Gentle Introduction to DOT calculus

ScalaMatsuri 2019

Kota Mizushima (DWANGO Co.,Ltd.)

Who am I?

- Twitter ID: @kmizu
- GitHub: kmizu
- Love: Scala/Rust/Nemerle/...
- Parsing algorithm enthuasiast

Scalaと構文解析アルゴリズムが大好きなプログラマです

Scala

- Designed by Martin Odersky in 2003
  - A FP researcher
- SCAlable LAnguage
- Latest: Scala 2.13.0
- Many industrial usages

- Martin Oderskyによって2003年に開発されました
- 多くの実用例があります

Dotty

- https://dotty.epfl.ch/
- Dotty will be Scala 3.0
- Many improvements
- Dotty is based on DOT calculus

- DottyはScala 3.0になる予定です
- DOT計算をベースにした言語です

DOT Calculus

- Core calculus of Scala 3.0
- DOT is abbreviation of *D*ependent *O*bject *T*ypes
- Core calculus:
  - Essense of a programming language
    - Abstract Syntax, Typing Rules, Semantics
  - Used to prove the behavior of the language

- DOT計算はScala 3.0の核言語です
- 核言語とはプログラミング言語の本質です

Why Core Calculus?

- Too difficult to model real programming languages
- Need good **subset** of programming languages

- 現実のプログラミング言語をモデル化することは難しすぎます
- 良いサブセットが必要です

History of Core Calculus in Scala

- vObj calculus (2003)
- Featherweight Scala (2006)
- DOT calculus (2012-)

- vObj calculus（2003年）
- Featherweight Scala（2006年）

Note

- DOT has several versions
  - since several DOT papers exists
- This presentation is based on
  - Nada Amin's dot repository
  - the paper of DOT appeared in OOPSLA'16
- Other papers may explain DOT in different manner

- Nada Aminさんのリポジトリと OOPSLA'16の論文をベースにします
- 他の論文では違う形で説明されているかもしれません

What is Needed for Core Calculus ?

- Abstract Syntax
  - drop concrete syntax
- Typing rule
  - if the language have type system
- (Operational) semantics
  - like a naive implementation of an interpreter

- 抽象構文、型付け規則、(操作的)意味論が必要です
- 操作的意味論は、ナイーヴなインタプリタ実装のようなものです

Abstract Syntax

- Concrete syntax have extra information
  - spaces, tabs, commas, parenthesis, etc.
- An example of pseudo EBNF:
  - very simple calculation

```
E ::= P ("+' S P | "-" S P)
P ::= "(" S E ")" S | I S
I ::= "0" | "1"
S ::= ("\t" | " " | "\r" | "\n)*
```

- 具象構文は余計な情報を含んでいます

Abstract Syntax

- Drop such extra information from concrete syntax
- It is important to define *value*
- An example of abstract syntax in EBNF-like notation
  - very simple calculation
  - V is value

```
E ::= E + E | V
V ::= 0 | 1  // value
```

- 抽象構文は具象構文から余計な情報を除いたものです
- *値*を定義するのが重要です

Typing Rules

- Rules for typing terms
- Γ is called as **typing environments**
- ⊢ is called as **typing judgements**
- An example:
  - very simple calculation

```
-----------   -----------
Γ⊢ 0 : int    Γ⊢ 1 : int    ...

Γ⊢ t1 : int  Γ⊢ t2 : int
--------------------------
Γ⊢ t1 + t2 : int
```

- Γは**型環境**と呼ばれます
- ⊢は**型判断**と呼ばれます

9

Operational Semantics

- Rules for evaluating terms
- It can be seen as a naive interpreter

```
t1 → t1'  t2 → t2'
--------------------
t1 + t2 → t1' + t2'
```

- ナイーヴなインタプリタのようなものと見ることができます

DOT: *D*ependent *O*bject *T*ypes

Characteristics of DOT's Type System

- Union and intersection types
  - `e: Int | String`, `e: Number & Serializable`
- Type members
  - `e: Any { type M }`
- Path-dependent types
  - `e: v.x` (v is constant)
- Combination of subtyping and these types

- 交差型（union type）、合併型（intersection type）など
- それらとサブタイピングが組み合わさっています

Syntax of DOT

- citation in https://github.com/namin/dot

DOT:

Syntax

$t ::=$		**terms**:	
	x	variable	
	$\{z \Rightarrow \bar{d}\}$	object	
	$t.m(t)$	method invocation	
$d ::=$		**initialization**:	
	$L = T$	type member	
	$m(x : T) = t$	method member	
$v ::=$		**values**:	
	$\{z \Rightarrow \bar{d}\}$	object	
$p ::=$		**paths**:	
	x	variable	
	v	value	

$S, T, U ::=$		**types**:
	$\top$	top
	$\bot$	bottom
	$T \wedge T$	intersection
	$T \vee T$	union
	$L : S..U$	type member
	$m(x : S) : U$	method member
	$p.L$	selection
	$\{z \Rightarrow T\}$	recursive self
$\Gamma ::=$		**contexts**:
	$\emptyset \mid \Gamma, x : T$	variable bindings

Figure 1: DOT: Syntax

Subtyping in DOT (1)

- citation in https://github.com/namin/dot

DOT:

Subtyping $\boxed{\Gamma \vdash S <: U}$

Lattice structure

$$\Gamma \vdash \bot <: T \quad (\textsc{Bot}) \qquad \Gamma \vdash T <: \top \quad (\textsc{Top})$$

$$\frac{\Gamma \vdash T_1 <: T}{\Gamma \vdash T_1 \wedge T_2 <: T} \quad (\textsc{And11}) \qquad \frac{\Gamma \vdash T <: T_1}{\Gamma \vdash T <: T_1 \vee T_2} \quad (\textsc{Or21})$$

$$\frac{\Gamma \vdash T_2 <: T}{\Gamma \vdash T_1 \wedge T_2 <: T} \quad (\textsc{And12}) \qquad \frac{\Gamma \vdash T <: T_2}{\Gamma \vdash T <: T_1 \vee T_2} \quad (\textsc{Or22})$$

$$\frac{\Gamma \vdash T <: T_1 \ , \ T <: T_2}{\Gamma \vdash T <: T_1 \wedge T_2} \quad (\textsc{And2}) \qquad \frac{\Gamma \vdash T_1 <: T \ , \ T_2 <: T}{\Gamma \vdash T_1 \vee T_2 <: T} \quad (\textsc{Or1})$$

Type and method members

$$\frac{\Gamma \vdash S_2 <: S_1 \ , \ U_1 <: U_2}{\Gamma \vdash L : S_1..U_1 <: L : S_2..U_2} \quad (\textsc{Typ}) \qquad \frac{\Gamma \vdash S_2 <: S_1 \quad \Gamma, x : S_2 \vdash U_1 <: U_2}{\Gamma \vdash m(x : S_1) : U_1 <: m(x : S_2) : U_2} \quad (\textsc{Fun})$$

Subtyping in DOT (2)

- citation in https://github.com/namin/dot

Type selections

$$\frac{\Gamma_{[x]} \vdash x :! (L : T..\top)}{\Gamma \vdash T <: x.L} \quad (\textsc{Sel2}) \qquad \frac{[z \mapsto \bar{d}]\bar{d} \ni L = T}{\Gamma \vdash T <: \{z \Rightarrow \bar{d}\}.L} \quad (\textsc{SSel2})$$

$$\frac{\Gamma_{[x]} \vdash x :! (L : \bot..T)}{\Gamma \vdash x.L <: T} \quad (\textsc{Sel1}) \qquad \frac{[z \mapsto \bar{d}]\bar{d} \ni L = T}{\Gamma \vdash \{z \Rightarrow \bar{d}\}.L <: T} \quad (\textsc{SSel1})$$

Recursive self types

$$\frac{\Gamma, z : T_1 \vdash T_1 <: T_2}{\Gamma \vdash \{z \Rightarrow T_1\} <: \{z \Rightarrow T_2\}} \quad (\textsc{Bind}) \qquad \frac{\Gamma, z : T_1 \vdash T_1 <: T_2 \quad z \notin fv(T_2)}{\Gamma \vdash \{z \Rightarrow T_1\} <: T_2} \quad (\textsc{Bind1})$$

Properties

$$\Gamma \vdash T <: T \quad (\textsc{Refl}) \qquad \frac{\Gamma \vdash T_1 <: T_2 \ , \ T_2 <: T_3}{\Gamma \vdash T_1 <: T_3} \quad (\textsc{Trans})$$

Typing Rules of DOT

- citation in https://github.com/namin/dot

Type assignment $\boxed{\Gamma \vdash t :_{(!)} T}$

$$\frac{\Gamma(x) = T}{\Gamma \vdash x :_{(!)} T} \text{ (Var)} \qquad \frac{\Gamma \vdash t :_{(!)} T_1 \ , \ T_1 <: T_2}{\Gamma \vdash t :_{(!)} T_2} \text{ (Sub)}$$

$$\frac{\Gamma \vdash p : [z \mapsto p]T}{\Gamma \vdash p : \{z \Rightarrow T\}} \text{ (Pack)} \qquad \frac{\Gamma \vdash p :_{(!)} \{z \Rightarrow T\}}{\Gamma \vdash p :_{(!)} [z \mapsto p]T} \text{ (Unpack)}$$

$$\frac{\Gamma \vdash t : (m(x : T_1) : T_2) \ , \ t_2 : T_1 \quad x \notin fv(T_2)}{\Gamma \vdash t.m(t_2) : T_2} \text{ (TApp)} \qquad \frac{\Gamma \vdash t : (m(x : T_1) : T_2) \ , \ p : T_1}{\Gamma \vdash t.m(p) : [x \mapsto p]T_2} \text{ (TAppDep)}$$

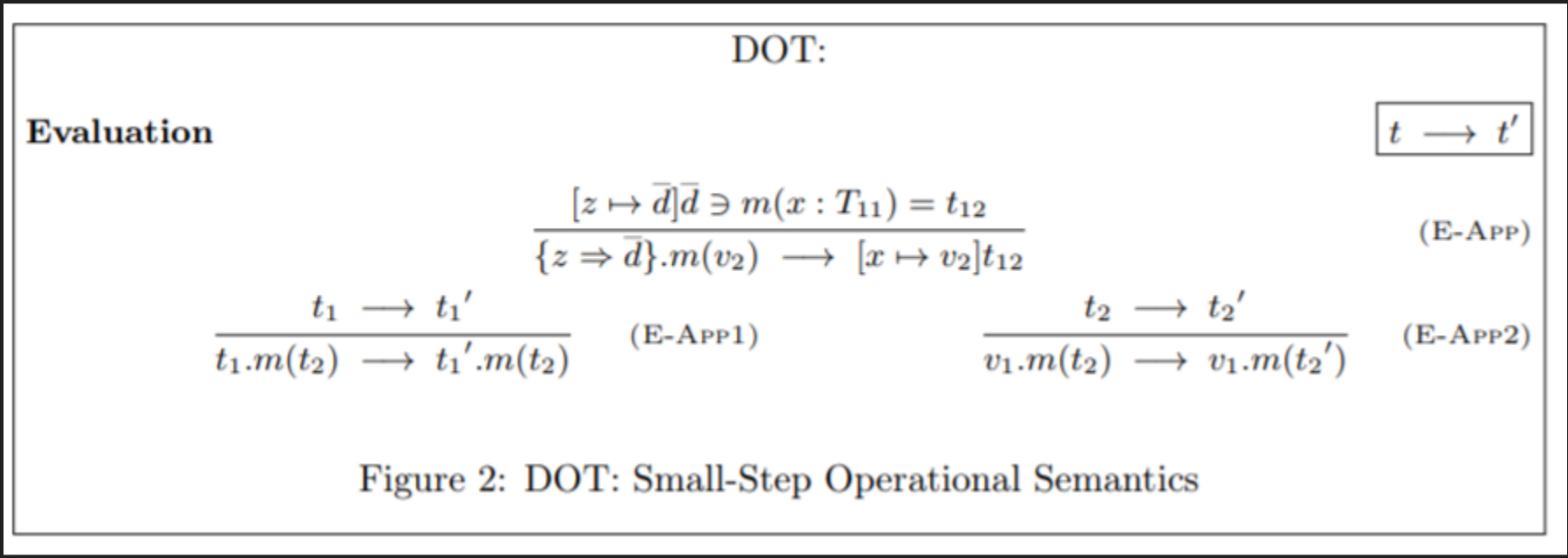
$$\frac{\text{(labels disjoint)} \quad \Gamma, x : T_1 \wedge \ldots \wedge T_n \vdash d_i : T_i \qquad \forall i, 1 \leq i \leq n}{\Gamma \vdash \{x \Rightarrow d_1 \ldots d_n\} : [x \mapsto \{x \Rightarrow d_1 \ldots d_n\}](T_1 \wedge \ldots \wedge T_n)} \text{ (TObj)}$$

Member initialization $\boxed{\Gamma \vdash d : T}$

$$\frac{\Gamma \vdash T <: T}{\Gamma \vdash (L = T) : (L : T..T)} \text{ (DTyp)} \qquad \frac{\Gamma, x : T_1 \vdash t : T_2}{\Gamma \vdash (m(x) = t) : (m(x : T_1) : T_2)} \text{ (DFun)}$$

Oerational Semantics of DOT

- citation in https://github.com/namin/dot



DOT:

Evaluation $\boxed{t \longrightarrow t'}$

$$\frac{[z \mapsto \bar{d}]\bar{d} \ni m(x : T_{11}) = t_{12}}{\{z \Rightarrow \bar{d}\}.m(v_2) \longrightarrow [x \mapsto v_2]t_{12}} \quad \text{(E-App)}$$

$$\frac{t_1 \longrightarrow t_1'}{t_1.m(t_2) \longrightarrow t_1'.m(t_2)} \quad \text{(E-App1)} \qquad \frac{t_2 \longrightarrow t_2'}{v_1.m(t_2) \longrightarrow v_1.m(t_2')} \quad \text{(E-App2)}$$

Figure 2: DOT: Small-Step Operational Semantics

Properties of DOT

- DOT type system is sound
  - soundness is very important property
  - A mechanized proof in Coq exists:
    - https://github.com/namin/dot
- Typing in DOT is undecidable
- Typing in many practical programming language is undedidable
  - Scala, Java, C++, TypeScript, Haskell(?), etc.

- DOTの型システムは健全です（Coqで証明されています）
- DOTの型付けは決定不能です（DOTはFsubにエンコード可能）

Conclusion

- Core calculus is essence of programming languages
- DOT is core calculus of Scala 3
- Type system of DOT is sound
- Typing in DOT is undecidable (!)