Lambda Cases

Dimitris Saridakis

1 Tokens

Keywords

`cases use_fields tuple_type or_type`

Value names

⟨*value-name*⟩ ::= ⟨*lower-case-letter*⟩ (⟨*lower-case-letter*⟩ | ‘`_`’)*

Type names

⟨*type-name*⟩ ::= ⟨*upper-case-letter*⟩ (⟨*upper-case-letter*⟩ | ⟨*lower-case-letter*⟩)*

2 Grammar

Program

⟨*program*⟩ ::= ⟨*value-defs*⟩ | ⟨*type-def*⟩

⟨*value-defs*⟩ ::= ⟨*value-names*⟩ ‘`:␣`’ ⟨*types*⟩ ‘`\n␣␣=`’ ⟨*value-exprs*⟩

⟨*value-names*⟩ ::= ⟨*value-name*⟩ (‘`,␣`’ ⟨*value-name*⟩)*

⟨*types*⟩ ::= ⟨*type*⟩ (‘`,␣`’ ⟨*type*⟩)*

⟨*value-exprs*⟩ ::= ⟨*value-expr*⟩ (‘`,␣`’ ⟨*value-expr*⟩)*

Types

⟨*type*⟩ ::= ⟨*func-type*⟩ | ⟨*prod-type*⟩ | ⟨*type-app*⟩

⟨*func-type*⟩ ::= ⟨*input-types-expr*⟩ ‘`␣->␣`’ ⟨*output-type*⟩

⟨*prod-type*⟩ ::= ⟨*prod-sub-type*⟩ (‘`␣x␣`’ ⟨*prod-sub-type*⟩)+

```
⟨type-app⟩            ::= [ ⟨t-inputs⟩ ‘==>’ ] ⟨type-name⟩ [ ‘<==’ ⟨t-inputs⟩ ]

⟨input-types-expr⟩    ::= ⟨many-ts-in-paren⟩ | ⟨one-type⟩

⟨output-type⟩         ::= ⟨prod-type⟩ | ⟨type-app⟩

⟨prod-sub-type⟩       ::= ‘(’ ( ⟨func-type⟩ | ⟨prod-type⟩ ) ‘)’ | ⟨type-app⟩

⟨one-type⟩            ::= ‘(’ ⟨func-type⟩ ‘)’ | ⟨prod-type⟩ | ⟨type-app⟩

⟨t-inputs⟩            ::= ⟨many-ts-in-paren⟩ | ‘(’ ⟨type⟩ ‘)’ | ⟨type-name⟩

⟨many-ts-in-paren⟩    ::= ‘(’ ⟨type⟩ (‘, ’ ⟨type⟩)+ ‘)’
```

Types

```
⟨type⟩                ::= ⟨prod-type⟩ [ ‘␣->␣’ ⟨prod-type⟩ ] | ⟨many-in-ts-func-t⟩

⟨many-in-ts-func-t⟩   ::= ⟨open-par-t⟩ ⟨comma-ts-close-par⟩ ‘␣->␣’ ⟨prod-type⟩

⟨prod-type⟩           ::= ⟨type-app⟩ ( ‘␣x␣’⟨type-app⟩ )*

⟨type-app⟩            ::= ⟨t-app-begin⟩ ( ⟨left-t-app⟩ | ⟨right-t-app⟩ )*

⟨t-name-t-app⟩        ::= ⟨type-name⟩ ( ⟨left-t-app⟩ | ⟨right-t-app⟩ )*

⟨t-app-begin⟩         ::= ⟨open-par-t⟩ ( ‘)’ | ⟨comma-ts-close-par⟩ ⟨right-t-app⟩ )

⟨open-par-t⟩          ::= ‘(’ ⟨type⟩

⟨paren-comma-types⟩   ::= ⟨open-par-t⟩ ‘,␣’ ⟨types⟩ ‘)’

⟨left-t-app⟩          ::= ‘<==’ ( ‘(’ ⟨types⟩ ‘)’ | ⟨type-name⟩ )

⟨right-t-app⟩         ::= ‘==>’ ( ‘(’ ⟨type⟩ ‘)’ | ⟨type-name⟩ )
```