Hybrid Type Systems

Jose A. Lopes

Max Planck Institute for Software Systems (MPI-SWS)

MOVEP 2012

Type systems

Type systems are a lightweight verification method

- Common in programming languages
- Increase software reliability
- Verify basic interface specifications
- Avoid complicated formalism

Type systems

Static

multiple types

- Earlier error detection
- Better documentation
- Allow more optimizations
- Increased runtime efficiency

Type systems

Dynamic

`type Dynamic`

- More expressive
- Fast adaptation to requirements
- Simpler component interaction
- Truly dynamic behavior

Problem

- Choosing between static/dynamic is not obvious
- Stronger formalism $\Leftrightarrow$ less flexibility

Hybrid type systems

Research goal

- Develop a hybrid type system
- Combine best of both static/dynamic
- Adjust type system to the development process

Type system properties

- Gradual typing (introduced by Siek [2])
- Type inference
- Polymorphism
- Generics & heterogeneous data structures
- Specifications
- Subtyping & covariance
- ...

Gradual typing & Type inference

Type annotations are optional and gradually strengthen the type system

```
// accepted
   (fn (x:Num) => x + 1) 1

// rejected
   (fn (x:Num) => x + 1) true
```

Gradual typing & Type inference

```
// accepted, cast failure at runtime
   (fn (x) => x + 1) true
```

Gradual typing & Type inference

```
// accepted, cast failure at runtime
   (fn (x) => x + 1) true

≈ (fn (x:Dyn) => x + 1) true
```

Gradual typing & Type inference

```
// accepted, cast failure at runtime
   (fn (x) => x + 1) true

≈ (fn (x:Dyn) => x + 1) true

≈ (fn (x:Dyn) => (<Num> x) + 1) (<Dyn> true)
```

Polymorphism

Identity function

```
let idI = (fun (x:Int) => x)
(idI 1) : Int

let idD = (fun (x:Double) => x)
(idD 2.0) : Double

let idIL = (fun (x:Int list) => x)
(idIL [1,2]) : Int list
```

Polymorphism

Polymorphic identity function

```
let id = (fun (x) => x) : a → a

(id 1) : Int

(id 2.0) : Double

(id [1,2]) : Int list
```

Generics & heterogeneous data structures

```
[1, 2, 3] : Int list

[1.0, 2.0, 3.0] : Double list

[1, 2.0, "Hi"] : Dyn list

[1, 2.0, "Hi"] : Int ∨ Double ∨ String list
```

Specifications

Fibonacci sequence with refinement types
(introduced by Flanagan [2])

```
let Pos0 = {x:Int | x >= 0}

let rec fib (n:Pos0):Pos0 =
  if (n < 2)
  then 1
  else ((fib (n - 1)) + (fib (n - 2)))
```

Specifications

Fibonacci sequence with refinement types
(introduced by Flanagan [2])

```
let Pos0 = {x:Int | x >= 0}

let rec fib (n:Pos0):Pos0 =
  if (n < 2)
  then 1
  else ((fib (n - 1)) + (fib (n - 2)))
```

Specifications

Fibonacci sequence with refinement types
(introduced by Flanagan [2])

```
let Pos0 = {x:Int | x >= 0}

let rec fib (n:Pos0):Pos0 =
  if (n < 2)
  then 1
  else ((fib (n - 1)) + (fib (n - 2)))
```

Work & Conclusions

- Bidirectional typechecking with polymorphic types (by Dunfield [1])
- `Dynamic` type encoding through union types (e.g., Furr [2])
- Integrate refinement types (by Flanagan [2])

Bibliography (1/3)

E. Meijer and P. Drayton

Static typing where possible, dynamic typing when needed: The end of the cold war between programming languages

In *OOPSLA'04 Workshop on Revival of Dynamic Languages*, 2004.

J. Siek and W. Taha

Gradual typing for functional languages

In *Scheme and Functional Programming Workshop*, 2006.

M. Abadi, L. Cardelli, B. Pierce, and G. Plotkin

Dynamic typing in a statically typed language

In *ACM Transactions on Programming Languages and Systems*, pages 237–268, 13(2), 1991.

Bibliography (2/3)

R. Cartwright and M. Fagan

Soft typing

In *PLDI'91*. 1991. ACM Press.

C. Flanagan

Hybrid type checking

In *POPL'06: The 33rd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pa es 245–256, Charleston, South Carolina, January 2006.

S. Thatte

Quasi-static typing

In *POPL'90: Proceedings of the 17th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 367–381, New York, NY, USA, 1990. ACM Press.

Bibliography (3/3)

J. Dunfield

Greedy bidirectional polymorphism

In *ML'09: ML Workshop*.

M. Furr, J. An, J. Foster, and M. Hicks

Static type inference for Ruby

In *Proceedings of the 24th Annual ACM Symposium on Applied Computing*, OOPS track, Honolulu, HI. March 2009.

K. Knowles, A. Tomb, J. Gronski, S. Freund, and C. Flanagan

Sage: Unified Hybrid Checking for First-Class Types, General Refinement Types, and Dynamic

In Scheme and Functional Programming workshop, 2006.