Cost-sensitive computational adequacy of higher-order recursion in synthetic domain theory

MFPS 2024
June 21, 2024

Yue Niu
Jon Sterling
Robert Harper

Carnegie Mellon University
✉ yuen@cs.cmu.edu
Acknowledgements

This work was funded by the United States Air Force Office of Scientific Research and the National Science Foundation. We thank Tristan Nguyen at AFOSR for support. Yue Niu was supported by the Air Force Research Laboratory through the NDSEG fellowship. Views and opinions expressed are however those of the authors only and do not necessarily reflect those of AFOSR, AFRL, or NSF.
Introduction

The story begins with a type theory calf developed to unify cost-sensitive and functional verification [Niu+22].

- Functional: IO-behavior of programs, data structure invariants
- Cost-sensitive: computational cost or resource usage (time, space, etc.)

Functional properties are about if a program is correct, cost-sensitive properties are about how much resource a program uses.
Introduction

calf supports a denotational style of cost analysis — connection to operational semantics via a cost-sensitive computational adequacy property à la Plotkin [Plo77].

Prior work: cost-sensitive adequacy for first-order recursion [NH23].

This talk: cost-sensitive adequacy for PCF (higher-order recursion).
Outline

Introduction to `calf`:
- Cost-sensitive and functional reasoning in `calf`
- Cost-sensitive adequacy property

Integrating higher-order recursion in `calf`:
- Introduction to `synthetic domain theory` (SDT)
- Cost-sensitive SDT
- Cost-sensitive adequacy in SDT
**Cost as an abstract effect**

In *calf*, cost is an *abstract* effect $F(A)$ supporting an operation $\text{step} : \mathbb{C} \to F(1)$. Think of $\text{step}^c$ as taking $c$ abstract steps:

\[
\text{insertSort} : \text{list} \to F(\text{list}) \\
\text{insertSort}(l) = \ldots \text{step}^c; e \ldots
\]

Under the hood define $F(A) = \mathbb{C} \times A$ and $\text{step}^c = (c, \star)$. Can reason about step’s equationally:

\[
\text{step}^{c_1}; \text{step}^{c_2} = \text{step}^{c_1 + c_2} \\
\text{step}^0; e = e
\]
Functional reasoning in calf

How to reason about the purely functional properties of cost-sensitive programs?

\[ \text{isSorted}(\text{insertSort}(l)) \iff \text{isSorted}(\text{mergeSort}(l)) \]

Should be automatic because both are sorting algorithms. But not because \( \text{insertSort} \neq \text{mergeSort} \) due to presence of cost structure!
Cost as a phase

The functional semantics of (total) programs is naturally modeled in \textbf{Set}.

\textbf{Set} is too “flat”: the cost effect $\mathbb{C} \times - : \text{Set} \to \text{Set}$ does not distinguish data from cost structure.

\textbf{calf}: cost as a new \textbf{dimension} or \textbf{phase}. 


Cost structure as families

calf = the internal type theory of the category of families \( \text{Set} \rightarrow \)

A type in \( \text{Set} \rightarrow \) is a cost-sensitive set equipped with a restriction action to the purely functional component:

\[
\begin{array}{ccc}
A^* & \downarrow & \pi_A \\
\downarrow & & \\
A^\circ & &
\end{array}
\]

Think Kripke/possible worlds semantics over \( I = \{ \circ \rightarrow \bullet \} \).
Functional vs. cost-sensitive phase

Presheaves over \( \{ \circ \rightarrow \bullet \} \) exhibits a phase distinction:

- World at \( \circ \) = **functional phase**
- World at \( \bullet \) = **cost-sensitive phase**
- In cost-sensitive phase, \( \text{insertSort} \neq \text{mergeSort} \).
- In functional phase, \( \text{insertSort} = \text{mergeSort} \).

Presheaf restriction \( \bullet \rightarrow \circ \) trivializes/redacts cost structure!
Modal types

A modal type is either purely functional or purely cost-sensitive.

Definition

A type is purely functional or function-modal when it is in the image of the constant presheaves functor $\text{Set} \to \text{Set}^{\to}$.

Definition

A type is purely cost-sensitive or cost-modal when it is given by a terminal map $A \to 1$. 
Cost effect with cost-modal types

Define $F(A)$ by using a *cost-modal* monoid object $C$:

$$ F\left( \downarrow A^\circ \right) = \downarrow \mathbb{N} \times \downarrow A^\circ = \downarrow \mathbb{N} \times A^\circ $$

Restriction deletes cost structure.
Internalization

Modal types can be phrased in the internal language of $\text{Set} \rightarrow$.

Let $\bot : \Omega$ be the *intermediate* proposition in $\text{Set} \rightarrow$:

$$
\begin{array}{ccc}
\bot & = & \downarrow \\
& 0 & \downarrow & 0 & = & \downarrow \\
\text{and} & & & & & \\
\top & = & \downarrow \\
& 1 & \downarrow & 1 & = & \downarrow \\
\end{array}
$$

Assuming $\bot = \text{restricting to the functional phase of } $calf$. 
Internal characterization of modal types

**Proposition**

A type $A$ is function-modal when $(\bot \rightarrow A) \equiv A$.

**Proposition**

A type $A$ is cost-modal when $(\bot \rightarrow A) \equiv 1$.

In other words, a function-modal type “thinks” the functional phase holds and a cost-modal type “thinks” the functional phase is false.
Constructing modal types

Given $A$, $\mathbb{P} \rightarrow A$ is function-modal. Dually, construct a cost-modal type $\mathbb{P} \lor A$ as follows:

![Diagram](image)

The cost modality $\mathbb{P} \lor$ — quotients the type $A$ to a unique point $*$ in the functional phase.
Functional and cost reasoning, internally

Semantically, \( F(A) = (∥ \lor C) \times A \).

Thus \( \text{insertSort} \neq \text{mergeSort} \) since the cost monoid \( ∥ \lor C \) is nontrivial.

But, \( ∥ \rightarrow ((∥ \lor C) \cong 1) \), so \( \text{insertSort} = \text{mergeSort} \) in the functional phase!
calf vs. programming languages

Cost analysis in **calf** is *equational* or *denotational* (\(\text{step}^{c_1}; \text{step}^{c_2} = \text{step}^{c_1 + c_2}\)).

Problems:

- How to relate cost analysis in **calf** to PLs with *operational* cost semantics?
- How to reconcile general recursive functions in PLs with total functions in **calf**?
calf vs. programming languages

Solution:

- Enrich calf with partiality via synthetic domain theory.
- Relate PLs and calf by an internal, cost-sensitive computational adequacy property.

Upshot:

- General recursive programming in calf
- Cost-sensitive generalization of Plotkin’s classic adequacy property.
Cost-sensitive computational adequacy

Example: take STLC equipped with the cost effect $F(A)$. Internal to $\text{calf}$, we have a language $\mathcal{L} = (\text{Ty} : \mathcal{U}, \text{Tm} : \text{Ty} \to \mathcal{U})$.

**Internal denotational cost semantics of $\mathcal{L}$:**

- $[\_]_{\text{Ty}} : \text{Ty} \to \mathcal{U}$
- $(\_[\_])_{\text{Tm}} : \text{Tm}(A) \to [A]_{\text{Ty}}$

As before $[F(A)] = \mathbb{C} \times [A]$.

**Internal operational cost semantics of $\mathcal{L}$:**

- $\Downarrow_A \subseteq \text{Tm}(A) \times \mathbb{C} \times \text{Tm}(A)$
Cost-sensitive computational adequacy

**Definition**

A language satisfies cost-sensitive computational adequacy when for all $e : F(2)$, $\llbracket e \rrbracket =_{C \times [A]} (c, \llbracket v \rrbracket)$ if and only if $e \Downarrow^c v$.

Classic Plotkin adequacy: $\llbracket - \rrbracket$ carves out functions that are definable operationally.

Cost-sensitive adequacy: $\llbracket - \rrbracket$ carves out calf functions that are definable operationally in a cost-reflecting way.
Cost-sensitive adequacy for higher-order recursion

Prior work: $\mathcal{L} =$ Algol-like languages with while loops [NH23].

This work: $\mathcal{L} =$ PCF.
Recursion in type theory

To define the denotational cost semantics of PCF in calf, we need a notion of partial functions in type theory.

Attempt: model calf in presheaves valued in ωcpo’s: ωCPO→.

Unfortunately not a model of dependent type theory.
Integrate higher-order recursion into type theory by means of synthetic domain theory (SDT):

- Intuitionistic type theory
- Class of predomains
- All definable predomain maps automatically continuous

Concretely: a topos $E$ equipped with a full subcategory Predom.
Axioms of SDT

To start, we need an object called the *dominance* that serves as the classifier of the *support* of partial maps.

**Definition**

A dominance subobject $\Sigma \hookrightarrow \Omega$ that is closed under $\top : \Omega$ and dependent sums.

Frequently $\Sigma$ is also required to be closed under $\bot : \Omega$. 
Lifting structure

The dominance $\Sigma$ induces a lifting structure $L(A) = \Sigma_{\phi: \Sigma}.\phi \to A$: partial maps $A \xleftarrow{\Sigma} D \to B$ as total maps $A \to L(B)$.

Lifting induces an incidence relation $\omega \xrightarrow{} \overline{\omega}$ including the initial lift algebra $\omega$ into the final lift coalgebra $\overline{\omega}$.

Think of $\omega \xrightarrow{} \overline{\omega}$ as a figure shape that we use to state the completeness properties of predomains.
Predomains in SDT

A predomain has the unique extension property along $\omega \rightarrow \omega$:

\[
\begin{array}{c}
\omega \\
\downarrow \\
X
\end{array}
\quad \xrightarrow{\sim} 
\quad \begin{array}{c}
\omega \\
\downarrow \\
\omega
\end{array}
\]

Synthetic counterpart to $\omega$-cpos, which extend along the figure shape $\{0 \leq 1 \leq \ldots\} \rightarrow \{0 \leq 1 \leq \ldots \leq \infty\}$. 
Model of SDT

A model of SDT is given by a topos $\mathcal{E}$ equipped with a predomain dominance $\Sigma$.

Every such model induces a full subcategory of predomains that is a reflective exponential ideal:

- Closed under limits and exponentials: types of $\textsf{PCF}$
- All colimits exist: used to define the cost-modal type $\bot \lor C$
- Every endomap of domains (predomains with lift algebras) has a fixed-point: $\text{fix}$ operator
Denotational semantics of \textbf{PCF} in cost-sensitive SDT

To interpret \textbf{PCF} with the cost effect \(F(A)\), need a proposition \(\phi\) for the \textit{functional phase}:

\textbf{Definition}

A \textit{model of SDT with a phase distinction} is a model of SDT \((\mathcal{E}, \Sigma, \phi)\) where \(\phi\) is a \(\Sigma\)-proposition.

Semantically: \([F(A)] = L(C \times [A])\) with \(C\) cost-modal.

Need \(\phi : \Sigma\) to ensure \(\phi \lor A\) is a predomain when \(A\) is one.
Operational semantics of PCF

Our proof of computational adequacy relies on the fact that $e \downarrow^c v$ is a $\Sigma$-proposition.

Define the operational semantics as a partial function $\text{eval} : \text{Tm}(F(A)) \rightarrow \text{Tm}(F(A)) \rightarrow \text{L}(\mathbb{C})$:

$$\text{eval}(e, v) = \begin{cases} c \boxplus \text{eval}(e', v) & \text{out}(e) = \text{inr} \cdot (c, e') \\ (e = v, \lambda u.0) & \text{out}(e) = \text{inl} \cdot * \end{cases}$$

In the above, we write $\text{out} : \text{Tm}(A) \rightarrow 1 + (\mathbb{C} \times \text{Tm}(A))$ for the one step transition relation, and $- \boxplus -$ for the cost algebra map.
Logical relation for computational adequacy

Define a family of relations $\ll_A \subseteq \llbracket A \rrbracket \times \text{Tm}(A)$ between the syntax and semantics of \textbf{PCF}.

A technical point is the definition of $\ll_{F(A)}$:

$$
e \ (R \Rightarrow S) \ e' = \forall [a \ R \ a'] (e \ a) \ S \ (e' \ a')$$
$$e \ll_{F(A)} e' = \forall [f \ (\ll_A \Rightarrow \leq) \ f'] e; f \leq e'; f'$$

In the above we write $e \leq e'$ for the \textit{specialization order} or \textit{definedness order} on $F(1) \cong L(C)$.

Ensures that $(\neg \ll_{F(A)} e') \subseteq \llbracket F(A) \rrbracket$ is always a sub-predomain or \textit{admissible}. 
Fundamental lemma and computational adequacy

We may prove the fundamental lemma of the logical relation:

**Theorem**

Given $\Gamma \vdash e : A$, we have $\Gamma \vdash \llbracket e \rrbracket \triangleleft_A e$.

Cost-sensitive computational adequacy follows directly from the fundamental lemma:

**Theorem**

Given $e : F(1)$, we have that $\llbracket e \rrbracket = \text{eval}(e, \star)$. 
Model of cost-sensitive SDT

To incorporate cost structure as a phase distinction, define a model of SDT fibred over $\mathbf{Set}^{\to}$.

Isolate a (small) category $\mathcal{C}$ of internal dcpos in $\mathbf{Set}^{\to}$.

- Presheaves on $\mathcal{C}$ is almost a model of SDT.
- Restrict to sheaves on $\mathcal{C}$ for the extensive coverage: preserves $\emptyset$ and $+$.

**Theorem**

The category of (internal) sheaves on $\mathcal{C}$ furnishes a model of SDT such that the functional phase proposition $\mathbb{P} : \mathcal{C}$ is preserved by the Yoneda embedding.
Related work

- Computational adequacy in SDT [Sim99; Sim04]
- Relative sheaf models of SDT [SH22]
- Rooted in the type-theoretic framework calf
- Extended the results of Niu and Harper [NH23] to PCF
- Denotational cost semantics based on prior work on effectful PCF [Kav+19]
Conclusion

- Integrated higher-order recursion into calf type theory
- Internal cost-sensitive computational adequacy theorem for PCF
- Connecting denotational and operational reasoning for cost analysis in type theory
- Relative sheaf model of the function-cost phase distinction
Future work

- Recursive types [Sim04]
- Relating internal and external cost-sensitive adequacy
Thanks for listening!
References I


References II


References III
