Type-Driven Partial Evaluation without Code Duplication

No Author Given

No Institute Given

Abstract. [1]

Keywords: Partial Evaluation

1 Introduction

```
def dot[V: Numeric](v1: Vector[V], v2: Vector[V]): V =
  (v1 zip v2).foldLeft(zero[V]){ case (prod, (cl, cr)) =>
    prod + cl * cr
  }
```

Fig. 1. Function for computing the dot product over generic `Numeric` values.

2 Formalization $F_{i<:}$

$t ::=$	Terms:
x, y	identifier
$(x : iT) \Rightarrow t$	function
$t(t)$	application
$\{\overline{x = t}\}$	record
$t.x$	selection
$\mathit{inline}\ t$	inlining request
$[X <: iT] \Rightarrow t$	type abstraction
$t[iT]$	type application
$S,\ T,\ U ::=$	Types:
$iS \Rightarrow jT$	function type
$\{\overline{x : iS}\}$	record type
$[X <: iS] \Rightarrow jT$	universal type
$\top$	top type
$iT,\ jT,\ kT,\ lT ::=$	Inlineable Types:
$T,\ \mathit{dynamic}\ T$	dynamic type
$\mathit{static}\ T$	static type
$\mathit{inline}\ T$	must inline type
$\Gamma ::=$	Contexts:
$\emptyset$	empty context
$\Gamma,\ x : iT$	term binding
$\Gamma,\ X <: iT$	type binding

Fig. 2. Syntax

$$\frac{x : iT \in \Gamma}{\Gamma \vdash x : iT} \qquad \text{(T-IDENT)}$$

$$\frac{\Gamma,\ x : iT_1 \vdash t : jT_2}{\Gamma \vdash (x : iT_1) \Rightarrow t : static\ iT_1 \Rightarrow jT_2} \qquad \text{(T-FUNC)}$$

$$\frac{\Gamma \vdash \overline{t : iT}}{\Gamma \vdash \{\overline{x = t}\} : static\ \{\overline{x : iT}\}} \qquad \text{(T-REC)}$$

$$\frac{\Gamma \vdash t_1 : i(jT_1 \Rightarrow kT_2) \quad \Gamma \vdash t_2 : jT_2}{\Gamma \vdash t_1(t_2) : (i \wedge j \wedge k)T_2} \qquad \text{(T-APP)}$$

$$\frac{\Gamma \vdash t : i\{x = jT_1, \overline{y = kT_2}\}}{\Gamma \vdash t.x : (i \wedge j)T_1} \qquad \text{(T-SEL)}$$

$$\frac{\Gamma \vdash t : static\ T}{\Gamma \vdash inline\ t : inline\ T} \qquad \text{(T-INLINE)}$$

$$\frac{\Gamma,\ X <: iT_1 \vdash t_2 : jT_2}{\Gamma \vdash [X <: iT_1] \Rightarrow t_2 : static\ [X <: iT_1] \Rightarrow jT_2} \qquad \text{(T-TABS)}$$

$$\frac{\Gamma \vdash t_1 : i([X <: jT_{11}] \Rightarrow kT_{12}) \quad \Gamma \vdash lT_2 <: jT_{11}}{\Gamma \vdash t_1[lT_2] : [X \mapsto lT_2](i \wedge k)T_{12}} \qquad \text{(T-TAPP)}$$

$$\frac{\Gamma \vdash t : iS \quad \Gamma \vdash iS <: jT}{\Gamma \vdash t : jT} \qquad \text{(T-SUB)}$$

Fig. 3. typing $\Gamma \vdash t : iT$

$$\frac{\forall i.\ i <: dynamic}{} \qquad \text{(IS-DYNAMIC)}$$

$$\frac{\forall i \in \{static,\ inline\}.\ i <: static}{} \qquad \text{(IS-STATIC)}$$

$$\frac{inline <: inline}{} \qquad \text{(IS-INLINE)}$$

Fig. 4. Inlinity Subtyping $i <: j$

$$\frac{\forall i,\ \forall j,\ i <: j.\ i \wedge j = j}{}$$

Fig. 5. Inlinity Intersection $i \wedge j$

$$\frac{}{\Gamma \vdash S <: S} \quad \text{(S-Refl)}$$

$$\frac{\Gamma \vdash S <: U \quad \Gamma \vdash U <: T}{\Gamma \vdash S <: U} \quad \text{(S-Trans)}$$

$$\frac{}{\Gamma \vdash S <: \top} \quad \text{(S-Top)}$$

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

$$\frac{\Gamma \vdash kT_1 <: iS_1 \quad \Gamma \vdash jS_2 <: lT_2}{\Gamma \vdash iS_1 \Rightarrow jS_2 <: kT_1 \Rightarrow lT_2} \quad \text{(S-Arrow)}$$

$$\frac{\Gamma,\ X <: iU_1 \vdash jS_2 <: kT_2}{\Gamma \vdash [X <: iU_1] \Rightarrow jS_2 <: [X <: iU_1] \Rightarrow kT_2} \quad \text{(S-All)}$$

$$\frac{}{\{x_p : i_p T_p \ ^{p \in 1..n+m}\} <: \{x_p : i_p T_p \ ^{p \in 1..n}\}} \quad \text{(S-RecWidth)}$$

$$\frac{\forall p \in 1..n.\ i_p S_p <: j_p T_p}{\{x_p : i_p S_p \ ^{p \in 1..n}\} <: \{x_p : j_p T_p \ ^{p \in 1..n}\}} \quad \text{(S-RecDepth)}$$

$$\frac{\{x_p : i_p S_p \ ^{p \in 1..n}\} \text{ is permutation of } \{y_p : j_p T_p \ ^{p \in 1..n}\}}{\{x_p : i_p S_p \ ^{p \in 1..n}\} <: \{y_p : j_p T_p \ ^{p \in 1..n}\}} \quad \text{(S-RecPerm)}$$

$$\frac{i <: j \quad \Gamma \vdash S <: T}{\Gamma \vdash iS <: jT} \quad \text{(S-Inline)}$$

Fig. 6. Subtyping $\Gamma \vdash iS <: jT$

3 Translating Scala to the Core Calculus

4 Case Studies

4.1 Integer Power Function

- Explain what happens.
- Typical partial evaluation example. Can be handled by D and Idris and not without duplication with type-driven partial evaluation.

```
@i? def pow(base: Double, exp: Int @i?): Double =
  if (exp == 0) 1 else base * pow(base, exp)
```

Fig. 7. Function for computing the non-negative power of a real number.

4.2 Variable Argument Functions

- `@i?` in argument position is a macro that expands the function to an underlying function `@i? def min_underlying[T: Numeric](values:Seq[T]@i?): T` and a macro that will call it according to the input parameters.
- Comparison to other approaches.

```
@i? def min[T: Numeric](@i? values:T*): T =
  values.tail.foldLeft(values.head)((min, el) => if (el < min) el else min)
```

Fig. 8. Function for computing the non-negative power of a real number.

4.3 Butterfly Networks

- Reference LMS. Discuss @i! annotation on classes. Works for both dynamic and static inputs.
- Comparison to LMS. Mention a pervasive number of annotations. Discuss duality of Exp[T] and @i!.

4.4 Dot Product

- Explain the removal of type classes together with inline. Explain how type classes are @i? and how they will completely evaluate if they are passed a static value.
- Comparison to other approaches.

```
object Numeric {
  @i! implicit def dnum: Numeric[Double] @i! = DoubleNumeric
  @i! def zero[T](implicit num: Numeric[T]): T = num.zero
  object Implicits {
    @i! implicit def infixNumericOps[T](x: T)(implicit num: Numeric[T]): Numeric[T]#Ops = new
  }
}
trait Numeric[T] {
  def plus(x: T, y: T): T
  def times(x: T, y: T): T
  def zero: T

  @i! class Ops(lhs: T) {
    @i! def +(rhs: T) = plus(lhs, rhs)
    @i! def *(rhs: T) = times(lhs, rhs)
  }
}

object DoubleNumeric extends Numeric[Double] {
  @i! def plus(x: Double @i?, y: Double @i?): Double = x + y
  @i! def times(x: Double @i?, y: Double @i?): Double = x * y
  @i! def zero: Double = 0.0
}
```

Fig. 9. Function for computing the non-negative power of a real number.

5 Evaluation

6 Related Work

7 Conclusion

References

1. Eugene Burmako and Martin Odersky. Scala Macros, a Technical Report. In *Third International Valentin Turchin Workshop on Metacomputation*, 2012.