On Optimality of Ownership Type Inference

Wei Huang and Ana Milanova

Rensselaer Polytechnic Institute, Troy NY, USA

Abstract. Despite the benefits of ownership type systems, practical adoption is lacking. This is due to some extent to the onerous annotation requirements most systems impose. Therefore, ownership type inference is an important problem.
One reason (arguably, the primary reason) why ownership type inference is difficult, is that ownership type systems permit many possible type assignments, and there is no notion of an optimal assignment. In this paper, we define the notion of the optimal ownership type assignment, and outline several research directions.

We consider the classical ownership type system from [1] restricted to one ownership parameter. Type annotation $\langle q_0|q_1\rangle$ consists of two parts: q_0 is the owner of the object and q_1 is the ownership parameter passed to that object. An annotated field or local variable is written as $\langle q_0|q_1\rangle\ \mathsf{C}\ \mathsf{x}$ and allocation site $\mathsf{new}\ \langle q_0|q_1\rangle\ \mathsf{C}()$. q_0 and q_1 can be one of the following values: rep, own, and p. rep denotes the object is owned by this and belongs to this's representation; own denotes the object is owned by the owner of this, and p denotes the object's owner is the ownership parameter of this.

$$\frac{E(\mathsf{x}) = t\ \mathsf{C}}{E \vdash \mathsf{x} = \mathsf{new}\ t\ \mathsf{C}}\ (\textsc{tnew}) \qquad \frac{E(\mathsf{x}) = t\ \mathsf{C} \quad E(\mathsf{y}) = t\ \mathsf{C}}{E \vdash \mathsf{x} = \mathsf{y}}\ (\textsc{tassign})$$

$$\frac{\mathsf{x} \neq \mathsf{this} \quad E(\mathsf{x}) = t_x\ \mathsf{C} \quad typeof(\mathsf{C.f}) = t_f\ \mathsf{D} \quad E(\mathsf{y}) = t_y\ \mathsf{D} \quad adapt(t_f, t_x) = t_y}{E \vdash \mathsf{x.f} = \mathsf{y}}\ (\textsc{twrite}) \qquad \frac{E(\mathsf{this}) = t'\ \mathsf{C} \quad typeof(\mathsf{C.f}) = t\ \mathsf{D} \quad E(\mathsf{y}) = t\ \mathsf{D}}{E \vdash \mathsf{this.f} = \mathsf{y}}\ (\textsc{twritethis})$$

$$\frac{\mathsf{y} \neq \mathsf{this} \quad E(\mathsf{y}) = t_y\ \mathsf{C} \quad typeof(\mathsf{C.f}) = t_f\ \mathsf{D} \quad E(\mathsf{x}) = t_x\ \mathsf{D} \quad adapt(t_f, t_y) = t_x}{E \vdash \mathsf{x} = \mathsf{y.f}}\ (\textsc{tread}) \qquad \frac{E(\mathsf{this}) = t'\ \mathsf{C} \quad typeof(\mathsf{C.f}) = t\ \mathsf{D} \quad E(\mathsf{x}) = t\ \mathsf{D}}{E \vdash \mathsf{x} = \mathsf{this.f}}\ (\textsc{treadthis})$$

$$\frac{E(\mathsf{y}) = t_y\ \mathsf{C} \quad typeof(\mathsf{C.m}) = \overline{t\ \mathsf{D}} \to t'\ \mathsf{D}' \quad \mathsf{y} \neq \mathsf{this} \quad E(\mathsf{x}) = t_x\ \mathsf{D}' \quad E(\overline{\mathsf{z}}) = \overline{t_z\ \mathsf{D}} \quad adapt(\overline{t}, t_y) = \overline{t_z} \quad adapt(t', t_y) = t_x}{E \vdash \mathsf{x} = \mathsf{y.m}(\overline{\mathsf{z}})}\ (\textsc{tcall})$$

$$\frac{E(\mathsf{this}) = t''\ \mathsf{C} \quad typeof(\mathsf{C.m}) = \overline{t\ \mathsf{D}} \to t'\ \mathsf{D}' \quad E(\mathsf{x}) = t'\ \mathsf{D}' \quad E(\overline{\mathsf{z}}) = \overline{t\ \mathsf{D}}}{E \vdash \mathsf{x} = \mathsf{this.m}(\overline{\mathsf{z}})}\ (\textsc{tcallthis})$$

Fig. 1. Typing Rules

Fig. 1 shows the typing rules (see [2] for additional details). E is a type mapping from variables to the annotated types; $t\ \mathsf{C}$ is an annotated type, in

which ownership type t can be $\langle \mathsf{rep}|\mathsf{rep}\rangle$, $\langle \mathsf{rep}|\mathsf{own}\rangle$, $\langle \mathsf{rep}|p\rangle$, $\langle \mathsf{own}|\mathsf{own}\rangle$, $\langle \mathsf{own}|p\rangle$ or $\langle p|p\rangle$. Viewpoint adaptation is used to adapt ownership type t from the point of view of ownership type t'. *adapt* is defined below:

$$\begin{aligned} adapt(\langle \mathsf{own}|\mathsf{own}\rangle, \langle q_0|q_1\rangle) &= \langle q_0|q_0\rangle \\ adapt(\langle \mathsf{own}|p\rangle, \langle q_0|q_1\rangle) &= \langle q_0|q_1\rangle \\ adapt(\langle p|p\rangle, \langle q_0|q_1\rangle) &= \langle q_1|q_1\rangle \end{aligned}$$

We begin by giving a graphical interpretation of the ownership type inference problem. The type constraints in Fig. 1 fall into two categories, *unification constraints* (for rules (TNEW), (TASSIGN), (TWRITETHIS), (TREADTHIS), (TCALLTHIS)) and *adapt constraints* (for rules (TWRITE), (TREAD), (TCALL)). These constraints induce a graph G as follows.

The unification constraints partition the set of variables[1] into k equivalence classes P_i, where members of each P_i receive the same ownership type. For the example in Fig. 2, statement $\mathsf{j1} = \mathsf{new\ J()}$ results in equivalence class $P_1 = \{\mathsf{j1}, j\}$, $\mathsf{i1} = \mathsf{new\ I()}$ results in $P_2 = \{\mathsf{i1}, i\}$ and $\mathsf{f2} = \mathsf{j2}$ results in $P_3 = \{\mathsf{f2}, \mathsf{j2}\}$. The equivalence classes correspond one-to-one to the edges of G. The adapt constraints "combine" those edges to define the nodes in G. For example, statement $\mathsf{i1.init(j1)}$ combines edges P_2 (the class of i1), P_1 (the class of j1), and P_3 (the class of formal parameter j2) as follows: it merges the source of P_2 with the source of P_1, it merges the target of P_2 with the source of P_3, and it merges the target of P_1 with the target of P_3.

Fig. 3(a) shows G for the program in Fig. 2. The unification constraints partition the variables into 14 equivalence classes: $P_1 = \{\mathsf{j1}, j\}$, $P_2 = \{\mathsf{i1}, i\}$, $P_3 = \{\mathsf{f2}, \mathsf{j2}\}$, $P_4 = \{\mathsf{f1}, k\}$, $P_5 = \{\mathsf{m1}\}$, $P_6 = \{\mathsf{n1}\}$, $P_7 = \{\mathsf{n2}\}$, $P_8 = \{\mathsf{g1}, m, \mathsf{n}\}$, $P_9 = \{\mathsf{g2}, l\}$, $P_{10} = \{\mathsf{h1}, n1\}$, $P_{11} = \{\mathsf{h2}, n2\}$, $P_{12} = \{\mathsf{j3}\}$, $P_{13} = \{\mathsf{j4}\}$, $P_{14} = \{\mathsf{m2}\}$. These classes correspond to the edges of G. Subsequently, the adapt constraints combine the edges to determine the nodes of G.

One important observation is that G is an abstraction of the run-time object graph (see [1] for a discussion on object graphs). The nodes in G correspond to abstract objects, and the edges correspond to the access relations between these objects. It is however, a somewhat unexpected abstraction, in the sense that the abstraction for objects is incomparable to the typical abstraction for objects which maps each run-time object to its allocation site. In our running example, the nodes in G correspond one-to-one to the allocation sites. This is not the case in general: one can easily construct cases where many nodes in G correspond to one allocation site, and vice versa, many allocation sites correspond to one node in G. One direction of future work is to study the degree of approximation inherent in the ownership type system — although in theory it is incomparable to allocation-based abstraction schemes, our intuition tells us that in practice it will be significantly less precise than those schemes.

A *valid type assignment* $T(P_i)$ is one that satisfies all adapt constraints. E.g., the adapt constraint due to statement $\mathsf{i1.init(j1)}$ requires $adapt(T(P_3), T(P_2)) = T(P_1)$. One can see that there are many valid type assignments, which gives rise

[1] "Variables" refers to local variables, fields, allocation sites and method return values.

```
1  class I {
2    K f1; J f2;
3    static void main(String[] arg) {
4      J j1 = new J();    [j]
5      I i1 = new I(); i1.init(j1);    [i]
6      i1.m();
7    }
8    void init(J j2) {
9      f2 = j2;
10     f1 = new K(); f1.init(f2);    [k]
11   }
12   void m() {
13     M m1 = f1.n();
14     N n1 = m1.h1;
15     N n2 = m1.h2;
16   }
17 }
```

```
1  class K {
2    M g1; L g2;
3    void init(J j3) {
4      g1 = new M(); g1.init();    [m]
5      g2 = new L(); g2.init(j3,g1);    [l]
6    }
7    M n() {
8      return g1;
9    }
10 }
11 class M {
12   N h1; N h2;
13   void init() {
14     h1 = new N();    [n1]
15     h2 = new N();    [n2]
16   }
17 }
18 class L {
19   void init(J j4, M m2) { ... }
20 }
```

Fig. 2. Simple program.

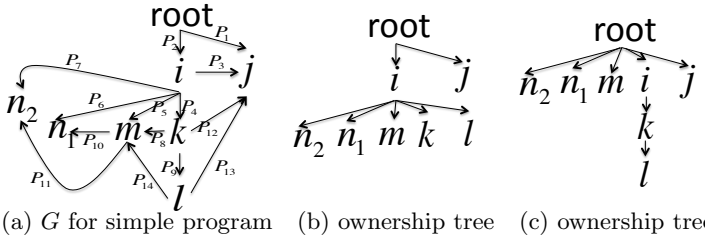

Fig. 3. (a) G for simple program (b) ownership tree (c) ownership tree

to a number of interesting questions. What is an optimal assignment? Can we formalize the notion of the optimal assignment?

We claim that the optimal assignment should *preserve as much of the inherent (existing) dominance as possible*. Formally, the optimal assignment $T(P_i)$ is such that (1) it satisfies the adapt constraints, and (2) it maximizes the objective function $\sum_{i=1}^{k} f(T(P_i))$ where $f(\langle q_0|q_1\rangle)$ returns 1 if q_0 is rep and 0 otherwise.

In our example, we aim at rep assignments for P_1 and P_2 (root dominates j and i), P_4, P_5, P_6 and P_7 (i dominates k, m, $n1$ and $n2$), and for P_9 (k dominates l), but unfortunately it is impossible to type all these edges as rep because of the adapt constraints.

One valid type assignment is $T(P_1) = \langle \mathsf{rep}|\mathsf{rep}\rangle$, $T(P_2) = \langle \mathsf{rep}|\mathsf{rep}\rangle$, $T(P_3) = \langle \mathsf{own}|\mathsf{own}\rangle, T(P_4) = \langle \mathsf{rep}|\mathsf{own}\rangle, T(P_5) = \langle \mathsf{own}|\mathsf{own}\rangle, P_6 = \langle \mathsf{own}|\mathsf{own}\rangle, T(P_7) = \langle \mathsf{own}|\mathsf{own}\rangle, T(P_8) = \langle p|p\rangle$, $T(P_9) = \langle \mathsf{rep}|p\rangle$, $T(P_{10}) = \langle \mathsf{own}|\mathsf{own}\rangle$, $T(P_{11}) = \langle \mathsf{own}|\mathsf{own}\rangle$, $T(P_{12}) = \langle p|p\rangle$, $T(P_{13}) = \langle p|p\rangle$, $T(P_{14}) = \langle p|p\rangle$. This assignment results in the ownership

tree in Fig. 3(c) and the value of its objective function is 4. Note that the typing of edge P_9 as rep forces edges P_5, P_6 and P_7 to become own.

Another valid type assignment is $T(P_1) = \langle \mathsf{rep}|\mathsf{rep}\rangle, T(P_2) = \langle \mathsf{rep}|\mathsf{rep}\rangle$, $T(P_3) = \langle \mathsf{own}|\mathsf{own}\rangle, T(P_4) = \langle \mathsf{rep}|\mathsf{own}\rangle$, $T(P_5) = \langle \mathsf{rep}|\mathsf{own}\rangle$, $P_6 = \langle \mathsf{rep}|\mathsf{own}\rangle$, $T(P_7) = \langle \mathsf{rep}|\mathsf{own}\rangle$, $T(P_8) = \langle \mathsf{own}|p\rangle$, $T(P_9) = \langle \mathsf{own}|p\rangle$, $T(P_{10}) = \langle \mathsf{own}|p\rangle$, $T(P_{11}) = \langle \mathsf{own}|p\rangle$, $T(P_{12}) = \langle p|p\rangle$, $T(P_{13}) = \langle p|p\rangle$, $T(P_{14}) = \langle \mathsf{own}|p\rangle$. This assignment results in the ownership tree in Fig. 3(b) and the value of its objective function is 6. Essentially, we sacrifice the choice of rep for edge P_9, which allows us to type edges P_5, P_6 and P_7 as rep. We argue that the second typing (and tree (b)) is better than the first typing (tree (c)) because it preserves more of the existing dominance.

Our ongoing research centers around the following questions. Is the above notion of optimality sensible? What are other useful notions of optimality? How do we combine optimality with programmer intent expressed with annotations? Given an ownership type system with a fixed number n of ownership parameters, can we design an efficient algorithm which computes an optimal assignment?

References

1. D. G. Clarke, J. M. Potter, and J. Noble. Ownership types for flexible alias protection. *ACM SIGPLAN Notices*, 33(10):48–64, Oct. 1998.
2. A. Milanova and J. Vitek. Static Dominance Inference. In *Proceedings of TOOLS Europe 2011, to appear*, 2011.