Context Constrained Computation

Robert Atkey
Computer and Information Sciences
University of Strathclyde
robert.atkey@strath.ac.uk

James Wood
Computer and Information Sciences
University of Strathclyde
james.wood.100@strath.ac.uk

1 Introduction

In normal typed λ-calculi, variables may be used multiple times, in multiple contexts, for multiple reasons, as long as the types agree. The disciplines of linear types [Girard 1987] and coeffects [Brunel et al. 2014; Ghica and Smith 2014; Petricek et al. 2014] refine this by tracking variable usage. We might track how many times a variable is used, or if it is used co-, contra-, or invariantly. Such a discipline yields a general framework of "context constrained computing", where constraints on variables in the context tell us something interesting about the computation being performed.

We will present work in progress on capturing the "intensional" properties of programs via a family of Kripke indexed relational semantics that refines a simple set-theoretic semantics of programs. The value of our approach lies in its generality. We can accommodate the following examples:

1. Linear types that capture properties like "all list manipulating programs are permutations". This example uses the Kripke-indexing to track the collection of datums currently being manipulated by the program.
2. Monotonicity coeffects that track whether a program uses inputs co-, contra-, or in-variantly (or not at all).
3. Sensitivity typing, tracking the sensitivity of programs in terms of input changes. This forms the core of systems for differential privacy [Reed and Pierce 2010].
4. Information flow typing, in the style of the Dependency Core Calculus [Abadi et al. 1999].

Through discusssion at the workshop, we hope to discover more applications of our framework. In future work, we plan to extend our framework with type dependency, and to explore the space of inductive data types and elimination principles possible in the presence of usage information.

The syntax and semantics we present here have been formalised in Agda: https://github.com/laMudri/quantitative/. Formalisation of the examples is in progress.

2 Syntax and Typing

We define our type system with respect to a partially ordered semiring R for tracking how variables are used. A *partially ordered semiring* $(R, \leq, 0, +, 1, \cdot)$ is a poset $(R, \leq)$, commutative monoid $(R, 0, +)$, and monoid $(R, 1, \cdot)$, such that $\cdot$ distributes over 0 and $+$, and $+$ and $\cdot$ are monotonic with respect to $\leq$. We take $\rho, \pi \in R$.

PL'18, January 01–03, 2018, New York, NY, USA
2018.

Examples 1. The zero-one-many semiring $\{0, 1, \omega\}$ simulates linear typing in our system. 2. Monotonicity typing uses the semiring with carrier $\{0, \uparrow, \downarrow, \equiv\}$, where the multiplicative unit is $\uparrow$ (covariance). The $\downarrow$ represents contra-variance, and $\equiv$ represents invariance. 3. Sensitivity analysis uses the semiring with carrier $\mathbb{R} \cup \{\infty\}$ with min and $+$ as the addition and multiplication. 4. Information flow analysis uses the semiring with carrier $\mathcal{P}(L)$, where L is some set of labels.

The base language we consider is a bidirectional [Pierce and Turner 2000] simply typed λ calculus with the following types, where ι ranges over some set of base types:

$$S, T ::= S \multimap T \mid !_\rho S \mid 1 \mid S \otimes T \mid \top \mid S \,\&\, T \mid 0 \mid S \oplus T \mid \iota$$

Bidirectional typing reduces the type annotations required. Since our language is bidirectionally typed, we have two syntactic categories of terms: s ranges over checked terms, and e ranges over synthesising terms. We use t for both.

$$s ::= \lambda x.s \mid \text{bang } s \mid *_\otimes \mid (s_0, s_1)_\otimes \mid *_\& \mid (s_0, s_1)_\& \mid \text{inj}_i(s) \mid \underline{e}$$
$$e ::= x \mid e\ s \mid \text{bm}_T(e, \{x\}s) \mid \text{del}_T(e, s) \mid \text{pm}_T(e, \{x, y\}s)$$
$$\mid \text{proj}_i(e) \mid \text{ex-falso}_T(e) \mid \text{case}_T(e, \{x\}s_0, \{y\}s_1) \mid s : S$$

where curly braces and λ denote variable binding and we take $i \in \{0, 1\}$ wherever it appears.

Contexts Γ assign to each variable a type S and a usage $\rho \in R$: $\Gamma = x_1 \overset{\rho_1}{:} S_1, \ldots, x_n \overset{\rho_n}{:} S_n$. Contexts whose variables and types match form a left R-semimodule, by pointwise addition and scaling of the usage annotations. The partial order on R is extended pointwise to contexts. Typing judgements for checked and synthesising terms have the same contexts, but either record that a term is checked against a type ($\Gamma \vdash T \ni s$) or synthesise a type ($\Gamma \vdash e \in T$).

The typing rules consist of a variable rule, two rules for change of direction, and introduction and elimination rules for each type former. The following rules for variables and function and $!_\rho$ introduction and elimination illustrate how usage information is tracked:

$$\frac{\Gamma \leq 0\Gamma_1, x \overset{1}{:} S, 0\Gamma_2}{\Gamma \vdash x \in S} \qquad \frac{\Gamma, x \overset{1}{:} S \vdash T \ni s[x]}{\Gamma \vdash S \multimap T \ni \lambda x.s[x]}$$

$$\frac{\Gamma_1 \vdash e \in S \multimap T \qquad \Gamma_2 \vdash S \ni s \qquad \Gamma \leq \Gamma_1 + \Gamma_2}{\Gamma \vdash e\ s \in T}$$

$$\frac{\Gamma_1 \vdash S \ni s \qquad \Gamma \leq \rho \cdot \Gamma_1}{\Gamma \vdash\ !_\rho S \ni \text{bang } s} \qquad \frac{\begin{array}{l}\Gamma_1 \vdash e \in\ !_\rho S \\ \Gamma_2, x \overset{\rho}{:} S \vdash T \ni s[x] \\ \Gamma \leq \Gamma_1 + \Gamma_2\end{array}}{\Gamma \vdash \text{bm}_T(e, \{x\}s[x]) \in T}$$

Sub-resourcing, weakening (adding 0-use variables to the context) and substitution are all admissible. In our Agda formalisation, we have constructed our type system in two levels: a non-usage tracked simply-typed λ-calculus, with a usage-tracking system layered above. This emphasises the use of coeffect annotations as an *analysis* of programs, they do not affect the underlying semantics, but comment on it. We introduce our semantic framework in the next section.

3 Semantics

Underlying Semantics We give a standard semantics of types and well typed terms into sets and functions. This semantics ignores the usage information. For types, we have:

$$\begin{array}{ll} [\![\iota]\!] = A_\iota & \\ [\![S \multimap T]\!] = [\![S]\!] \to [\![T]\!] & [\![!_\rho S]\!] = [\![S]\!] \\ [\![1]\!] = [\![\top]\!] = \{*\} & [\![S \otimes T]\!] = [\![S \,\&\, T]\!] = [\![S]\!] \times [\![T]\!] \\ [\![0]\!] = \{\} & [\![S \oplus T]\!] = [\![S]\!] \uplus [\![T]\!] \end{array}$$

Contexts are interpreted as left-nested products. Terms are assigned the usual semantics as functions $[\![t]\!] : [\![\Gamma]\!] \to [\![S]\!]$.

Usage-tracking semantics To derive interesting properties from our type system, we refine the set-theoretic semantics by Kripke-indexed binary relations. This gives a fundamental lemma for our system, that when instantitated in different ways captures the examples in the introduction.

Our framework is parameterised by a category $\mathcal{W}$ of *possible worlds* that track how resources are distributed by programs. To interpret resource separation, we assume that $\mathcal{W}$ has *symmetric promonoidal structure*: profunctors $J : 1 \nrightarrow \mathcal{W}$ and $P : \mathcal{W} \times \mathcal{W} \nrightarrow \mathcal{W}$ such that $P \odot (J \times 1) \cong 1$, $P \odot (1 \times J) \cong 1$, $P \odot (1 \times P) \cong P \odot (P \times 1)$, and $P \cong P \odot (\pi_2 \times \pi_1)$, and the triangle, pentagon, and hexagon laws hold[1].

We now assign to each type T a functor $[\![T]\!]^R : \mathcal{W}^{op} \to \mathrm{Rel}\,[\![T]\!]$ that captures a notion of $\mathcal{W}$-indexed "indistinguishability". To interpret $!_\rho S$, we assume we are given a relation transformer $!_A : R^{op} \to \mathrm{Rel}(A)^{\mathcal{W}^{op}} \to \mathrm{Rel}(A)^{\mathcal{W}^{op}}$ that satisfies the axioms of a monoidal exponential comonad. The interesting cases are for functions, $\otimes$-products and the $!_\rho$ modality:

$$\begin{array}{l} [\![S \multimap T]\!]^R\ w\ (f, f') = \\ \quad \forall x, y.\ P(y, w)x \Rightarrow \forall s, s'.\ [\![S]\!]^R y (s, s') \Rightarrow [\![T]\!]^R x (f\ s, f'\ s') \\ [\![S \otimes T]\!]^R\ w\ ((s, t), (s', t')) = \\ \quad \exists x, y.\ P(x, y)w \wedge [\![S]\!]^R x (s, s') \wedge [\![T]\!]^R y (t, t') \\ [\![!_\rho S]\!]^R\ w\ (s, s') = !_\rho [\![S]\!]^R\ w\ (s, s') \end{array}$$

Contexts $x_1 \overset{\rho_1}{:} S_1, \ldots, x_n \overset{\rho_n}{:} S_n$ are interpreted as if they were $[\![(\cdots (1 \otimes !_{\rho_1} S_1) \cdots \otimes !_{\rho_n} S_n)]\!]$. With these definitions, we can prove the following fundamental lemma for our Kripke-indexed relational semantics.

Theorem 3.1 (Fundamental Lemma)**.**

$$\Gamma \vdash t : T \implies [\![\Gamma]\!]^R w\ (\gamma, \gamma') \implies [\![T]\!]^R w\ ([\![t]\!]\gamma, [\![t]\!]\gamma')$$

[1]We don't need the laws to hold to prove the fundamental lemma.

Example Instantiations The ingredients of our fundamental lemma are perhaps well known (relational interpretations, Kripke-indexing), but the value of our framework lies in the generality of being able to choose $\mathcal{W}$ and its promonoidal structure, and the interpretion the $!_\rho$ modality as a relation transformer. Examples include:

Permutation Types With the $\{0, 1, \omega\}$ semiring, we take the category $\mathcal{W}$ to consist of lists of some type of keys, and permutations between them. The relation transformer is defined as: $!_0 R\ l = \top$, where $\top$ is the total relation, $!_1 R\ l = R\ l$ and $R_\omega\ R\ l = (l = []) \wedge R\ l$. With suitable types of keys and lists of keys, the fundamental lemma states that all programs are permutations. This result has already been formalised in a one-off type system at https://github.com/bobatkey/sorting-types.

Monotonicity Types With R the partially ordered semiring with carrier $\{0, \uparrow, \downarrow, \equiv\}$ ordered $\equiv \leq \uparrow, \downarrow$ and $\uparrow, \downarrow \leq 0$, we take $\mathcal{W}$ to be the one-object, one-arrow category, and define the relation transformer ! to be:

$$!_0\ R = \top \qquad !_\uparrow\ R = R \qquad !_\downarrow\ R = R^{op} \qquad !_\equiv\ R = R \cap R^{op}$$

If we let our base type be natural numbers with the relational interpretation $R_{\mathrm{nat}}(n, n') \Leftrightarrow n \leq n'$, then the fundamental lemma states that a program of type $x \overset{\uparrow}{:} \mathrm{nat} \vdash t : \mathrm{nat}$ is covariant (and similarly for contravariant and invariant).

Sensitivity Analysis With the $R = \mathbb{R} \cup \{\infty\}$ semiring, we let $\mathcal{W}$ be R as well. The relation transformer is given by scaling. With a base type of real numbers with relational intepretation $R_{\mathrm{real}}\ k\ (x, x') \Leftrightarrow |x - x'| \leq k$, then the fundamental lemma states that the usage annotations on the input variables tracks the extent to which the program is sensitive to changes in those variables.

Information Flow With the $R = \mathcal{P}(L)$ semiring, we again take $\mathcal{W} = R$, and let the relation transformer to be $!_l R\ l' = \{\top \text{ when } l \geq l'; R \text{ otherwise}\}$. Then the fundamental lemma yields the same non-interference properties as stated by Abadi et al. for the DCC [Abadi et al. 1999].

Acknowledgments

James Wood is supported by a EPSRC Studentship.

References

M. Abadi, A. Banerjee, N. Heintze, and J. G. Riecke. 1999. A Core Calculus of Dependency. In *POPL '99*. 147–160.

A. Brunel, M. Gaboardi, D. Mazza, and S. Zdancewic. 2014. A Core Quantitative Coeffect Calculus. In *ESOP 2014*. 351–370.

Dan R. Ghica and Alex I. Smith. 2014. Bounded Linear Types in a Resource Semiring. In *ESOP 2014*. 331–350.

Jean-Yves Girard. 1987. Linear Logic. *Theor. Comp. Sci.* 50 (1987), 1–101.

Tomas Petricek, Dominic A. Orchard, and Alan Mycroft. 2014. Coeffects: a calculus of context-dependent computation. In *ICFP 2014*. 123–135.

Benjamin C. Pierce and David N. Turner. 2000. Local type inference. *ACM TOPLAS* 22, 1 (2000), 1–44.

J. Reed and B. C. Pierce. 2010. Distance Makes the Types Grow Stronger. In *ICFP 2010*, P. Hudak and S. Weirich (Eds.). 157–168.