Constraints as a Denotational Semantics for Object Calculus

Midterm presentation

Thomas Sixuan Lou

February 3, 2017

Project Overview & Motivations

Motivation: We want to solve *local equations* and *local behaviors* over arbitrary algebras.

Project Overview & Motivations

Motivation: We want to solve *local equations* and *local behaviors* over arbitrary algebras.

- local equations (constraints) $\sim$ classes.

Project Overview & Motivations

Motivation: We want to solve *local equations* and *local behaviors* over arbitrary algebras.

- local equations (constraints) $\sim$ classes.
- local solutions $\sim$ objects.

Project Overview & Motivations

Motivation: We want to solve *local equations* and *local behaviors* over arbitrary algebras.

- local equations (constraints) $\sim$ classes.
- local solutions $\sim$ objects.
- constraint resolution $\sim$ semantics for object calculus.

Object Calculus in a Nutshell

- Values ("objects" / "instances") of a type are classified into "classes".

Object Calculus in a Nutshell

- Values ("objects" / "instances") of a type are classified into "classes".
- Functions ("methods") have different behaviors depending on the "class" of its arguments.

Object Calculus in a Nutshell

- Values ("objects" / "instances") of a type are classified into "classes".
- Functions ("methods") have different behaviors depending on the "class" of its arguments.
- Each "class" enforces different internal structures and different behaviors when passed to methods.

Object Calculus in a Nutshell

- Values ("objects" / "instances") of a type are classified into "classes".
- Functions ("methods") have different behaviors depending on the "class" of its arguments.
- Each "class" enforces different internal structures and different behaviors when passed to methods.
- We represent each class by a record of field and function types.

Object Calculus in a Nutshell

- Values ("objects" / "instances") of a type are classified into "classes".
- Functions ("methods") have different behaviors depending on the "class" of its arguments.
- Each "class" enforces different internal structures and different behaviors when passed to methods.
- We represent each class by a record of field and function types.
- We represent the behavior of each method with a "dispatch matrix"

$$\prod_{c \in \mathcal{C}} \prod_{d \in \mathcal{D}} (\tau^c \rightharpoonup \rho_d)$$

Object Calculus: examples

$$\tau^{cart} = Class\ \{x : \mathbb{R}, y : \mathbb{R}\}$$
$$\tau^{polar} = Class\ \{r : \mathbb{R}, \theta : \mathbb{R}\}$$

Object Calculus: examples

$$\tau^{cart} = Class\ \{x : \mathbb{R}, y : \mathbb{R}\}$$
$$\tau^{polar} = Class\ \{r : \mathbb{R}, \theta : \mathbb{R}\}$$
$$p = Obj\ \{x \mapsto 1, y \mapsto 1\} : \tau^{cart}$$
$$q = Obj\ \{r \mapsto 1, \theta \mapsto \frac{\pi}{4}\} : \tau^{polar}$$

Object Calculus: examples

$$
\begin{aligned}
\tau^{cart} &= Class\ \{x : \mathbb{R}, y : \mathbb{R}\} \\
\tau^{polar} &= Class\ \{r : \mathbb{R}, \theta : \mathbb{R}\} \\
p &= Obj\ \{x \mapsto 1, y \mapsto 1\} : \tau^{cart} \\
q &= Obj\ \{r \mapsto 1, \theta \mapsto \frac{\pi}{4}\} : \tau^{polar} \\
e &: \begin{bmatrix} \tau^{cart} \to \rho_{distSq} & \tau^{cart} \to \rho_{xCoord} \\ \tau^{polar} \to \rho_{distSq} & \tau^{polar} \to \rho_{xCoord} \end{bmatrix} \\
e &= \begin{bmatrix} \lambda p : \tau^{cart}.(p.x^2 + p.y^2) & \lambda p : \tau^{cart}.p.x \\ \lambda p : \tau^{polar}.(p.r^2) & \lambda p : \tau^{polar}.(p.r \times \cos(p.\theta)) \end{bmatrix}
\end{aligned}
$$

Adding Constraints in Classes

Now, we allow the declaration of *local equations* in objects.

Adding Constraints in Classes

Now, we allow the declaration of *local equations* in objects.

$$Class\ \{s : A, t : B, s = t\}$$

Adding Constraints in Classes

Now, we allow the declaration of *local equations* in objects.

$$Class\ \{s : A, t : B, s = t\}$$

An instance solves constraints declared by its class.

Adding Constraints in Classes

Now, we allow the declaration of *local equations* in objects.

$$\mathit{Class}\ \{s : A, t : B, s = t\}$$

An instance solves constraints declared by its class.

$$\mathit{Obj}\ \{s \mapsto a, t \mapsto b\} : \mathit{Class}\ \{s : A, t : B, s = t\}$$

Adding Constraints in Classes

Now, we allow the declaration of *local equations* in objects.

$$Class\ \{s : A, t : B, s = t\}$$

An instance solves constraints declared by its class.

$$Obj\ \{s \mapsto a, t \mapsto b\} : Class\ \{s : A, t : B, s = t\}$$

and also first-order constraints!

Adding Constraints in Classes

Now, we allow the declaration of *local equations* in objects.

$$Class\ \{s : A, t : B, s = t\}$$

An instance solves constraints declared by its class.

$$Obj\ \{s \mapsto a, t \mapsto b\} : Class\ \{s : A, t : B, s = t\}$$

and also first-order constraints!

$$Class\ \{s = t, \neg\phi, \phi_1 \wedge \phi_2, \phi_1 \vee \phi_2, \exists x.\phi\}$$

Example: classes as problems

An arithmetic problem class:

$$\mathit{Class}\ \{x : \mathbb{N}, y : \mathbb{N}, (3 + x) \cdot 2 = y \wedge y = 2 \cdot 3 + 2\}$$

Example: classes as problems

An arithmetic problem class:

$$\mathit{Class}\ \{x : \mathbb{N}, y : \mathbb{N}, (3 + x) \cdot 2 = y \wedge y = 2 \cdot 3 + 2\}$$

evaluate!

$$\Downarrow$$

$$[\![\mathit{Class}\ \{x : \mathbb{N}, y : \mathbb{N}, (3 + x) \cdot 2 = y \wedge y = 2 \cdot 3 + 2\}]\!]_{AE}$$

Example: classes as problems

An arithmetic problem class:

$$Class\ \{x : \mathbb{N}, y : \mathbb{N}, (3 + x) \cdot 2 = y \wedge y = 2 \cdot 3 + 2\}$$

evaluate!

$$\Downarrow$$

$$[\![Class\ \{x : \mathbb{N}, y : \mathbb{N}, (3 + x) \cdot 2 = y \wedge y = 2 \cdot 3 + 2\}]\!]_{AE}$$

$$\Downarrow$$

$$Class\ \{x : \mathbb{N}, y : \mathbb{N}, [\![(3 + x) \cdot 2 = y]\!]_{AE} \wedge [\![y = 2 \cdot 3 + 2]\!]_{AE}\}$$

Example: classes as problems

An arithmetic problem class:

$$\mathit{Class}\ \{x : \mathbb{N}, y : \mathbb{N}, (3 + x) \cdot 2 = y \wedge y = 2 \cdot 3 + 2\}$$

evaluate!

$$\Downarrow$$

$$[\![\mathit{Class}\ \{x : \mathbb{N}, y : \mathbb{N}, (3 + x) \cdot 2 = y \wedge y = 2 \cdot 3 + 2\}]\!]_{AE}$$

$$\Downarrow$$

$$\mathit{Class}\ \{x : \mathbb{N}, y : \mathbb{N}, [\![(3 + x) \cdot 2 = y]\!]_{AE} \wedge [\![y = 2 \cdot 3 + 2]\!]_{AE}\}$$

$$\Downarrow$$

$$\mathit{Class}\ \{x : \mathbb{N}, y : \mathbb{N}, (3 + x) \cdot 2 = y \wedge y = 8\}$$

Example: objects as solutions

Possible solutions to a problem given by the semantics of equations

$$Obj\ \{x \mapsto 1, y \mapsto 8\} : C$$

We denote the "solving" process into the CLP land:
let $\theta = \{x \mapsto 1, y \mapsto 8\}$

$$[\![Class\ \{x : \mathbb{N}, y : \mathbb{N}, (3 + x) \cdot 2 = y \wedge y = 8\}]\!](\theta)$$

Example: objects as solutions

Possible solutions to a problem given by the semantics of equations

$$\mathit{Obj}\ \{x \mapsto 1, y \mapsto 8\} : C$$

We denote the “solving” process into the CLP land:
let $\theta = \{x \mapsto 1, y \mapsto 8\}$

$$[\![\mathit{Class}\ \{x : \mathbb{N}, y : \mathbb{N}, (3 + x) \cdot 2 = y \wedge y = 8\}]\!](\theta)$$
$$\Downarrow$$
$$8 = [\![(3 + x) \cdot 2]\!]\theta \equiv [\![y]\!]\theta$$
$$8 = [\![y]\!]\theta = [\![8]\!]\theta$$

Example: objects as solutions

Possible solutions to a problem given by the semantics of equations

$$\mathit{Obj}\ \{x \mapsto 1, y \mapsto 8\} : C$$

We denote the "solving" process into the CLP land:
let $\theta = \{x \mapsto 1, y \mapsto 8\}$

$$\llbracket \mathit{Class}\ \{x : \mathbb{N}, y : \mathbb{N}, (3 + x) \cdot 2 = y \land y = 8\} \rrbracket(\theta)$$
$$\Downarrow$$
$$8 = \llbracket (3 + x) \cdot 2 \rrbracket \theta \equiv \llbracket y \rrbracket \theta$$
$$8 = \llbracket y \rrbracket \theta = \llbracket 8 \rrbracket \theta$$
$$\Downarrow$$
$$\{\theta\}$$

OO : inheritance

As in ordinary object calculus, there are inheritance to create class hierarchies.

$$\tau^{cart} = Class\ \{x : \mathbb{R}, y : \mathbb{R}\}$$
$$\tilde{\tau}^{cart} <: \tau^{cart} = Class\ \{color : colorT, x : \mathbb{R}, y : \mathbb{R}\}$$

same for constraints

$$\tilde{\tau}^{cart} <: \tau^{cart} = Class\ \{x + y = 1, x : \mathbb{R}, y : \mathbb{R}\}$$
$$\hat{\tau}^{cart} <: t\tilde{a}u^{cart} = Class\ \{x = 1 \land x + y = 1, x : \mathbb{R}, y : \mathbb{R}\}$$

OO : methods

Recall methods are (internally) represented as a dispatch matrix, when adding method to a class, we are adding a column vector into the class.

$$\tilde{\tau}^{cart} <: \tau^{cart} = Class\ \{dist : \prod_{c \in C} \tau^{c} \to \mathbb{R}\}$$

$$\hat{\tau}^{cart} <: \tilde{\tau}^{cart} = Class\ \{dist(self) = 1\}$$

OO : methods

Recall methods are (internally) represented as a dispatch matrix, when adding method to a class, we are adding a column vector into the class.

$$\tilde{\tau}^{cart} <: \tau^{cart} = Class\ \{dist : \prod_{c \in C} \tau^{c} \to \mathbb{R}\}$$
$$\hat{\tau}^{cart} <: \tilde{\tau}^{cart} = Class\ \{dist(self) = 1\}$$

$$Obj\{x \mapsto 1, y \mapsto 0\} : \hat{\tau}^{cart}$$

Dispatch methods

Let $\theta = \{x \mapsto 1, y \mapsto 0\}$,

$$[\![Class\{x : \mathbb{R}, y : \mathbb{R}, dist : \prod_{c \in \mathcal{C}} \tau^c \to \mathbb{R}\}, dist(self) = 1]\!]\theta$$

$$\Downarrow$$

$$[\![dist(self)]\!]\theta \overset{\text{dispatch}}{=} [\![(\lambda p : \tau^{cart}.(p.x^2 + p.y^2))self]\!]\theta$$
$$= [\![x^2 + y^2]\!]\theta = 1 = [\![1]\!]\theta$$

$$\Downarrow$$

$$\{\theta\}$$