Open Issues in Type Systems

LaME 2012: Languages for the Multicore Era

Rob Bocchino
Carnegie Mellon University
June 13, 2012

Some Issues

What should type systems do?

Static vs. dynamic checks

Usability

Composability

Alternatives to type systems

Other issues?

What Should Type Systems Do?

Here are some things they can do

- Provide semantic guarantees for concurrent programs
- Track concurrent effects on shared memory
- Manage locality
- Merge concurrent updates

What else?

Semantic Guarantees

Type safety

- Sometimes (C/C++) vs. all the time (Java, C#, Scala)

Determinism

- Sequential equivalence, or not?
- External vs. internal

Isolation

- Strong vs. weak
- For whole tasks vs. for atomic sections

Race freedom

Others?

Effect Checking

Regions/ownership [DPJ, JOE, MOJO]

Permissions/linear types [Vault, Plaid]

Monads [Haskell]

Other methods?

Mix and match?

Other Uses for Types

Manage locality

- X10 places

Resolve conflicts

- Revisions/isolation types
- Deterministic consistency
- Semantic commutativity

What else?

Static vs. Dynamic Checks

Benefits of static checking

- Early detection
- Strong guarantee
- Minimize runtime overhead

Benefits of dynamic checking

- More flexibility for programmer
- More precise info
- Less compile-time overhead (compute, annotate)
- Better at dynamic dependences, casts

Blending Static and Dynamic

Gradual types

- Assignment of blame
- Efficiency

Supporting dynamic checks with types

- Jade: Use type system to reduce # of runtime checks
- DPJ: Similar approach using transactions

Galois, Prometheus, concurrent revisions, etc.

- Provide guarantees if code follows guidelines
- Could use types to check guidelines

Usability

Complexity

- More power usually means more complexity
- Can we hide the complexity from the user?
  - This is why compiler techology has succeeded

Annotation overhead

- Programmers don't want to write extra code
- Probably need type inference, IDE assistance
  - Functional languages have this issue too
- Language translation tools (e.g., DPJizer)

Natural abstractions (e.g., `unique` vs. points-to graph)

Composability

State of the art: New problem ⇒ new type system

What if we want to solve several problems?

- Union of all the type systems doesn't scale
- It's too complex

Can we apply different systems to the same code?

- Say in an IDE?
- Use extensible syntax? Annotations?
- Pluggable type systems?

Can a single framework express several systems?

- E.g., different flavors of ownership?

Alternatives to Type Systems

Static analysis

- Type checking is programmer-guided analysis
- Also a form of abstract interpretation

Verification, e.g., using separation logic

- Types can help here
- Provide clean syntax for what they are good at
  - E.g., use of ownership in OO verification
- Can check parts of programs (e.g., library/framework uses)

Dynamic checks (race detectors, determinism checkers)

Testing

Other Issues?