Towards Practical Runtime Type Instantiation

Dispatch on Generics

Karl Naden
TLDI
January 28, 2012

Scientific Programming

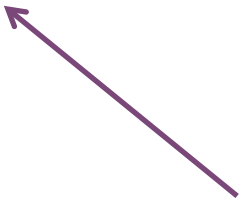
```
forecast() = do

    ... (*) number crunching

    temp = bigMatrix * reallyBigMatrix

    ... (*) more number crunching

end
```



When can this multiplication be optimized?

Algorithm choice

$$\texttt{temp} = \texttt{bigMatrix} * \texttt{reallyBigMatrix}$$

- Depends on the type of the entries

$$\texttt{bigMatrix} : \texttt{Matrix}[\![\texttt{Number}]\!]$$
$$\texttt{reallyBigMatrix} : \texttt{Matrix}[\![\texttt{Number}]\!]$$

  - General numbers – use standard algorithm
    - Many individual multiplications

$$\texttt{bigMatrix} : \texttt{Matrix}[\![\mathbb{Z}_2]\!]$$
$$\texttt{reallyBigMatrix} : \texttt{Matrix}[\![\mathbb{Z}_2]\!]$$

  - Binary Digits – create bit vectors and use bit operators for multiplication
    - Many multiplications at the same time
    - More efficient multiplication

How to Choose Best Algorithm?

$$\texttt{temp} = \texttt{bigMatrix} * \texttt{reallyBigMatrix}$$

- If know statically operands are binary matrixes
  - Can define a separate function for the programmer to call

$\texttt{binaryMatrixMult}(x : \texttt{Matrix}[\![\mathbb{Z}_2]\!], y : \texttt{Matrix}[\![\mathbb{Z}_2]\!]) = \ldots$

$\texttt{temp} = \texttt{binaryMatrixMult}(\texttt{bigMatrix}, \texttt{reallyBigMatrix})$

- Otherwise, could use multiple dispatch
  - Define two cases of * operator

$*(x : \texttt{Matrix}[\![\texttt{Number}]\!], y : \texttt{Matrix}[\![\texttt{Number}]\!]) = \textbf{do}$
$\quad \ldots (*)\ standard\ algorithm$
$\textbf{end}$
$*(x : \texttt{Matrix}[\![\mathbb{Z}_2]\!], y : \texttt{Matrix}[\![\mathbb{Z}_2]\!]) = \textbf{do}$
$\quad \ldots (*)\ bit\ vector\ algorithm$
$\textbf{end}$

  - Binary algorithm chosen dynamically if operands are binary matrixes at runtime
    - **Runtime type** must include **generic** information

Generic Dispatch

- Instantiated type of objects matters at runtime
  - Used to choose what code is executed – **No type erasure!**
- Instantiated type of generic functions matters
  - Instantiated generic used to create objects – must be tagged at runtime

$$\texttt{append}[\![X <: \texttt{Any}]\!](l : \texttt{List}[\![X]\!], n : X) : \texttt{List}[\![X]\!] = \textbf{do}$$
$$\quad \ldots (*)\ duplicate\ l,\ add\ n\ to\ duplicate,\ return\ duplicate$$
$$\textbf{end}$$

  - Type of resulting List depends on the instantiation of X
  - Want it to be as specific as possible

$$\texttt{removeNegatives}(l : \texttt{List}[\![\texttt{Number}]\!]) : \texttt{List}[\![\texttt{Number}]\!] = \ldots (*)\ iterate$$
$$\texttt{removeNegatives}(l : \texttt{List}[\![\mathbb{N}]\!]) : \texttt{List}[\![\mathbb{N}]\!] = \ldots (*)\ no\ op$$

- Don't always know the best answer statically
  - **Need runtime type instantiation!**

Problem – Runtime Instantiation

- Given
  - a generic function signature with type variables X_i
  - the runtime types of the inputs
  - Static return type of the function call
- Find an instantiation of the X_i such that
  - Type variable bounds met (Valid instantiation)
  - Runtime inputs are subtypes of parameter types
    (Function applicable to arguments)
  - Return type is a subtype of static return type (Type safe)
  - Minimize return type (As specific as possible)

heuristic

Fortress Overview

- Symmetric Multiple Dispatch
  - Statically guaranteed a most specific function definition exists
  - Chosen at runtime, static information for optimization of search
- Objects
  - Nominal (declared) subtyping
  - Variant Generics
    - Covariant: If Baz covariant, then $\texttt{Baz}[\![X]\!] <: \texttt{Baz}[\![Y]\!]$ if and only if $X <: Y$
    - Invariant: If Baz invariant, then $\texttt{Baz}[\![X]\!] <: \texttt{Baz}[\![Y]\!]$ if and only if $X = Y$
  - Types form a lattice
    - Meet (greatest lower bound) and join (least upper bound) defined
- Generic Functions $f[\![\overline{X <: \tau_X}]\!](\overline{y : \tau_y}) : \tau_{\mathrm{r}}$
  - Type variables X bound within
    - Upper bounds of type variables τ_X
    - Parameter types τ_y
    - Return type τ_{r}

Problem Example

- List Definition $\texttt{List}[\![\textbf{covariant } X <: \texttt{Any}]\!]$
- Number hierarchy $\mathbb{Z}_2 <: \mathbb{N} <: \texttt{Number} <: \texttt{Any}$
- Function

$\texttt{append}[\![X <: \texttt{Any}]\!](l : \texttt{List}[\![X]\!], n : X) : \texttt{List}[\![X]\!] = \textbf{do}$
$\quad \ldots (*)\ duplicate\ l,\ add\ n\ to\ duplicate,\ return\ duplicate$
$\textbf{end}$

- Code

$\texttt{theList} : \texttt{List}[\![\texttt{Number}]\!]$
$\texttt{toAdd} : \texttt{Number}$
$\ldots$
$\texttt{temp} = \texttt{append}(\texttt{theList}, \texttt{toAdd})$ Static return type: $\texttt{List}[\![\texttt{Number}]\!]$
$\texttt{removeNegatives}(\texttt{temp})$

Instantiation Calculation for X

$\texttt{append}[\![X <: \texttt{Any}]\!](l : \texttt{List}[\![X]\!], n : X) : \texttt{List}[\![X]\!] = \textbf{do}$
$\quad \ldots (*)\ duplicate\ l,\ add\ n\ to\ duplicate,\ return\ duplicate$
$\textbf{end}$

- Given - l : $\texttt{List}[\![\mathbb{N}]\!]$, n : $\mathbb{Z}_2$, $T_r = \texttt{List}[\![\texttt{Number}]\!]$
- Parameters:
  - $\texttt{List}[\![\mathbb{N}]\!] <: \texttt{List}[\![X]\!]$ implies $\mathbb{N} <: X$ (covariance)
  - $\mathbb{Z}_2 <: X$
- Type Variable Bounds
  - $X <: \texttt{Any}$
- Return Type Bounds
  - $\texttt{List}[\![X]\!] <: \texttt{List}[\![\texttt{Number}]\!]$ implies $X <: \texttt{Number}$ (covariance)
- Upper Bound
  - $meet(\texttt{Any}, \texttt{Number}) = \texttt{Number}$
- Lower Bound
  - $join(\mathbb{N}, \mathbb{Z}_2) = \mathbb{N}$
- Best (most specific) choice = $\mathbb{N}$

General Algorithm

Given $f[\![\overline{X <: \tau_X}]\!](\overline{y : \tau_y}) : \tau_r$, runtime input types T_y , and static return type T_r

1. Generate upper and lower bounds for each X from
   - Parameter constraints
   - Return type constraints
   - Type variable bound constraints
2. Let X_L=join(lowerBounds(X)) and X_U=meet(upperBounds(X))
3. Check X_L <: X_U
4. If true, return X_L as the instantiation

- Single pass algorithm
  - If type variables scoped left to right

$$\texttt{badlyScoped}[\![X <: Y, Y <: \texttt{List}[\![X]\!]]\!](x : X, y : Y) : \texttt{Any}$$

  - Otherwise could result in infinite iteration on bound generation

Remaining Challenges

1. Multiple potentially applicable function definitions
   - Run this algorithm on each to find most specific applicable
2. Contravariance
   - Function types in Fortress
   - May be undecidable
3. Efficient computation of meets, joins, and comparisons in subtype lattice
4. Enough performance gains to offset overhead?

```
result = forecast() do

    ... (*) number crunching

    temp = bigMatrix * reallyBigMatrix

    ... (*) more number crunching

end
```