4 Types

“Now! *That* should clear up a few things around here!”

4 Types

Types, Type Errors, Type Systems

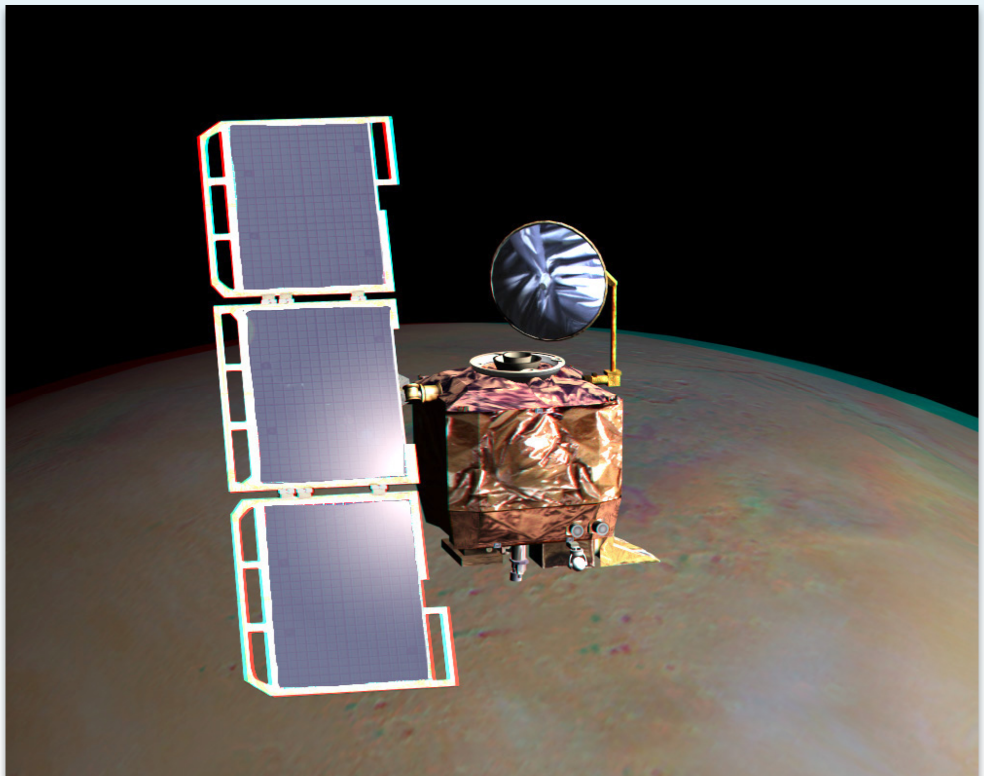
Why Type Checking?

Static vs. Dynamic Typing

Polymorphism

Parametric Polymorphism

Lost ...



mars.jpl.nasa.gov/msp98/orbiter/

No Sugar ...

www.gcn.com/print/17_17/33727-1.html

How to not Excel ...

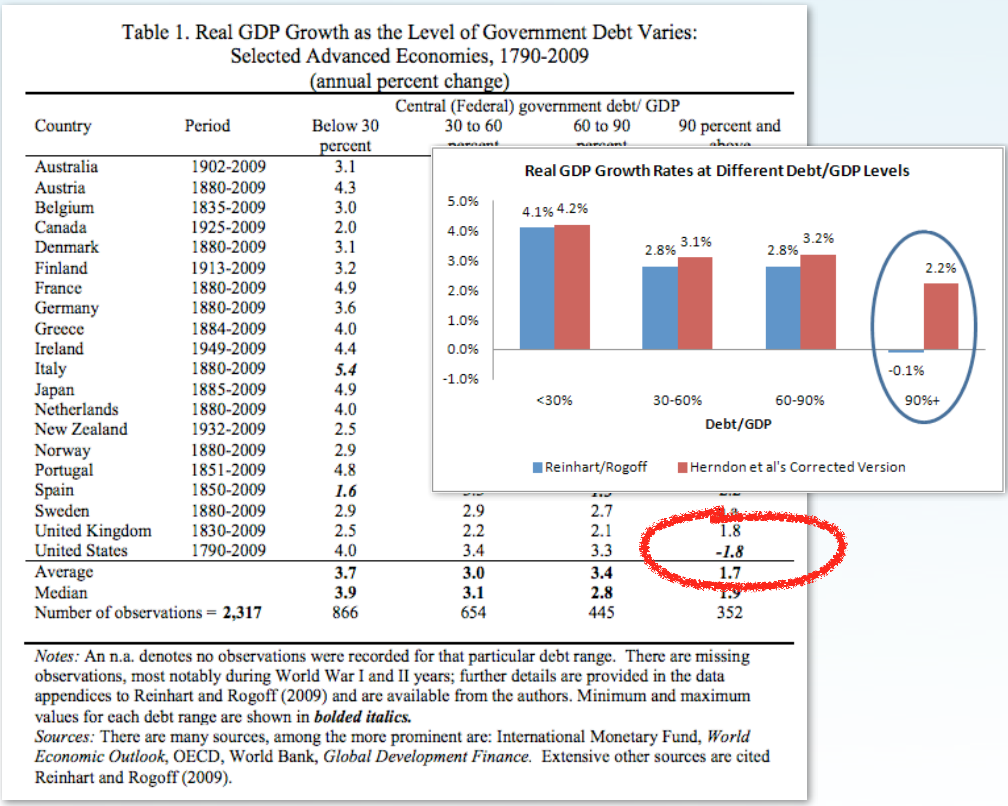
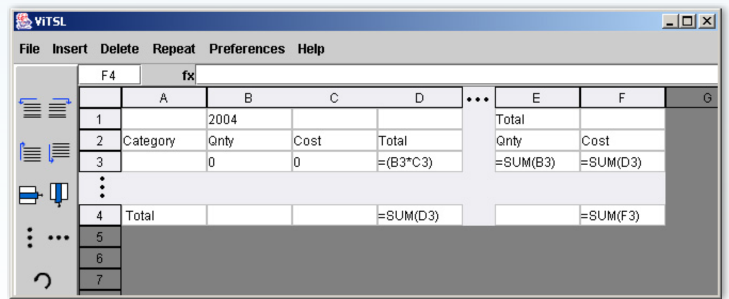


Table 1. Real GDP Growth as the Level of Government Debt Varies: Selected Advanced Economies, 1790-2009 (annual percent change)

Country	Period	Central (Federal) government debt/ GDP: Below 30 percent	30 to 60 percent	60 to 90 percent	90 percent and above
Australia	1902-2009	3.1			
Austria	1880-2009	4.3			
Belgium	1835-2009	3.0			
Canada	1925-2009	2.0			
Denmark	1880-2009	3.1			
Finland	1913-2009	3.2			
France	1880-2009	4.9			
Germany	1880-2009	3.6			
Greece	1884-2009	4.0			
Ireland	1949-2009	4.4			
Italy	1880-2009	***5.4***			
Japan	1885-2009	4.9			
Netherlands	1880-2009	4.0			
New Zealand	1932-2009	2.5			
Norway	1880-2009	2.9			
Portugal	1851-2009	4.8			
Spain	1850-2009	***1.6***			
Sweden	1880-2009	2.9	2.9	2.7	[illegible]
United Kingdom	1830-2009	2.5	2.2	2.1	1.8
United States	1790-2009	4.0	3.4	3.3	***-1.8***
Average		**3.7**	**3.0**	**3.4**	**1.7**
Median		**3.9**	**3.1**	**2.8**	**1.9**
Number of observations = **2,317**		866	654	445	352

Notes: An n.a. denotes no observations were recorded for that particular debt range. There are missing observations, most notably during World War I and II years; further details are provided in the data appendices to Reinhart and Rogoff (2009) and are available from the authors. Minimum and maximum values for each debt range are shown in ***bolded italics.***
Sources: There are many sources, among the more prominent are: International Monetary Fund, *World Economic Outlook*, OECD, World Bank, *Global Development Finance*. Extensive other sources are cited Reinhart and Rogoff (2009).

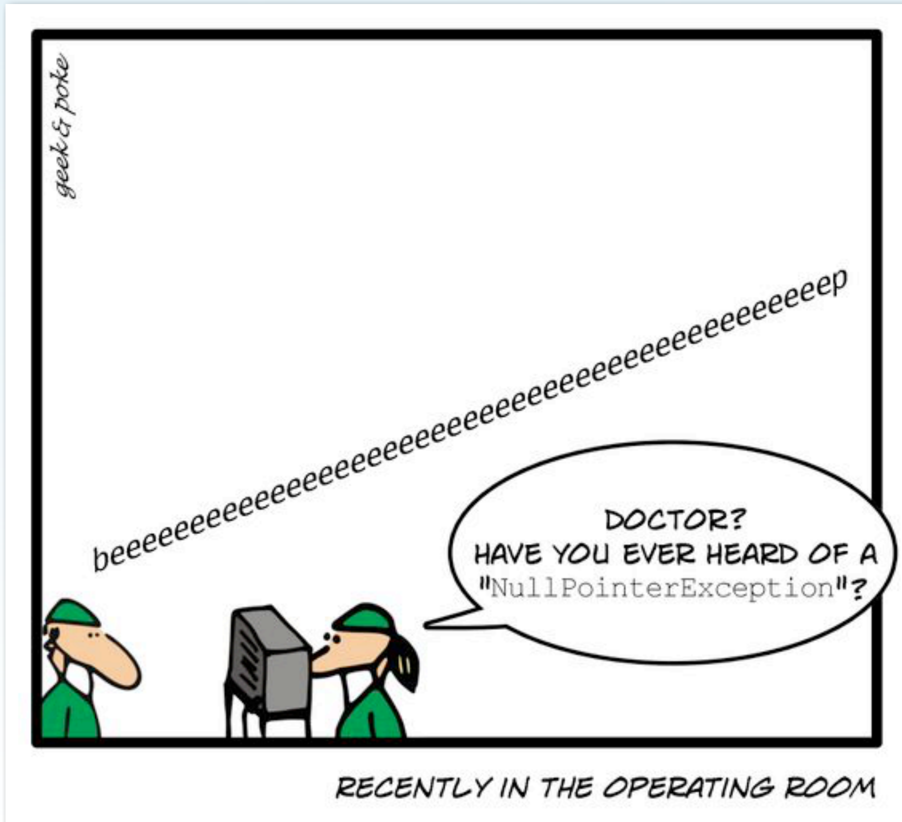
eecs.oregonstate.edu/~erwig/UCheck/



eecs.oregonstate.edu/~erwig/Gencel/

Wednesday, May 1, 13

Your Stay in the Hospital ...



More Motivation ...

Video clip

More examples of non-CS use of types ... next class

What is a Type (System) ?

- *Type*
  Collection of PL elements that share the same behavior
- *Type System*
  Formal system to determine the types of PL elements and to prove the absence of type errors
- *Purpose of Type Systems*
  Prevent programming errors, i.e. type errors

Type Errors

- *Type Error*
  Illegal combination of PL elements
  (typically: applying an operation to a value of the wrong type)
- *Why are type errors bad?*
  Lead to program crashes
  Cause incorrect computations

Why Types are a Good Thing

(1) Types provide *precise documentation* of programs

(2) Types *summarize* a program on an abstract level

(3) Type correctness means *partial correctness* of programs;
a type checker delivers partial *correctness proofs*

(4) Type systems can *prevent* runtime *errors* and
can save a lot of debugging

(5) Type information can be exploited for *optimization*

Things to Know About Type Systems

(1) Notion of Type Safety

(2) Strong vs. Weak Typing

(3) Static vs. Dynamic Typing

(4) Approximation & Undecidability of Static Typing

(5) Type Checking vs. Type Inference

(6) Polymorphism (*parametric*, subtype, ad hoc)

Example: Expression Language with 2 Types

Expr2.hs

Type Safety

Type Safety
A programming language is called *type safe* if all type errors are detected

Type Safe Languages
Lisp (*ridiculous type system*)
Java
Haskell
Expr + eval
Expr + evalDynTC

Unsafe Languages
(type casts, pointers)
C
C++

Exercises

(1) Implement an unsafe `eval` function for the language `Expr`

(a) Use `Int` as the semantic domain

(b) Map boolean values to `0` and `1`

```
data Expr = N Int
          | Plus Expr Expr
          | Equal Expr Expr
          | Not Expr
```

(2) Evaluate unsafe expressions

Expr2Unsafe.hs

Strong vs. Weak Typing

Strong Typing
Each value has one precisely determined type

Weak Typing
Values can be interpreted in different types (e.g. "17" can be used as a string or number, or 0 can be used as a number or boolean)

In practice: Only strongly typed languages are safe (although strong typing does not guarantee safety)

A Type Checker for the Expression Language

Expr2.hs

TypeCheck.hs

Dynamic vs. Static Typing

Dynamic Typing
Types are checked during runtime

Static Typing
Types and type errors are found during compile time

Statically Typed	*Dynamically Typed*
Haskell	Lisp
(Java)	
Expr + evalStatTC	Expr + evalDynTC

Static Typing is Conservative

What is the type of the following expression?

```
if 3>4 then "hello" else 17
```

Under dynamic typing: Int
Under static typing: type error

How about:

```
f x = if test x then x+1 else False
```

Under dynamic typing: ?
Under static typing: type error

Exercises

(1) What is the type of the following function under static and dynamic typing?

```
f x = if not x then x+1 else x
```

(2) What is the type of the following function under static and dynamic typing?

```
f x = f (x+1) * 2
```

Undecidability of Static Typing

```
mayLoop :: Int -> Bool
f x = if mayLoop x then x+1 else not x
```

`f` is *type correct* if `mayLoop x` yields `True`

`f` contains a *type error* if `mayLoop x` yields `False`

Since `mayLoop x` might not terminate, we cannot determinate the value statically because of the undecidability of the halting problem.

Static typing *approximates* by assuming a type error when type correctness cannot be shown

Advantages & Disadvantages of Static/Dynamic Typing

	Advantage	*Disadvantage*
Static Typing	prevents type errors smaller & faster code early error detection (saves debugging)	rejects some o.k. programs
Dynamic Typing	prevents type errors faster compilation (& development?)	slower execution released programs may stop unexpectedly with type errors

A Type Checker for a Geometric Language

TypedGeoLang.hs

Exercises

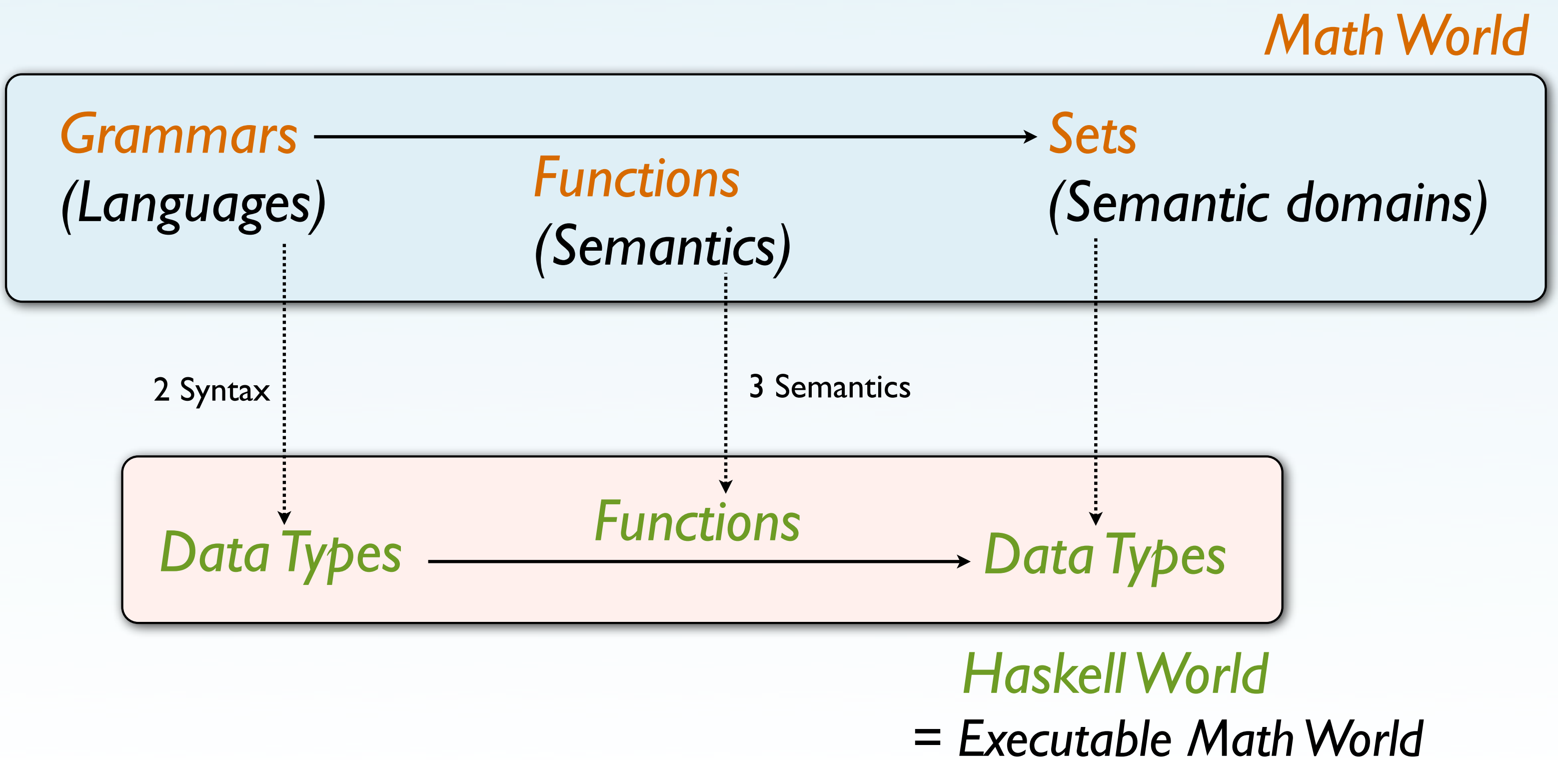
(1) Extend the geometric language by a predicate `Inside` that determines whether one object is inside of another

(2) Extend the data type `Type` and the type checker `tc` to typecheck the operation `Inside`

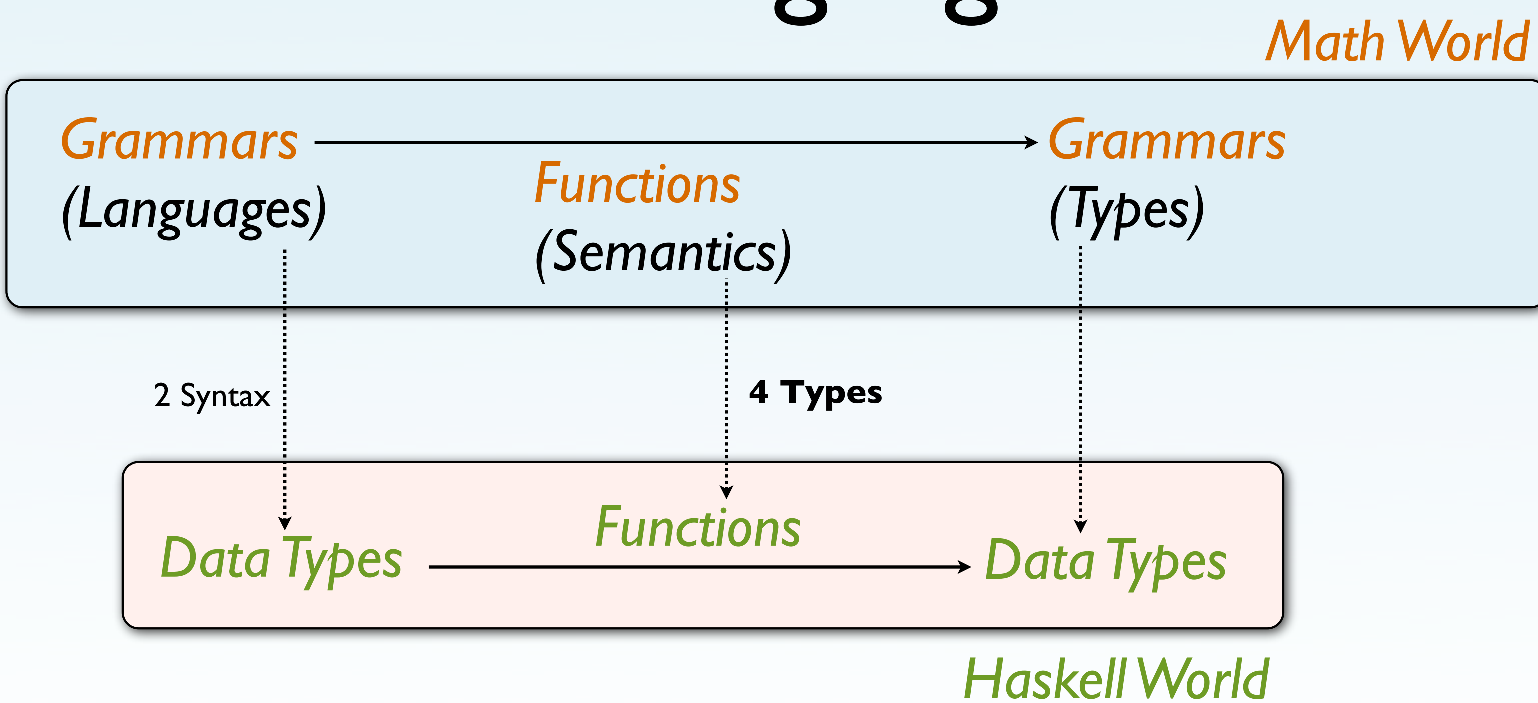
A Type Checker for Arithmetic Language with Pairs

ExprPair.hs
ExprNPair.hs

Haskell as a Mathematical Metalanguage



Haskell as a Mathematical Metalanguage



Typing = ***Static Semantics***
Semantics = *Dynamic Semantics*

Polymorphism

A value (function, method, ...) is *polymorphic* if it has more than one type

Different forms of polymorphism can be distinguished based on:

(a) relationship between types

(b) implementation of functions

Forms of Polymorphism

Parametric Polymorphism
(a) All types match a common "type pattern"
(b) One implementation, i.e., there is only one function

Ad Hoc Polymorphism (aka *Overloading*)
(a) Types are unrelated
(b) Implementation differs for each type, i.e., different functions are referred to by the same name

Subtype Polymorphism
(a) Types are related by a subtype relation
(b) One implementation (methods can be applied to objects of any subtype)

Parametric Polymorphism

Haskell demo