Reasoning about Staged Computation

Frank Pfenning

SAIG 2000, Montreal, Canada
September 20, 2000

Disclaimer: Work in Progress!

Acknowledgments: Alberto Momigliano

Terminology

- **Staged Computation:** explicit division of a computation into stages. Used in algorithm derivation and program optimization.
- **Partial Evaluation:** static specialization of a program based on partial input data.
- **Run-Time Code Generation:** dynamic generation of code during the evaluation of a program.

Intensionality

- Staged computation is concerned with **how** a value is computed.
- Staging is an **intensional** property of a program.
- Most research has been motivated **operationally**.
- Modal and temporal λ-calculi provide a **logical** way to understand staging.
  [Davies & Pf.'96] [Davies'96] [Davies & Pf.'00]
  [Moggi, Taha, Benaissa & Sheard'99]

Reasoning about Functional Programs

- Specifications as dependent types.
- Verification via type-checking.
- Extracting programs from proofs **or**
  reconstructing proof obligations from programs.

This Talk

1. Modal λ-Calculus
2. Specifications via Dependent Types
3. Hidden Variables
4. Dependent Types Revisited
5. Erasure Interpretation
6. Conclusion

Hypothetical Judgments

- Basic judgment: $M : A$ "M *has type* A"
- Hypothetical judgment:

$$\underbrace{x_1{:}A_1, \ldots, x_n{:}A_n}_{\text{value variables}} \vdash M : A$$

- Hypothesis rule: $\Gamma_1, x{:}A, \Gamma_2 \vdash x : A$
- Substitution property:

  If $\Gamma \vdash M : A$
  and $\Gamma, x{:}A \vdash N : C$
  then $\Gamma \vdash [M/x]N : C$.

- λ-abstraction, application, β-reduction as usual.

Categorical Judgments

- For run-time code generation, a **source expression** must be available at run-time (at least conceptually).
- Judgment: $M :: A$ "M *is a source expression of type* A"
- Definition: $M :: A$ if $\bullet \vdash M : A$.
- Hypothetical judgment:

$$\underbrace{u_1{::}B_1, \ldots, u_k{::}B_k}_{\text{expression variables}} \;;\; \underbrace{x_1{:}A_1, \ldots, x_n{:}A_n}_{\text{value variables}} \vdash N : C$$

- Additional hypothesis rule: $(\Delta_1, u{::}A, \Delta_2); \Gamma \vdash u : A$
- Additional substitution property:

If $\Delta; \bullet \vdash M : A$
and $(\Delta, u{::}A); \Gamma \vdash N : C$
then $\Delta; \Gamma \vdash [M/u]N : C$.

Intensional Types

- Type: $\Box A$ "*terms denoting source expressions of type* A"
- Constructor:

$$\frac{\Delta; \bullet \vdash M : A}{\Delta; \Gamma \vdash \mathsf{box}\, M : \Box A}\, \Box I$$

- Destructor:

$$\frac{\Delta; \Gamma \vdash M : \Box A \qquad (\Delta, u{::}A); \Gamma \vdash N : C}{\Delta; \Gamma \vdash \mathsf{let\,box}\, u = M \;\mathsf{in}\; N \;\mathsf{end} : C}\, \Box E$$

- Evaluation:

$$\frac{}{\mathsf{box}\, M \hookrightarrow \mathsf{box}\, M} \qquad \frac{M \hookrightarrow \mathsf{box}\, M_1 \qquad [M_1/u]N \hookrightarrow V}{\mathsf{let\,box}\, u = M \;\mathsf{in}\; N \;\mathsf{end} \hookrightarrow V}$$

Some Properties

- Type preservation and progress [Davies & Pf'96].
- Staging correctness [Davies & Pf'00].
- Staging interpretation via multiple world semantics.
- Curry-Howard isomorphism with modal logic S4.

Design Philosophy

- Put the power of staging into the hands of the programmer.
- Simple and declarative.
- Conservative extension of ML based on modal types.
- Safe in the presence of effects.
- Staging errors are type errors.
- Enables, but does not prescribe optimizations.
- Compiler implementation in progress (PML).
- Some experiments in run-time code generation:
  Fabius [Leone & Lee'94'96]
  CCAM [Wickline, Lee & Pf.'98]

Example: Power Function

- Unstaged $\vdash power : \mathsf{nat} \to (\mathsf{nat} \to \mathsf{nat})$

```
fun power 0 b = 1
  | power (s n) b = b * (power n b)
```

- Incorrectly staged $\not\vdash power : \mathsf{nat} \to \Box(\mathsf{nat} \to \mathsf{nat})$

```
fun power 0 = box (λb. 1)
  | power (s n) = box (λb. b * (power n) b)
```

- Correctly staged $\vdash power : \mathsf{nat} \to \Box(\mathsf{nat} \to \mathsf{nat})$

```
fun power 0 = box (λb. 1)
  | power (s n) =
    let box u = power n
    in box (λb. b * u b) end
```

Examples: Laws of S4

- Curry-Howard isomorphism applied to modal logic S4.
- $\vdash \mathit{eval} : \Box A \to A$

```
fun eval x = let box u = x in u end
```

- $\vdash \mathit{subst} : \Box(A \to B) \to \Box A \to \Box B$

```
fun subst f x =
    let box f' = f
     in let box x' = x
         in box (f' x') end end
```

- $\vdash \mathit{quote} : \Box A \to \Box\Box A$

```
fun quote x = let box u = x in box (box u) end
```

Specifications via Dependent Types

- Dependent function type $\Pi x{:}A.\, B(x)$ ($\sim A \to B$)
- Dependent sum type $\Sigma x{:}A.\, B(x)$ ($\sim A \times B$)
- Propositions $m \doteq n$ ($\sim$ proof) as type of proofs.
- Specification of $power$ function:

$$power : \Pi n{:}\mathsf{nat}.\, \Pi b{:}\mathsf{nat}.\, \Sigma m{:}\mathsf{nat}.\, m \doteq b^n$$

- Erasing dependencies we obtain:

$$power : \mathsf{nat} \to \mathsf{nat} \to (\mathsf{nat} \times \mathsf{proof})$$

- Usually do not compute elements of type proof.

Dependent Function Types

- Constructor

$$\frac{\Gamma \vdash A : \mathsf{type} \qquad \Gamma, x{:}A \vdash M : B(x)}{\Gamma \vdash \lambda x{:}A.\, M : \Pi x{:}A.\, B(x)}\,\Pi I$$

- Destructor

$$\frac{\Gamma \vdash M : \Pi x{:}A.\, B(x) \qquad \Gamma \vdash N : A}{\Gamma \vdash M\, N : B(N)}\,\Pi E$$

- Evaluation

$$\frac{}{\lambda x{:}A.\, M \hookrightarrow \lambda x{:}A.\, M}$$

$$\frac{M \hookrightarrow \lambda x{:}A.\, M_1 \quad N \hookrightarrow W \quad [W/x]M_1 \hookrightarrow V}{M\, N \hookrightarrow V}$$

Dependent Sum Types

- Constructor

$$\frac{\Gamma \vdash M : A \qquad \Gamma \vdash N : B(M)}{\Gamma \vdash \langle M, N \rangle : \Sigma x{:}A.\, B(x)}\, \Sigma I$$

- Destructors

$$\frac{\Gamma \vdash M : \Sigma x{:}A.\, B(x)}{\Gamma \vdash \pi_1\, M : A}\, \Sigma E_1 \qquad \frac{\Gamma \vdash M : \Sigma x{:}A.\, B(x)}{\Gamma \vdash \pi_2\, M : B(\pi_1\, M)}\, \Sigma E_1$$

- Evaluation

$$\frac{M \hookrightarrow V \quad N \hookrightarrow W}{\langle M, N \rangle \hookrightarrow \langle V, W \rangle}$$

$$\frac{M \hookrightarrow \langle V, W \rangle}{\pi_1\, M \hookrightarrow V} \qquad \frac{M \hookrightarrow \langle V, W \rangle}{\pi_2\, M \hookrightarrow W}$$

Judgments Revisited

- Well-formed types: $\Gamma \vdash A : \text{type}$
- Definitional equality: $\Gamma \vdash A = B : \text{type}$
- Definitional equality derived from $M \hookrightarrow V$.
- Type conversion:

$$\frac{\Gamma \vdash M : A \quad \Gamma \vdash A = B : \text{type}}{\Gamma \vdash M : B}$$

- Generalized substitution property:

  If $\Gamma \vdash M : A$
  and $\Gamma, x{:}A \vdash N : C(x)$
  then $\Gamma \vdash [M/x]N : C(M)$.

Example Revisited

- Proof objects:

$$p_0 \;:\; \Pi b{:}\mathsf{nat}.\, 1 \doteq b^0$$

$$p_1 \;:\; \Pi b{:}\mathsf{nat}.\, \Pi n{:}\mathsf{nat}.\, \Pi m{:}\mathsf{nat}.\, m \doteq b^n \rightarrow b * m \doteq b^{n+1}$$

- Recall: $power : \Pi n{:}\mathsf{nat}.\, \Pi b{:}\mathsf{nat}.\, \Sigma m{:}\mathsf{nat}.\, m \doteq b^n$

- Power function with correctness proof:

```
fun power 0 b = ⟨1, p0 b⟩
  | power (s n) b =
    ⟨b * π1 (power n b),
     p1 b n (π1 (power n b))
        (π2 (power n b))⟩
```

The Problem with Staging

- Consider the staged version of the power program:

```
fun power 0 = box (λb. ⟨1, p0 b⟩)
  | power (s n) =
    let box u = power n
     in box (λb. ⟨b * (π1 (u b)),
                  p1 b n (π1 (u b)) (π2 (u b))⟩) end
```

- $\not\vdash power : \Pi n{:}\mathsf{nat}.\,\Box(\Pi b{:}\mathsf{nat}.\,\Sigma m{:}\mathsf{nat}.\,m \doteq b^n)$
- Violates staging by accessing n.
- n is used only in the proof object!

Solution: Hidden Variables

- Generalize judgment: $\Delta; \Gamma; \Omega \vdash M : A$.
- $\Delta = u_1{::}A_1, \ldots, u_k{::}A_k$ (expression variables), available in all stages.
- $\Gamma = x_1{:}B_1, \ldots, x_n{:}B_n$ (value variables), available only in current stage.
- $\Omega = w_1 \div C_1, \ldots, w_m \div C_m$ (hidden variables), **were available in a past stage**.
- Uniform preservation of information.
- **No** Hypothesis rule: $\Delta; \Gamma; (\Omega_1, w \div A, \Omega_2) \not\vdash w : A$!

Modal Types Revisited

- Modal constructor revisited:

$$\frac{\Delta; \bullet; (\Gamma, \Omega) \vdash M : A}{\Delta; \Gamma; \Omega \vdash \mathsf{box}\, M : \Box A}\, \Box I$$

- Destructor:

$$\frac{\Delta; \Gamma; \Omega \vdash M : \Box A \qquad (\Delta, u{::}A); \Gamma; \Omega \vdash N : C}{\Delta; \Gamma; \Omega \vdash \mathsf{let\,box}\, u = M\ \mathsf{in}\ N\ \mathsf{end} : C}\, \Box E$$

- Substitution principle:

  If $\Delta; \bullet; (\Gamma, \Omega) \vdash M : A$
  and $(\Delta, u{::}A); \Gamma; \Omega \vdash N : A$
  then $\Delta; \Gamma; \Omega \vdash N : C$.

Hidden Types

- $M : [A]$ "*M is a past term of type A*"
- Constructor:

$$\frac{\Delta; (\Gamma, \Omega); \bullet \vdash M : A}{\Delta; \Gamma; \Omega \vdash [M] : [A]}\,[-]I$$

- Destructor:

$$\frac{\Delta; \Gamma; \Omega \vdash M : [A] \qquad \Delta; \Gamma; (\Omega, w \div A) \vdash N : C}{\Delta; \Gamma; \Omega \vdash \mathsf{let}\,[w] = M\ \mathsf{in}\ N\ \mathsf{end} : C}\,[-]E$$

- Substitution principle:

If $\Delta; (\Gamma, \Omega); \bullet \vdash M : A$
and $\Delta; \Gamma; (\Omega, w \div A) \vdash N : C$
then $\Delta; \Gamma; \Omega \vdash [M/w]N : C$

Example Revisited

- "Hide" proof objects.
- $\vdash power$: Πn:nat. $\Box(\Pi b$:nat. Σm:nat. $[m \doteq b^n])$
- Implementation:

```
fun power 0 = box (λb. ⟨1, [p0 b]⟩)
  | power (s n) =
    let box u = power n
     in box (λb. ⟨b * (π1 (u b)),
                  let [w] = π2 (u b)
                  in [p1 b n (π1 (u b)) w] end⟩) end
```

Dependent Types Revisited

- We cannot sort $u{::}A$, $x{:}A$, $w \div A$ into zones because of dependencies.
- $\Psi ::= \bullet \mid \Psi, x{::}A \mid \Psi, x{:}A \mid \Psi, x \div A$
- $\Psi^{\ominus}$ replaces $x{:}A$ by $x \div A$.
- $\Psi^{\oplus}$ replaces $x \div A$ by $x{:}A$.
- Valid contexts: $\vdash \Psi$ ctx (other choices possible?)

$$\frac{}{\vdash \bullet \text{ ctx}} \qquad \frac{\vdash \Psi \text{ ctx} \quad \Psi \vdash A : \text{type}}{\vdash (\Psi, x{:}A) \text{ ctx}}$$

$$\frac{\vdash \Psi \text{ ctx} \quad \Psi^{\ominus} \vdash A : \text{type}}{\vdash (\Psi, x{::}A) \text{ ctx}} \qquad \frac{\vdash \Psi \text{ ctx} \quad \Psi^{\oplus} \vdash A : \text{type}}{\vdash (\Psi, x \div A) \text{ ctx}}$$

Dependent Substitution Principles

- For expression variables:

  If $\Psi^{\ominus} \vdash M : A$
  and $\Psi, u{::}A \vdash N : C(u)$
  then $\Psi \vdash [M/u]N : C(M)$.

- For value variables:

  If $\Psi \vdash M : A$
  and $\Psi, x{:}A \vdash N : C(x)$
  Then $\Psi \vdash [M/x]N : C(M)$.

- For hidden variables:

  If $\Psi^{\oplus} \vdash M : A$
  and $\Psi, w \div A \vdash N : C(w)$
  then $\Psi \vdash [M/w]N : C(M)$.

Hidden Types Revisited

- Constructor:

$$\frac{\Psi^{\oplus} \vdash M : A}{\Psi \vdash [M] : [A]}\,[-]I$$

- Destructor:

$$\frac{\Psi \vdash M : [A] \qquad \Psi, w \div A \vdash N : C}{\Psi \vdash \mathsf{let}\,[w] = M\ \mathsf{in}\ N\ \mathsf{end} : C}\,[-]E$$

Dependent Function Types Revisited

- Constructor: (other choices possible?)

$$\frac{\Psi \vdash A : \mathsf{type} \qquad \Psi, x{:}A \vdash M : B(x)}{\Psi \vdash \lambda x{:}A.\, M : \Pi x{:}A.\, B(x)}\,\Pi I$$

- Destructor:

$$\frac{\Psi \vdash M : \Pi x{:}A.\, B(x) \qquad \Psi \vdash N : A}{\Psi \vdash M\, N : B(N)}\,\Pi E$$

- Dependent sums are straightforward.

Application: Subset Types

- Define $\{x{:}A \mid B(x)\}$ as $\Sigma x{:}A.\,[B(x)]$.
- Derived rules:

$$\frac{\Psi \vdash M : A \quad \Psi^{\oplus} \vdash N : B(M)}{\Psi \vdash \langle M, N\rangle : \{x{:}A \mid B(x)\}}\,\{-\}I$$

$$\frac{\Psi \vdash M : \{x{:}A \mid B(x)\}}{\Psi \vdash \pi_1\, M : A}\,\{-\}E_1$$

$$\frac{\Psi \vdash M : \{x{:}A \mid B(x)\} \quad \Psi, w \div B(\pi_1\, M) \vdash N : C}{\Psi \vdash \mathsf{let}\,[w] = \pi_2\, M\ \mathsf{in}\ N\ \mathsf{end} : C}\,\{-\}E_2$$

The Erasure Interpretation

- "*Forget about the Past!*"
- Type $\odot$ "*terms with no computational contents*"
- Type erasure ($A', B' \neq \odot$)

$$\frac{A \mapsto A' \quad B(x) \mapsto B'}{\Pi x{:}A.\, B(x) \mapsto A' \to B'} \qquad \frac{A \mapsto \odot \quad B(x) \mapsto B'}{\Pi x{:}A.\, B(x) \mapsto B'}$$

$$\frac{A \mapsto A' \quad B(x) \mapsto \odot}{\Pi x{:}A.\, B(x) \mapsto \odot} \qquad \frac{A \mapsto \odot \quad B(x) \mapsto \odot}{\Pi x{:}A.\, B(x) \mapsto \odot}$$

$$\frac{}{[A] \mapsto \odot}$$

Erasure Continued

- Dependent sums:

$$\frac{A \mapsto A' \quad B(x) \mapsto B'}{\Sigma x{:}A.\, B(x) \mapsto A' \times B'} \qquad \frac{A \mapsto \odot \quad B(x) \mapsto \odot}{\Sigma x{:}A.\, B(x) \mapsto \odot}$$

$$\frac{A \mapsto \odot \quad B(x) \mapsto B'}{\Sigma x{:}A.\, B(x) \mapsto B'} \qquad \frac{A \mapsto A' \quad B(x) \mapsto \odot}{\Sigma x{:}A.\, B(x) \mapsto A'}$$

- Type-directed translation:

$$\Psi \vdash M : A \mapsto M' : A'$$

- Omit the straightforward definition.

Erasure

- **Theorem:** Erasure commutes with type-checking and evaluation.
- **Theorem:** Hidden variables are *useless* under erasure interpretation.
- Dependently typed programs are correct and their erasure executes in a well-staged manner.

Future Work on Type Theory

- Decidability of type-checking?
- Variations on rules?
- Definitional equality of type theory:
  1. Intensional vs. extensional interpretation of $\Box A$.
  2. Erasure semantics and *proof irrelevance*?
- Integrate with DML, where index objects are restricted to (tractable) constraint domains. [Xi & Pf'98] [Xi & Pf'99]
- Develop dependently typed linear logic?

Application: Linear Logic

- Traditional judgment: $\Delta; \Gamma \vdash M : A$
  Δ (unrestricted variables), Γ (linear variables)

- Traditional sample rule:

$$\frac{\Delta; \Gamma_1 \vdash M : A \quad \Delta; \Gamma_2 \vdash N : B}{\Delta; (\Gamma_1, \Gamma_2) \vdash M \otimes N : A \otimes B} \otimes I$$

- New rule (adding hidden variables):

$$\frac{\Delta; \Gamma_1; (\Gamma_2, \Omega) \vdash M : A \quad \Delta; \Gamma_2; (\Gamma_1, \Omega) \vdash N : B}{\Delta; (\Gamma_1, \Gamma_2); \Omega \vdash M \otimes N : A \otimes B} \otimes I$$

- This permits more programs under type assignment:

$$\vdash \lambda x.\, \lambda y.\, x \otimes (\lambda w.\, y)\, x : A \multimap B \multimap A \otimes B$$

Related Work

- Useless variable elimination (many—one later in this workshop).
- Vacuous and strict variables [Momigliano'00].
- Substructural λ-calculi [Wright'98].
- Hidden variables in Nuprl.
- Proof irrelevance interpretation of propositions in type theory.

Summary

- Modal types for staged computation.
- Dependent types for full specifications.
- The problem with dependent types.
- Hidden variables as references to the past.
- Clear logical solution.
- Admits erasure interpretation.
- Other applications (subset types, linear type theory).