Types with units of measure

Adam Gundry
University of Strathclyde

Scottish Programming Languages Seminar

15 March 2011

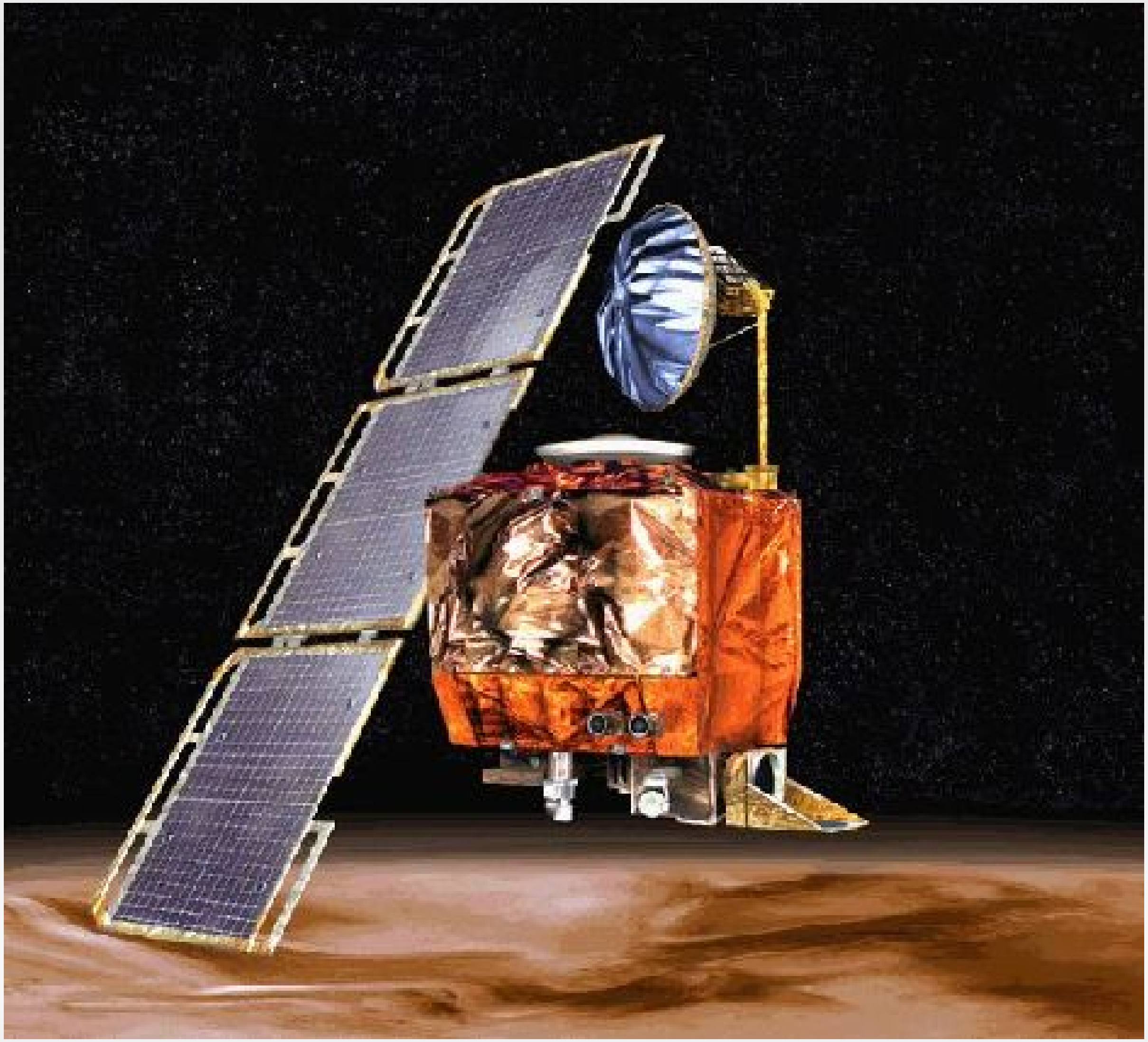
What are units of measure?

- A **dimension** is a physical quantity
  - length
  - time
  - mass
- A **unit** is a standard measure of quantity
  - metres
  - feet
  - seconds

What are units of measure?

- Arithmetic only works if units are compatible:
  - 10 m + 5 m = 15 m
  - 120 m / 60 s = 2 ms^{-1}
  - 6 m - 3 s = ???
- Can we enforce unit compatibility with types?

Why?



NASA/JPL

The plan

- Algebraic structure of units
- Units of measure in F#
- Type inference going wrong
- Type inference done differently
- Future speculations

Algebraic structure

- Base units
  - metres (m)
  - seconds (s)
  - ...
- Derived units
  - square metres (m^2)
  - metres per second (ms^{-1})
  - ...

Algebraic structure

- We have:
  - Multiplication: e.g. $m^2 = m \cdot m$
  - Dimensionless quantities: 1
  - Inverses: e.g. s^{-1}
- Subject to:
  - $d \cdot (e \cdot f) = (d \cdot e) \cdot f$ (associativity)
  - $1 \cdot d = d = d \cdot 1$ (identity)
  - $d \cdot d^{-1} = 1$ (inverse)
  - $d \cdot e = e \cdot d$ (commutativity)

Algebraic structure

- Units form an abelian group
- Specifically, the **free** abelian group generated by the base units
- No fractional powers...
- ...but we probably don't need them

Units of measure in F#

- Andrew Kennedy pioneered work on units of measure with **polymorphism**
- He introduced them in F#
- I'm following his design

Units of measure in F#

```
type [<Measure>] m;
type [<Measure>] s;
let vel   = 2.0<m/s>;
let accel = 3.8<m/s^2>;
let distance t =
    vel * t + accel * t * t;


val distance : float<s> → float<m>
```

Type inference is possible

- Free abelian group unification
  - has most general unifiers
  - is decidable
- We can infer types with Damas and Milner's Algorithm W

Type inference is tricksy

```
> fun x ->
-             (div x 5<m>, div x 2<s>);;

val it : int<'u> -> int<'u/m> * int<'u/s>
```

Type inference is tricksy

```
> fun x -> let f = div x in
-            (f 5<m>, f 2<s>);;
```

Type inference is tricksy

```
> fun x -> let f = div x in
-           (f 5<m>, f 2<s>);;

  --------------------^^^^

error FS0001: Type mismatch. Expecting a

    int<m>

but given a

    int<s>

The unit of measure 'm' does not match the unit
of measure 's'
```

Type inference is tricksy

- F# doesn't always infer principal types

- Let-generalisation is syntactic:
- does a occur free in the typing environment?

- This doesn't respect group equivalence:
- e.g. $a \cdot a^{-1} \equiv 1$ but a only occurs on one side

Type inference going wrong

`fun x -> let f = div x in (f 5<m>, f 2<s>)` : ?

x : t ⊢ `let f = div x in (f 5<m>, f 2<s>)` : ?

x : t ⊢ `div x` : ?

x : t ⊢ `div` : int<a b> → int<a> → int<b>

x : t ⊢ `div x` : int<a> → int<b> (if t = int<a b>)

x : int<a b> ⊢ `div x` : int<a> → int<b>

x : int<a b>, f : int<a> → int<b> ⊢ `(f 5<m>, f 2<s>)` : ?

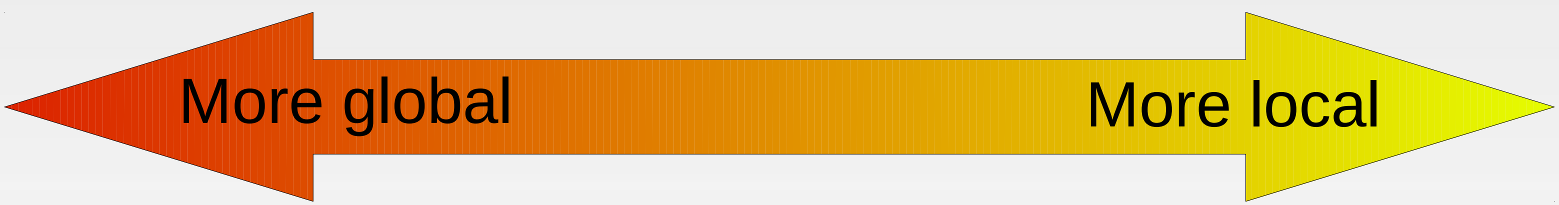
x : int<a b>, f : int<a> → int<b> ⊢ `f 5<m>` : int<b> (if a = m)

x : int<m b>, f : int<m> → int<b> ⊢ `f 2<s>` : int<b> (if m = s) ※

Type inference done differently

- Types go in the context
- Ordered by dependency

a := int<m>, ?b, x : b, c := a → b, ?d, ...



Type inference done differently

- Context divided into 'localities'
- Mark generalisation points for let-expressions

a := int<m>, ?b ◆ x : b, c := a → b ◆ ?d, ...

Type inference done differently

- Type variables only moved when necessary
- Most general unifier is a more precise notion
- Generalisation is easy: collect variables from the current locality

Type inference example

```
fun x -> let f = div x in (f 5<m>, f 2<s>) : ?
```

?t, x : t ♦ ⊢ div x : ?

?t, x : t ♦ ?a, ?b ⊢ div : int<a b> → int<a> → int<b>

?t, x : t ♦ ?a, ?b ⊢ div x : int<a> → int<b> (if t = int<ab>)

?t, x : t ♦ ?a, ?b, ?c ⊢ div x : int<a> → int<b> (if t = int<c>, c = a b)

?c, x : int<c> ♦ ?a, ?b ⊢ div x : int<a> → int<b> (if c = a b)

?c, x : int<c> ♦ ?a, b := c a^{-1} ⊢ div x : int<a> → int<b>

?c, x : int<c> ⊢ f : ∀a. int<a> → int<c a^{-1}>

?c, x : int<c>, f : … ⊢ (f 5<m>, f 2<s>) : int<c m^{-1}> × int<c s^{-1}>

⊢ … : ∀c. int<c> → int<c m^{-1}> × int<c s^{-1}>

Where do we go from here?

- Another free abelian group: the integers
- Extend this approach to type inference with:
  - Numeric inequalities
  - Local constraints (GADTs)
  - Higher-rank types
- I'm building a Haskell dialect with such features

Photo by Mike Gogulski
via Wikimedia Commons

References

- Andrew Kennedy
  Programming Languages and Dimensions
  Ph.D. Thesis (1996)

- Andrew Kennedy
  Types for units-of-measure: theory and practice
  CEFP '09 (2010)

References

- Adam Gundry, Conor McBride, James McKinna
  Type Inference in Context
  MSFP '10 (2010)
- Adam Gundry
  Type Inference for Units of Measure
  Draft (2011)