Scala vs Idris

Dependent Types, Now and in the Future

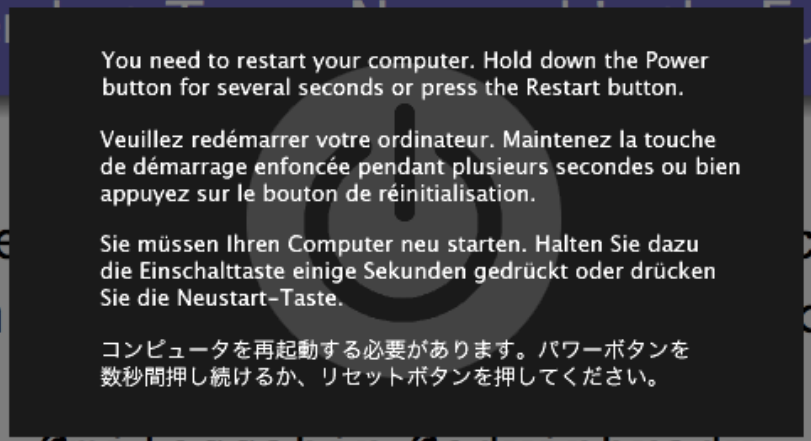
Miles Sabin (miles@milessabin.com)
Edwin Brady (ecb10@st-andrews.ac.uk)

@milessabin @edwinbrady

Strange Loop, September 20th 2013

Scala vs Idris

Depe uture



Mile com)
Edwin c.uk)

@milessabin @edwinbrady

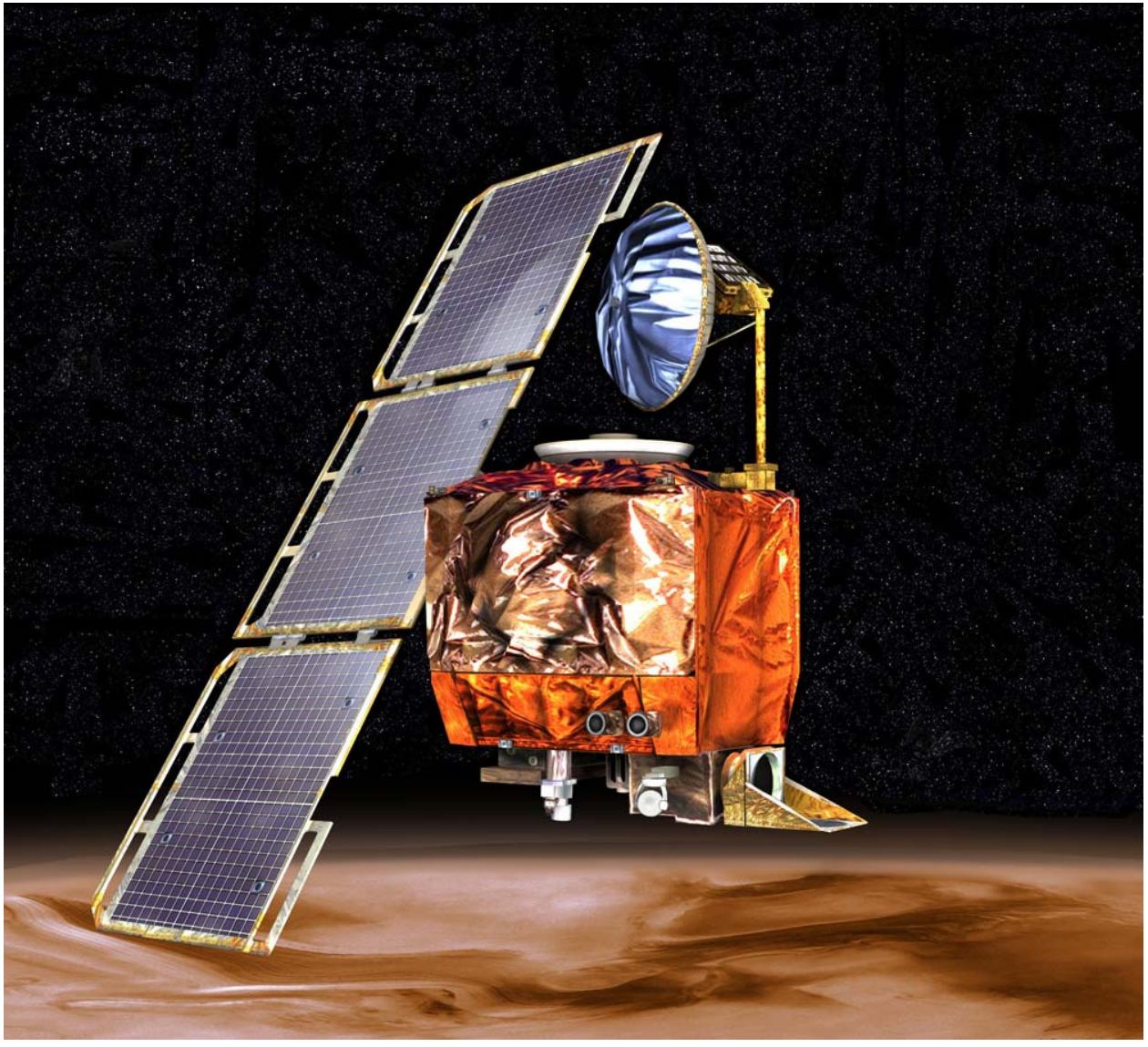
Strange Loop, September 20th 2013

Software Correctness

Software Correctness

By secretlondon123 (Flickr: card reader segfault) [CC-BY-SA-2.0 (http://creativecommons.org/licenses/by-sa/2.0)], via Wikimedia

Software Correctness



Types for Correctness

We can use *type systems* for:

- *Checking* a program has the intended properties
- *Guiding* a programmer towards a correct program
- Building *expressive* and *generic* libraries

In This Talk: Dependent Types

Dependent Types allow types to be predicated on *values*. Two approaches:

In This Talk: Dependent Types

Dependent Types allow types to be predicated on *values*. Two approaches:

- *Idris* (Edwin)
  - Building a language with dependent types from the ground up

In This Talk: Dependent Types

Dependent Types allow types to be predicated on *values*. Two approaches:

- *Idris* (Edwin)
  - Building a language with dependent types from the ground up
- *Scala* (Miles)
  - Exploring the limits of dependently-typed programming in a mainstream language

Unary natural numbers

```
data Nat = Z | S Nat
```

Unary natural numbers

```
data Nat = Z | S Nat
```

Polymorphic lists

```
data List : Type -> Type where
     Nil  : List a
     (::) : a -> List a -> List a
```

Unary natural numbers

```
data Nat = Z | S Nat
```

Polymorphic lists

```
data List : Type -> Type where
     Nil  : List a
     (::) : a -> List a -> List a
```

Vectors — polymorphic lists with length

```
data Vect : Nat -> Type -> Type where
     Nil  : Vect Z a
     (::) : a -> Vect k a -> Vect (S k) a
```

Dependent Types — Examples

Append

```
(++) : Vect m a -> Vect n a -> Vect (m + n) a
(++) []        ys = ys
(++) (x :: xs) ys = x :: xs ++ ys
```

Dependent Types — Examples

Append

```
(++) : Vect m a -> Vect n a -> Vect (m + n) a
(++) []        ys = ys
(++) (x :: xs) ys = x :: xs ++ ys
```

Pairwise addition

```
vAdd : Num a => Vect n a -> Vect n a -> Vect n a
vAdd []        []        = []
vAdd (x :: xs) (y :: ys) = x + y :: vAdd xs ys
```

Dependent Types — Examples

zipWith

```
zipWith : (a -> b -> c) ->
          Vect n a -> Vect n b -> Vect n c
zipWith f []          []      = []
zipWith f (x :: xs) (y :: ys)
                              = f x y :: zipWith f xs ys
```

Types are First Class

Calculate a type

```
adderTy : Nat -> Type
adderTy Z     = Nat
adderTy (S k) = Nat -> adderTy k
```

Types are First Class

Calculate a type

```
adderTy : Nat -> Type
adderTy Z     = Nat
adderTy (S k) = Nat -> adderTy k
```

Use a calculated type

```
adder : (n : Nat) -> (acc : Nat) -> adderTy n
adder Z     acc = acc
adder (S k) acc = \n => adder k (n + acc)
```

Over to Miles. . .

State, Interaction, Resources

Dependent types work very well with *pure* programs. But what about:

- State
- Interaction
- External (e.g. C) libraries

In general, what about *side effects*?

State, Interaction, Resources

Dependent types work very well with *pure* programs. But what about:

- State
- Interaction
- External (e.g. C) libraries

In general, what about *side effects*?

Demo: Managing *side effects* in Idris

Summary

Why are we interested in dependent types?

- *Safety*
  - Programs checked against precise specifications

Summary

Why are we interested in dependent types?

- *Safety*
  - Programs checked against precise specifications
- *Expressivity*
  - Better, more descriptive APIs
  - *Type directed* development

Summary

Why are we interested in dependent types?

- *Safety*
  - Programs checked against precise specifications
- *Expressivity*
  - Better, more descriptive APIs
  - *Type directed* development
  - Type system should be *helping*, not telling you off!

Summary

Why are we interested in dependent types?

- *Safety*
  - Programs checked against precise specifications
- *Expressivity*
  - Better, more descriptive APIs
  - *Type directed* development
  - Type system should be *helping*, not telling you off!
- *Genericity*
  - e.g. program generation

Summary

Why are we interested in dependent types?

- *Safety*
  - Programs checked against precise specifications
- *Expressivity*
  - Better, more descriptive APIs
  - *Type directed* development
  - Type system should be *helping*, not telling you off!
- *Genericity*
  - e.g. program generation
- *Efficiency* (one day...)
  - More precise type information should help the compiler

Scala vs Idris: Judge for yourself

Idris:

- `http://idris-lang.org`
- `cabal install idris`

Scala/Shapeless:

- `https://github.com/milessabin/shapeless`
- `https://github.com/milessabin/strangeloop-2013`

Summary — applications

Dependent types are an active research topic, and we're having lots of fun. Some things we've been working on:

- Concurrency (e.g. verify absence of deadlock)
- Network transport protocols
- Packet formats
- Type-safe web applications
- Scientific programming
- ...