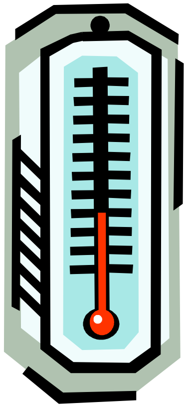
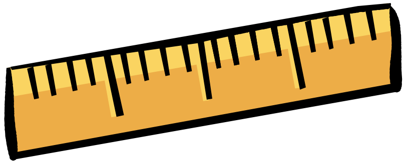
# Types for Units-of-Measure in F#

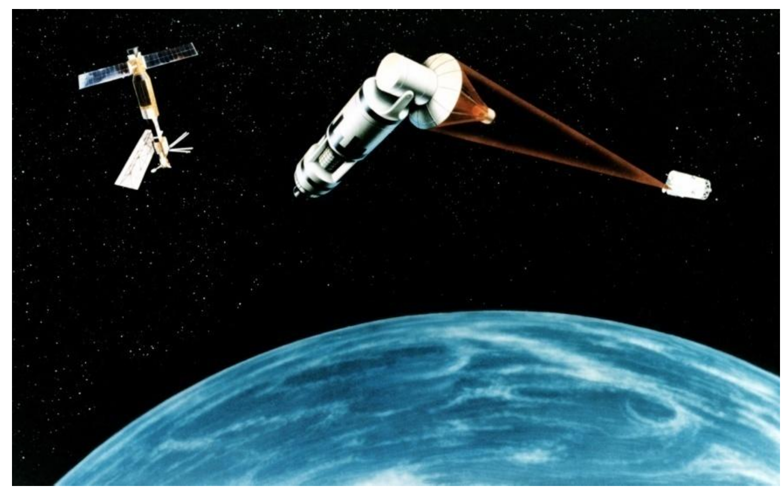
Andrew Kennedy
Microsoft Research
Cambridge



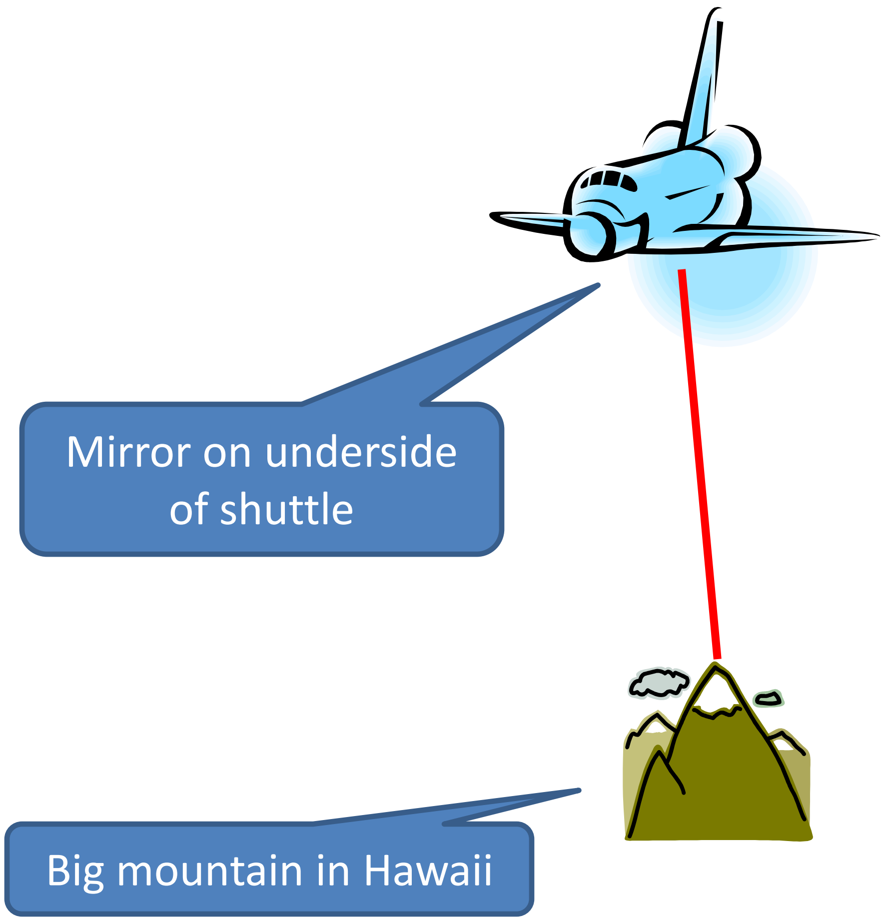


# NASA "Star Wars" experiment, 1983

23rd March 1983. Ronald Reagan announces SDI (or "Star Wars"): ground-based and space-based systems to protect the US from attack by strategic nuclear ballistic missiles.

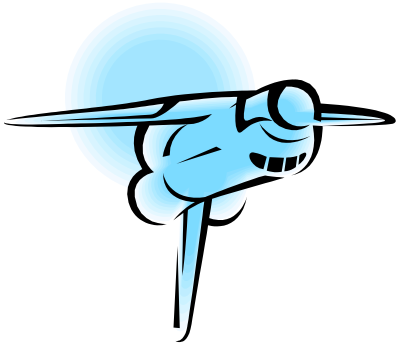


# 1985

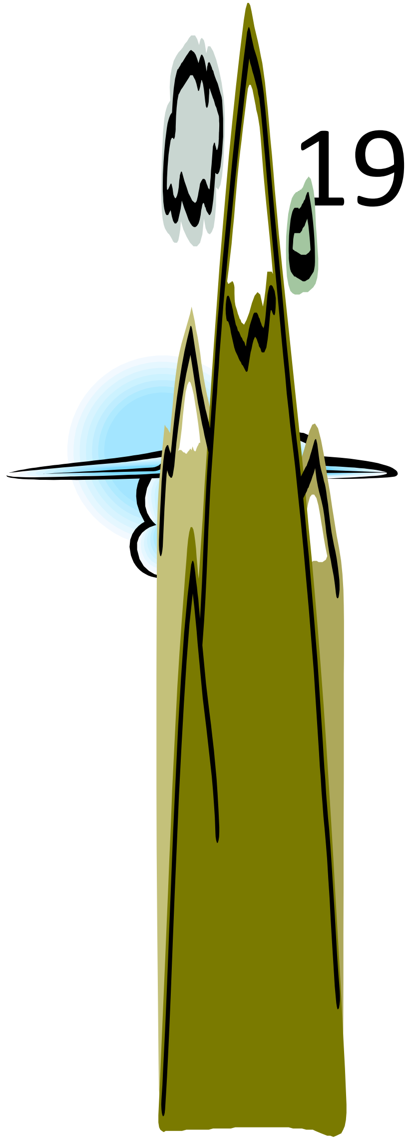

SDI experiment:
The plan

# 1985

SDI experiment:
The reality

1985

The reality

## Attention All Units, Especially Miles and Feet!

Much to the surprise of Mission Control, the space shuttle Discovery flew upside-down over Maui on 19 June 1985 during an attempted test of a Star-Wars-type laser-beam missile defense experiment. The astronauts reported seeing the bright-blue low-power laser beam emanating from the top of Mona Kea, but the experiment failed because the shuttle's reflecting mirror was oriented upward! A statement issued by NASA said that the shuttle was to be repositioned so that the mirror was pointing (downward) at a spot *10,023 feet* above sea level on Mona Kea; that number was supplied to the crew in units of feet, and was correctly fed into the onboard guidance system -- which unfortunately was expecting units in nautical miles, not feet. Thus the mirror wound up being pointed (upward) to a spot *10,023 nautical miles* above sea level. The San Francisco Chronicle article noted that "the laser experiment was designed to see if a low-energy laser could be used to track a high-speed target about 200 miles above the earth. By its failure yesterday, NASA unwittingly proved what the Air Force already knew -- that the laser would work only on a 'cooperative target' -- and is not likely to be useful as a tracking device for enemy missiles." [This statement appeared in the S.F. Chronicle on 20 June, excerpted from the L.A. Times; the NY Times article on that date provided some controversy on the interpretation of the significance of the problem.] The experiment was then repeated successfully on 21 June (using nautical miles). The important point is not whether this experiment proves or disproves the viability of Star Wars, but rather that here is just one more example of an unanticipated problem in a human-computer interface that had not been detected prior to its first attempted actual use.

# NASA Mars Climate Orbiter, 1999

CNN.com

sci-tech> space> story page

MAIN PAGE
WORLD
U.S.
LOCAL
POLITICS
WEATHER
BUSINESS
SPORTS
TECHNOLOGY
SPACE
HEALTH
ENTERTAINMENT
BOOKS
TRAVEL
FOOD
ARTS & STYLE
NATURE
IN-DEPTH
ANALYSIS
myCNN

Headline News brief
news quiz
daily almanac

MULTIMEDIA:
video
video archive
audio
multimedia showcase
more services

E-MAIL:
Subscribe to one of our news e-mail lists.
Enter your address:

exploringmars

in-depth specials

## Metric mishap caused loss of NASA orbiter

***September 30, 1999***
*Web posted at: 4:21 p.m. EDT (2021 GMT)*

**In this story:**

Metric system used by NASA for many years

Error points to nation's conversion lag

**RELATED STORIES, SITES**

NASA's Climate Orbiter was lost September 23, 1999

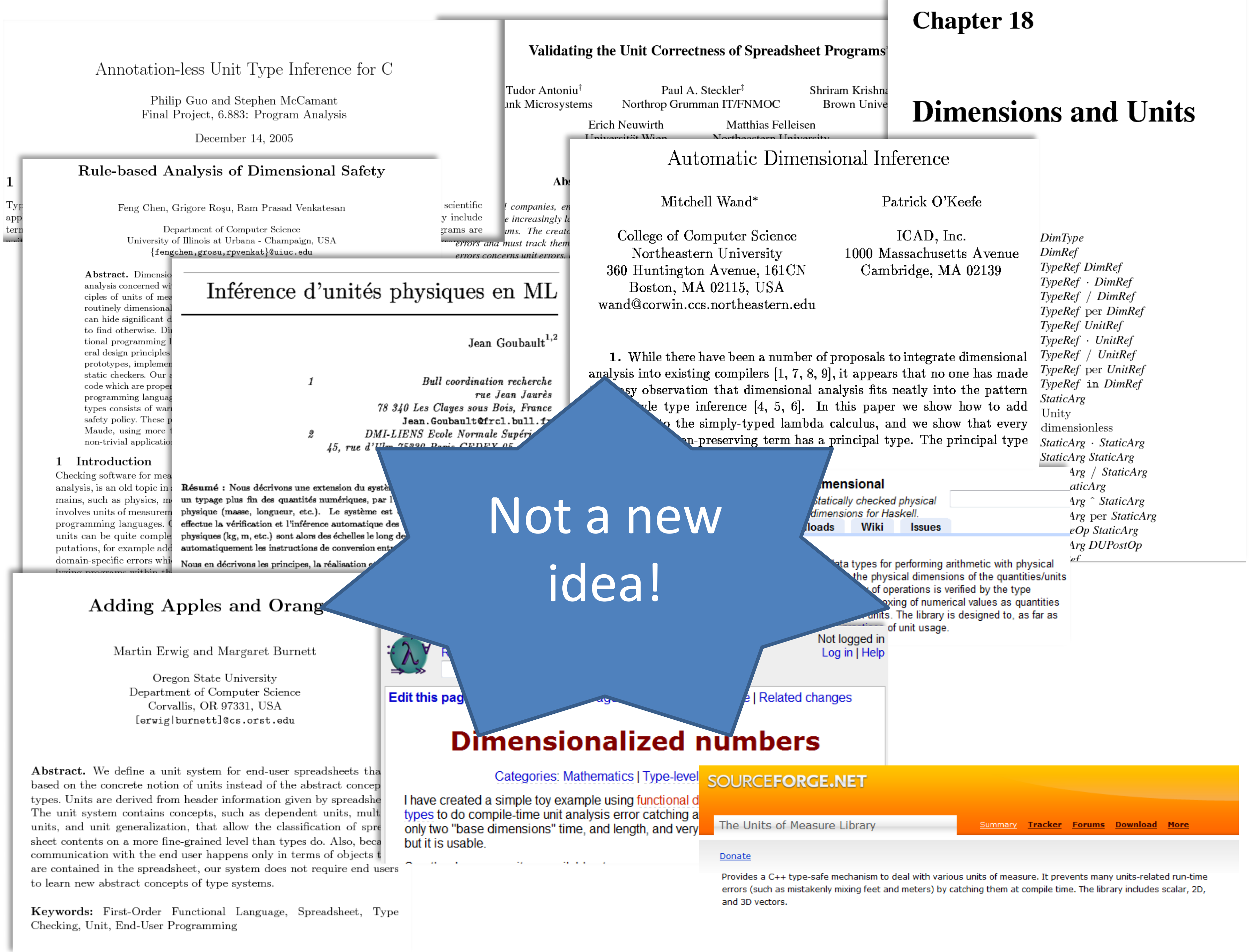
*By Robin Lloyd*
*CNN Interactive Senior Writer*

(CNN) -- NASA lost a $125 million Mars orbiter because a Lockheed Martin engineering team used English units of measurement while the agency's team used the more conventional metric system for a key spacecraft operation, according to a review finding released Thursday.

The units mismatch prevented navigation information from transferring between the Mars Climate Orbiter spacecraft team in at Lockheed Martin in Denver and the flight team at NASA's Jet Propulsion Laboratory in Pasadena, California.

# Solution

- Check units at development time, by
  - Static analysis, *or*
  - Type checking


Not a new idea!
Annotation-less Unit Type Inference for C
Philip Guo and Stephen McCamant
Final Project, 6.883: Program Analysis
December 14, 2005
Validating the Unit Correctness of Spreadsheet Programs
Tudor Antoniu
Paul A. Steckler
Northrop Grumman IT/FNMOC
Erich Neuwirth
Matthias Felleisen
Chapter 18
Dimensions and Units
Rule-based Analysis of Dimensional Safety
Feng Chen, Grigore Roşu, Ram Prasad Venkatesan
Department of Computer Science
University of Illinois at Urbana - Champaign, USA
{fengchen,grosu,rpvenkat}@uiuc.edu
1 Introduction
Automatic Dimensional Inference
Mitchell Wand*
Patrick O'Keefe
College of Computer Science
Northeastern University
360 Huntington Avenue, 161CN
Boston, MA 02115, USA
wand@corwin.ccs.northeastern.edu
ICAD, Inc.
1000 Massachusetts Avenue
Cambridge, MA 02139
Inférence d'unités physiques en ML
Jean Goubault[1,2]
1 Bull coordination recherche
rue Jean Jaurès
78 340 Les Clayes sous Bois, France
Jean.Goubault@frcl.bull.fr
2 DMI-LIENS Ecole Normale Supérieure
Adding Apples and Oranges
Martin Erwig and Margaret Burnett
Oregon State University
Department of Computer Science
Corvallis, OR 97331, USA
[erwig|burnett]@cs.orst.edu
Abstract. We define a unit system for end-user spreadsheets that is based on the concrete notion of units instead of the abstract concept of types. Units are derived from header information given by spreadsheets. The unit system contains concepts, such as dependent units, multiple units, and unit generalization, that allow the classification of spreadsheet contents on a more fine-grained level than types do. Also, because communication with the end user happens only in terms of objects that are contained in the spreadsheet, our system does not require end users to learn new abstract concepts of type systems.
Keywords: First-Order Functional Language, Spreadsheet, Type Checking, Unit, End-User Programming
Statically checked physical dimensions for Haskell.
Wiki
Issues
Not logged in
Log in | Help
Edit this page
Related changes
Dimensionalized numbers
Categories: Mathematics | Type-level
I have created a simple toy example using functional dependencies and types to do compile-time unit analysis error catching at only two "base dimensions" time, and length, and very but it is usable.
SOURCEFORGE.NET
The Units of Measure Library
Summary Tracker Forums Download More
Donate
Provides a C++ type-safe mechanism to deal with various units of measure. It prevents many units-related run-time errors (such as mistakenly mixing feet and meters) by catching them at compile time. The library includes scalar, 2D, and 3D vectors.

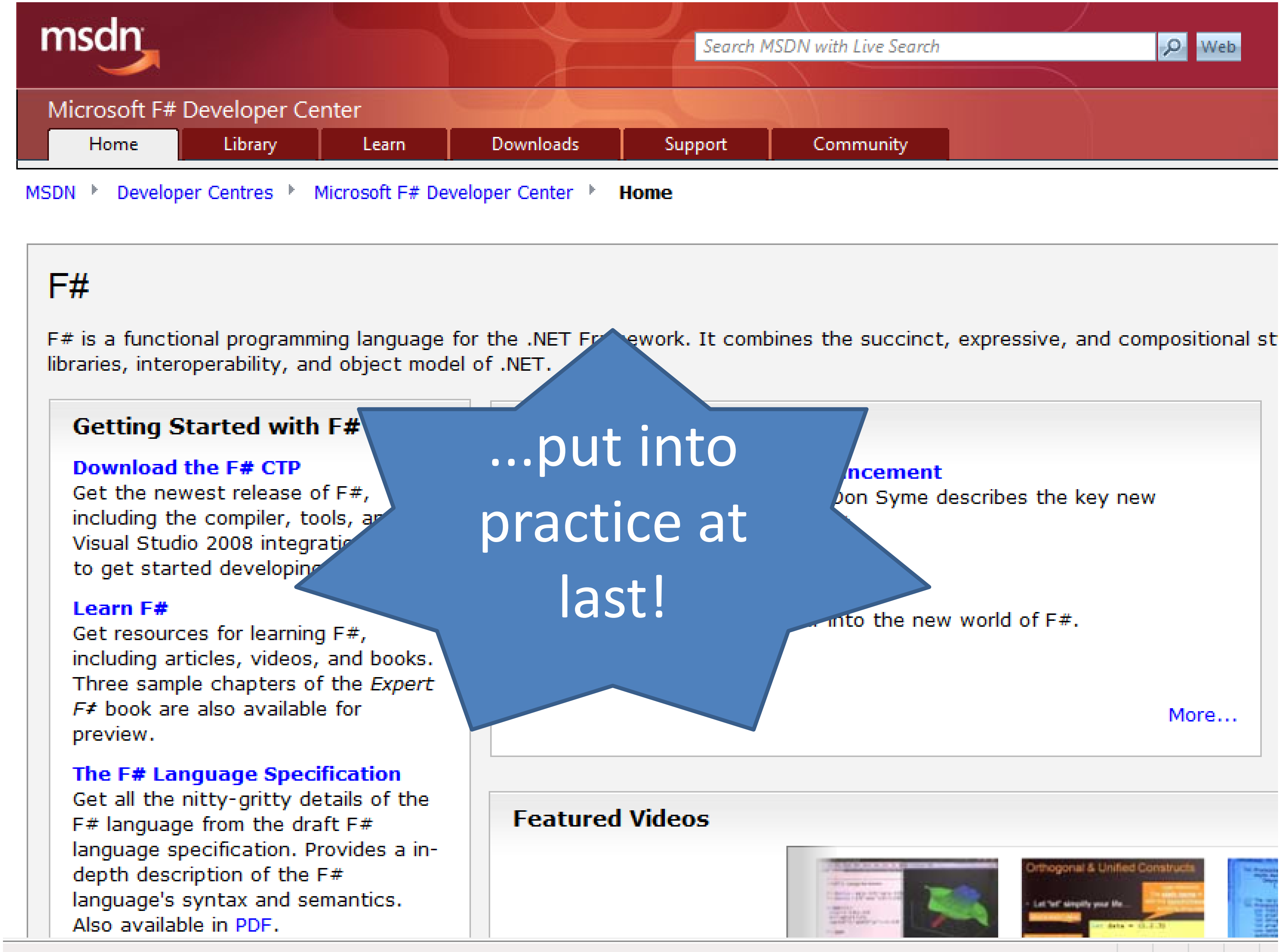
# Programming Languages and Dimensions

Andrew John Kennedy
St. Catharine's College

**Last century...**

A dissertation submitted to the University of Cambridge
towards the degree of Doctor of Philosophy

November 1995


msdn
Search MSDN with Live Search
Web
Microsoft F# Developer Center
Home
Library
Learn
Downloads
Support
Community
MSDN
Developer Centres
Microsoft F# Developer Center
Home
F#
F# is a functional programming language for the .NET Fr
ework. It combines the succinct, expressive, and compositional st
libraries, interoperability, and object model of .NET.
Getting Started with F#
Download the F# CTP
Get the newest release of F#,
including the compiler, tools, an
Visual Studio 2008 integratio
to get started developin
Learn F#
Get resources for learning F#,
including articles, videos, and books.
Three sample chapters of the Expert
F# book are also available for
preview.
The F# Language Specification
Get all the nitty-gritty details of the
F# language from the draft F#
language specification. Provides a in-
depth description of the F#
language's syntax and semantics.
Also available in PDF.
ncement
Don Syme describes the key new
into the new world of F#.
More...
Featured Videos
Orthogonal & Unified Constructs
...put into
practice at
last!

# Talk overview

- Practice
  - What is F#?
  - A tour of units in F#
  - Case studies
- Theory
  - Type system
  - Type inference
- Future

# What is F#?

- It's a functional language in the ML tradition

  core is compatible with core of Caml

  + .NET object model , builds on experience of SML.NET, MLj

  + active patterns, quotations, monad comprehensions, units-of-measure, lightweight syntax and other features

- Shipping as a product with next release of Visual Studio
  - Community Tech Preview released September 08
  - Also available for Mac/Linux via the Mono runtime
  - Come to Don Syme's CUFP talk (9am Friday), or the DEFUN tutorial (Sunday pm)

# Units-of-measure in F#

- *Type system* extension
  - Not just a static analysis tool
- Minimally invasive
  - Type inference, in the spirit of ML & Haskell
    - Annotate literals with units, let inference do the rest
    - But overloading must be resolved
  - No run-time cost (erasure)
- Support F# object model as far as possible
- Extensible
  - Not just for floats!

# Feature Tour

# Case studies

- We've been using the units feature at Microsoft for a few months now
  - Machine learning (Ralf Herbrich)
  - Games (Phil Trelford)
  - Physics simulation (Philipp Hennig, Don Syme, Chris Smith)
  - Finance (Luca Bolognese)

# Feedback from users

- Units are useful
  - They really do catch unit errors (Ralf, Phil, Philipp)
  - They *inform* the developer, and “correct” types help catch errors e.g.

```
let doublesqr x = sqr x + x
```

val doublesqr : float -> float

```
let doublesqr x = sqr x + sqr x
```

val doublesqr : float<'u> -> float<'u ^ 2>

- Automatic unit conversions: would be nice, but surprisingly not a big request
- Need for “unit asserts” for external code e.g.

```
type System.Math =
  with
    val Sqrt : float<'u^2> -> float<'u>
  end
```

# Theory

# The type system, informally

- Take the ML type system with Hindley-Milner inference
- Add a new *sort*: Measure

```
[<Measure>] type kg
type ([<Measure>] 'a) complex = {real:float<'a>; imag:float<'a>}
```

- Add operators on Measures (product, inverse, no units)

```
val (*) : float<'a> -> float<'b> -> float<'a 'b>
[<Measure>] type Hz = s^-1
val norm : Vector<'a> -> Vector<1>
```

inverse

product

no units

- Build in equational theory on Measures (commutativity, associativity, identity, inverses i.e. Abelian group)
- Refine the types of arithmetic operators e.g.

```
val sqrt : float<'a^2> -> float<'a>
val (/) : float<'u> -> float<'v> -> float<'u/'v>
```

# Toy type system, formally

**Syntax**

| | |
|---|---|
| units | $\mu ::= u \mid b \mid \mathbf{1} \mid \mu_1 \cdot \mu_2 \mid \mu^{-1}$ |
| types | $\tau ::= \alpha \mid \texttt{float}\,\mu \mid \tau_1 \rightarrow \tau_2 \mid \texttt{bool} \mid \cdots$ |
| type schemes | $\sigma ::= \forall \vec{u}.\tau$ |
| expressions | $e ::= x \mid c \in \mathbb{R} \mid e\ e \mid \lambda x.e \mid \texttt{let}\ x = e_1\ \texttt{in}\ e_2 \mid \cdots$ |
| judgments | $u_1, \ldots, u_m; x_1 : \sigma_1, \ldots, x_n : \sigma_n \vdash e : \tau$ |

**Equational theory of units (Abelian groups)**

| | | | |
|---|---|---|---|
| identity | $\mathbf{1} \cdot \mu =_U \mu$ | inverse | $\mu \cdot \mu^{-1} =_U \mathbf{1}$ |
| assoc | $(\mu_1 \cdot \mu_2) \cdot \mu_3 =_U \mu_1 \cdot (\mu_2 \cdot \mu_3)$ | comm | $\mu_1 \cdot \mu_2 =_U \mu_2 \cdot \mu_1$ |

**Typing rules (excerpt)**

$$\frac{(x : \forall \vec{u}.\tau) \in \Gamma}{\Delta; \Gamma \vdash x : \tau[\vec{\mu}/\vec{u}]} \qquad \frac{\Delta; \Gamma \vdash e : \tau_1 \qquad \tau_1 =_U \tau_2}{\Delta; \Gamma \vdash e : \tau_2} \qquad \frac{}{\Delta; \Gamma \vdash c : \texttt{float}\,\mathbf{1}}$$

$$\frac{\Delta, \vec{u}; \Gamma \vdash e_1 : \tau_1 \quad \Delta; \Gamma, x : \forall \vec{u}.\tau_1 \vdash e_2 : \tau_2}{\Delta; \Gamma \vdash \texttt{let}\ x = e_1\ \texttt{in}\ e_2 : \tau_2} \qquad \frac{\Delta; \Gamma, x : \tau_1 \vdash e : \tau_2}{\Delta; \Gamma \vdash \lambda x.e : \tau_1 \rightarrow \tau_2}$$

# Type inference and principal types

- The type systems of SML, Caml, Haskell and F# have (in principle, at least) the *principal types* property:
  - if expression *e* is typeable there exists a unique type scheme $\sigma$ such that all valid types are instances of $\sigma$
  - moreover, an inference algorithm will find the principal type
- If type checking *e* produces a type scheme that instantiates to $\tau$ write $\mathrm{tc}(e) \leq \tau$
- We can express correctness as
  - Soundness: $\mathrm{tc}(e) \leq \tau \Rightarrow \vdash e : \tau$
  - Completeness: $\vdash e : \tau \Rightarrow \mathrm{tc}(e) \leq \tau$
- Units-of-measure also have principal types, but algorithm is trickier

# ML type inference algorithm

- Two essential ingredients
  1. **Unification**. A *unifier* of two types $\tau_1$ and $\tau_2$ is a substitution S on type variables such that S($\tau_1$)=S($\tau_2$). For unifiable types, there is a *most general unifier.*
  2. **Generalization**. To type let x = $e_1$ in $e_2$ find a type $\tau$ for $e_1$ and then quantify on the variables that are free in $\tau$ but not free in the type environment $\Gamma$.

# The good news...

- For units, a unifier of two unit expressions $\mu_1$ and $\mu_2$ is a substitution S on unit variables such that S($\mu_1$)$=_U$ S($\mu_2$)
- Fortunately, Abelian Group unification is
  - *unitary* (unique most general unifiers exist with respect to the equational theory), and
  - *decidable* (algorithm is a variation of Gaussian elimination)

# Unification algorithm

$\text{Unify}(\mu_1, \mu_2) = \text{UnifyOne}(\mu_1 \cdot {\mu_2}^{-1})$

$\text{UnifyOne}(\mu) =$

$\quad \text{let } \mu = u_1^{x_1} \cdots u_m^{x_m} \cdot b_1^{y_1} \cdots b_n^{y_n} \text{ where } |x_1| \leqslant |x_2|, \cdots, |x_m|$

$\quad \text{in}$

$\quad\quad \text{if } m = 0 \text{ and } n = 0 \text{ then } I$

$\quad\quad \text{if } m = 0 \text{ and } n \neq 0 \text{ then fail}$

$\quad\quad \text{if } m = 1 \text{ and } x_1 \mid y_i \text{ for all } i \text{ then } \{u_1 \mapsto b_1^{-y_1/x_1} \cdots b_m^{-y_n/x_1}\}$

$\quad\quad \text{if } m = 1 \text{ otherwise then fail}$

$\quad\quad \text{else } S_2 \circ S_1 \text{ where}$

$\quad\quad\quad S_1 = \{u_1 \mapsto u_1 \cdot u_2^{-\lfloor x_2/x_1 \rfloor} \cdots u_m^{-\lfloor x_m/x_1 \rfloor} \cdot b_1^{-\lfloor y_1/x_1 \rfloor} \cdots b_n^{-\lfloor y_n/x_1 \rfloor}\}$

$\quad\quad\quad S_2 = \text{UnifyOne}(S_1(\mu))$

# Unification in action

$$u^3 \cdot v^2 =_U \mathrm{kg}^6$$

⇩ rewrite

$$u^3 \cdot v^2 \cdot \mathrm{kg}^{-6} =_U \mathbf{1}$$

⇩ apply $\{v \mapsto v \cdot u^{-1} \cdot \mathrm{kg}^3\}$

$$u \cdot v^2 =_U \mathbf{1}$$

⇩ apply $\{u \mapsto u \cdot v^{-2}\}$

$$u =_U \mathbf{1}$$

⇩ apply $\{u \mapsto \mathbf{1}\}$

$$\mathbf{1} =_U \mathbf{1}$$

Success!

# The bad news...

- Generalization based on free variables is sound but not complete for units-of-measure
- Why? Because the notion of syntactic free variables is not stable under various transformations.

1. "Free variables" is not stable under equivalence of types e.g.
   $\mathrm{fv}(u \cdot v \cdot u^{-1}) \neq \mathrm{fv}(v)$.
   Solution: *normalize*

2. "Free variables" is not stable under equivalence of type schemes e.g.
   fv ($\forall$ u. float<u· v>$\rightarrow$ float<u· v>) $\neq$ fv($\forall$ u.float<u>$\rightarrow$ float<u>)
   Solution: *normalize*

3. "Generalizable variables" is not stable under equivalence of typings e.g.
   d : float<u· v> $\vdash$ *expr* : float<u>$\rightarrow$ float<v>
   d : float<u· v> $\vdash$ *expr* : float<u· w>$\rightarrow$ float<v· $w^{-1}$>
   Solution: *normalize*

# Type Scheme Equivalence

- Two type schemes are equivalent if they instantiate to the same set of types
- For vanilla ML, this just amounts to renaming quantified type variables or removing redundant quantifiers.
- For ML + units, there are many non-trivial equivalences. E.g.

$$/ : \forall uv.\texttt{float}\, u \to \texttt{float}\, v \to \texttt{float}\, u \cdot v^{-1}$$
$$/ : \forall uvw.\texttt{float}\, w \cdot u \to \texttt{float}\, v \to \texttt{float}\, w \cdot u \cdot v^{-1}$$
$$/ : \forall uv.\texttt{float}\, u^{-1} \to \texttt{float}\, v^{-1} \to \texttt{float}\, u^{-1} \cdot v$$
$$/ : \forall uv.\texttt{float}\, u \cdot v \to \texttt{float}\, u \to \texttt{float}\, v$$
$$/ : \forall uv.\texttt{float}\, u \to \texttt{float}\, v^{-1} \to \texttt{float}\, u \cdot v$$
$$/ : \forall uv.\texttt{float}\, w \cdot u \to \texttt{float}\, w \cdot v \to \texttt{float}\, u \cdot v^{-1}$$

# Simplifying type schemes

- It's possible to show that two type schemes are equivalent iff there is an invertible substitution on the bound variables that maps between them (this is a "change of basis")
- *Idea*: compute such a substitution that puts a type scheme in some kind of preferred "normal form". Desirable properties:
  - No redundant bound or free variables (so number of variables = number of "degrees of freedom")
  - Minimize size of exponents
  - Use positive exponents if possible
  - Unique up to renaming
- Such a form does exist, and corresponds to Hermite Normal Form from algebra
  - Pleasant side-effect: deterministic ordering on variables in type

# Simplification in action

$$\forall uv.\texttt{float}\, w \cdot u \to \texttt{float}\, w \cdot v^{-1} \to \texttt{float}\, u \cdot v$$

$$\Downarrow \{u \mapsto u \cdot w^{-1}\}$$

$$\forall uv.\texttt{float}\, u \to \texttt{float}\, w \cdot v^{-1} \to \texttt{float}\, w^{-1} \cdot u \cdot v$$

$$\Downarrow \{v \mapsto v^{-1}\}$$

$$\forall uv.\texttt{float}\, u \to \texttt{float}\, w \cdot v \to \texttt{float}\, w^{-1} \cdot u \cdot v^{-1}$$

$$\Downarrow \{v \mapsto v \cdot w^{-1}\}$$

$$\forall uv.\texttt{float}\, u \to \texttt{float}\, v \to \texttt{float}\, u \cdot v^{-1}$$

# Generalization: example problem

- Suppose

$$\Gamma = \{/ : \forall u v.\texttt{float}\, u \cdot v \to \texttt{float}\, u \to \texttt{float}\, v\}$$

- Infer a type for

$$\emptyset; \Gamma \vdash \lambda x.\texttt{let}\ f = /x\ \texttt{in}\ (f\ \texttt{1.0<kg>}, f\ \texttt{2.0<s>}) : ?$$

- Problem: when typing let, we can’t generalize:

$$\Gamma, x : \texttt{float}\, u \cdot v \vdash /x : \texttt{float}\, u \to \texttt{float}\, v$$

- Solution: apply a “simplifying” substitution to the environment:

$$u \mapsto u \cdot v^{-1}$$

$$\Gamma, x : \texttt{float}\, u \vdash /x : \texttt{float}\, u \cdot v^{-1} \to \texttt{float}\, v$$

# Generalization

- Recipe:
  - Use the "simplify" algorithm on the *free variables* of the type environment $\Gamma$ to compute an invertible change-of-basis substitution *S*
  - Apply *S* to both $\Gamma$ and the inferred type $\tau$
  - Compute generalizable variables in the usual way i.e. fv(S($\tau$)) \ fv(S($\Gamma$))
  - Apply $S^{-1}$ to the resulting type scheme.
- Summary:

$$\begin{aligned}&\mathrm{Gen}(\Gamma, \tau) = S^{-1}(\forall u_1, \ldots, u_n . S(\tau))\\ &\text{where } \mathrm{fv}(S(\tau)) \setminus \mathrm{fv}(S(\Gamma)) = \{u_1, \ldots, u_n\}\\ &\text{and } S \text{ is simplifier of free variables of } \Gamma.\end{aligned}$$

# Generalization in action

$$\Gamma, x : \texttt{float}\, u \cdot v \vdash /x : \texttt{float}\, u \rightarrow \texttt{float}\, v$$

$u \mapsto u \cdot v^{-1}$

$$\Gamma, x : \texttt{float}\, u \vdash /x : \texttt{float}\, u \cdot v^{-1} \rightarrow \texttt{float}\, v$$

quantify

$$\Gamma, x : \texttt{float}\, u \vdash /x : \forall v. \texttt{float}\, u \cdot v^{-1} \rightarrow \texttt{float}\, v$$

rename

$$\Gamma, x : \texttt{float}\, u \vdash /x : \forall w. \texttt{float}\, u \cdot w^{-1} \rightarrow \texttt{float}\, w$$

$u \mapsto u \cdot v$

$$\Gamma, x : \texttt{float}\, u \cdot v \vdash /x : \forall w. \texttt{float}\, u \cdot v \cdot w^{-1} \rightarrow \texttt{float}\, w$$

# Beyond Hindley-Milner

- Non-regular datatypes

```
// Non-regular datatype: a list of derivatives of a function
type derivs<[<Measure>] 'u, [<Measure>] 'v> =
  Nil
| Cons of (float<'u> -> float<'v>) * derivs<'u,'v/'u>
```

- Polymorphic recursion

```
let rec makeDerivs<[<Measure>] 'u, [<Measure>] 'v>
  (h:float<'u>)
  (f:float<'u> -> float<'v>)
  (n:int) : derivs<'u,'v> =
  if n=0 then Nil else Cons(f, makeDerivs h (diff h f) (n-1))
```

# Future developments (F# v.next?)

- Automatic unit conversion
  - FAQ, but not a top priority for developers who have tried out the feature
  - Implicit insertion of floating-point operations considered harmful?
- Units for external code: asserting a type
  - Should be controlled: not a general cast mechanism
- Higher kinds e.g. Consider

```
type Matrix<[<Measure>] 'u, 't> = 't<'u> array array
```

# Conclusion

- Units-of-measure types occupy a “sweet spot” in the space of type systems
  - Type system is easy to understand for novices
  - Typing rules are very simple
  - Types have a simple form (e.g. no constrained polymorphism)
  - Types don’t intrude (there is rarely any need for annotation)
  - Behind the scenes, inference is non-trivial but practical

# Pointers

- F# download: http://msdn.microsoft.com/fsharp
- Blog on units: http://blogs.msdn.com/andrewkennedy
- Thesis and papers: http://research.microsoft.com/~akenn/units