# Scientific Units As Data Types

Mohamedali Basrai and Sam Fernandez

# Goals

- Allow for consistent manipulation of measurements “for free”
- Support user-defined units
- Provide runtime checks for scientific units

# Dimensions and Units

Dimension: Quantity of interest

Unit: Concrete measure of a dimension

| **Base Dimension** | Length | Mass | Time | Electric Current | Temperature | Substance | Luminous Intensity |
|---|---|---|---|---|---|---|---|
| **SI Base Unit** | Meter | Kilogram | Second | Ampere | Kelvin | Mole | Candela |

# DSL Grammar

prog ::= assign [prog] | alias [prog] | compare [prog] |
    convert [prog] | print [prog]

assign ::= <id> ‘:=’ base | expression

base ::= <id> | <number> units

units ::= unitProduct ‘/’ unitProduct

unitProduct ::= [unitProduct ‘*’ ] unit

unit ::= [<scale>’-’]<base unit>

expression ::= [expression op] base

op ::= ‘+’, ‘-’, ‘*’, ‘/’

# DSL Grammar Continued

alias ::= ‘DEFINE’ <name> ‘:=’ units |

‘DEFINE’ <name> ‘:=’ units ‘SCALE’ <number> |

‘DEFINE’ <name> ‘:=’ units ‘^’ <integer>

convert ::= <id> ‘TO’ units

compare ::= <id> comp <id>

comp ::= ‘==’ | ‘=:=’ | ‘===’ | ‘<=’ | ‘>=’ | ‘<’ | ‘>’

print ::= ‘PRINT’ <id> [‘AS’ units]

# Assignment

Assignments variables are scala symbols:

‘a := 5 (“kilo-grams”)

Units are strings consistent with the unit grammar

Scalars

# Aliasing

Aliases are defined with the DEFINE keyword:

DEFINE “km” := “kilo-meter”

Aliases can be scaled:

DEFINE “mi” := “‘kilo-meter“ SCALE 1.6

Or raised to an arbitrary power

DEFINE “mi2” := “mi” ^ 2 // mi2 = mi*mi

# Math

Variables can be assigned to expressions between symbols:

‘a := ‘b*(‘c+’d)

# Printing

Symbols are printed with their user defined units:

‘a := 5 (“kilo-meters”)

PRINT ‘a // 5 “kilo-meters”

We also allow formatted print:

PRINT ‘a  AS “mi” // 3.125

# Unit maintenance and cancelling

Units are automatically cancelled in arithmetic operations:

‘a := 10 (“kilo-meters*kilo-meters”)

‘b := 5 (“kilo-meters”)

‘c := ‘a / ‘b // 2 “kilo-meters”

# Conversion

The TO can be used to change the display units for a symbol:

‘a := 10 (“kilo-meters”)

PRINT ‘a // 10 “kilo-meters”

‘a TO “mi”

PRINT ‘a // 6.25 “mi”

# Comparisons

Comparisons are done between symbols:

‘a =:= ‘b // true if a and b are compatible

‘a === ‘b // true if a and b are compatible and have the same value

‘a == ‘b // true if a and b have the same value, throws exception if incompatible

<=, >=, <, >

# Interaction with Scala

Code can mix easily with Scala

VAL and STRING are used to retrieve data from symbols

# Unit Types: User Units and Fundamental Units

Units types for a symbol are represented by values

Each symbol has 2 unit representations:

Fundamental Units and User Units

Fundamental units are used for math and fast type checking

User units are used for printing and canceling

# Unit Compatibility

Two symbols are compatible if they have the same fundamental units

Addition, subtraction, and comparisons are only valid between compatible units

# User Units

User Units are represented as a tree

User Units are left associative in addition and subtraction

# Error Handling

Exceptions are thrown for unit type errors and illegal unit strings

# Demos

Cancelling, Aliasing (hotdog)

Left Associativity

Comparison(value vs units)

Error checking