DOT

(Dependent Object Types)

Nada Amin

with

Samuel Grütter Martin Odersky Sandro Stucki Tiark Rompf

LAMP

May 17, 2016

Why DOT?

- DOT as a type-theoretic foundation:
  - few yet powerful concepts,
    with uniform means of abstraction and combination
    e.g. quantification only over term, yet supports polymorphism
  - "user-extensible" subtyping
  - mixture of nominal and structural
  - nominality is "scoped"
    e.g. no global class table – nice for static analysis?
  - no imposed notion of code sharing
    such as prototype vs class inheritance, mixins, ...
- Impact on Scala/Dotty:
  - characterizing soundness issues,
    e.g. type selection on `Null` or $\bot$ paths
  - suggesting simplifications,
    e.g. a core type system based on DOT
  - lifting ad-hoc restrictions,
    e.g. recursive structural types are more powerful in DOT than in Scala.

DOT: The Essence of Scala

> *What do you get if you boil Scala on a slow flame and wait until all incidental features evaporate and only the most concentrated essence remains? After doing this for 8 years we believe we have the answer: it's DOT, the calculus of dependent object types, that underlies Scala.*

– Martin Odersky

`http://www.scala-lang.org/blog/2016/02/03/essence-of-scala.html`

DOT (Syntax)

$t ::=$	**terms**:
x	variable
$\{x \Rightarrow \overline{d}\}$	object
$t.l$	field. sel.
$t.m(t)$	meth. app.
$d ::=$	**init.**:
$l = p$	field mem.
$m(x : T) = t$	meth. mem.
$L = T$	type mem.
$v ::=$	**values**:
$\{x \Rightarrow \overline{d}\}$	object
$p ::=$	**paths**:
x	variable
v	value
$p.l$	field. sel.

$S, T, U ::=$	**types**:
$\top$	top
$\bot$	bottom
$T \wedge T$	intersection
$T \vee T$	union
$l : U$	field mem.
$m(x : S) : U$	meth. mem.
$L : S..U$	type mem.
$p.L$	type sel.
$\{x \Rightarrow T\}$	rec. self

Deriving DOT

From $F_{<:}$ to DOT

1. lower bound
2. type member and selection
3. subtyping lattice
4. records
5. recursion over self
6. normalization in paths

System $F_{<:}$

$$t ::= x \mid \lambda x : T.t \mid t\ t \mid \lambda X <: T.t \mid t\ [T]$$

$$T ::= T \to T \mid \top \mid X \mid \forall X <: T.T$$

- combines System F (polymorphic lambda-calculus) and subtyping
- generalizes universal quantification to upper-bounded quantification
  - example of universal quantification

    ```
    id = λX<:⊤.λx:X.x
    ```
    - `id : ∀X<:⊤.X→X`
  - example of upper-bounded quantification

    ```
    p = λX<:{a:Nat}.λx:X.{orig_x=x, s=succ(x.a)};
    ```
    - `p : ∀X<:{a:Nat}.X → {orig_x:X, s:Nat}`

    ```
    n = (p [{a:Nat,b:Nat}] (a=0,b=0)).orig_x.b
    ```
    - `n: Nat`

Soundness

Theorem (Type-Safety)

If t is a closed well-typed term, $\emptyset \vdash t : T$, then either t is a value or else there is some t' with $t \longrightarrow t'$ and $\emptyset \vdash t' : T$.

Theorem (Preservation)

If $\Gamma \vdash t : T$ and $t \longrightarrow t'$, then $\Gamma \vdash t' : T$.

Theorem (Progress)

If t is a closed well-typed term, then either t is a value or else there is some t' with $t \longrightarrow t'$.

Properties

Narrowing

Substitution

Inversion of Subtyping

Inversion of Value Typing

1. If $\Gamma \vdash \lambda x : S_1.t_2 : T$ and $\Gamma \vdash T <: U_1 \to U_2$, then $\Gamma \vdash U_1 <: S_1$ and there is some S_2 such that $\Gamma, x : S_1 \vdash t_2 : S_2$ and $\Gamma \vdash S_2 <: U_2$.
2. If $\Gamma \vdash \lambda X <: S_1.t_2 : T$ and $\Gamma \vdash T <: \forall X <: U_1.U_2$, then $\Gamma \vdash U_1 <: S_1$ and there is some S_2 such that $\Gamma, x <: S_1 \vdash t_2 : S_2$ and $\Gamma, X <: U_1 \vdash S_2 <: U_2$.

Canonical Forms

1. If v is a closed value of type $T_1 \to T_2$, then v has the form $\lambda x : S_1.t_2$.
2. If v is a closed value of type $\forall X <: T_1.T_2$, then v has the form $\lambda X <: S_1.t_2$.

1. Lower Bound

$$\frac{X <: T \in \Gamma}{\Gamma \vdash X <: T} \quad \text{(S-TVar)}$$

vs

$$\frac{X : S..U \in \Gamma}{\Gamma \vdash S <: X <: U} \quad \text{(S-TVar)}$$

System $F_{<:>}$

$$\lambda X <: T.t \text{ becomes } \lambda X : S..U.t$$

$$\forall X <: T.T \text{ becomes } \forall X : S..U.T$$

$\bot$ to recover upper-bounded quantification

- example of lower-bounded quantification:

```
  p = λX:{a:Nat,b:Nat}..⊤.λf:X→⊤.{orig=f, r=(f {a=0,b=0})};
▸ p : ∀X:{a:Nat,b:Nat}..⊤.(X→⊤)→{orig:X→⊤, r:⊤}
  pa = p [{a:Nat}] (λx:{a:Nat}. x.a);
▸ pa : {orig:{a:Nat}→⊤, r:⊤}
```

- example of "translucent" quantification:

```
  p = λX:{a:Nat,b:Nat}..{a:Nat}.λf:X→X.(f {a=0,b=0}).a
▸ p : ∀X:{a:Nat,b:Nat}..{a:Nat}.(X → X) → Nat
  n = p [{a:Nat}] (λx:{a:Nat}. {a=succ(x.a)})
▸ n : Nat
```

Dealing with "bad" bounds

- Restrict Preservation to $\Gamma = \emptyset$.
  If $\Gamma \vdash t : T$ and $t \longrightarrow t'$, then $\Gamma \vdash t' : T$.
- Define invertible value typing, aka "possible types": $v :: T$.
  1. $v :: \top$.
  2. If $x : S_1 \vdash t_1 : T_1$ and $\emptyset \vdash S_2 <: S_1, T_1 <: T_2$ then $\lambda x : S_1.t_1 :: S_2 \rightarrow T_2$.
  3. If $X : S_1..U_1 \vdash t_1 : T_1$ and $\emptyset \vdash S_1 <: S_2, U_2 <: U_1$ and $X : S_2..U_2 \vdash T_1 <: T_2$ then $\lambda X : S_1..U_1.t_1 :: \forall X : S_2..U_2.T_2$.
- Prove subtyping closure aka widening of possible types.
  If $v :: T$ and $\emptyset \vdash T <: U$ then $v :: U$.
- Prove value typing implies possible types.
  If $\emptyset \vdash v : T$ then $v :: T$.
- Prove inversion of value typing and canonical forms via (direct) inversion of possible types.

2. System D: $D_{<:}$, $D_{<:>}$

$$t ::= x \mid \lambda x : T.t \mid t\ t \mid \{L = T\}$$

$$T ::= \top \mid \bot \mid \forall x : S.T \mid \{L : S..U\} \mid p.L$$

- System D unifies term and type abstraction.
- A term can hold a type: a term $\{L = T\}$ introduces a type $\{L : S..U\}$.
- Path-dependent type: $p.L$ is a type that depends on some term p.
- What terms are paths p?
  Here, only normal forms (variables or values).

$$p ::= x \mid v$$

$$v ::= \lambda x : T.t \mid \{L = T\}$$

- λ-values are paths???

Subtyping of Type Selections aka Path-Dependent Types

$$\frac{\Gamma \vdash p : \{L : S..U\}}{\Gamma \vdash S <: p.L <: U} \quad \text{(S-TSel)}$$

$$\Gamma \vdash T <: \{L = T\}.L <: T \quad \text{(S-TSel-Tight)}$$

- Define subtyping generally (non-tight), so that substitution is easier: no need for narrowing while substituting a value.
- Define tight subtyping for "possible types". Prove widening of "possible types". For lambda values, delegate to regular subtyping for non-empty context and also define "shallow" variant that does not care about lambda values beyond shape.
- Prove tight $\equiv$ general subtyping in empty context. Use shallow variant of possible types for inverting path typing in subtyping type selections.
- Now adjusted back to $F_{<:>}$ proof strategy.

3. Full Subtyping Lattice

$$\Gamma \vdash \bot <: T \qquad \text{(Bot)}$$

$$\Gamma \vdash T <: \top \qquad \text{(Top)}$$

$$\frac{\Gamma \vdash T_1 <: T}{\Gamma \vdash T_1 \wedge T_2 <: T} \quad \text{(And11)}$$

$$\frac{\Gamma \vdash T <: T_1}{\Gamma \vdash T <: T_1 \vee T_2} \quad \text{(Or21)}$$

$$\frac{\Gamma \vdash T_2 <: T}{\Gamma \vdash T_1 \wedge T_2 <: T} \quad \text{(And12)}$$

$$\frac{\Gamma \vdash T <: T_2}{\Gamma \vdash T <: T_1 \vee T_2} \quad \text{(Or22)}$$

$$\frac{\Gamma \vdash T <: T_1 \ , \ T <: T_2}{\Gamma \vdash T <: T_1 \wedge T_2} \quad \text{(And2)}$$

$$\frac{\Gamma \vdash T_1 <: T \ , \ T_2 <: T}{\Gamma \vdash T_1 \vee T_2 <: T} \quad \text{(Or1)}$$

4. Records, typed via Intersection Types

$t ::=$	**terms**:	$m(x : S) : U$	meth. mem.
$\quad x$	variable	$L : S..U$	type mem.
$\quad \{\overline{d}\}$	record		
$\quad t.m(t)$	meth. app.		
$d ::=$	**init.**:		
$\quad m(x : T) = t$	meth. mem.		
$\quad L = T$	type mem.		

- A record $\{d_1, \ldots, d_n\}$ has type $T_1 \wedge \ldots \wedge T_n$.

5. Recursion: From Records to Objects

- An object is a record which closes over a self variable z: $\{z \Rightarrow \overline{d}\}$.
- An object introduces a recursive type: $\{z \Rightarrow T\}$.

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

- Store to keep track of object identities?
- Recursive types bring lots of power: F-bounded abstraction and beyond, non-termination, nominality through type abstraction, etc.

Typing and Subtyping of Recursive Types

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

$$\frac{\Gamma \vdash p : [z \mapsto p]T}{\Gamma \vdash p : \{z \Rightarrow T\}} \quad \text{(Pack)}$$

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

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

$$\frac{\Gamma, z : T_1 \vdash T_1 <: T_2}{\Gamma \vdash \{z \Rightarrow T_1\} <: \{z \Rightarrow T_2\}} \quad \text{(Bind)}$$

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

Restrictions in Type Selections of Abstract Variables

$$\frac{\Gamma_{[x]} \vdash x :_! (L : T..\top)}{\Gamma \vdash T <: x.L} \quad \text{(Sel2)}$$

$$\frac{\Gamma_{[x]} \vdash x :_! (L : \bot..T)}{\Gamma \vdash x.L <: T} \quad \text{(Sel1)}$$

- To prove tight $\equiv$ non-tight subtyping in empty context, we need substitution because of type selection on recursive types. But if we use substitution, we cannot use the IH.
- With the restriction on Γ, we can use tight subtyping – on values only – from the outset.
- Restrict substitution so that substituted variable is first in context, i.e. $\Gamma' = \emptyset$.
  If $\Gamma', x : U, \Gamma \vdash t : T$ and $\Gamma' \vdash v : U$, then
  $\Gamma', [x \mapsto v]\Gamma \vdash [x \mapsto v]t : [x \mapsto v]T$

Possible Types (Base Cases)

$$v :: \top \qquad \text{(V-Top)}$$

$$\frac{(L = T) \in \overline{[x \mapsto \{x \Rightarrow \overline{d}\}]d} \quad \emptyset \vdash S <: T \ , \ T <: U}{\{x \Rightarrow \overline{d}\} :: (L : S..U)} \qquad \text{(V-Typ)}$$

$$\frac{\text{(labels disjoint)} \quad \forall i, 1 \leq i \leq n \quad \emptyset, (x : T_1 \wedge \ldots \wedge T_n) \vdash d_i : T_i \quad \exists j,\ [x \mapsto \{x \Rightarrow \overline{d}\}]d_j = (m(z : S) = t) \quad [x \mapsto \{x \Rightarrow \overline{d}\}]T_j = (m(z : S) : U) \quad \emptyset \vdash S' <: S \quad \emptyset, (z : S') \vdash U <: U'}{\{x \Rightarrow \overline{d}\} :: (m(x : S') : U')} \qquad \text{(V-Fun)}$$

Possible Types (Inductive Cases)

$$\frac{v :: T \qquad (L = T) \in \overline{[x \mapsto \{x \Rightarrow \overline{d}\}]d}}{v :: (\{x \Rightarrow \overline{d}\}.L)} \tag{V-Sel}$$

$$\frac{v :: [x \mapsto v]T}{v :: \{x \Rightarrow T\}} \tag{V-Bind}$$

$$\frac{v :: T_1 \qquad v :: T_2}{v :: T_1 \wedge T_2} \tag{V-And}$$

$$\frac{v :: T_1}{v :: T_1 \vee T_2} \tag{V-Or1}$$

$$\frac{v :: T_2}{v :: T_1 \vee T_2} \tag{V-Or2}$$

Proof Sketch

- Let $v ::_m T$ denote a derivation of $v :: T$ with no more than m uses of (V-Bind).
- Let Widen_m denote the assumption that $v ::_m T$ can be widened:
  If $v ::_m T$ and $\emptyset \vdash T <: U$ then $v ::_m U$.
- Prove some substitution lemmas assuming widening.
  1. If $v ::_m T$ and Widen_m and $x : T, \Gamma \vdash S <: U$, then $[x \mapsto v]\Gamma \vdash [x \mapsto v]S <: [x \mapsto v]U$.
  2. If $v ::_m T$ and Widen_m and $x \neq z$ and $x : T, \Gamma \vdash z :_! T$, then $[x \mapsto v]\Gamma \vdash z :_! [x \mapsto v]T$.
  3. If $v ::_m T$ and Widen_m and $x : T, \Gamma \vdash x :_! U$, then $v ::_m [x \mapsto v]U$.
- Prove widening: $\forall m.\text{Widen}_m$.
- Prove empty-context value typing implies "possible types":
  If $\emptyset \vdash v : T$ then $v :: T$.

6. Beyond Normal Paths

- Relating paths across reduction steps?
- Use evaluation/normalization of paths to just relate values.
- Properties:

uniqueness	If $p \Downarrow v_1$ and $p \Downarrow v_2$ then $v_1 = v_2$.
confluence	If $p \longrightarrow p'$ and $p \Downarrow v$ then $p' \Downarrow v$.
strong normalization	If $\emptyset \vdash p :_! T$ then there is some v with $p \Downarrow v$.
preservation	If $\emptyset \vdash p :_! T$ and $p \Downarrow v$ then $v :: T$.

- The approach works well for simple paths, i.e. immutable fields of chain selections.
- For *application* in paths, work-in-progress. Once more, the issue is bootstrapping the lemmas given *substitution* in paths.

Conclusion

- Deriving DOT: $F_{<:}$, $F_{<:>}$, $D_{<:}$, $D_{<:>}$, Lattice, Records, Objects, ...
- A bottom-up exploration:
  - + interesting intermediary points in the landscape
  - survival of the fittest designs and proofs
    - + survival bias means design and proof are quite robust to variations...
    - – ...but also stuck in local sweet spots
    - = lots of time spent on dead ends
- Sound DOT design "discovered" rather than invented.

Bonus

DOT: Some Unsound Variations

- Add subsumption to member initialization.

$$\frac{\Gamma \vdash d : T \quad \Gamma \vdash T <: U}{\Gamma \vdash d : U} \qquad \text{(DSub)}$$

$$\{x \Rightarrow L = \top\} : \{x \Rightarrow L : \top..\bot\}$$

- Change type member initialization from $\{L = T\}$ to $\{L : S..U\}$.

$$\frac{\Gamma \vdash S <: U}{\{L : S..U\} : \{L : S..U\}} \qquad \text{(DTyp)}$$

$$\{x \Rightarrow L : \top..\bot\} : \{x \Rightarrow L : \top..\bot\}$$

Retrospective on Proving Soundness

A good proof is one that makes us wiser. – Yuri Manin

- Static semantics should be monotonic. All attempts to prevent bad bounds broke it.
- Embrace subsumption, don't requires precise calculations in arbitrary contexts.
- Create recursive objects concretely, enforcing good bounds and shape syntactically not semantically. Then subsume/abstract, if desired.
- Inversion lemmas need only hold in empty abstract environment.
- Tension between preservation and abstraction. Rely on precise types for runtime values.

Unsoundness in Scala (fits in a Tweet)

```
trait A { type L >: Any}
def id1(a: A, x: Any): a.L = x
val p: A { type L <: Nothing } = null
def id2(x: Any): Nothing = id1(p, x)
id2("oh")
```

Unsoundess in Java (thanks Ross Tate!)

```
class Unsound {
  static class Bound<A, B extends A> {}
  static class Bind<A> {
    <B extends A> A bad(Bound<A,B> bound, B b) {
      return b;
    }
  }
  public static <T,U> U coerce(T t) {
    Bound<U,? super T> bound = null;
    Bind<U> bind = new Bind<U>();
    return bind.bad(bound, t);
  }
}
```

Formal Model

Formal Model

This is a formal model for DOT at step 5 (including recursive subtyping, but excluding fields and full paths).

DOT Syntax

$t ::=$	**terms**:
x	variable
$\{z \Rightarrow \overline{d}\}$	object
$t.m(t)$	meth. app.
$d ::=$	**init.**:
$L = T$	type mem.
$m(x : T) = t$	meth. mem.
$v ::=$	**values**:
$\{z \Rightarrow \overline{d}\}$	object
$p ::=$	**paths**:
x	variable
v	value

$S, T, U ::=$	**types**:
$\top$	top
$\bot$	bot.
$T \wedge T$	inter.
$T \vee T$	union
$L : S..U$	type mem.
$m(x : S) : U$	meth. mem.
$p.L$	sel.
$\{z \Rightarrow T\}$	rec. sel.
$\Gamma ::=$	**contexts**:
$\emptyset \mid \Gamma, x : T$	var. bind.

DOT Subtyping $\Gamma \vdash S <: U$

Lattice structure

$$\Gamma \vdash \bot <: T \quad \text{(Bot)}$$

$$\Gamma \vdash T <: \top \quad \text{(Top)}$$

$$\frac{\Gamma \vdash T_1 <: T}{\Gamma \vdash T_1 \wedge T_2 <: T} \quad \text{(And11)}$$

$$\frac{\Gamma \vdash T <: T_1}{\Gamma \vdash T <: T_1 \vee T_2} \quad \text{(Or21)}$$

$$\frac{\Gamma \vdash T_2 <: T}{\Gamma \vdash T_1 \wedge T_2 <: T} \quad \text{(And12)}$$

$$\frac{\Gamma \vdash T <: T_2}{\Gamma \vdash T <: T_1 \vee T_2} \quad \text{(Or22)}$$

$$\frac{\Gamma \vdash T <: T_1 \ , \ T <: T_2}{\Gamma \vdash T <: T_1 \wedge T_2} \quad \text{(And2)}$$

$$\frac{\Gamma \vdash T_1 <: T \ , \ T_2 <: T}{\Gamma \vdash T_1 \vee T_2 <: T} \quad \text{(Or1)}$$

Properties

$$\Gamma \vdash T <: T \quad \text{(Refl)}$$

$$\frac{\Gamma \vdash T_1 <: T_2 \ , \ T_2 <: T_3}{\Gamma \vdash T_1 <: T_3} \quad \text{(Trans)}$$

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

Method and type members

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

$$\frac{\Gamma \vdash S_2 <: S_1 \ , \ U_1 <: U_2}{\Gamma \vdash L : S_1..U_1 <: L : S_2..U_2} \quad \text{(Typ)}$$

Type selections

$$\frac{\Gamma_{[x]} \vdash x :_! (L : T..\top)}{\Gamma \vdash T <: x.L} \quad \text{(Sel2)}$$

$$\frac{[z \mapsto \overline{d}]\overline{d} \ni L = T}{\Gamma \vdash T <: \{z \Rightarrow \overline{d}\}.L} \quad \text{(SSel2)}$$

$$\frac{\Gamma_{[x]} \vdash x :_! (L : \bot..T)}{\Gamma \vdash x.L <: T} \quad \text{(Sel1)}$$

$$\frac{[z \mapsto \overline{d}]\overline{d} \ni L = T}{\Gamma \vdash \{z \Rightarrow \overline{d}\}.L <: T} \quad \text{(SSel1)}$$

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

Recursive self types

$$\frac{\Gamma, z : T_1 \vdash T_1 <: T_2}{\Gamma \vdash \{z \Rightarrow T_1\} <: \{z \Rightarrow T_2\}} \quad \text{(BindX)}$$

$$\frac{\begin{array}{c}\Gamma, z : T_1 \vdash T_1 <: T_2 \\ z \notin fv(T_2)\end{array}}{\Gamma \vdash \{z \Rightarrow T_1\} <: T_2} \quad \text{(Bind1)}$$

DOT Typing $\boxed{\Gamma \vdash t :_{(!)} T}$

$$\frac{\Gamma(x) = T}{\Gamma \vdash x :_{(!)} T} \quad \text{(Var)}$$

$$\frac{\Gamma \vdash t :_{(!)} T_1 \;,\; T_1 <: T_2}{\Gamma \vdash t :_{(!)} T_2} \quad \text{(Sub)}$$

$$\frac{\Gamma \vdash p : [z \mapsto p]T}{\Gamma \vdash p : \{z \Rightarrow T\}} \quad \text{(Pack)}$$

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

DOT Typing $\boxed{\Gamma \vdash t : T}$

$$\frac{\Gamma \vdash t : (m(x : T_1) : T_2)\ ,\ t_2 : T_1 \quad x \notin fv(T_2)}{\Gamma \vdash t.m(t_2) : T_2} \quad \text{(TApp)}$$

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

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

DOT Member Initialization $\boxed{\Gamma \vdash d : T}$

$$\frac{\Gamma \vdash T <: T}{\Gamma \vdash (L = T) : (L : T..T)} \qquad \text{(DTyp)}$$

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

DOT Small-Step Operational Semantics $\boxed{t \longrightarrow t'}$

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

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

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