

The background of the slide is a dense, three-dimensional field of numbers. The numbers are rendered in a light blue, translucent style with soft shadows, giving them a sense of depth and volume. They are scattered across the entire frame, with some numbers appearing larger and more prominent than others, creating a complex, abstract pattern. The overall color palette is a range of blues, from light sky blue to a deeper, muted blue.

Exponential Non- Determinism

Ross Tate

Desirable Features

Annotation-Free Get/Set

- ◊ Get/set are likely frequent
- ◊ Type annotation is half of instruction size
- ◊ Other VMs and TALs have no annotation

Multiple Upper Bounds

- ◊ Used in Java to abstract type hierarchy
- ◊ Used to support user-defined casting
- ◊ Other VMs and TALs support it



NP-Hard Type-Checking
in current MVP!

The Problem

Given graph with

- ◇ V vertices
- ◇ E edges
- ◇ For each edge e , vertices v_e^{in} and v_e^{out}

Assign a color $c_v \in \{R, G, B\}$ to each vertex v

- ◇ such that $\forall e. c_{v_e^{in}} \neq c_{v_e^{out}}$

The Program

pick $n : [t_n, \dots, t_0] \rightarrow [t_n, \dots, t_0, t_n]$

(func \$color (param \$v (ref \$V))

 V times: (struct.get 0 (local.get \$v))

 for each e: (struct.set 0 (pick v_e^{in}) (pick v_e^{out}))

 V times: drop

)

With right definition of \$V,
\$color type-checks iff
graph is colorable

Setting up the non-determinism

```
(import type $V
```

```
  <: (struct (ref $R))
```

```
  <: (struct (ref $G))
```

```
  <: (struct (ref $B))
```

```
)
```

```
struct.get 0 : [(ref $V)] -> [(ref $R)]
```

```
struct.get 0 : [(ref $V)] -> [(ref $G)]
```

```
struct.get 0 : [(ref $V)] -> [(ref $B)]
```



No
principal
typing

The Program

```
(func $color (param $v (ref $V))  
  V times: (struct.get 0 (local.get $v))  
  ;; V choices of (ref $R) or (ref $G) or (ref $B)  
  for each e: (struct.set 0 (pick  $v_e^{in}$ ) (pick  $v_e^{out}$ ))  
  V times: drop  
)
```

Cranking the search

```
(import types $NotR $NotG $NotB)
```

```
(import type $R  
  <: $NotG  
  <: $NotB  
  <: (struct (mut (ref $NotR))))
```

```
(import type $G  
  <: $NotB  
  <: $NotR  
  <: (struct (mut (ref $NotG))))
```

```
(import type $B  
  <: $NotR  
  <: $NotG  
  <: (struct (mut (ref $NotB))))
```

```
struct.set 0 : [(ref $R) (ref $G)] -> []
```

```
struct.set 0 : [(ref $R) (ref $B)] -> []
```

```
struct.set 0 : [(ref $R) (ref $R)] -> []
```

```
struct.set 0 : [(ref $G) (ref $B)] -> []
```

```
struct.set 0 : [(ref $G) (ref $R)] -> []
```

```
struct.set 0 : [(ref $G) (ref $G)] -> []
```

```
struct.set 0 : [(ref $B) (ref $R)] -> []
```

```
struct.set 0 : [(ref $B) (ref $G)] -> []
```

```
struct.set 0 : [(ref $B) (ref $B)] -> []
```

◇ Type-checks iff colors are not equal!

The Program

```
(func $color (param $v (ref $V))  
  V times: (struct.get 0 (local.get $v))  
  ;; V choices of (ref $R) or (ref $G) or (ref $B)  
  for each e: (struct.set 0 (pick  $v_e^{in}$ ) (pick  $v_e^{out}$ ))  
  ;; type-checks iff choices make edges distinct  
  V times: drop  
)
```


Recognizing Patterns

Non-Determinism



Exponentiation