IV Type System Inference Rules

April 3, 2025

Language

Declarations

Data declaration

The data types are represented by Algebraic Data Types (ADT) in the language.

```
data Either a b:
  [a] left,
  [b] right.
```

If a constructor does not take any types its input parameters can be omitted.

```
data Maybe a:
  nothing,
  [a] just.
```

ADT's can be recursive.

```
data Nat:
  zero,
  [Nat] suc.
```

Operator declaration

The operators are represented by a sequence of operators. All operator definitions must have a type annotation.

```
define [Nat, Nat] add [Nat]:
  case { zero { }, suc { add suc } }.

define [Nat] addThree [Nat]:
  zero suc suc suc add.
```

An operator's body can be empty

```
define [] nop []:.
```

Standard operations

Non-parametric

- `pop`
  Delete the top element.
- `dup`
  Duplicate the top element.

Parametric

Parameter `n` is a natural number without zero ($\mathbb{N} \setminus \{0\}$).

- `br-n`
  Move the topmost element to be the n-th element.
- `dg-n`
  Move the n-th element to the top of the stack.

Operator type system

Environment

Environment is constant throughout the type checking process. It contains operator definitions and data definitions.

$$\Gamma = (\text{opDefs}, \text{dataDefs})$$

Operator definitions consist of standard operators, user-defined operators, user-defined data constructors.

$$\text{opDefs} = \text{stdOps} \cup \text{userOps} \cup \text{userDataConstrs}$$

Data definitions consist of user-defined data types (Algebraic Data Types).

$$\text{dataDefs} = \text{userDatas}$$

Operator Type separation

- **Type** - represents the type of a value stored on the stack.
- **Operator Type** - represents the type of an element of an operator body.

Type definition

A type can be one of the following

- Monomorphic type, e.g., Int, Bool, Nat.
- Polymorphic type (type variable), e.g., a, b, c.
- Type application, e.g. Maybe a, Either a b.

Definition in Haskell

```
data Type
  = Mono String
  | Poly String
  | App Type Type
```

Operator Type definition

An operator has only one constructor that has the following fields

- pre - types of elements that the operator takes as input arguments.
- post - types of elements that the operator returns as output arguments.

Definition in Haskell

```
data OpType
  = OpType {
    pre :: [Type],
    post :: [Type]
  }
```

Notation

- "[*pre*][*post*]" represents an operator type. Where *pre* and *post* represent the input and output parameters.
- "{a $\mapsto$ Foo}[*pre*][*post*]" represents an application of a substitution "{a $\mapsto$ Foo}" on an operator type "[*pre*][*post*]"
- "$\alpha \cdot \beta$" represents list concatenation.
- An operator(s) between the stack descriptions is a shorthand, e.g., writing $[a, b]\texttt{foo bar}[c, d]$ is equivalent to $\texttt{foo bar} : [a, b][c, d]$.
- The leftmost element in the stack type description is the most recently pushed, e.g., $[]\texttt{foo bar baz}[\mathrm{Baz}, \mathrm{Bar}, \mathrm{Foo}]$.
- Greek letters denote lists of types, while Latin letters denote single types.

Type inference rules

Specialization Rule (Operator Type)

An operator type is considered a specific of a general operator type if there exists a substitution that turns the general type into the specific type.

$$\frac{[\alpha][\beta] = \{a' \mapsto a\}[\alpha'][\beta']}{[\alpha][\beta] \sqsubseteq [\alpha'][\beta']}$$

Empty rule

Allows to use an empty sequence of operators, that does not take any input arguments and does not return any output arguments.

$$\frac{}{\Gamma \vdash [][]}$$

Name rule

Allows to use previously defined operators.

$$\frac{[\alpha]\texttt{op}[\beta] \in \Gamma}{\Gamma \vdash [\alpha]\texttt{op}[\beta]}$$

Specialization and augmentation rule

Allows specialization and augmentation of operator types of operators.

- (Specialization) allows to use $[a]\texttt{id}[a]$ in place of $[\mathrm{Nat}]\texttt{inc}[\mathrm{Nat}]$
- (Augmentation) allows to use [Nat] `inc` [Nat] in place of [Nat, Nat] `inc2` [Nat, Nat]

$$\frac{\Gamma \vdash [\alpha']\texttt{x}[\beta'] \qquad [\alpha][\beta] \sqsubseteq [\alpha'][\beta']}{\Gamma \vdash [\alpha \cdot \gamma]\texttt{x}[\beta \cdot \gamma]}$$

Chain rule

Allows to compose operators. To be chained, LHS post should be equal (i.e., equal length and elements, including type variables) to the RHS pre.

$$\frac{\Gamma \vdash [\alpha]\texttt{x}[\beta] \qquad \Gamma \vdash [\psi]\texttt{y}[\omega] \qquad \beta = \psi}{\Gamma \vdash [\alpha]\texttt{x y}[\omega]}$$

Case rule

Operator type of the whole case expression must be a specific of all case arms. Pattern matching should be total on all constructors of the data type.

$$\frac{\{\texttt{constr1}, \ldots\} = \text{constrs}(t) \qquad \Gamma \vdash [t, \alpha']\texttt{constr}^{-1}\ \texttt{body}[\beta'] \qquad [t, \alpha][\beta] \sqsubseteq [t, \alpha'][\beta'] \qquad \ldots}{\Gamma \vdash [t, \alpha]\texttt{case}\{\texttt{constr1}\{\texttt{body1}\}\texttt{,}\ \texttt{...}\}[\beta]}$$

A case arm operator type is destructor $\texttt{constr}^{-1}$ chained with the body.
Where $\texttt{constr}^{-1}$ is the destructor of a constructor `constr`, i.e., Operator Type of `constr` with pre and post swapped.

$$\frac{\Gamma \vdash [\alpha]\texttt{constr}[t]}{\Gamma \vdash [t]\texttt{constr}^{-1}[\alpha]}$$

Stack operations

Dup

$$\frac{}{\Gamma \vdash [a]\texttt{dup}[a, a]}$$

Pop

$$\frac{}{\Gamma \vdash [a]\texttt{pop}[]}$$

Bury

$$\frac{||\alpha|| = n}{\Gamma \vdash [b, \alpha]\texttt{br-n}[\alpha, b]}$$

Dig

$$\frac{||\alpha|| = n}{\Gamma \vdash [\alpha, b]\texttt{dg-n}[b, \alpha]}$$