Permission-Based Programming Languages

Jonathan Aldrich    Ronald Garcia    Mark Hahnenberg
Manuel Mohr    Karl Naden    Darpan Saini    Sven Stork
Joshua Sunshine    Éric Tanter    Roger Wolff

ICSE New Ideas and Emerging Results
May 25, 2011
Permission systems associate every reference with both a type and a permission that restricts aliasing and mutability.

```java
var unique InputStream stream = new FileInputStream(...);
```

• Some permissions and their intuitive semantics [Boyland][Noble][…]

- **unique**: no other references to the object
- **immutable**: no-one can modify the object

Type system checks permission consistency
• A language whose type system, object model, and run-time are co-designed with permissions in mind
  – Contrast: prior permission systems layered static permission checking onto existing languages

• Potential benefits
  – Design and encapsulation enforcement
  – Parallel execution
  – Explicit state change in the object model
  – Compile-time and run-time checking
method unique Data createData();

val d = createData();
print(d);
val s = getStats(d);
manipulate(d, s);
**Automatic Parallelization**

```java
method unique Data createData();

val d = createData();
print(d);
val s = getStats(d);
manipulate(d, s);
```

---

**Permission-Based Programming Languages**

---

**Plaid**
**Automatic Parallelization**

```java
method unique Data createData();
method void print(immutable Data d);
method unique Stats getStats(immutable Data d);

val d = createData();
print(d);
val s = getStats(d);
manipulate(d, s);
```
method unique Data createData();
method void print(immutable Data d);
method unique Stats getStats(immutable Data d);

val d = createData();
print(d);
val s = getStats(d);
manipulate(d, s);
method unique Data createData();
method void print(immutable Data d);
method unique Stats getStats(immutable Data d);

val d = createData();
print(d);
val s = getStats(d);
manipulate(d, s);
method unique Data createData();
method void print(immutable Data d);
method unique Stats getStats(immutable Data d);
method void manipulate(unique Data d, immutable Stats s);

val d = createData();
print(d);
val s = getStats(d);
manipulate(d, s);
method unique Data createData();
method void print(immutable Data d);
method unique Stats getStats(immutable Data d);
method void manipulate(unique Data d, immutable Stats s);

val d = createData();
print(d);
val s = getStats(d);
manipulate(d, s);
Automatic Parallelization

```java
method unique Data.createData();
method void print(immutable Data d);
method unique Stats.getStats(immutable Data d);
method void manipulate(unique Data d, immutable Stats s);

val d = createData();
print(d);
val s = getStats(d);
manipulate(d, s);
```

Casts can also be used to recover unique
The runtime checks the cast using reference counts
Explicit State Change

```java
state File {
    val String filename;
}
```

Permission-Based Programming Languages
Explicit State Change

```java
state File {
    val String filename;
}
state ClosedFile = File with {
    method void open() [unique ClosedFile >> OpenFile] {
```

Permission-Based Programming Languages
Explicit State Change

```scala
state File {
  val String filename;
}
state ClosedFile = File with {
  method void open() [unique ClosedFile >> OpenFile] {
    State transition
  }
}
```
Explicit State Change

```scala
state File {
    val String filename;
}
state ClosedFile = File with {
    method void open() [unique ClosedFile>>OpenFile] {
        open transition
    }
}
state OpenFile = File with {
    private val CFile fileResource;
    method int read();
    method void close() [OpenFile>>ClosedFile];
}
```

Permission-Based Programming Languages
Explicit State Change

state File {
    val String filename;
}

state ClosedFile = File with {
    method void open() [unique ClosedFile>>OpenFile] {
        this <- OpenFile {
            fileResource = fopen(filename);
        }
    }
}

state OpenFile = File with {
    private val CFile fileResource;

    method int read();
    method void close() [OpenFile>>ClosedFile];
}
Plaid: A Permission-Based Language

• Currently exploring these ideas with Plaid
  – First-class abstractions for changing state
  – Naturally safe concurrent execution
  – Practical mix of static & dynamic checking

• Other research directions possible
  – Systems languages: permissions support memory management
  – Security: permissions help control access, information flow

• Status: compiler implemented, typechecker underway
  – Web-based interface available

http://www.plaid-lang.org/