CS225 Spring 2018—Final Project Proposal

Duncan Enzmann `git:@MrDJEnz` Devin Beckim `git:@dbeckim` Jacob Normyle `git:@jnormyle`

Andrew Edwards `git:@andrewgedwards`

April 24, 2018

Project:
Effect Types and Regions – ATAPL Chapter 3

We are going to create a project based off stack allocation memory lifetimes. We will design functions which will be allowed to allocate memory on its stack for a value, and then pass a reference to that value to functions called within the body of the function. The point of introducing effect types and regions into OCaml code allows us to use it as an application domain to develop fundamental concepts of effect type systems step by step.

Base Language We will be working with simple typed lambda calculus. Which include the Terms.
Terms (t ::=)

1. A value expression written as v
2. A variable written as x.
3. An application written as t t.
4. A conditional written as if t then t else t

Value Expression (v ::=)

1. An abstraction written as λ x.t
2. A truth value written as written as bv.

Truth Values (bv ::=)

1. True written as tt.
2. False written as ff

With the Evaluation rules as:

1. (E-Beta) written as: $(\lambda x.t_{12})v_2 \rightarrow [x \rightarrow v_2]t_{12}$
2. (E-FixBeta) written as: fix x.t $\rightarrow [x \rightarrow \texttt{fix x.t}]t$
3. (E-IfTrue) written as: if tt then t_2 else $t_3 \rightarrow t_2$
4. (E-IfFalse) written as: if ff then t_2 else $t_3 \rightarrow t_3$
5. (E-App1) written as: $\frac{t_1 \rightarrow t_1'}{t_1\ t_2 \rightarrow t_1'\ t_2}$
6. (E-App2) written as: $\frac{t_2 \rightarrow t_2'}{v_1\ t_2 \rightarrow v_1\ t_2'}$
7. (E-If) written as: $\frac{t_1 \rightarrow t_1'}{if\ t_1\ then\ t_2\ else\ t_3 \rightarrow if\ t_1'\ then\ t_2\ else\ t_3}$

Also includes the types:
Types (T ::=)

1. Boolean types written as bool.
2. Function type written as T $\rightarrow$ T

With the Typing Rules being:

1. (T-Var) Written as: $\frac{x \notin \Gamma'}{\Gamma, x{:}\ T, \Gamma' \vdash x : T}$
2. (T-Bool) Written as: $\Gamma \vdash bv : bool$ or $\Gamma \vdash t_1 : bool$
3. (T-If) Written as: $\frac{\Gamma \vdash t_2 {:}\ T \qquad \Gamma \vdash t_3 : T}{\Gamma \vdash if\ t_1\ then\ t_2\ else\ t_3 : T}$
4. (T-ABS) Written as: $\frac{\Gamma, x : T_1 \vdash t : T_2}{\Gamma \vdash \lambda x.t : T_1 \rightarrow T_2}$
5. (T-App) Written as: $\frac{\Gamma \vdash t_0 : T_1 \rightarrow T_2 \qquad \Gamma \vdash t_1 : T_1}{\Gamma \vdash t_0\ t_1 : T_2}$
6. (T-Fix) Written as: $\frac{\Gamma, x : T \vdash t : T}{\Gamma \vdash fix\ x.t : T}$

Extended Language Extend this language with tagging and un-tagging terms.
This consists of one new type (T::=) written as:

1. A tagged value type, written T at p

Two new terms (t ::=)

1. Type tagging, written t at p
2. Type untagging, written t ! p

One new value expressions (v ::=):

1. Tagged Value: $< v >_\rho$

One new label expressions (p ::=):

1. Tagged Value: ρ

New Evaluation Rules:

1. (E-Tag) Written as: $\frac{t \to t'}{t\ at\ \rho \to t'\ at\ \rho}$
2. (E-TagBeta) Written as: $v\ at\ \rho\ \to < v >_\rho$
3. (E-UnTag) Written as: $\frac{t \to t'}{t\ !\ \rho \to t'\ !\ \rho}$
4. (E-UnTagBeta) Written as: $< v >_\rho\ !\ \rho \to v$

Applications

Label

1. Labels let us distinguish values in different places although they are extensionally equal.
2. For purposes, labels can be though of as "regions".

Tag/UnTag Variables

1. The tagging and untagging operations serve to name certain sets of values and to mark where such values are constructed and used.
2. Note: multiple sub terms of a term may have the same label. Even though a label may occur once in a program it may tag multiple values at run time.

Typing based on terms:

1. Consistent with our material, every term has a "type" (and a label) and operations are restricted to terms of a certain type.

Constructor/De-constructor completion

1. A completion where each expression is tagged and untagging takes place.

Project Goals For this project, we plan to complete:

1. Type existing variables via explicit tagging and untagging operations
2. Create outline for reference-passing function designed to allocated memory on a stack by using new effect types for system variables.
3. We hope to get to implement the effect type judgments by using an effect expression, if there is an empty effect we write it as $\emptyset$, and $^{\emptyset}$T is abbreviated to T.
4. The typing rule may depend not only on the results of evaluations, but on certain aspects of the evaluation itself, in other words on how a value is computed, not just which value is computed. To capture properties of evaluation we will introduce effects.

Expected Challenges We expect that implementing effects into our call by value language will cause some issue in that capturing properties of the evaluation, since there cannot be any judgments using a deleted or accessible region or free variables, rather we need to denote $\Gamma \vdash t :^{\phi} T$ to express free variables are going to be bound to the values of the types. Rather than the type of it self.

Timeline and Milestones By the checkpoint we hope to have completed:

1. A complete writeup of this document.
2. A plan for a comprehensive list of test cases for our new terms and types.
3. Research more into Effect Typed Languages.

By the final project draft we hope to have completed:

1. Full implementation of the test cases of the newly introduced terms types
2. Having all the tests passing the suite.
3. Having a fully comprehensive working program.
4. A draft writeup that explains the on-paper formalism of our implementation
5. A draft of a presentation with 5 slides as the starting point for our in-class presentation

By the final project submissions we hope to have completed:

1. The final writeup and presentation
2. Any remaining implementation work that was missing in the final project draft