\Gamma \vdash_{\mathsf{A}} P : \tau_1$, and $\Gamma \vdash_{\mathsf{v}} A_{\mathsf{v}} : \tau_2$, and $\Gamma \vdash_{<:} \tau_2 <: \tau_1$, then there exists Γ', which extend Γ, such that $\sigma_1 \cdot \rho_2 \vdash_{\mathsf{coh}} \Gamma'$.

Then, this result is important to state type soundness. TH: Subject Reduction for

Closed Terms If $\emptyset \vdash_{\mathsf{A}} A : \tau$, and $\emptyset \cdot \emptyset \vdash A \Downarrow_{\mathsf{val}} A_{\mathsf{v}} \cdot \sigma$, then there exists Γ' which extend Γ, and τ' such that $\Gamma' \vdash_{\sigma} \sigma : ok$, and $\Gamma' \vdash_{\mathsf{v}} A_{\mathsf{v}} : \tau'$, and $\Gamma' \vdash_{<:} \tau' <: \tau$. Run-time errors for this interpreter correspond to stuck-states when using the rules to evaluate a closed expression. An inspection of the three judgments shows that there are only few ways in which evaluation may "get-stuck":

($\Downarrow_{\mathsf{val}}$) This judgment gets stuck when we access to a variable not defined in the environment, when the evaluation of A in ($*A$, or $A = B$) gives a fresh location ($i.e.$ not in the current used store), or if one premise in some judgment gets stuck;

($\Downarrow_{\mathsf{call}}$) This judgment gets stuck when the we try to apply a location-value to a value ($e.g.$ $\langle \iota \cdot A_{\mathsf{v}} \rangle$), or a failure-value to a value, ($e.g.$ $\langle \langle [P \ll_{\Delta} B_{\mathsf{v}}].C \cdot \rho \rangle \cdot A_{\mathsf{v}} \rangle$), or if one premise in some judgment gets stuck;

($\Downarrow_{\mathsf{match}}$) This judgment gets stuck in all cases (left to the reader) we try to match a pattern against a value with a different (unmatchable) shape, $e.g.$ $\langle P\, P \cdot A; B \rangle$, etc.

The following theorem proves the absence of such errors in the evaluation of a

well-typed closed expression: type soundness follows from this result.

TH: Absence of stuck states

($\Downarrow_{\mathsf{val}}$) Let A_0 be a closed expression such that $\emptyset \vdash_{\mathsf{A}} A_0 : \tau$ is derivable. Then:

- if $A_0 \equiv \mathsf{C}[X]$, then $\sigma \cdot \rho \vdash X \Downarrow_{\mathsf{val}} \rho(X)$, and $\rho(X)$ is defined (for some σ, and ρ);
- if $A_0 \equiv *A$, then $\emptyset \cdot \emptyset \vdash A \Downarrow_{\mathsf{val}} \iota \cdot \sigma_1$, and $\iota \in \mathsf{Dom}(\sigma_1)$ (for some σ_1);
- if $A_0 \equiv (A = B)$, and $\emptyset \cdot \emptyset \vdash B \Downarrow_{\mathsf{val}} B_{\mathsf{v}} \cdot \sigma_1$, and $\sigma_1 \cdot \emptyset \vdash A \Downarrow_{\mathsf{val}} \iota \cdot \sigma_2$, and $\iota \in \mathsf{Dom}(\sigma_2)$ (for some σ_1, and σ_2);

($\Downarrow_{\mathsf{call}}$) Let A, and B be two closed term such that $\emptyset \cdot \emptyset \vdash A \Downarrow_{\mathsf{val}} A_{\mathsf{v}} \cdot \sigma_0$, and $\sigma_0 \cdot \emptyset \vdash B \Downarrow_{\mathsf{val}} B_{\mathsf{v}} \cdot \sigma_1$. Then:

- if $\sigma_1 \vdash \langle A_{\mathsf{v}} \cdot B_{\mathsf{v}} \rangle \Downarrow_{\mathsf{call}} C_{\mathsf{v}}$, then $A_{\mathsf{v}} \not\equiv \iota$, or $A_{\mathsf{v}} \not\equiv [P \ll_{\Delta} B_{\mathsf{v}}].C$;

($\Downarrow_{\mathsf{match}}$) Let A, and B be two closed term such that $\emptyset \cdot \emptyset \vdash A \Downarrow_{\mathsf{val}} \langle P{:}\Delta \rightarrowtriangle C \cdot \rho \rangle \cdot \sigma_0$, and $\sigma_0 \cdot \emptyset \vdash B \Downarrow_{\mathsf{val}} B_{\mathsf{v}} \cdot \sigma_1$.

- if $\sigma_1 \cdot \rho \vdash \langle P \cdot B_{\mathsf{v}} \rangle \Downarrow_{\mathsf{match}} \rho'$, then P will successfully match with B_{v}.

TH: Decidability Given a closed expression A, the following propositions are decidable:

1. **(Type Checking)** The judgment $\emptyset \vdash_{\mathsf{A}} A : \tau$ is derivable;
2. **(Type Reconstruction)** There exists a type τ', such that $\emptyset \vdash_{\mathsf{A}} A : \tau'$.