Denotational Semantics of first-order Object Calculus

Thomas Sixuan Lou
University of Washington
lous@cs.washington.edu

Abstract

This paper is an extension of the work by Apt (Apt 2000) to a pure object calculus.

Keywords Semantics, Constraint Logic Programming, Object Calculus

ACM Reference format:
Thomas Sixuan Lou. 2016. Denotational Semantics of first-order Object Calculus. In *Proceedings of ACM Conference, Washington, DC, USA, July 2017 (Conference'17),* 7 pages.
DOI: 10.1145/nnnnnnn.nnnnnnn

1 Introduction

At its core, constraint logic programming may be seen as a formalization of both logic and computation by solving systems of relations. From a proof-theoretic perspective, given a language $\mathcal{L}$, a logic on $\mathcal{L}$ is a structure generated from a set of primitive rules, which may be seen as a set of constraints on the language; while a constraint solver corresponds to the computational interpretation of the logic. It's thus natural of using constraints and substitutions (i.e. solutions to constraints) to give a denotational semantics for the logic system we want to formalize.

This project will be an extension to the work by Apt (Apt 2000), where he gave a denotational semantics for first-order logic by embedding first-order terms into constraints. He also proved the desired notion of "truth" in the logic coincides with constraint resolution in this denotation. In this project, we propose to incorporate objects into the logic so that we can solve local constraints for each object - for example, an (insntance) solution to a object is an (possibly non-ground) substitution of the object's local fields satisfying constraints declared in the object itself.

The idea is to embed first-order terms with equality into each object and use substitution for field update and constraint resolution. We will explore the usual object oriented concepts (Abadi and Cardelli 1996) such as (possibly delegation and) inheritance, as well as whether it is possible to establish the soundness of such denotation.

Team member will include myself.

2 A Pure Calculus with Constraints

In this section, we will define $\mathbf{O}_c$, an pure object-based calculus with constraints introduced through a special binder $\iota(o).e$.

The calculus is parametrized over an algebra and an theory on that algebra (definitions below are similar to those formulated in (Birkhoff 1935; Burris and Sankappanavar 1981)). Intuitively, the algebra describes what are the primitive expressions in our calculus. An theory defines a equivalence relation on that algebra, which our calculus should be invariant under. In addition, the solver will solve constraints with respect to the theory defined on the algebra.

2.1 Algebra and Theory

Definition 2.1 (Type)**.** A set of function symbols $\mathscr{F}$ is called an *type* if for every $f \in \mathscr{F}$, it is assigned a non-negative integer n, called the *arity* of f. Let $\mathscr{F}_k$ denote the set of function symbols with arity k.

Definition 2.2 (Algebra)**.** An *algebra* $\mathcal{A}$ of type $\mathscr{F}$ is a pair $\langle A, F\rangle$, where $A \neq \varnothing$ is called a *carrier set* and F is a set of finitary operations on A so that for every n-ary function symbol $f \in \mathscr{F}_n$, there is an n-ary operation $f^{\mathcal{A}}$ on A (i.e. $f^{\mathcal{A}} : A^n \longrightarrow A$).

The map from every $f \in \mathscr{F}$ to $f^{\mathcal{A}}$ is called an *interpretation.*

The superscript $(-)^{\mathcal{A}}$ is often omitted when the context is clear.

Definition 2.3 (Language)**.** Given a set of set of distinct objects $\mathbf{X}$ (variables), and a type $\mathscr{F}$, a language $\mathcal{L}(\mathbf{X}, \mathscr{F})$ of type $\mathscr{F}$ over $\mathbf{X}$ is the smallest set such that

1. $\mathbf{X} \cup \mathscr{F}_0 \subset \mathcal{L}(\mathbf{X}, \mathscr{F})$
2. if $p_1, \ldots, p_n \in \mathcal{L}(\mathbf{X}, \mathscr{F})$, $f \in \mathscr{F}_n$, then the string $f(p_1, \ldots, p_n) \in \mathcal{L}(\mathbf{X}, \mathscr{F})$.

The parameters $\mathbf{X}$ and $\mathscr{F}$ are omitted when the context is clear.

Definition 2.4 (Theory)**.** Given type $\mathscr{F}$, a *theory* "$\sim$" is a relation between terms in $\mathcal{L}(\mathbf{X}, \mathscr{A})$, written as $p(x_1, \ldots, x_n) \sim q(x_1, \ldots, x_n)$. Arguments are often omitted when we are not making pointwise statements.

An algebra $\mathcal{A}$ of type $\mathscr{F}$ satisfies $p \sim q$ if and only if

$$\forall a_1, \ldots a_n \in A \left[p^{\mathcal{A}}(a_1, \ldots, a_n) = q^{\mathcal{A}}(a_1, \ldots, a_n)\right]$$

Example 2.5 (Modular arithmetic)**.** The arithmetic algebra $\mathbb{Z}/_m = \langle \mathbb{Z}, 0, 1, m, +, \times \rangle$ is an algebra over with carriers of integers $\mathbb{Z}$, characteristic elements $0, 1 : \varnothing \to \mathbb{N}$ and the following binary operations:

$$+, \times : \mathbb{Z}^2 \to \mathbb{Z}$$

satisfying

$$\begin{array}{cc} x + y \sim x \times y & x \times y \sim y \times x \\ 0 + x \sim x & x + (y + z) \sim (x + y) + z \\ 1 \times x \sim x & x \times (y \times z) \sim (x \times y) \times z \\ x \times (y + z) \sim x \times y + x \times z & x + y \times m \sim x \end{array}$$

Example 2.6 (Boolean algebra)**.** A distributive lattice $\mathcal{L}$ on set L is the algebra $\langle L, \wedge, \vee \rangle$ with operations $\wedge, \vee : L^2 \to L$ satisfying

$$\begin{array}{cc} x \wedge x \sim x & x \vee x \sim x \\ x \wedge y \sim y \wedge x & x \vee (x \wedge y) \sim x \\ x \vee y \sim y \vee x & x \wedge (x \vee y) \sim x \\ x \wedge (y \wedge z) \sim (x \wedge y) \wedge z & x \wedge (y \vee z) \sim (x \wedge y) \vee (x \wedge z) \\ x \vee (y \vee z) \sim (x \vee y) \vee z & x \vee (y \wedge z) \sim (x \vee y) \wedge (x \vee z) \end{array}$$

A boolean algebra $\mathcal{B} = \langle B, 0, 1, \wedge, \vee, ' \rangle$ is an extension to $\mathcal{L}$ with characteristic elements $0, 1$ and an complement operator $' : B \to B$, satisfying

$$\begin{array}{cc} x \wedge 0 \sim 0 & x \wedge 1 \sim x \\ x \vee 1 \sim 1 & x \vee 0 \sim x \\ x' \wedge x \sim 0 & x' \vee x \sim 1 \end{array}$$

An instance of boolean algebra is $\mathbf{Su}(X)$ (subsets of X) with $B = \mathcal{P}(X)$, $0 = \varnothing$, 1 is any singleton set. The theory on $\mathbf{Su}(X)$ is interpreted as (note this of the semantics portion, which is independent from the syntactic/algebraic definitions):

$$x \sim y \text{ iff } \forall z\, [z \in x \leftrightarrow z \in y]$$

In this paper, the constraint solver is abstracted as a blackbox that is able to find a single best solution from a constraint whenever possible. The solver is also parametrized over an algebra and a theory so that it's general enough to solve equivalence constraints of arbitrary interpretations.

For example, an solver $\mathfrak{S}$ over $\mathbb{Z}/_7$ is expected to be able to obtain a solution from constraints on the relation $\pmod 7$: given constraint $c = x \sim 0 \wedge y \sim x$ (where x and y are objects in the set of variable $\mathbf{X}$), it should return a solution $\{x \mapsto 7, y \mapsto 0\}$. Which solution to return is up to specific implementations of solvers.

2.2 Syntax Formalization

We now formally introduce the calculus. The syntax of the calculus is constructed from a language $\mathcal{L}(\mathbf{X}, \mathscr{F})$, and various special syntactical constructs. There are two syntactical categories, the object fragment ("a") and the constraint fragment ("c","e"). They are separated so that rules involving constraint solving are easier to write.

The complete syntax definition is given in Figure 1. Each object is declared as a collection of ⟨field name, method⟩ and ⟨constraint name, constraint⟩ pairs. Each method (resp. constraint) is given as a binder "$\varsigma(x).a$" (resp. "$\iota(x).e$"), where x is bound to which object it's invoked from.

Constraints in our calculus are of the form $\iota(x).e$, where x is a variable which will bind to the object this constraint is invoked from before solving the constraint term e. Constraint terms are constructed from atomic $\sim$-relations and using connectives including $\wedge$, $\vee$ and $\neg$. Notice, there are no $\exists$ quantifications on constraint terms since every name(label) is already existentially quantified in each of the object.

We define free variables of $\mathbf{O}_c$ terms and substitutions in Figure 2 and Figure 3 respectively.

Since our calculus was parametrized over arbitrary algebras and theories, we define the notion of evaluation that evaluate a term involving primitives (i.e. with terms in $\mathcal{L}$) down to its general form by substituting primitives with terms and formulas in the algebra.

In evaluating terms using the equational facts from the algebra, we have avoided adding rules of evaluating arbitrary terms into our calculus. Thus, we only need to focus the core object and constraint solving fragment in our calculus.

Definition 2.7 (Evaluation)**.** Evaluation is the extended notion of interpretation from languages to algebras.

Let $\mathcal{L}(\mathbf{X}, \mathscr{F})$ be a language, let $\mathcal{A} = \langle A, F \rangle$ be an algebra. A *evaluation* of term a is defined inductively in Figure 4. The evaluated term is called a *generalized term.*

2.3 Semantics of Constraints

Let $\mathfrak{S}_{\mathcal{A},\mathcal{T}}$ be a solver parametrized by an algebra $\mathcal{A}$ and a theory $\mathcal{T}$. It interacts with our calculus by the following satisfaction judgment $o(c) \models_{\mathfrak{S}_{\mathcal{A},\mathcal{T}}} S$, where o is an object, S is a set of objects and c is a constraint. The judgment reads: S is the set of objects that satisfies constraint c when evaluated from an particular object o.

This satisfaction judgment gives the semantics of equations (more generally, $\sim$-relations) in our calculus. If we

$$\begin{array}{lll}
c ::= \iota(x).e & \text{constraint binder} & \\
e ::= & \text{constraint terms} & \\
\quad a_1 \sim a_2 & \quad \text{relation} & \\
\quad e_1 \wedge e_2 & \quad \text{conjunction} & \\
\quad e_1 \vee e_2 & \quad \text{disjunction} & \\
\quad \neg e_1 & \quad \text{negation} & \\
\mathcal{E}[-] & \text{context} &
\end{array}
\qquad
\begin{array}{ll}
a, b ::= & \text{terms} \\
\quad t(a_1, \ldots, a_n) & \quad \text{primitive terms } t \in \mathcal{L} \\
\quad x & \quad \text{variable} \\
\quad a.l_i \quad a.r_i & \quad \text{method/constraint invocation} \\
\quad [l_i = \varsigma(x).b_i, r_j = c_j]_{\substack{i=1..n \\ j=1..m}} & \quad \text{object} \\
\quad a.l_i \Leftarrow \varsigma(x).b \quad a.r_i \Leftarrow c & \quad \text{method/constraint update}
\end{array}$$

Figure 1. Syntax of $\mathbf{O}_c$ under theory $\mathcal{T}$ on algebra $\mathcal{A}$

$$\begin{aligned}
FV(t(a_1, \ldots, a_n)) &= \bigcup_{i \in [n]} FV(a_i) \\
FV(x) &= \{x\} \\
FV(l) &= \{l\} \\
FV(a.l_i) &= FV(a) \cup \{l_i\} \\
FV(a.r_i) &= FV(a) \cup \{r_i\} \\
FV([l_i = \varsigma(x).b_i, r_j = c_j]_{\substack{i \in [n] \\ j \in [m]}}) &= \bigcup_{i \in [n]} FV(\varsigma(x).b_i) \cup \bigcup_{j \in [m]} FV(c_j) \\
FV(a.l_i \Leftarrow \varsigma(x).b) &= FV(a) \cup FV(\varsigma(x).b)
\end{aligned}
\qquad
\begin{aligned}
FV(a.r_i \Leftarrow c) &= FV(a) \cup FV(c) \\
FV(\varsigma(x).b) &= FV(b) \backslash \{x\} \\
FV(\iota(x).e) &= FV(e) \backslash \{x\} \\
FV(a_1 \sim a_2) &= FV(a_1) \cup FV(a_2) \\
FV(e_1 \wedge e_2) &= FV(e_1) \cup FV(e_2) \\
FV(e_1 \vee e_2) &= FV(e_1) \cup FV(e_2) \\
FV(\neg e) &= FV(e)
\end{aligned}$$

Figure 2. Free variables in $\mathbf{O}_c$

$$\begin{array}{ll}
[a/y]t(a_1, \ldots, a_n) = t([a/y]a_1, \ldots, [a/y]a_n) & \\
[a/y]x = x & [a_1/y]a.l_i \Leftarrow \varsigma(x).b = ([a_1/y]a).l_i \Leftarrow [a_1/x]\varsigma(x).b \\
[a/y]y = a & [a_1/y]a.r_i \Leftarrow c = ([a_1/y]a).r_i \Leftarrow [a_1/x]c \\
[a/y]b.l_i = ([a/y]b).l_i & [a/y]a_1 \sim a_2 = [a/y]a_1 \sim [a/y]a_2 \\
[a/y]b.r_i = ([a/y]b).r_i & [a/y]e_1 \wedge e_2 = [a/y]e_1 \wedge [a/y]e_2 \\
[a/y]\varsigma(x).b = \varsigma(z)[a/y][z/x]b & [a/y]e_1 \vee e_2 = [a/y]e_1 \vee [a/y]e_2 \\
[a/y]\iota(x).b = \iota(z)[a/y][z/x]b & [a/y]\neg e = \neg([a/y]e) \\
[a/y][l_i = \varsigma(x).b_i, r_j = c_j] = [l_i = [a/y]\varsigma(x).b_i, r_j = [a/y]c_j] &
\end{array}$$

Figure 3. Substitution in $\mathbf{O}_c$

$$\begin{array}{ll}
\mathfrak{e}(f(a_1, \ldots, a_2)) = f^{\mathcal{A}}(\mathfrak{e}(a_1), \ldots, \mathfrak{e}(a_n)) & \\
\mathfrak{e}(x) = x & \mathfrak{e}(\iota(x).e) = \iota(x).\mathfrak{e}(e) \\
\mathfrak{e}(a.l_i) = \mathfrak{e}(a).l_i & \mathfrak{e}(a_1 \sim a_2) = \iota(a_1) \sim \iota(a_2) \\
\mathfrak{e}(a.r_i) = \mathfrak{e}(a).r_i & \mathfrak{e}(e_1 \wedge e_2) = \mathfrak{e}(e_1) \wedge \mathfrak{e}(e_2) \\
\mathfrak{e}([l_i = \varsigma(x).b_i, r_i = c_i]) = [l_i = \varsigma(x).\mathfrak{e}(b_i), r_i = \mathfrak{e}(c_i)] & \mathfrak{e}(e_1 \vee e_2) = \mathfrak{e}(e_1) \vee \mathfrak{e}(e_2) \\
\mathfrak{e}(a.l_i \Leftarrow \varsigma(x).b) = \mathfrak{e}(a).l_i \Leftarrow \varsigma(x).\mathfrak{e}(b) & \mathfrak{e}(\neg e) = \neg\mathfrak{e}(e) \\
\mathfrak{e}(a.r_i \Leftarrow c) = \mathfrak{e}(a).l_i \Leftarrow \mathfrak{e}(c) &
\end{array}$$

Figure 4. Evaluation

leave out the constraint c and the initial object o is the judgment, it induces a semantic map on constraints:

$$[\![c]\!] : Object \rightarrow \mathcal{P}(Object)$$

Therefore, having defined the satisfaction judgment of constraints on objects, the semantics for constraints are defined through the following relationship:

$$o(c) \models_{\mathfrak{S}_{\mathcal{A},\mathcal{T}}} [\![c]\!](o)$$

On the other hand, the judgment is an externalization of the semantics of constraints, this formulation hides away the computational contents in interpreting constraints.

We may sometimes want to construct a new construct c by combining several constraints $\{c_i\}_{i\in I}$. Aside from creating a new object with all of c_i inserted, we may also internalize it using first-order connectives we've defined earlier. This leads us to define the following auxiliary functions:

$$(\iota(x).e_1) \otimes (\iota(x).e_2) \triangleq \iota(x).e_1 \wedge e_2$$

$$(\iota(x).e_1) \oplus (\iota(x).e_2) \triangleq \iota(x).e_1 \vee e_2$$

There is another series of design decision to make involving when the solver is fired, when the object is updated and how the object's methods are updated in solving constraints. In our formulation of the calculus, we insist the following:

1. The only place the solver interacts with the satisfaction judgment is on the resolution of $\sim$-relations that present as premises of multiple satisfaction rules.
2. Constraints are solved eagerly, meaning (1) on constraint invocation the rule RED-C-UPDATE is fired (given in the next section) ensuring the resulting object satisfies the newly declared constraint and (2) on method updates, the rule RED-M-UPDATE is fired to get a new solution from the newly updated method.
3. Objects are updated monotonically, on constraint update, we do not attempt to backtrack and first solve with the rest of the constraints, instead we simply run the solver on the current object (solution).
4. Method updates are all handled functionally, a new object is returned on each invocation to the satisfaction judgment. Thus, there is no notion of references or object identities in this calculus. And we don't need to keep track the object identities environment.

2.4 Operational Semantics: Reduction

Now, we are ready to define the notion of reduction given the semantics of constraints. $o \rightarrowtail o'$ is a single step reduction between two top-level terms, $o \rightarrow o'$ is a single step reduction between two terms within a context and $\twoheadrightarrow$ is the reflexive, transitive closure of $\rightarrow$. The reduction judgment serves as the operational semantics of $\mathbf{O}_c$.

Now having defined both the satisfaction judgment and the reduction relation, we present several examples of derivations in the calculus to provide some intuitions.

Example 2.8. Suppose we are working over algebra $\mathbb{N}/_7$ - modular arithmetic over natural numbers (similar to $\mathbb{Z}/_m$ as we introduced in 2.5 but on $\mathbb{N}$ instead of $\mathbb{Z}$). Suppose we have an object $pt = [px = \varsigma(x).(x.px), py = \varsigma(x).(x.py), c = \iota(x).(x.px \sim x.py)]$, which represents a point whose x and y coordinates are congruent modulo 7. We want to show the following reduction is derivable in our calculus:

$$pt.px \Leftarrow \varsigma(x).2 \twoheadrightarrow pt'$$

where $pt' = [px = \varsigma(x).2, py = \varsigma(x).9, c = \iota(x).(x.px \sim x.py)]$.

Proof. Let $o = [px = \varsigma(x).2, py = \varsigma(x).(x.py), c = \iota(x).(x.px \sim x.py)]$. We first derive $pt.px \Leftarrow (x).2 \rightarrow o.py \Leftarrow \varsigma(x).9$, then derive $o.py \Leftarrow \varsigma(x).9 \rightarrow pt'$, then by transitivity, we have our result.

$$\dfrac{\dfrac{\begin{array}{c}2 \sim 9\\ py \text{ is a label in } o\\ py \notin FV(o.px)\\ [o/x](x.px) = o.px\\ [o/x](x.py) = o.py\end{array} \quad \dfrac{}{o.px \rightarrowtail 2}\,\textsc{red-m-invoke}}{o(\iota(x).(x.px \sim x.py)) \models_{\mathfrak{S}_{\mathbb{N}/_7}} \{o.py \Leftarrow \varsigma(x).9\}}\,\textsc{sat-update}_2}{pt.px \Leftarrow \varsigma(x).2 \rightarrow o.py \Leftarrow \varsigma(x).9}\,\textsc{red-m-update}$$

$$\dfrac{\dfrac{\begin{array}{c}[pt'/x](x.py) = pt'.py\\ [pt'/x](x.px) = pt'.px\end{array} \quad \begin{array}{c}\overline{pt'.px \rightarrowtail 9}\\ \overline{pt'.px \rightarrowtail 2}\end{array} \quad 2 \sim 9}{pt'(\iota(x).(x.px \sim x.py)) \models \{pt'\}}\,\textsc{sat-eq}}{o.py \Leftarrow \varsigma(x).9 \rightarrow pt'}\,\textsc{red-m-update}$$

□

Theorem 2.9 (monotonicity of objects). *For any object $o = [l_i = \varsigma(x).b_i, r_j = c_j]_{i\in[n], j\in[m]}$, given any constraint c, indices $p, q \in [m]$, if $o(c_p) \models_{\mathfrak{S}_{\mathcal{A},\mathcal{T}}} S$ and $o.r_q \Leftarrow c \rightarrowtail o'$, then $o' \in S$.*

3 Typing the Pure Calculus

In this section, we will type the untyped calculus $\mathbf{O}_c$ as defined in the previous section, and prove the reduction judgment preserves typing relation (subject reduction). We will first type the core calculus (with familiar record typing), and then extend the calculus with object-oriented features such as inheritance and a formulation of dynamic constraint dispatch.

3.1 Typing the Core Calculus

The $\mathbf{O}_{ct}$, the type system induced from the core calculus will be really familiar to those who have implemented objects using records in functional languages. This type system is similar to usual record semantics in the sense that (1) we are not dealing with object identities or references (2) state updates are all handled functionally (i.e. on every method update or constraint satisfaction, a new object with desired fields are returned) and (3) we have restricted ourselves to only solve $\sim$-constraints between primitive terms in a given

$$\frac{[o/x]a_1 \twoheadrightarrow v_1 \in \mathcal{A} \quad [o/x]a_2 \twoheadrightarrow v_2 \in \mathcal{A} \quad v_1 \sim v_2}{o(\iota(x).a_1 \sim a_2) \models_{\mathfrak{S}_{\mathcal{A},\mathcal{T}}} \{o\}} \ \textsc{sat-eq}$$

$$\frac{[o/x]a_1 \in \mathcal{A} \quad [o/x]a_2 \in \mathcal{A} \quad [o/x]a_1 \twoheadrightarrow v_1 \quad [o/x]a_2 \twoheadrightarrow v_2 \quad v_1 \nsim v_2}{o(\iota(x).a_1 \sim a_2) \models_{\mathfrak{S}_{\mathcal{A},\mathcal{T}}} \varnothing} \ \textsc{sat-empty}$$

$$\frac{\begin{array}{c} l \notin FV([o/x]a_2) \quad [o/x]a_2 \longrightarrow v_2 \\ o.l = [o/x]a_1 \quad l \text{ is a label in } o \quad v_2 \sim v_1 \end{array}}{o(\iota(x).a_1 \sim a_2) \models_{\mathfrak{S}_{\mathcal{A},\mathcal{T}}} \{o.l \Leftarrow \varsigma(z).v_1\}} \ \textsc{sat-update}_1$$

$$\frac{\begin{array}{ccc} l \text{ is a label in } o & l \notin FV(\hat{a}_1) & v_1 \sim v_2 \\ v_1 \text{ is not a label or a variable} & o.l = [o/x]a_2 & \hat{a}_1 \longrightarrow v_1 \end{array}}{o(\iota(x).a_1 \sim a_2) \models_{\mathfrak{S}_{\mathcal{A},\mathcal{T}}} \{o.l \Leftarrow \varsigma(z).v_2\}} \ \textsc{sat-update}_2$$

Figure 5. Satisfaction Judgment

$$\frac{o(\iota(x).e_1) \models_{\mathfrak{S}_{\mathcal{A},\mathcal{T}}} S_1 \quad S_1(\iota(x).e_2) \models_{\mathfrak{S}_{\mathcal{A},\mathcal{T}}} S}{o(\iota(x).e_1 \wedge e_2) \models_{\mathfrak{S}_{\mathcal{A},\mathcal{T}}} S} \ \textsc{sat-conj}$$

$$\frac{o(\iota(x).e_1) \models_{\mathfrak{S}_{\mathcal{A},\mathcal{T}}} S_1 \quad o(\iota(x).e_2) \models_{\mathfrak{S}_{\mathcal{A},\mathcal{T}}} S_2}{o(\iota(x).e_1 \vee e_2) \models_{\mathfrak{S}_{\mathcal{A},\mathcal{T}}} S_1 \cup S_2} \ \textsc{sat-disj}$$

$$\frac{o(\iota(x).e) \models_{\mathfrak{S}_{\mathcal{A},\mathcal{T}}} S \quad o \in S}{o(\iota(x).\neg e) \models_{\mathfrak{S}_{\mathcal{A},\mathcal{T}}} \varnothing} \ \textsc{sat-neg}_1$$

$$\frac{o(\iota(x).\neg e) \models_{\mathfrak{S}_{\mathcal{A},\mathcal{T}}} \varnothing}{o(\iota(x).\neg e) \models_{\mathfrak{S}_{\mathcal{A},\mathcal{T}}} \{o\}} \ \textsc{sat-neg}_2$$

Figure 6. Satisfaction Judgment on First-order Constraints

$$\frac{}{[l_i = \varsigma(x).b_i, r_j = c_j].l_k \rightarrowtail [[l_i = \varsigma(x).b_i, r_i = c_i]_{\substack{i \in [n] \\ j \in [m]}} /x]b_k} \ \textsc{red-m-invoke}$$

$$\frac{[l_i = \varsigma(x).b_i, r_j = c_j](c_k) \models_{\mathfrak{S}_{\mathcal{A},\mathcal{T}}} S \quad o' \in S}{[l_i = \varsigma(x).b_i, r_j = c_j].c_k \rightarrowtail o'} \ \textsc{red-c-invoke}$$

$$\frac{[l_k = \varsigma(x).b, l_{i \neq k} = \varsigma(x).b_i, r_j = c_j](\bigotimes_{i \in [m]} c_i) \models_{\mathfrak{S}_{\mathcal{A},\mathcal{T}}} S \quad o' \in S}{[l_i = \varsigma(x).b_i, r_j = c_j].l_k \Leftarrow \varsigma(x).b \rightarrowtail o'} \ \textsc{red-m-update}$$

$$\frac{[l_i = \varsigma(x).b_i, r_{j \neq k} = c_j, r_k = c](c) \models_{\mathfrak{S}_{\mathcal{A},\mathcal{T}}} S \quad o' \in S}{[l_i = \varsigma(x).b_i, r_j = c_j].c_k \Leftarrow c \rightarrowtail o'} \ \textsc{red-c-update}$$

$$\frac{a \rightarrowtail a'}{\mathcal{E}[a] \longrightarrow \mathcal{E}[a']} \ \textsc{red-congr}$$

Figure 7. Reduction relation in $\mathbf{O}_c$

algebra (so that there are no higher-order unification happening).

In addition to the syntax of types presented in 8, we also need to modify the syntax for terms from $\mathbf{O}_c$ to O_{ct} by annotating types in methods. Therefore, what was of the form

$A, B ::=$	types
$\mathcal{A}$	atomic types
$[l_i : B_i, r_j : C_j]$	object type
$Id(\mathcal{A})$	Identity Type in $\mathcal{A}$

$\Gamma ::=$	Context
$\langle\rangle$	empty context
$\Gamma, x : A$	context augmentation

Figure 8. Syntax of types $\mathbf{O}_{ct}$ under theory $\mathcal{T}$ on algebra $\mathcal{A}$

$\varsigma(x).b$ before should be annotated into term of the form $\varsigma(x : A).b$, where A is the type of the object x binds to.

Few comments on typing rules...

Lemma 3.1. *For arbitrary object $o = [l_i = \varsigma(x : A).b_i, r_i = c_i]$, if field l_k has method $\varsigma(x : A).(x.l_k)$, then the field l_k may be typed by any type B. This will be the analogous notion of Falsum (i.e. bottom "$\bot$") in our calculus.*

Proof. Without loss of generality, suppose our object only has one method field named l. Let B be an arbitrary type, let $A = [l : B]$,

$$\dfrac{\dfrac{\dfrac{}{\Gamma, x : [l : B] \vdash x : [l : B]}\ \textsc{tm-var}}{\Gamma, x : [l : B] \vdash x.l : B}\ \textsc{tm-m-invoke}}{\Gamma \vdash [l = \varsigma(x : A).(x.l)] : [l : B]}\ \textsc{tm-obj}$$

□

In order for our calculus to even make sense (in a sense that typing rules interact well with other judgments), we need to prove several important property. One of which is to show our constraints satisfaction judgment as we defined in Section 2 preserves the typing judgment. Maybe we can call this subject satisfaction (in reference to subject reduction).

Before we going into proof of the theorem, we define another syntactic apparatus, that is, the typing judgment for a set of objects. Let S be a set of objects, A is a type, we write $\Gamma \vdash S : A$ to mean for all $o \in S$, we have $\Gamma \vdash o : A$. This theorem is proved below:

Lemma 3.2 (substitution preserves typing)**.** *Suppose our calculus was parametrized over algebra $\mathcal{A}$ and theory $\mathcal{T}$. If $\Gamma, x : A \vdash t : T$ and $\langle\rangle \vdash v : A$, then $\Gamma \vdash [v/x]t : T$.*

Proof. Induction on the structure of term t: we will only prove the case for primitives here, rest of the cases all follows directly by the definition of the substitution function and inductive hypotheses.

1. $t = t_1(a_1, \ldots, a_n)$ for primitive term, since we have $\Gamma, x : A \vdash t_1(a_1, \ldots, a_n) : T$, it must be the case that $T = \mathcal{A}$ and have premises:

   $t \in \mathscr{F}$ and $\Gamma, x : A \vdash a_i : \mathcal{A}$ for all $i \in [n]$

   By IH, we have $\Gamma \vdash [v/x]a_i : \mathcal{A}$ for all $i \in [n]$. Since by the definition of substitution, we have

   $$[v/x](t_1(a_1 \ldots, a_n)) = t_1([v/x]a_1, \ldots, [v/x]a_n)$$

   therefore we have

   $$\dfrac{t_1 \in \mathscr{F} \quad \Gamma \vdash [v/x]a_i : \mathcal{A} \quad \forall i \in [n]}{\Gamma \vdash t_1([v/x]a_1, \ldots, [v/x]a_n) : T}\ \textsc{tm-atomic}$$

□

Theorem 3.3 ($\models$ preserves typing (subject satisfaction))**.** *Suppose our calculus was parametrized over algebra $\mathcal{A}$ and theory $\mathcal{T}$. If $o(\iota(x).e) \models_{\mathfrak{S}_{\mathcal{A},\mathcal{T}}} S$, $\Gamma \vdash e : Id(\mathcal{A})$ and $\Gamma \vdash o : A$, then $\Gamma \vdash S : A$.*

Proof. Induction on the derivation of $o(\iota(x).e) \models_{\mathfrak{S}_{\mathcal{A},\mathcal{T}}} S$:

1. SAT-EQ, then $S = \{o\}$, $\Gamma \vdash o : A$ by assumption.
2. SAT-EMPTY, then $S = \varnothing$, vacuously true.
3. SAT-UPDATE$_1$, then $e = a_1 \sim a_2$, $S = \{o.l \Leftarrow \varsigma(z).[o/x]a_2\}$ with $o.l = [o/x]a_1$. Induction on the judgment $\Gamma \vdash e : Id(\mathcal{A})$ together with the fact that $o.l = [o/x]a_i$ gives us only one case to consider: ID-VAR$_1$. Then we have $\Gamma \vdash a_2 \in \mathcal{A}$, thus $\Gamma \vdash [o/x]a_2 : \mathcal{A}$. Therefore, applying TM-M-UPDATE gives us the conclusion $\Gamma \vdash o.l \Leftarrow \varsigma(z).[o/x]a_2 : A$.

□

3.2 Subtyping

3.3 Inheritance

3.4 Constraint Dispatch

A Appendix

Text of appendix ...

References

Martín Abadi and Luca Cardelli. 1996. *Object-Based Languages.* Springer New York, New York, NY, 35–49. DOI:http://dx.doi.org/10.1007/978-1-4419-8598-9_5

Krzysztof R. Apt. 2000. *A Denotational Semantics for First-Order Logic.* Springer Berlin Heidelberg, Berlin, Heidelberg, 53–69. DOI:http://dx.doi.org/10.1007/3-540-44957-4_4

Garrett Birkhoff. 1935. On the Structure of Abstract Algebras. *Mathematical Proceedings of the Cambridge Philosophical Society* 31, 4 (Oct 1935), 433–454. DOI:http://dx.doi.org/10.1017/S0305004100013463

S. Burris and H.P. Sankappanavar. 1981. *A course in universal algebra.* Springer-Verlag. https://books.google.com/books?id=2grvAAAAMAAJ

$$\frac{}{\langle\rangle \vdash}\ \textsc{ctxt-empty} \qquad \frac{\Gamma \vdash A}{\Gamma, x : A \vdash}\ \textsc{ctxt-aug} \qquad \frac{}{\Gamma \vdash \mathcal{A}}\ \textsc{ty-atomic}$$

$$\frac{}{\Gamma \vdash Id(\mathcal{A})}\ \textsc{ty-id} \qquad \frac{\Gamma \vdash B_i \qquad \forall i \in [n]}{\Gamma \vdash [l_1 : B_1, \ldots, l_n : B_n]}\ \textsc{ty-obj}$$

$$\frac{\Gamma \vdash a : \mathcal{A} \quad \Gamma \vdash b : \mathcal{A}}{\Gamma \vdash a \sim b : Id(\mathcal{A})}\ \textsc{id-refl} \qquad \frac{\Gamma \vdash a_1 \sim a_2 : Id(\mathcal{A}) \quad \Gamma \vdash a_2 \sim a_3 : Id(\mathcal{A})}{\Gamma \vdash a_1 \sim a_3 : Id(\mathcal{A})}\ \textsc{id-trans} \qquad \frac{\Gamma \vdash a_1 \sim a_2 : Id(\mathcal{A})}{\Gamma \vdash a_2 \sim a_1 : Id(\mathcal{A})}\ \textsc{id-symm}$$

$$\frac{\Gamma \vdash e : Id(\mathcal{A})}{\Gamma \vdash \neg e : Id(\mathcal{A})}\ \textsc{id-neg} \qquad \frac{\Gamma \vdash e1 : Id(\mathcal{A}) \quad \Gamma \vdash e2 : Id(\mathcal{A})}{\Gamma \vdash e_1 \wedge e_2 : Id(\mathcal{A})}\ \textsc{id-conj} \qquad \frac{\Gamma \vdash e1 : Id(\mathcal{A}) \quad \Gamma \vdash e2 : Id(\mathcal{A})}{\Gamma \vdash e_1 \vee e_2 : Id(\mathcal{A})}\ \textsc{id-disj}$$

$$\frac{x : A \in \Gamma}{\Gamma \vdash x : A}\ \textsc{tm-var} \qquad \frac{t \in \mathscr{F} \quad \Gamma \vdash a_i : \mathcal{A} \quad \forall i \in [n]}{\Gamma \vdash t(a_1, \ldots, a_n) : \mathcal{A}}\ \textsc{tm-atomic}$$

$$\frac{\Gamma, x : A \vdash b_i : B_i\ \forall i \in [n] \quad \Gamma, x : A \vdash e_j : Id(\mathcal{A})\ \forall j \in [m] \quad A = [l_i : B_i]}{\Gamma \vdash [l_i = \varsigma(x : A).b_i, r_j = \iota(x : A).e_j] : [l_i : B_i]}\ \textsc{tm-obj}$$

$$\frac{\Gamma \vdash [l_i = \varsigma(x : A).b_i, r_j = c_j] : [l_i : B_i] \quad k \leq n}{\Gamma \vdash [l_i = \varsigma(x : A).b_i, r_j = c_j].l_k : B_k}\ \textsc{tm-m-invoke} \qquad \frac{\Gamma \vdash [l_i = \varsigma(x : A).b_i, r_j = c_j] : [l_i = B_i]}{\Gamma \vdash [l_i = \varsigma(x : A).b_i, r_j = c_j] : Id(\mathcal{A})}\ \textsc{tm-c-invoke}$$

$$\frac{\Gamma, x : A \vdash b : B_k \quad \Gamma \vdash [l_i = \varsigma(x : A).b_i, r_j = c_j] : [l_i : B_i]}{\Gamma \vdash [l_i = \varsigma(x : A).b_i, r_j = c_j].l_k \Leftarrow \varsigma(x : A).b : [l_i : B_i]}\ \textsc{tm-m-update}$$

$$\frac{\Gamma \vdash [l_i = \varsigma(x : A).b_i, r_j = c_j] : [l_i : B_i]}{\Gamma \vdash [l_i = \varsigma(x : A).b_i, r_j = c_j].r_k \Leftarrow c : [l_i : B_i]}\ \textsc{tm-c-update}$$

Figure 9. Typing Judgments in $\mathbf{O}_{ct}$