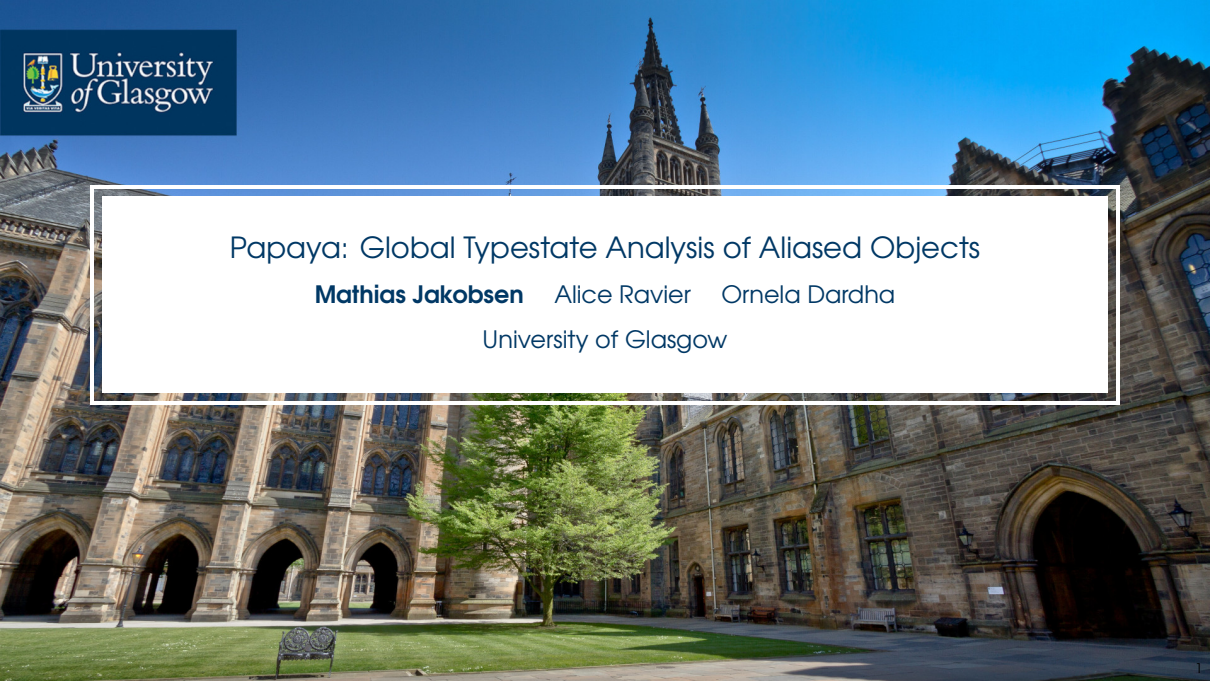


University of Glasgow

Papaya: Global Typestate Analysis of Aliased Objects

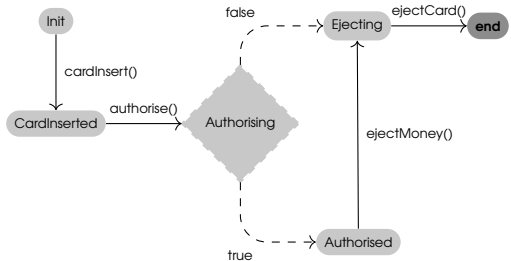
Mathias Jakobsen Alice Ravier Ornela Dardha

University of Glasgow

Typestates as Dynamic Interfaces for Objects

Typestates (introduced by Strom & Yemini (1986)) augment **static** typing information with **dynamic** properties

```
atm.cardInsert()
atm.authorise() match {
  case true => atm.ejectMoney()
  case false =>
}
atm.ejectCard()
```



The Language of Typestates

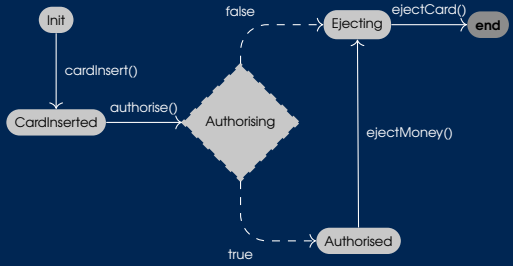
The class C of an object defines the static information about methods and fields

We extend this with a **usage**[a] $\mathcal{U}$ describing the protocol of an object:

- Branching (external choice): $\{m_i; w_i\}_{i \in I}$
- Choice (internal choice): $\langle l_i : u_i \rangle_{i \in I}$
- Recursion: $\mu X.u$
- Finished: end

This together forms a typestate $C[\mathcal{U}]$ to describe the type of an object

[a] Following the notion of Bravetti et. al (2020)



Mungo

```
@Typestate("ATMProtocol")
class ATM {
  def cardInsert(): Unit = { ... }

  def authorise(): Boolean = { ... }

  def ejectMoney(): Unit = { ... }

  def ejectCard(): Unit = { ... }

}
```

In Mungo (introduced by Kouzapas et. al (2016), continued by Bravetti et. al (2020)), class annotations allow us to statically verify that protocols are respected

Mungo is a Java toolchain for defining multiparty protocols with multiparty session types, and ensuring protocol conformance of the parties using typestates

A challenge with aliasing

Aliasing presents a challenge, in the presence of typestates

```
ATM a1 = new ATM();  // a1: ATM[{cardInsert; {authorise; <True: ..., False: ...>}}]
ATM a2 = a1;         // a2: ATM[{cardInsert; {authorise; <True: ..., False: ...>}}]
a1.cardInsert()      // a1: ATM[{authorise; <True: ..., False: ...>}]
a2;                  // a2: ATM[𝒰]
```

If $\mathcal{U} = \{\text{cardInsert}; \{\text{authorise}; \langle \text{True} : \ldots, \text{False} : \ldots \rangle\}\}$ then calling open will result in two calls to open on the underlying object

If $\mathcal{U} = \{\text{authorise}; \langle \text{True} : \ldots, \text{False} : \ldots \rangle\}$ then operations to a1 should affect a2 and vice versa

Prior Approaches to Aliasing

Previously, two different approaches has been used to deal with this:

Disallowing aliases by adopting **linear typing** is commonly seen in other behavioural type systems. This approach was used in the original work on Mungo

Managed aliases by introducing various restricted aliasing mechanisms such as **access-permissions** (Plaid), **Adoption & Focus** (Vault), **View equations** (Plaid-like), and **Parallel protocols** (Mungo)

In the case of *managed aliasing* severe restrictions are imposed on aliases, to ensure **compositionality**

Papaya

We present another approach to aliasing and typestates, **Papaya**:

- The language and the type system
- An example and it's properties
- An implementation for Scala

Aliasing as a Global Property

The idea behind the Papaya type system: *"Full aliasing is a **global property** - what if we treat it like that?"*

Why? Object references are often aliased and stored in other objects for later use (getter and setter methods)

If this should be tracked, we can no longer have an isolated look at each class

The Core Calculus

We define a small core calculus with OO features

$$
\begin{aligned}
D &::= \text{class } C\{\mathcal{U}, \overline{F}, \overline{M}\} \mid \text{enum } L\{\bar{l}\} \\
F &::= \text{val } f : t \\
M &::= \text{fun } m(x : t) : t\ \{e\} \\
r &::= o \mid o.f \mid x \\
e &::= o.f = e \mid o.f = \text{new } C \mid e; e \mid r.m(e) \mid \text{unit} \mid o.f \mid x \\
&\quad \mid \text{if } (e)\ \{e\} \text{ else } \{e\} \mid o.l \mid \text{match}(e)\{\overline{l : e}\} \mid \text{null} \\
&\quad \mid \text{true} \mid \text{false} \mid k : e \mid \text{continue } k
\end{aligned}
$$

Keeping Track of Aliases

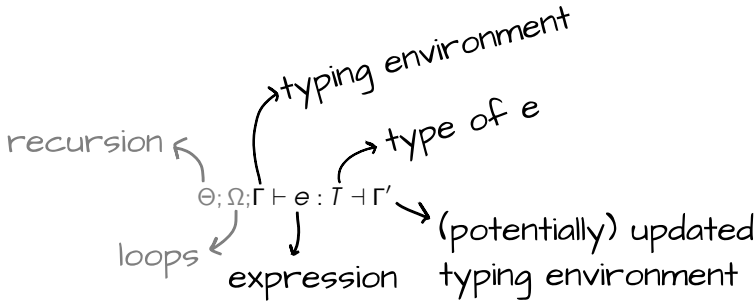
We further extend the type of an object to contain the object reference:

$$
\begin{aligned}
T &::= \ldots \mid o[C, \mathcal{U}] \\
\mathcal{U} &::= \mu X.\mathcal{U} \mid X \mid \{\overline{m; w}\} \mid \text{end} \\
w &::= \langle \overline{l : \mathcal{U}} \rangle \mid \mathcal{U}
\end{aligned}
$$

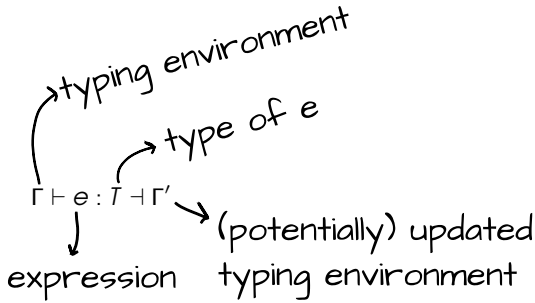
We then add a layer of indirection to our typing environment:

$$
\begin{aligned}
\Gamma &::= \emptyset \mid \Gamma, o \mapsto (T, \lambda) \\
\lambda &::= \emptyset \mid \lambda, f \mapsto z \\
z &::= \texttt{basetype}\ \mathit{bool} \mid \texttt{basetype}\ \mathit{void} \\
&\quad \mid \texttt{basetype}\ \bot \mid \ \texttt{basetype}\ L \\
&\quad \mid \texttt{reference}\ o
\end{aligned}
$$

The Type System



The Type System



The Type System

CALL-D

$$\frac{\Gamma \vdash e : T \dashv \Gamma'' \qquad \Gamma''(o) = (o[C, \mathcal{U}], \lambda) \qquad \mathcal{U} \xrightarrow{m} \mathcal{U}' \qquad \text{fun } m(x : t) : t'\{e'\} \in \overline{D}(C).\text{methods} \qquad \text{agree}(t, T) \qquad \Gamma''[o \mapsto (o[C, \mathcal{U}'], \lambda) \vdash e'\{\text{this}/o\}\{x/\text{getValue}(T)\} : T' \dashv \Gamma' \qquad \text{returns}(t', T')}{\Gamma \vdash o.m(e) : T' \dashv \Gamma'}$$

Global Analysis

Outline of type checking:

1. Start from initial expression (main method)
2. Check expressions as normal
3. When type checking method calls, *expand the method body* and continue checking

Example

```
class BankAccount
    [{setMoney;
     {applyInterest;
     {getMoney; end}}}] {

  val amount:float;

  fun setMoney(d:float):void  {
    this.amount = d;
  }

  fun getMoney():float {
    this.amount;
  }

  fun applyInterest(rate:float) {
    this.amount = this.amount *
        rate;
  }
}
```

```
class SalaryManager
    [{setAccount;
     {addSalary; end}}] {

  val account:BankAccount

  fun setAccount(ms:BankAccount):
       void {
    this.account = ms;
  }

  fun addSalary(amount:float) {
    this.account.setMoney(amount);
    this.account.applyInterest
        (1.05);
  }
}
```

```
class DataStorage
    [{setAccount;
     {store; end}}] {

  val account:BankAccount

  fun setAccount(ms:BankAccount):
       void {
    this.account = ms;
  }

  fun store() {
    this.account.getMoney();
  }
}
```

Example

```
class BankAccount
    [{setMoney;
     {applyInterest;
     {getMoney; end}}}] {

  val amount: float;

  fun setMoney(d: float): void  {
    this.amount = d;
  }

  fun getMoney(): float {
    this.amount;
  }

  fun applyInterest(rate: float) {
    this.amount = this.amount *
        rate;
  }
}
```

```
class SalaryManager
    [{setAccount;
     {addSalary; end}}] {

  val account: BankAccount

  fun setAccount(ms: BankAccount):
       void {
    this.account = ms;
  }

  fun addSalary(amount: float) {
    this.account.setMoney(amount);
    this.account.applyInterest
        (1.05);
  }
}
```

```
class DataStorage
    [{setAccount;
     {store; end}}] {

  val account: BankAccount

  fun setAccount(ms: BankAccount):
       void {
    this.account = ms;
  }

  fun store() {
    this.account.getMoney();
  }
}
```

Example

```
class BankAccount
    [{setMoney;
     {applyInterest;
     {getMoney; end}}}] {

  val amount:float;

  fun setMoney(d:float):void  {
    this.amount = d;
  }

  fun getMoney():float {
    this.amount;
  }

  fun applyInterest(rate:float) {
    this.amount = this.amount *
        rate;
  }
}
```

```
class SalaryManager
    [{setAccount;
     {addSalary; end}}] {

  val account:BankAccount

  fun setAccount(ms:BankAccount):
       void {
    this.account = ms;
  }

  fun addSalary(amount:float) {
    this.account.setMoney(amount);
    this.account.applyInterest
        (1.05);
  }
}
```

```
class DataStorage
    [{setAccount;
     {store; end}}] {

  val account:BankAccount

  fun setAccount(ms:BankAccount):
       void {
    this.account = ms;
  }

  fun store() {
    this.account.getMoney();
  }
}
```

Expressivity of Global Analysis

```
void main() {
    account = new BankAccount;
    manager = new SalaryManager;
    db = new DataStorage;

    manager.setAccount(account);
    db.setAccount(account);

    manager.addSalary(100.0);
    db.store();
}
```

We can use global analysis to allow aliasing of `account`

Aliases are tracked across class boundaries, meaning the references inside `manager` and `db` are also tracked

All aliases progress the same typestate since they share the object reference

Expressivity of Global Analysis

```
void main() {
    account = new BankAccount;
    manager = new SalaryManager;
    db = new DataStorage;

    manager.setAccount(account);
    db.setAccount(account);

    manager.addSalary(100.0);
    db.store();
}
```

$$\Gamma = \left\{ o_{\text{main}} \mapsto \begin{array}{l} (\text{Main[end]}, \\ \{\texttt{account} \mapsto \text{null}, \texttt{manager} \mapsto \text{null}, \\ \texttt{db} \mapsto \text{null}\}) \end{array} \right\}$$

Expressivity of Global Analysis

```
void main() {
    account = new BankAccount;
    manager = new SalaryManager;
    db = new DataStorage;

    manager.setAccount(account);
    db.setAccount(account);

    manager.addSalary(100.0);
    db.store();
}
```

$$
\Gamma = \left\{ \begin{array}{ll}
o_{\text{main}} \mapsto & (\text{Main}[\text{end}], \\
 & \{\texttt{account} \mapsto o_{\text{acc}}, \texttt{manager} \mapsto o_{\text{man}}, \\
 & \texttt{db} \mapsto o_{\text{db}}, \}) \\
o_{\text{acc}} \mapsto & (\text{BankAccount}[\ \{\text{setMoney}; \\
 & \quad \{\text{applyInterest}; \\
 & \quad \{\text{getMoney}; \text{end}\}\}\}], \\
 & \{\texttt{amount} \mapsto \texttt{double}\}) \\
o_{\text{man}} \mapsto & (\text{SalaryManager}[\ \{\text{setAccount}; \\
 & \quad \{\text{addSalary}; \text{end}\}], \\
 & \{\texttt{account} \mapsto \text{null}\}) \\
o_{\text{db}} \mapsto & (\text{DataStorage}[\ \{\text{setAccount}; \\
 & \quad \{\text{store}; \text{end}\}], \\
 & \{\texttt{account} \mapsto \text{null}\})
\end{array} \right\}
$$

Expressivity of Global Analysis

$o_{man}.\texttt{account} = o_{acc}$

...

$$\Gamma = \left\{ \begin{array}{ll} o_{\text{main}} \mapsto & (\text{Main}[\text{end}], \\ & \{\texttt{account} \mapsto o_{\text{acc}}, \texttt{manager} \mapsto o_{\text{man}}, \\ & \texttt{db} \mapsto o_{\text{db}}, \}) \\ o_{\text{acc}} \mapsto & (\text{BankAccount}[\ \{\text{setMoney}; \\ & \qquad \{\text{applyInterest}; \\ & \qquad \{\text{getMoney}; \text{end}\}\}\}], \\ & \{\texttt{amount} \mapsto \texttt{double}\}) \\ o_{\text{man}} \mapsto & (\text{SalaryManager}[\{\text{addSalary}; \text{end}\}], \\ & \{\texttt{account} \mapsto \text{null}\}) \\ o_{\text{db}} \mapsto & (\text{DataStorage}[\ \{\text{setAccount}; \\ & \qquad \{\text{store}; \text{end}\}], \\ & \{\texttt{account} \mapsto \text{null}\}) \end{array} \right\}$$

Expressivity of Global Analysis

$o_{man}.\texttt{account} = o_{acc}$

...

$$\Gamma = \left\{ \begin{array}{l} o_{\text{main}} \mapsto \text{(Main[end],} \\ \quad \{\texttt{account} \mapsto o_{\text{acc}}, \texttt{manager} \mapsto o_{\text{man}}, \\ \quad \texttt{db} \mapsto o_{\text{db}}, \}) \\ o_{\text{acc}} \mapsto \text{(BankAccount[\{setMoney;} \\ \quad \{applyInterest; \\ \quad \{getMoney; end\}\}\}], \\ \quad \{\texttt{amount} \mapsto \texttt{double}\}) \\ o_{\text{man}} \mapsto \text{(SalaryManager[\{addSalary; end\}],} \\ \quad \{\texttt{account} \mapsto o_{\text{acc}}\}) \\ o_{\text{db}} \mapsto \text{(DataStorage[\{setAccount;} \\ \quad \{store; end\}], \\ \quad \{\texttt{account} \mapsto \text{null}\}) \end{array} \right\}$$

Expressivity of Global Analysis

```
void main() {
    account = new BankAccount;
    manager = new SalaryManager;
    db = new DataStorage;

    manager.setAccount(account);
    db.setAccount(account);

    manager.addSalary(100.0);
    db.store();
}
```

$$
\Gamma = \left\{
\begin{array}{ll}
o_{\text{main}} \mapsto & (\text{Main}[\text{end}], \\
& \quad \{\texttt{account} \mapsto o_{\text{acc}}, \texttt{manager} \mapsto o_{\text{man}}, \\
& \quad \texttt{db} \mapsto o_{\text{db}}, \}) \\
o_{\text{acc}} \mapsto & (\text{BankAccount}[\ \{\text{setMoney}; \\
& \qquad \{\text{applyInterest}; \\
& \qquad \{\text{getMoney}; \text{end}\}\}\}], \\
& \{\texttt{amount} \mapsto \texttt{double}\}) \\
o_{\text{man}} \mapsto & (\text{SalaryManager}[\{\text{addSalary}; \text{end}\}], \\
& \{\texttt{account} \mapsto o_{\text{acc}}\}) \\
o_{\text{db}} \mapsto & (\text{DataStorage}[\ \{\text{setAccount}; \\
& \qquad \{\text{store}; \text{end}\}], \\
& \{\texttt{account} \mapsto \text{null}\})
\end{array}
\right\}
$$

Expressivity of Global Analysis

```
void main() {
    account = new BankAccount;
    manager = new SalaryManager;
    db = new DataStorage;

    manager.setAccount(account);
    db.setAccount(account);

    manager.addSalary(100.0);
    db.store();
}
```

$$
\Gamma = \left\{
\begin{array}{ll}
o_{\text{main}} \mapsto & (\text{Main}[\text{end}], \\
& \quad \{\texttt{account} \mapsto o_{\text{acc}}, \texttt{manager} \mapsto o_{\text{man}}, \\
& \quad \texttt{db} \mapsto o_{\text{db}}, \}) \\
o_{\text{acc}} \mapsto & (\text{BankAccount}[\ \{\text{setMoney}; \\
& \qquad \{\text{applyInterest}; \\
& \qquad \{\text{getMoney}; \text{end}\}\}\}], \\
& \{\texttt{amount} \mapsto \texttt{double}\}) \\
o_{\text{man}} \mapsto & (\text{SalaryManager}[\text{addSalary}; \text{end}\}], \\
& \{\texttt{account} \mapsto o_{\text{acc}}\}) \\
o_{\text{db}} \mapsto & (\text{DataStorage}[\{\text{store}; \text{end}\}], \\
& \{\texttt{account} \mapsto o_{\text{acc}}\})
\end{array}
\right\}
$$

Expressivity of Global Analysis

```
o_man.account.setMoney(100);
o_man.account.applyInterest(1.05f);
...
```

$$\Gamma = \left\{ \begin{array}{ll} o_{\text{main}} \mapsto & (\text{Main}[\text{end}], \\ & \{\texttt{account} \mapsto o_{\text{acc}}, \texttt{manager} \mapsto o_{\text{man}}, \\ & \texttt{db} \mapsto o_{\text{db}}, \}) \\ o_{\text{acc}} \mapsto & (\text{BankAccount}[\ \{\text{setMoney}; \\ & \qquad \{\text{applyInterest}; \\ & \qquad \{\text{getMoney}; \text{end}\}\}\}], \\ & \{\texttt{amount} \mapsto \texttt{double}\}) \\ o_{\text{man}} \mapsto & (\text{SalaryManager}[\text{end}], \\ & \{\texttt{account} \mapsto o_{\text{acc}}\}) \\ o_{\text{db}} \mapsto & (\text{DataStorage}[\{\text{store}; \text{end}\}], \\ & \{\texttt{account} \mapsto o_{acc}\}) \end{array} \right\}$$

Expressivity of Global Analysis

```
o_acc.amount = 100;
...
```

$$\Gamma = \left\{ \begin{array}{ll} o_{\text{main}} \mapsto & (\text{Main}[\text{end}], \\ & \{\texttt{account} \mapsto o_{\text{acc}}, \texttt{manager} \mapsto o_{\text{man}}, \\ & \texttt{db} \mapsto o_{\text{db}}, \}) \\ o_{\text{acc}} \mapsto & (\text{BankAccount}[\ \{\text{applyInterest}; \\ & \qquad \{\text{getMoney}; \text{end}\}\}], \\ & \{\texttt{amount} \mapsto \texttt{double}\}) \\ o_{\text{man}} \mapsto & (\text{SalaryManager}[\text{end}], \\ & \{\texttt{account} \mapsto o_{\text{acc}}\}) \\ o_{\text{db}} \mapsto & (\text{DataStorage}[\{\text{store}; \text{end}\}], \\ & \{\texttt{account} \mapsto o_{acc}\}) \end{array} \right\}$$

Expressivity of Global Analysis

```
o_man.account.setMoney(100);
o_man.account.applyInterest(1.05f);
...
```

$$\Gamma = \left\{ \begin{array}{ll} o_{\text{main}} \mapsto & (\text{Main[end]}, \\ & \quad \{\texttt{account} \mapsto o_{\text{acc}}, \texttt{manager} \mapsto o_{\text{man}}, \\ & \quad \texttt{db} \mapsto o_{\text{db}}, \}) \\ o_{\text{acc}} \mapsto & (\text{BankAccount}[\ \{\text{applyInterest}; \\ & \qquad \{\text{getMoney}; \text{end}\}\}], \\ & \quad \{\texttt{amount} \mapsto \texttt{double}\}) \\ o_{\text{man}} \mapsto & (\text{SalaryManager[end]}, \\ & \quad \{\texttt{account} \mapsto o_{\text{acc}}\}) \\ o_{\text{db}} \mapsto & (\text{DataStorage}[\{\text{store}; \text{end}\}], \\ & \quad \{\texttt{account} \mapsto o_{acc}\}) \end{array} \right\}$$

Expressivity of Global Analysis

```
void main() {
    account = new BankAccount;
    manager = new SalaryManager;
    db = new DataStorage;

    manager.setAccount(account);
    db.setAccount(account);

    manager.addSalary(100.0);
    db.store();
}
```

$$\Gamma = \left\{ \begin{array}{ll} o_{\text{main}} \mapsto & (\text{Main[end]}, \\ & \{\texttt{account} \mapsto o_{\text{acc}}, \texttt{manager} \mapsto o_{\text{man}}, \\ & \texttt{db} \mapsto o_{\text{db}}, \}) \\ o_{\text{acc}} \mapsto & (\text{BankAccount}[\ \{\text{getMoney; end}\}], \\ & \{\texttt{amount} \mapsto \texttt{double}\}) \\ o_{\text{man}} \mapsto & (\text{SalaryManager[end]}, \\ & \{\texttt{account} \mapsto o_{\text{acc}}\}) \\ o_{\text{db}} \mapsto & (\text{DataStorage}[\{\text{store; end}\}], \\ & \{\texttt{account} \mapsto o_{acc}\}) \end{array} \right\}$$

Expressivity of Global Analysis

```
void main() {
    account = new BankAccount;
    manager = new SalaryManager;
    db = new DataStorage;

    manager.setAccount(account);
    db.setAccount(account);

    manager.addSalary(100.0);
    db.store();
}
```

$$\Gamma = \left\{ \begin{array}{ll} o_{\text{main}} \mapsto & (\text{Main[end]}, \\ & \{\texttt{account} \mapsto o_{\text{acc}}, \texttt{manager} \mapsto o_{\text{man}}, \\ & \texttt{db} \mapsto o_{\text{db}}, \}) \\ o_{\text{acc}} \mapsto & (\text{BankAccount[end]}, \\ & \{\texttt{amount} \mapsto \texttt{double}\}) \\ o_{\text{man}} \mapsto & (\text{SalaryManager[end]}, \\ & \{\texttt{account} \mapsto o_{\text{acc}}\}) \\ o_{\text{db}} \mapsto & (\text{DataStorage[end]}, \\ & \{\texttt{account} \mapsto o_{acc}\}) \end{array} \right\}$$

Properties of the Type System

- The type system is **sound**
  - Safety & progress
- Usages are **respected** and **completed**
  - Protocol conformance - All operations follow their specified protocol
  - Protocol completion - At termination, all protocols have been completed

Properties of the Type System

- The type system is **sound**
  - Safety & progress
- Usages are **respected** and **completed**
  - Protocol conformance - All operations follow their specified protocol

Corollary: Protocol conformance

If

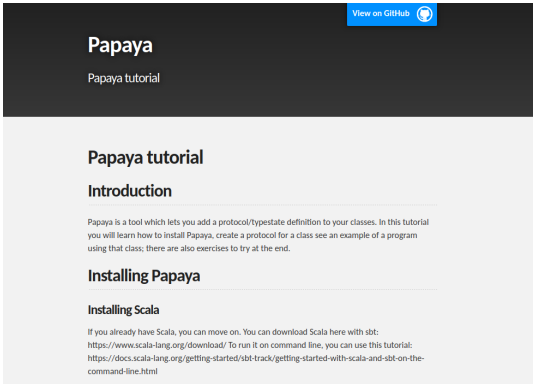
$$\Gamma \vdash e : T \dashv \Gamma'', \Gamma \vdash h, \text{ and } \langle h, e\rangle \xrightarrow{o.\alpha} \langle h', e'\rangle$$

then

$$\exists \Gamma'.\ \Gamma' \vdash e' : T \dashv \Gamma'' \text{ and } \Gamma(o).\text{usage} \xrightarrow{\alpha} \Gamma'(o).\text{usage}$$

  - Protocol completion - At termination, all protocols have been completed

Implementation



A prototype tool has been implemented for Scala[1]

Implements the core language features covered in this presentation as a Scala compiler plugin

Allows you to define protocols in a DSL and add them to classes with annotations

Emits **compile time errors and warnings** when program does not adhere to the specified protocol

[1] `https://github.com/Aliceravier/Scala-Mungo`

Implementation

```
import ProtocolDSL.ProtocolLang

object ATMProtocol extends ProtocolLang with App{
  in("init")
  when("takeCard()") goto "CardIn"
  when("beginNewTransaction()") goto "init"
  in("CardIn")
  when("authorise()") goto
    "Authorised" at "true" or
    "ShouldEject" at "false"
  in("Authorised")
  when("giveMoney()") goto "ShouldEject"
  in("ShouldEject")
  when("eject()") goto "end"
  in("end")
  when("beginNewTransaction()") goto "init"
  end()
}
```

```
import compilerPlugin.Typestate

@Typestate("ATMProtocol")
class ATM(){...}
```

A prototype tool has been implemented for Scala[1]

Implements the core language features covered in this presentation as a Scala compiler plugin

Allows you to define protocols in a DSL and add them to classes with annotations

Emits **compile time errors and warnings** when program does not adhere to the specified protocol

[1] https://github.com/Aliceravier/Scala-Mungo

Conclusion and Future Work

The contributions of this work:

- A global typestate checking approach for **aliased objects**
- An **implemented** prototype illustrating the concepts in practice

Interesting future work include:

- Split the program into non-linear and linear parts and typecheck each individually, reintroducing compositionality for linear part

Thanks for listening!