

lo elio. digamole el meior.  
q̄ caye en precio. del sc̄o confessor.  
to se aduzir. este ciego lazrado.  
a casa del monge. de fuso ementado.  
creye bien afirmes. estaua syuzado  
rie desta coyta. por ell. terminad  
fue ala puerta. de san̄o sabastia  
quiso el meiano. poder uino ni pan.  
dize ay poder. por señor san̄o. Al.  
prenda cordio. de este mi affan.  
ne alla do yaces yo ati vin buscar.  
tu oranda. ami alla tornar.  
no. yo non podria. partirme dell legar.  
tu me mandes. ser otorn  
re delos lazrados. deña ne usitas.  
sobre mi tu mano. signa me  
q̄ yo pudiesse. la tu mano besar.  
da esta coyta. cuydaria sanar  
adre beneyto. bien entro do estaua.  
los ayudados. q̄ este ciego daua.  
e p̄ quanto le. q̄l cosa demandaua.  
el a humer. ca el non cobdianua.  
me f̄a demia. en or tal. l un

**DSL**

Chol con el ysoy. del agua salada.  
Consigno li los oios con la cruz consagrada.  
La dolencia coyta. fue luego amansada.  
La burra q̄ yda. fue toda recobrada.  
Los amigos y señores.  
de diuersos colores.

V nos de ceguidad. al de ḡues dolores.  
Mas de todo biē sano. rendie adios lodores.

**P**oro ei padre sc̄o. amigo ue tu uia.  
O raderel. q̄ uas con meiora.  
C uria te q̄ no peques. y no fagas folia.  
E a seia en tu tidio. fages recadia.

**M**uchos son los mirados. q̄ dell pad̄ sabemos.  
el p̄mel vs unos o oymos. los otros q̄ leemos.

**E**ndubda. ei panim. en q̄l enpegaremos.  
uo q̄ Has aq̄l parte. q̄ sea. a dehiar aturemos.

**S**aca Desta lizon los est. q̄ero los fer esquos.

**E**n su Desir uno y mebre uos. miētre fuerdes uiuos.  
uida. Como gano la gra. q̄ saca los catiuos.

Por ond de luengas tierras. le enbia bodigos.

**Q**uā en essi tiempo. los motos muy ueyinos.



DSL

=

Domain Specific  
Language



# DSL Approach

1. develop a language
2. solve problems with that language
3. \$\$\$



# Types of DSLs

- **Stand-alone DSL**  
HTML, Verilog, SQL, YACC, GraphViz, ...
- **Embedded DSL (EDSL)**  
embedded in a host language like  
Haskell as library  
(but also in Scala, Groovy, ...)



# Geo-Server



Haskell vs. Ada vs. C++ vs. Awk vs. ...  
An Experiment in Software Prototyping Productivity

Paul Hudak, Mark P. Jones



# The Setup

- US Navy Experiment
- Study suitability of languages for rapid prototyping
- Languages: Haskell, Ada, Ada9X, C++, Awk, Rapide, Griffin, Proteus, Relational Lisp
- I expert programmer, small project







# Problem

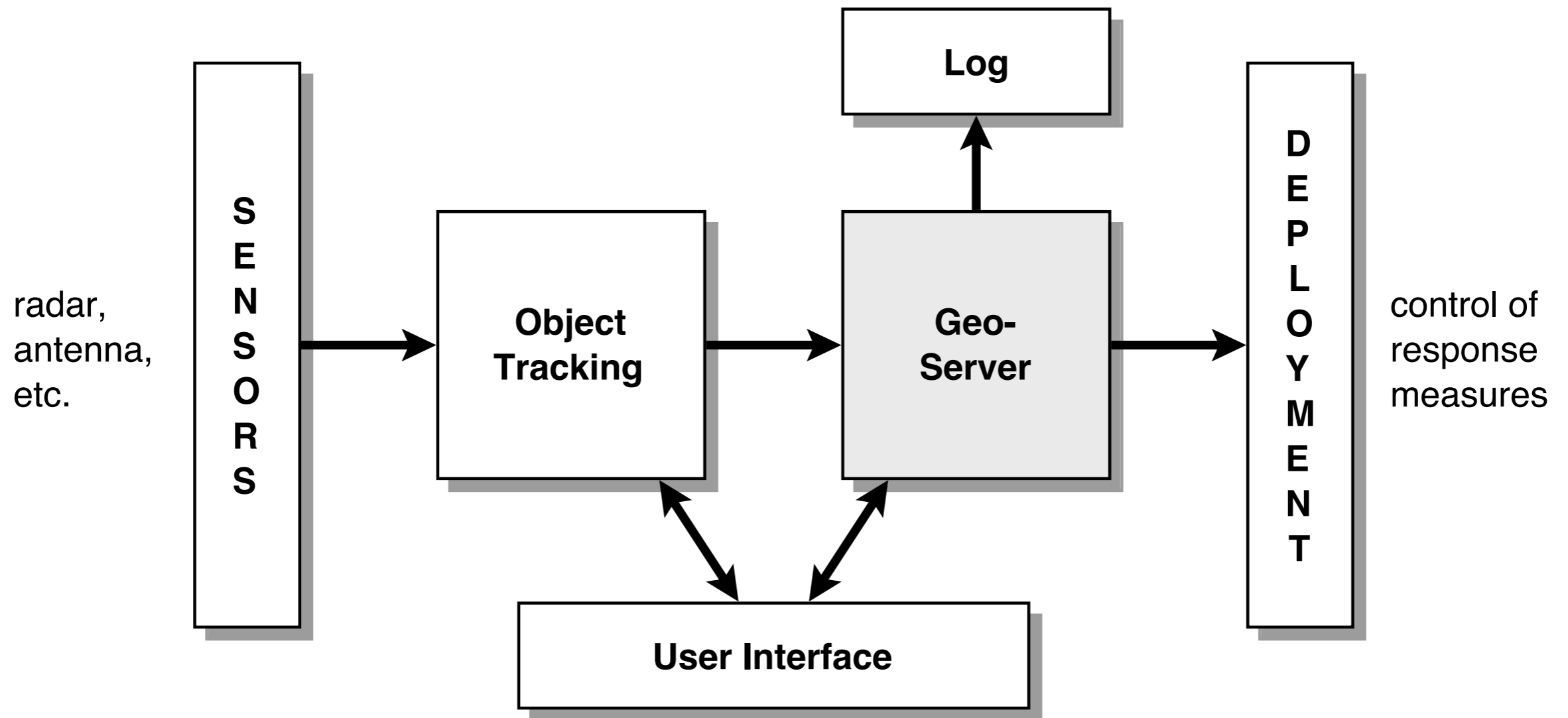


Figure 1: Simplified Aegis Weapons Systems Diagram



# Geo-Server Input

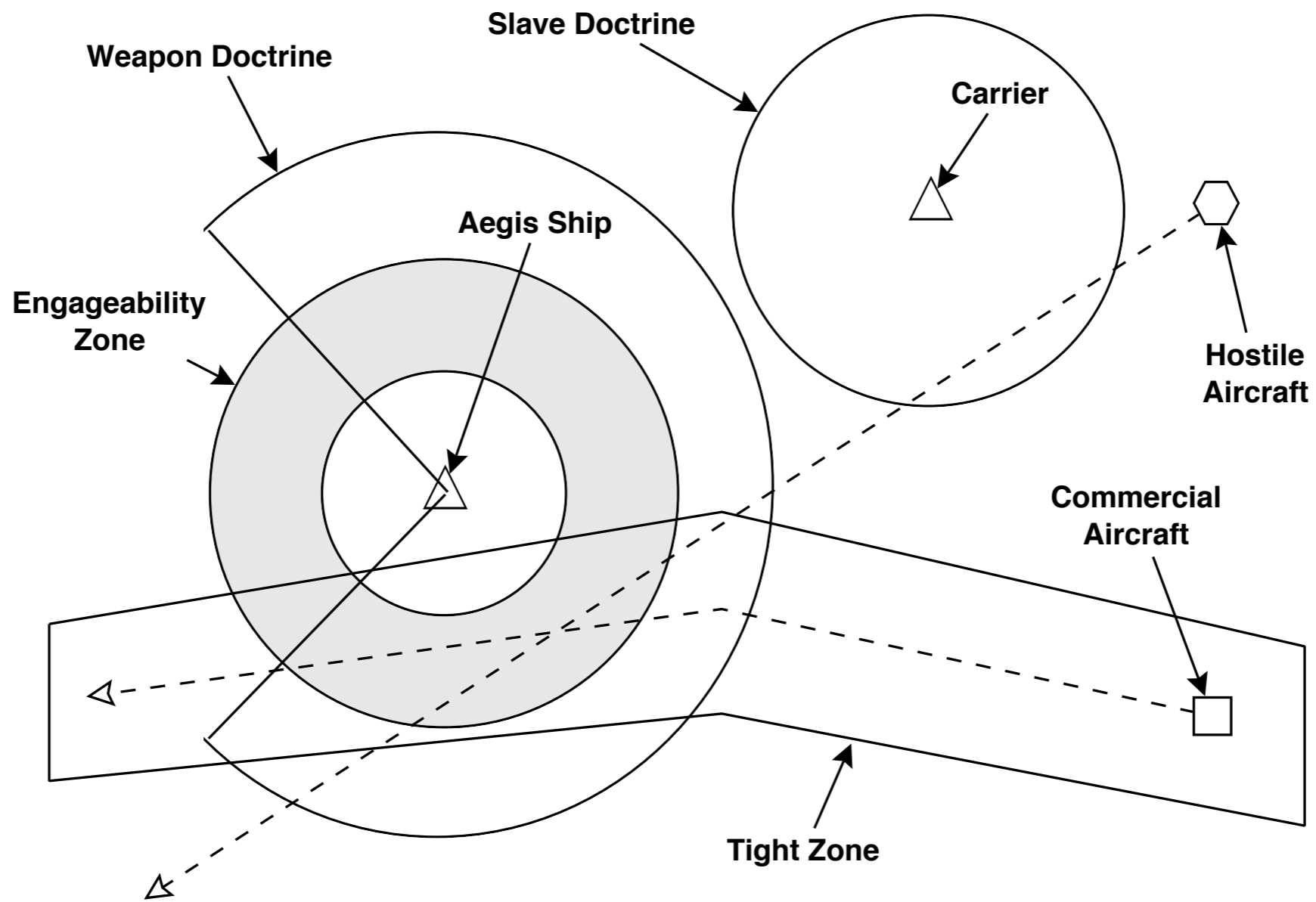


Figure 2: Geo-Server Input Data



# Geo-Server Output

Time 0.0:

commercial aircraft: (38.0,25.0)

-- In tight zone

hostile craft: (258.0,183.0)

Time 20.0:

commercial aircraft: (58.0,30.0)

-- In tight zone

hostile craft: (239.0,164.0)

Time 40.0:

commercial aircraft: (100.0,43.0)

-- In engageability zone

-- In tight zone

hostile craft: (210.0,136.0)

-- In carrier slave doctrine



# Haskell Solution

```
type Region
```

```
inRegion :: Point → Region → Bool
```

```
circle :: Radius → Region
```

```
outside :: Region → Region
```

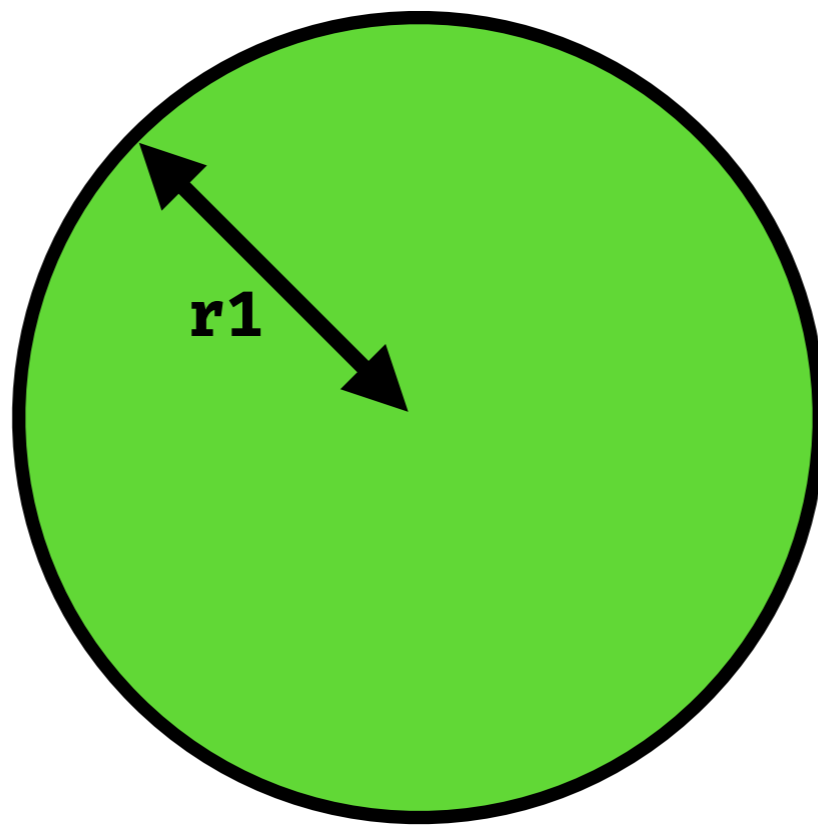
```
(/\) :: Region → Region → Region
```

```
annulus :: Radius → Radius → Region
```

```
annulus r1 r2 = outside (circle r1) /\  
                    circle r2
```

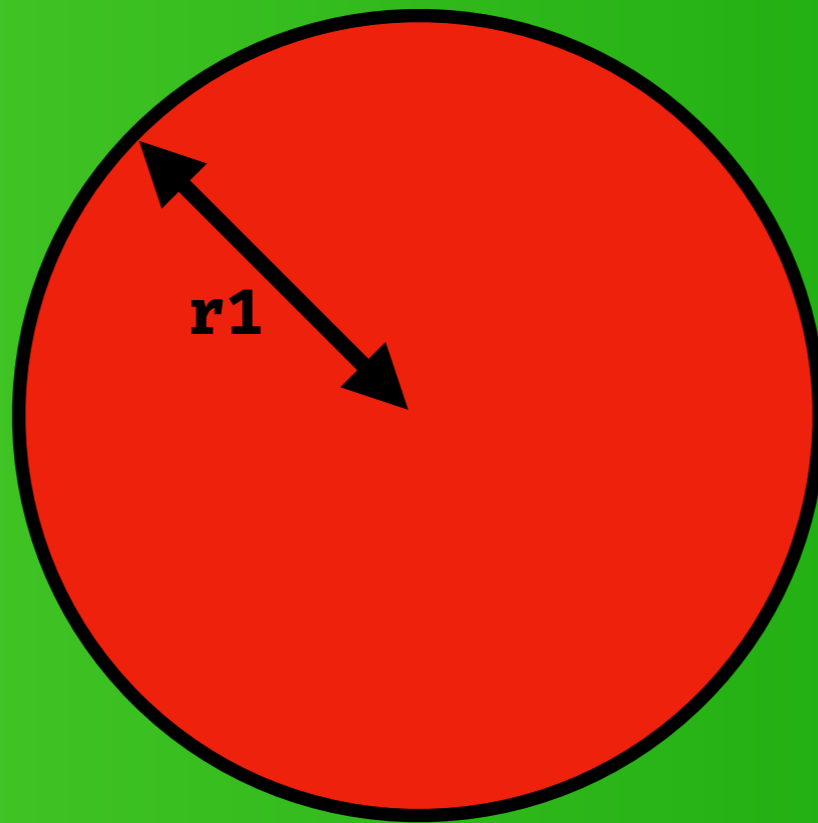


annulus r1 r2 =  
circle r1

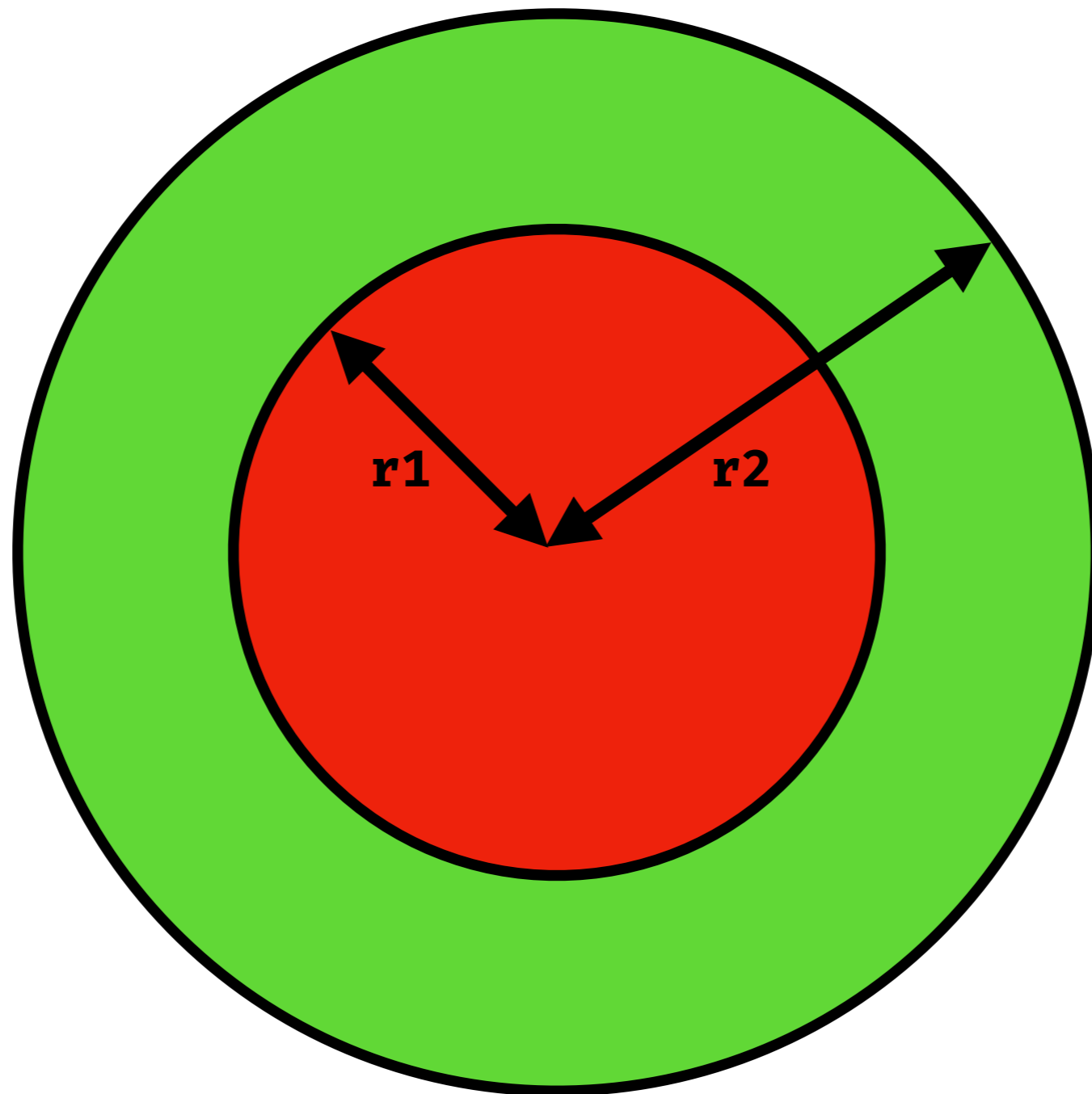




```
annulus r1 r2 =  
  outside (circle r1)
```



```
annulus r1 r2 =  
  outside (circle r1)  
  /\ circle r2
```





# Implementation

## Shallow embedding

- ★ implement regions as Haskell functions
- ★ semantics: `inRegion`
- ★ no “interpretative overhead”

# Shallow Embedding

```
type Region func(p Point) bool {  
    p `inRegion`  
    return distance((0,0),p) ≤ d  
}
```

```
circle d = \p → distance (0,0) p =< d
```

```
outside r = \p → not (r p)
```

```
r1 /\ r2 = \p → r1 p && r2 p
```



# Study Results

Language	Lines of code	Lines of documentation	Development time (hours)
(1) Haskell	85	465	10
(2) Ada	767	714	23
(3) Ada9X	800	200	28
(4) C++	1105	130	–
(5) Awk/Nawk	250	150	–
(6) Rapide	157	0	54
(7) Griffin	251	0	34
(8) Proteus	293	79	26
(9) Relational Lisp	274	12	3
(10) Haskell	156	112	8



The background features a close-up of Euro banknotes, showing the intricate security patterns and the yellow stars. Overlaid on this is a complex, three-dimensional wireframe architectural structure, possibly representing a building or a financial framework. The overall color palette is dominated by the blues and greens of the currency, with the yellow stars providing a contrasting accent.

# Financial Contracts

**Composing contracts: an adventure in financial  
engineering**

Simon Peyton Jones, Jean-Marc Eber, Julian Seward



# Example Contract

The owner of the contract  
has the right to choose on June 30 2000  
between:

$D_1$  Both of:

$D_{11}$  Receive £100 on 29 Jan 2001.

$D_{12}$  Pay £105 on 1 Feb 2002.

$D_2$  An option exercisable on 15 Dec 2000 to choose one of:

$D_{21}$  Both of:

$D_{211}$  Receive £100 on 29 Jan 2001.

$D_{212}$  Pay £106 on 1 Feb 2002.

$D_{22}$  Both of:

$D_{221}$  Receive £100 on 29 Jan 2001.

$D_{222}$  Pay £112 on 1 Feb 2003.



# Problems

- \$ Inaccurate, non-uniform language
- \$ Analysis and manipulation of contracts
  - £ calculate worth
  - £ simulate

# Simple Contract

```
-- receive £100 on 13/02/2003
```

```
c1 :: Contract
```

```
c1 = zcb t1 100 GBP
```

```
-- zero coupon bond
```

```
zcb :: Date → Double → Currency → Contract
```

```
mkDate :: String → Date
```

```
t1 :: Date
```

```
t1 = mkDate "0800GMT 13 Feb 2003"
```

# Composing Contracts

**and** :: Contract → Contract → Contract

$c_2, c_3$  :: Contract

$c_2 = \text{zcb } t_2 \text{ 200 GBP}$

$c_3 = c_1 \text{ `and` } c_2$

**give** :: Contract → Contract

**andGive** :: Contract → Contract → Contract

**andGive** c d = c `and` give d

$c_4 = c_1 \text{ `andGive` } c_2$



# Haskell in Industry



Finance



ABN·AMRO

BARCLAYS  
CAPITAL



Deutsche Bank

CREDIT SUISSE



Telecom

QUALCOMM®



at&t



Alcatel·Lucent



Many Others

Google intel®

The  
New York  
Times



# Summary

# Summary

◆ **D**omain **S**pecific

◆ **L**anguage

◆ **E**Embedded



**Liked this material?**

*Functional Programming in Industry*  
<https://dtai.cs.kuleuven.be/events/fpcourse/>

# Deep Embedding

```
type Region = R
```

```
data R = Circle Radius  
      | Outside R  
      | Intersect R R
```

```
circle    = Circle  
outside   = Outside  
(/\)     = Intersect
```

```
p `inRegion` (Circle d)  
= distance (0,0) p =< d  
p `inRegion` (Outside r)  
= not (p `inRegion` r)  
p `inRegion` (Intersect r1 r2)  
= (p `inRegion` r1) && (p `inRegion` r2)
```

# Smart Constructors

```
type Region = R  
data R = Circle Radius  
      | Outside R  
      | Intersect R R
```

```
circle = opt . Circle  
outside = opt . Outside  
(/\) = ...
```

```
opt :: R → R  
opt (Intersect (Circle d1) (Circle d2))  
    = Circle (max d1 d2)  
...  
opt r = r
```