OOPSLA'22

THE UNIVERSITY OF HONG KONG
111TH ANNIVERSARY
香港大學111周年

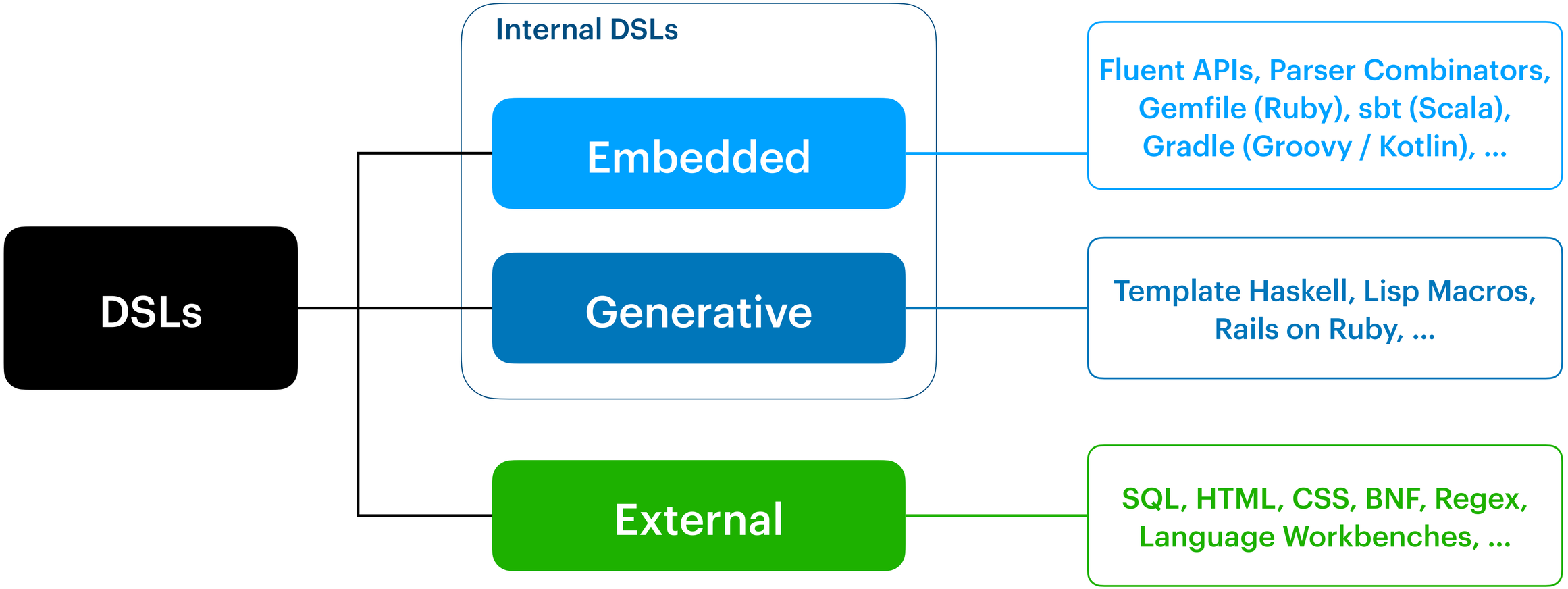
Compositional Embeddings

of Domain-Specific Languages

Yaozhu Sun, Utkarsh Dhandhania, Bruno C. d. S. Oliveira

10 December 2022

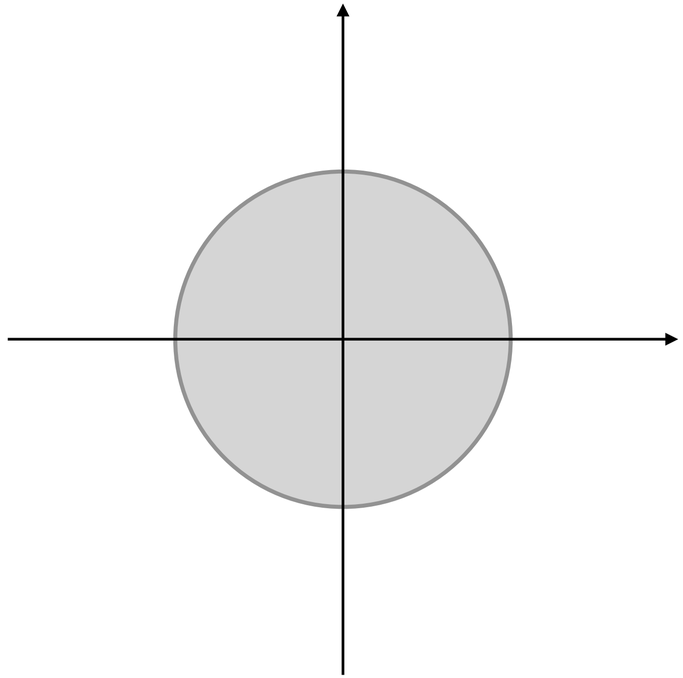
Taxonomy



Debasish Ghosh. *DSLs in Action*. Manning Publications Co. 2010.

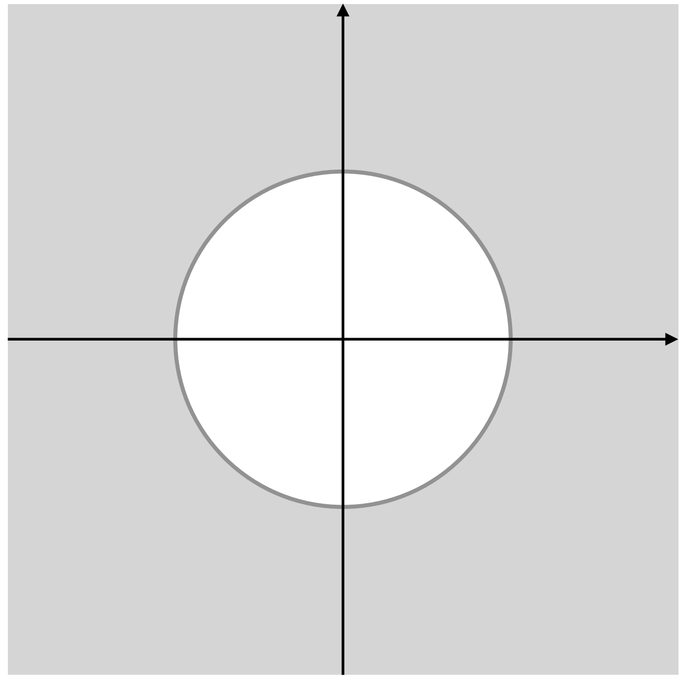
An Embedded DSL for Regions

- `(circle r)` a circular region of radius `r`



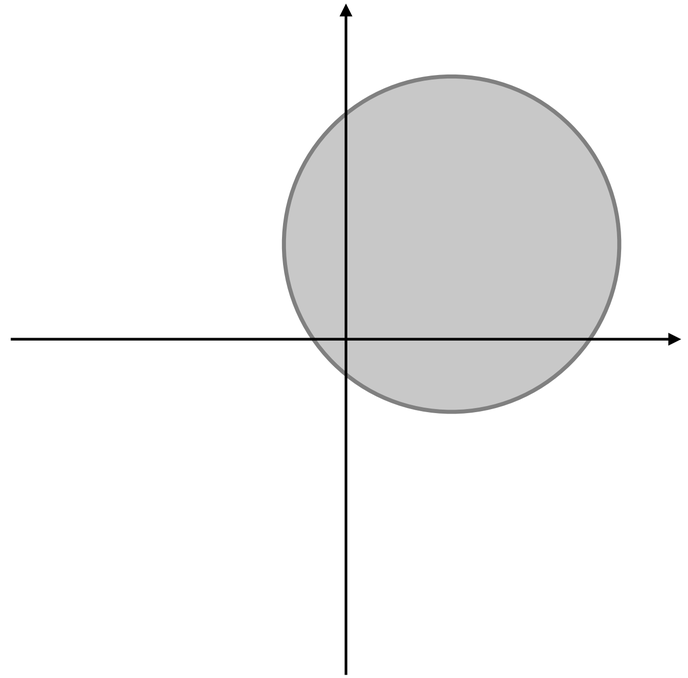
An Embedded DSL for Regions

- `(circle r)` a circular region of radius `r`
- `(outside a)` the complement of region a



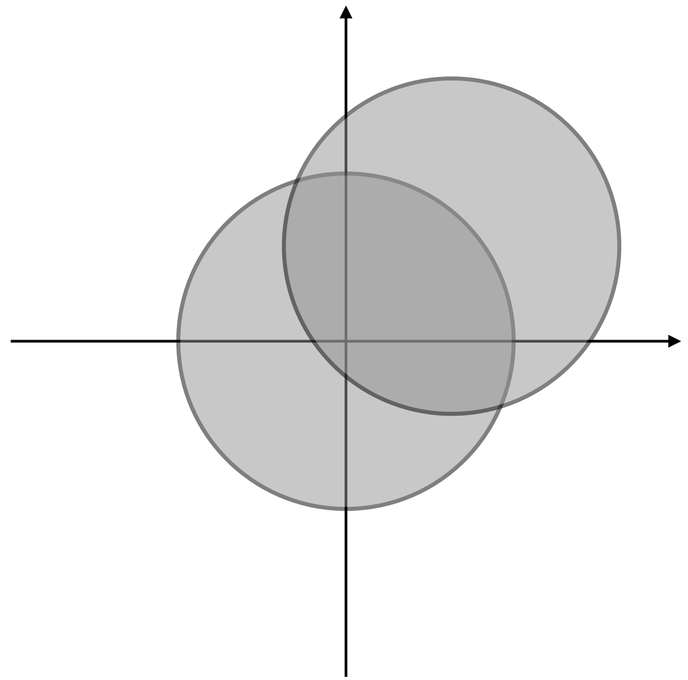
An Embedded DSL for Regions

- `(circle r)` a circular region of radius `r`
- `(outside a)` the complement of region a
- `(translate v a)` translate a by v



An Embedded DSL for Regions

- `(circle r)` a circular region of radius `r`
- `(outside a)` the complement of region a
- `(translate v a)` translate a by v
- `(union a b)` the union of a and b
- `(intersection a b)` the intersection of a and b



Shallow

```
type Region = Int
```

syntactic size
(number of AST nodes)

```
circle    :: Double → Region
circle      _   =  1
outside   :: Region → Region
outside     a   =  a + 1
union     :: Region → Region → Region
union     a b   =  a + b + 1
intersect :: Region → Region → Region
intersect a b   =  a + b + 1
translate :: Vector → Region → Region
translate _ a   =  a + 1
```

Shallow vs. Deep

syntactic size
(number of AST nodes)

```
type Region = Int

circle    :: Double → Region
circle      _   =  1
outside   :: Region → Region
outside     a   =  a + 1
union     :: Region → Region → Region
union     a b   =  a + b + 1
intersect :: Region → Region → Region
intersect a b   =  a + b + 1
translate :: Vector → Region → Region
translate _ a   =  a + 1
```

```
data Region where
  Circle    :: Double → Region
  Outside   :: Region → Region
  Union     :: Region → Region → Region
  Intersect :: Region → Region → Region
  Translate :: Vector → Region → Region

size :: Region → Int
size (Circle      _) = 1
size (Outside     a) = size a + 1
size (Union     a b) = size a + size b + 1
size (Intersect a b) = size a + size b + 1
size (Translate _ a) = size a + 1
```

Linguistic Reuse

(Shallow)

```
circles = go 20 (2 ** 18)
  where go :: Int → Double → Region
        go 0 offset = circle 1
        go n offset = let shared = go (n - 1) (offset / 2)
                      in union (translate Vector { x = -offset, y = 0 } shared)
                               (translate Vector { x =  offset, y = 0 } shared)
```

duplicate computation avoided (very fast!)

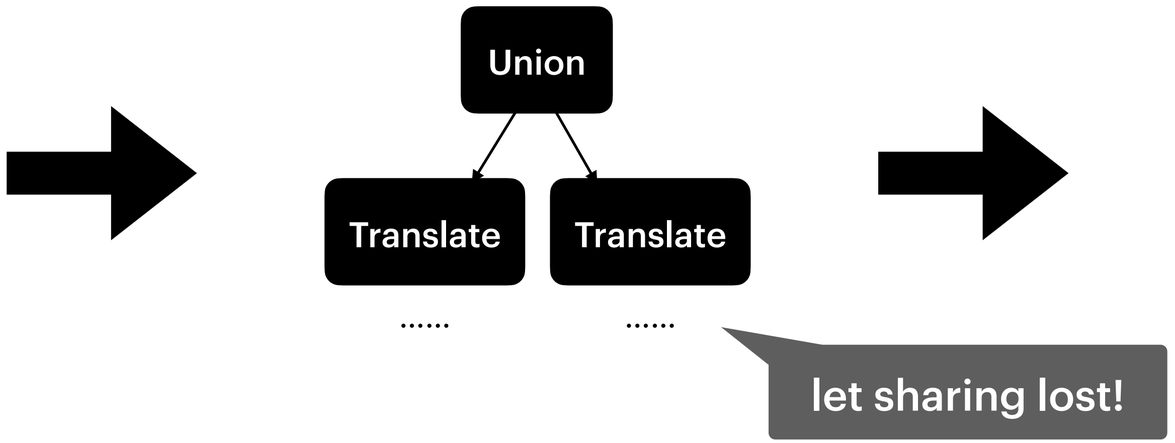
Linguistic Reuse

(Shallow)

```
circles = go 20 (2 ** 18)
  where go :: Int → Double → Region
        go 0 offset = circle 1
        go n offset = let shared = go (n - 1) (offset / 2)
                      in union (translate Vector { x = -offset, y = 0 } shared)
                               (translate Vector { x =  offset, y = 0 } shared)
```

duplicate computation avoided (very fast!)

(Deep)
AST



duplicate computation (very slow!)

The Expression Problem

(Shallow) **new language constructs** (Deep) 🙁

```
univ :: Region
univ = 1
```

```
empty :: Region
empty = 1
```

```
scale :: Vector → Region → Region
scale _ a = a + 1
```

The Expression Problem

(Shallow) **new language constructs** (Deep) 😕

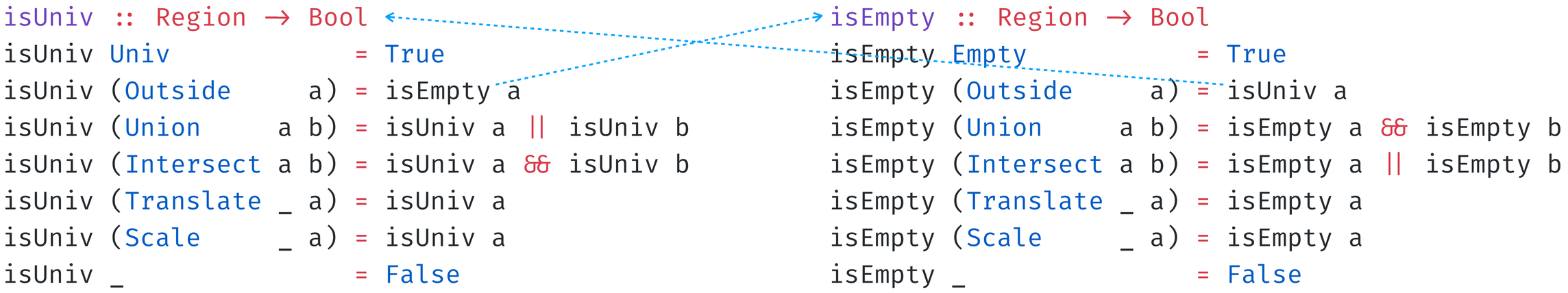
```
univ :: Region
univ = 1

empty :: Region
empty = 1

scale :: Vector → Region → Region
scale _ a = a + 1
```

(Deep) **new semantic interpretations** (Shallow) 😕

```
contains :: Region → Vector → Bool
Circle                  r `contains` p = p.x ** 2 + p.y ** 2 ⩽ r ** 2
Outside                 a `contains` p = not (a `contains` p)
Union                 a b `contains` p = a `contains` p || b `contains` p
Intersect             a b `contains` p = a `contains` p && b `contains` p
Translate Vector {..} a `contains` p = a `contains` Vector { x = p.x - x, y = p.y - y }
```

Dependencies without Code Duplication

(Deep)



```
isUniv :: Region → Bool
isUniv Univ            = True
isUniv (Outside     a) = isEmpty a
isUniv (Union     a b) = isUniv a || isUniv b
isUniv (Intersect a b) = isUniv a && isUniv b
isUniv (Translate _ a) = isUniv a
isUniv (Scale     _ a) = isUniv a
isUniv _               = False
```

```
isEmpty :: Region → Bool
isEmpty Empty            = True
isEmpty (Outside     a) = isUniv a
isEmpty (Union     a b) = isEmpty a && isEmpty b
isEmpty (Intersect a b) = isEmpty a || isEmpty b
isEmpty (Translate _ a) = isEmpty a
isEmpty (Scale     _ a) = isEmpty a
isEmpty _               = False
```

Dependencies without Code Duplication

(Deep)

```
isUniv :: Region → Bool
isUniv Univ              = True
isUniv (Outside     a)   = isEmpty a
isUniv (Union     a b)   = isUniv a || isUniv b
isUniv (Intersect a b)   = isUniv a && isUniv b
isUniv (Translate _ a)   = isUniv a
isUniv (Scale     _ a)   = isUniv a
isUniv _                 = False
```

```
isEmpty :: Region → Bool
isEmpty Empty              = True
isEmpty (Outside     a)    = isUniv a
isEmpty (Union     a b)    = isEmpty a && isEmpty b
isEmpty (Intersect a b)    = isEmpty a || isEmpty b
isEmpty (Translate _ a)    = isEmpty a
isEmpty (Scale     _ a)    = isEmpty a
isEmpty _                  = False
```

(Shallow) non-modular workaround (code duplication needed)

(isUniv, isEmpty)

```
type Region = (Bool, Bool)

univ      = (True,  False)
empty     = (False, True)
circle  _ = (False, False)
```

isEmpty a　　isUniv a

```
outside a     = (snd a,                fst a)
union     a b = (fst a || fst b,       snd a && snd b)
intersect a b = (fst a && fst b,       snd a || snd b)
translate _ a = (fst a,                snd a)
scale     _ a = (fst a,                snd a)
```

The 3rd Dimension of the EP

	FP	OOP	Poly.	Comp.
New constructs	○	●	●	●
New functions	●	○	●	●
Dependencies	●	●	○	●

● no code duplication ○ code duplication needed

Poly. = Polymorphic embeddings, tagless-final embeddings, and other modern embeddings
Comp. = Compositional embeddings

“The holy grail is to combine the advantages of shallow and deep in a single implementation.”

— Josef Svenningsson and Emil Axelsson

Compositional Programming

Compositional Embeddings

(written in the CP language)

```
type Vector = { x : Double; y : Double };

type HudakSig<Region> = {
  Circle    : Double → Region;
  Outside   : Region → Region;
  Union     : Region → Region → Region;
  Intersect : Region → Region → Region;
  Translate : Vector → Region → Region;
};
```

sort

constructors

```
data Region where
  Circle    :: Double → Region
  Outside   :: Region → Region
  Union     :: Region → Region → Region
  Intersect :: Region → Region → Region
  Translate :: Vector → Region → Region
```

Compositional Embeddings

(written in the CP language)

```
type Vector = { x : Double, y : Double };

type HudakSig<Region> = {
  Circle    : Double → Region;
  Outside   : Region → Region;
  Union     : Region → Region → Region;
  Intersect : Region → Region → Region;
  Translate : Vector → Region → Region;
};
```

sort

constructors

```
type Size = { size : Int };

sz = trait implements HudakSig<Size> ⇒ {
  (Circle      _).size = 1;
  (Outside     a).size = a.size + 1;
  (Union     a b).size = a.size + b.size + 1;
  (Intersect a b).size = a.size + b.size + 1;
  (Translate _ a).size = a.size + 1;
};
```

instantiates sort Region

method patterns

```
data Region where
  Circle    :: Double → Region
  Outside   :: Region → Region
  Union     :: Region → Region → Region
  Intersect :: Region → Region → Region
  Translate :: Vector → Region → Region

size :: Region → Int
size (Circle      _) = 1
size (Outside     a) = size a + 1
size (Union     a b) = size a + size b + 1
size (Intersect a b) = size a + size b + 1
size (Translate _ a) = size a + 1
```

Linguistic Reuse

```
sharing Region = trait [self : RegionSig<Region>] ⇒ {
  circles = letrec go (n:Int) (offset:Double) : Region =
              if n == 0 then Outside (Outside (Circle 1.0))
              else let shared = go (n-1) (offset/2.0)
                   in Union (Translate { x = -offset; y = 0.0 } shared)
                            (Translate { x =  offset; y = 0.0 } shared)
            in let n = 20 in go n (pow 2.0 (n-2));
};
```

duplicate computation avoided (very fast!)

New Constructs & Interpretations

```
type HoferSig<Region> = {
  Univ  : Region;
  Empty : Region;
  Scale : Vector → Region → Region;
};

sz' = trait implements HoferSig<Size> ⇒ {
  (Univ     ).size = 1;
  (Empty    ).size = 1;
  (Scale _ a).size = a.size + 1;
};


type RegionSig<Region> =
  HudakSig<Region> & HoferSig<Region>;
```

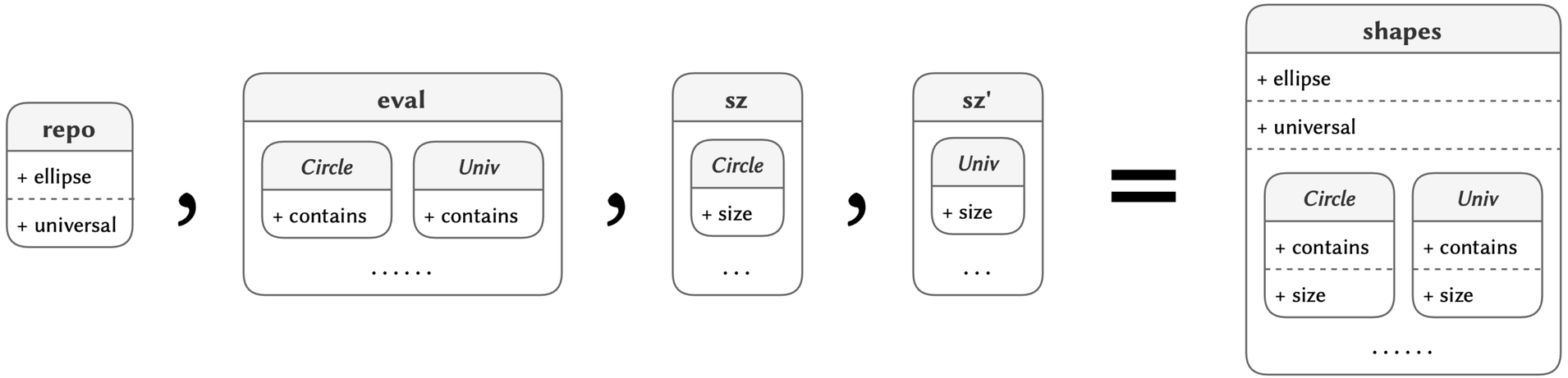
intersection type

```
type Eval = { contains : Vector → Bool };
eval = trait implements RegionSig<Eval> ⇒ {
  (Circle         r).contains p = pow p.x 2 + pow p.y 2 ⩽ pow r 2;
  (Outside        a).contains p = not (a.contains p);
  (Union        a b).contains p = a.contains p || b.contains p;
  (Intersect    a b).contains p = a.contains p && b.contains p;
  (Translate {..} a).contains p = a.contains { x = p.x - x;
                                               y = p.y - y };
  (Univ             ).contains _ = true;
  (Empty            ).contains _ = false;
  (Scale     {..} a).contains p = a.contains { x = p.x / x;
                                               y = p.y / y };
};

shapes = new repo @(Eval&Size) , eval , sz , sz';
```

nested trait composition

Nested Trait Composition

```
shapes = new   repo @(Eval&Size)  , eval , sz , sz' ;
shapes = new ((repo @(Eval&Size)) , eval , sz , sz');
```



Dependency Injection

(Interface Type Refinement)

```
type IsUniv  = { isUniv  : Bool };

chkUniv = trait
    implements RegionSig<IsEmpty ⇒ IsUniv> ⇒ {
  (Univ         ).isUniv = true;
  (Outside     a).isUniv = a.isEmpty;
  (Union     a b).isUniv = a.isUniv || b.isUniv;
  (Intersect a b).isUniv = a.isUniv && b.isUniv;
  (Translate _ a).isUniv = a.isUniv;
  (Scale     _ a).isUniv = a.isUniv;
                _.isUniv = false;
};
```

input

output

```
type IsEmpty = { isEmpty : Bool };

chkEmpty = trait
    implements RegionSig<IsUniv ⇒ IsEmpty> ⇒ {
  (Empty        ).isEmpty = true;
  (Outside     a).isEmpty = a.isUniv;
  (Union     a b).isEmpty = a.isEmpty && b.isEmpty;
  (Intersect a b).isEmpty = a.isEmpty || b.isEmpty;
  (Translate _ a).isEmpty = a.isEmpty;
  (Scale     _ a).isEmpty = a.isEmpty;
                _.isEmpty = false;
};
```

input

output

Comparison of Embeddings

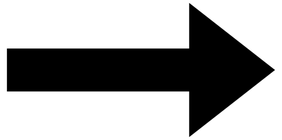
	Shallow	Deep	Hybrid	Poly.	Comp.
Transcoding free	●	●	○	●	●
Linguistic reuse	●	○	●	●	●
Language construct extensibility	●	○	◐[1]	●	●
Interpretation extensibility	○	●	●	●	●
Transformations and optimizations	○	●	●	◐[2]	◐[2]
Linguistic reuse after transformations	*n/a*	○	○	●	●
Modular dependencies	○	◐[3]	◐[3]	○	●
Nested pattern matching	○	●	●	○	◐[4]

[1] The extensibility of language constructs is limited or precludes exhaustive pattern matching.
[2] Transformations require some ingenuity and are sometimes awkward to write.
[3] Dependencies do not require code duplication but still refer to concrete implementations.
[4] Nested pattern matching is implemented as delegated method patterns.

ExT

(A Compositionally Embedded DSL for Document Authoring)

```
\Section[
  Welcome to \Href("https://plground.org")[PLGround]!
]
\Bold[CP] is a \Emph[compositional] programming
language. \\ There are \(1+1+1+1) key concepts in CP:
\Itemize[
  \Item[Compositional interfaces;]
  \Item[Compositional traits;]
  \Item[Method patterns;]
  \Item[Nested trait composition.]
]
```



WELCOME TO PLGROUND!

CP is a *compositional* programming language.
There are 4 key concepts in CP:

- Compositional interfaces;
- Compositional traits;
- Method patterns;
- Nested trait composition.

Implementation can be found in PLGround.org

Applications

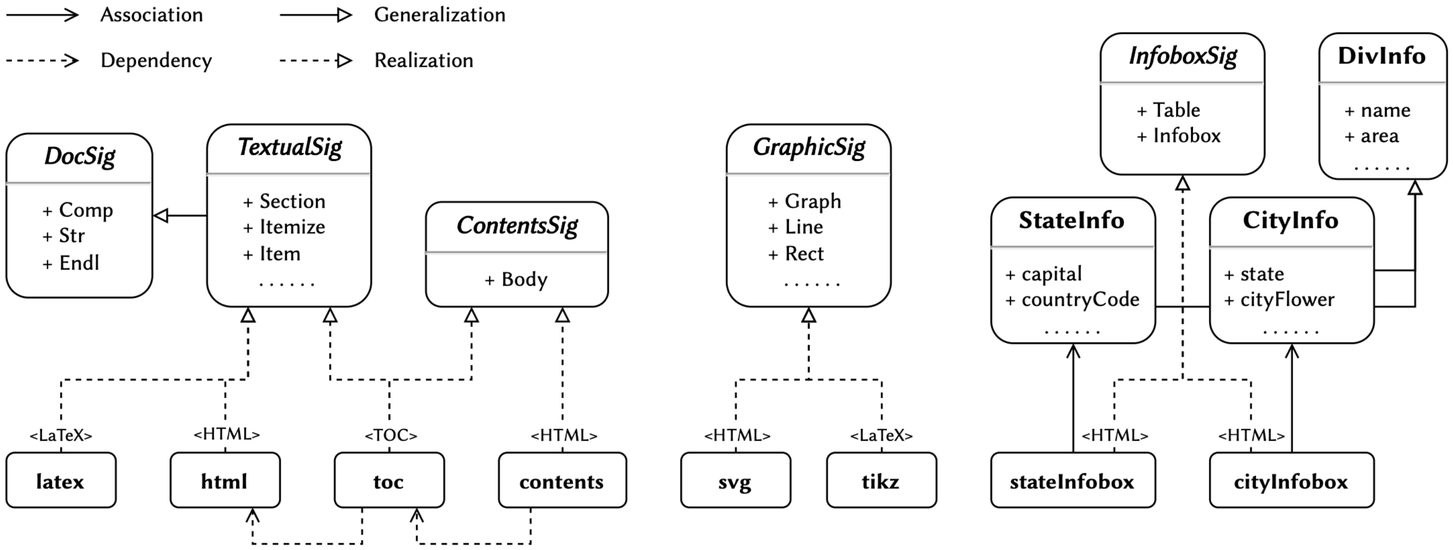
1. Minipedia

No.	Country	Area (km²)	Population	Capital
1	Vatican City State	0	1000	
2	Principality of Monaco	2	38300	
3	Republic of Nauru	21	10671	Yaren
4	Tuvalu	26	11646	Funafuti
5	Republic of San Marino	61	33600	City of San Marino
6	Principality of Liechtenstein	160	38896	Vaduz
7	Marshall Islands	181	54701	Majuro
8	Federation of Saint Christopher and Nevis	261	52441	Basseterre
9	Republic of Maldives	300	383135	Malé
10	Republic of Malta	316	514564	Valletta

Microstate

A **microstate** or **ministate** is a sovereign state having a very small population or very small land area, usually both. The meanings of "state" and "very small" are not well-defined in international law.[1] Recent attempts, since

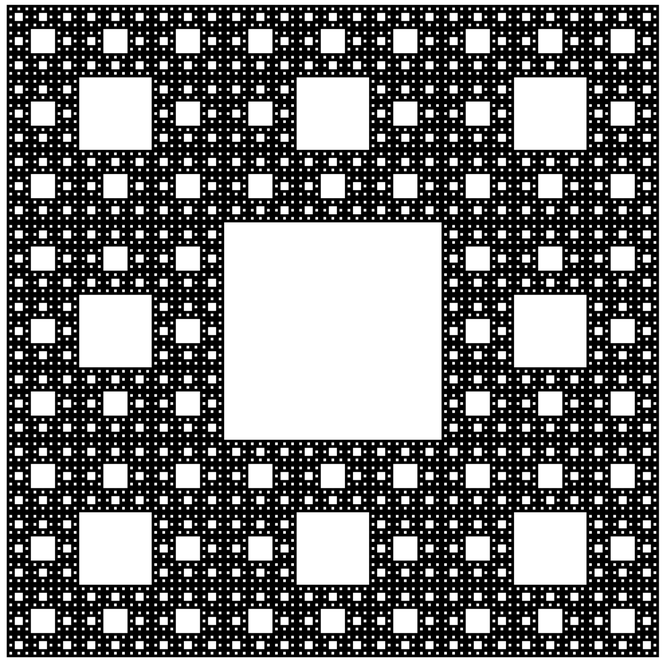
A simplified diagram of ExT components:



Case studies can also be found in PLGround.org

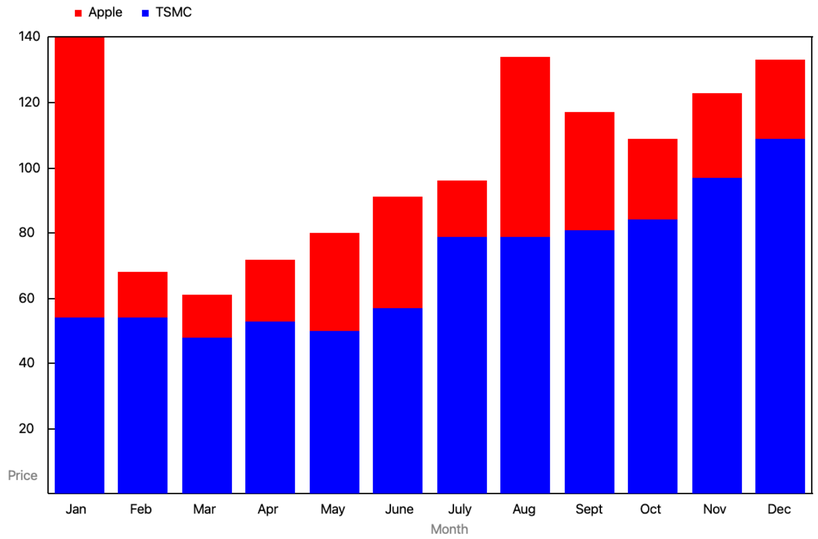
Applications

2. Fractals



Sierpiński Carpet

3. Charts



(Both are powered by modular extensions for SVG)

Case studies can also be found in PLGround.org

Key Features of Compositional Embeddings

can also be encoded in ...

Xuan Bi, Ningning Xie, Bruno C. d. S. Oliveira, and Tom Schrijvers.
Distributive Disjoint Polymorphism for Compositional Programming.
In ESOP 2019.

Bruno C. d. S. Oliveira, Tijs van der Storm, Alex Loh, and William R. Cook.
Feature-Oriented Programming with Object Algebras. In ECOOP 2013.

with a lot of boilerplate code 😕

More in Our Paper

- Our novel way to support nested pattern matching (§3.5);
- A detailed comparison of different embeddings (§3.6);
- The design of ExT (§4);
- The complete code explaining why the tagless-final embedding cannot handle dependencies modularly (Appendix A);
- The complete code extending the tagless-final embedding to modularize dependencies with a lot of boilerplate code (Appendix B).

Q&A