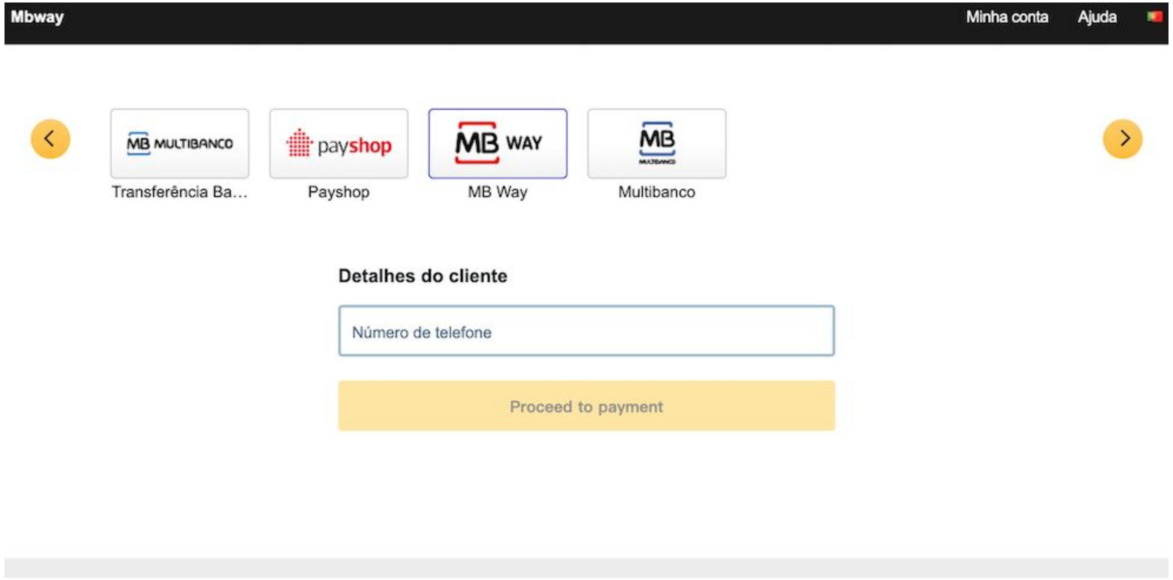
Java Typestate Checker

Lorenzo Bacchiani[1], Mario Bravetti[1], Marco Giunti[2], João Mota[2], António Ravara[2]

[1] University of Bologna, Italy
[2] NOVA LINCS and NOVA School of Science and Technology, Portugal

Protocols are common

- Example: MBWay
- Payment flow:
  - Choose MBWay and enter the mobile phone number…

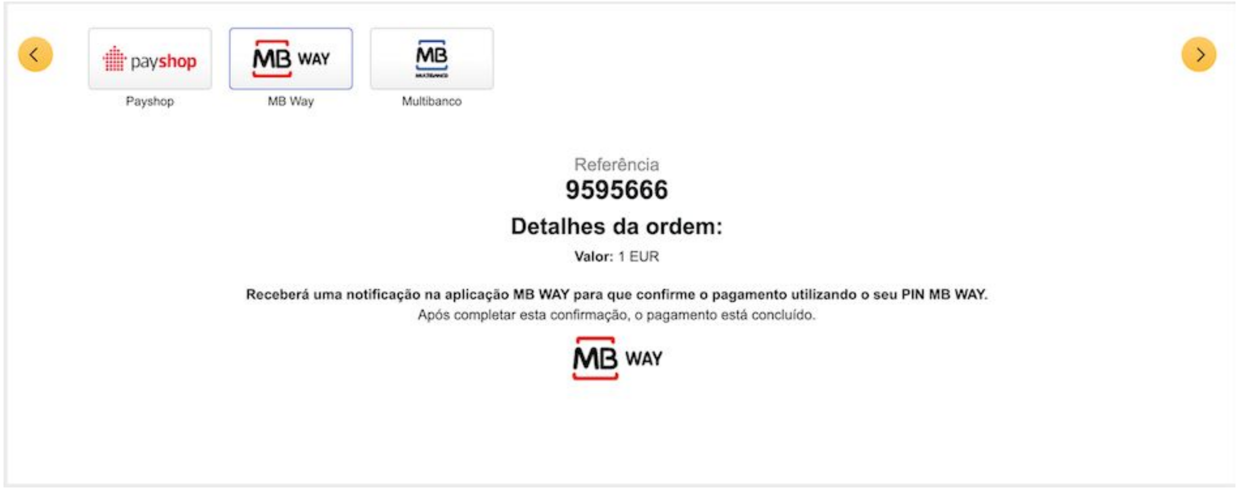


Detalhes do cliente

Número de telefone

Protocols are common

- Example: MBWay
- Payment flow:
    - The payment reference is generated…

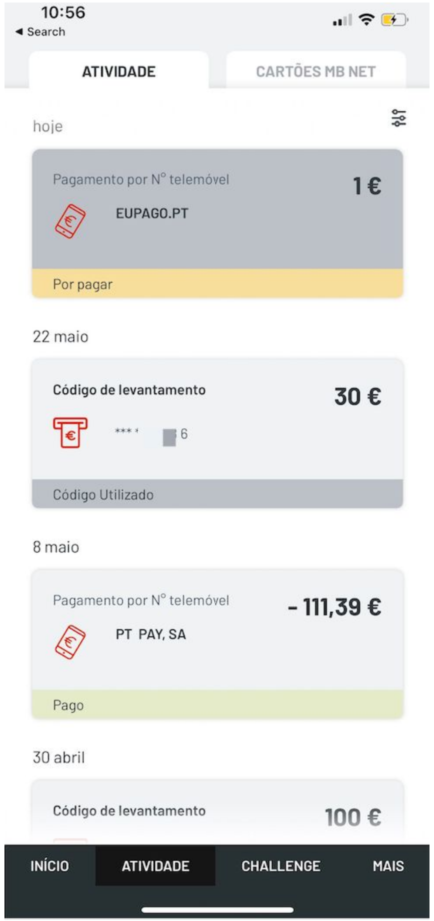


Referência
9595666
Detalhes da ordem:
Valor: 1 EUR

Receberá uma notificação na aplicação MB WAY para que confirme o pagamento utilizando o seu PIN MB WAY.
Após completar esta confirmação, o pagamento está concluído.

Protocols are common

- Example: MBWay
- Payment flow:
    - The order notification is sent to the MBWay app…


10:56
◂ Search

ATIVIDADE | CARTÕES MB NET

hoje

Pagamento por Nº telemóvel 1 €
EUPAGO.PT
Por pagar

22 maio

Código de levantamento 30 €
*** [redacted] 6
Código Utilizado

8 maio

Pagamento por Nº telemóvel - 111,39 €
PT PAY, SA
Pago

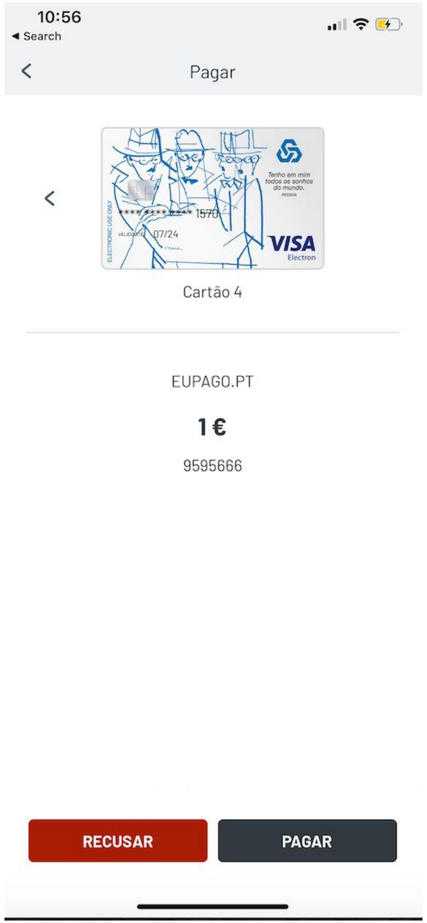
30 abril

Código de levantamento 100 €

INÍCIO | ATIVIDADE | CHALLENGE | MAIS

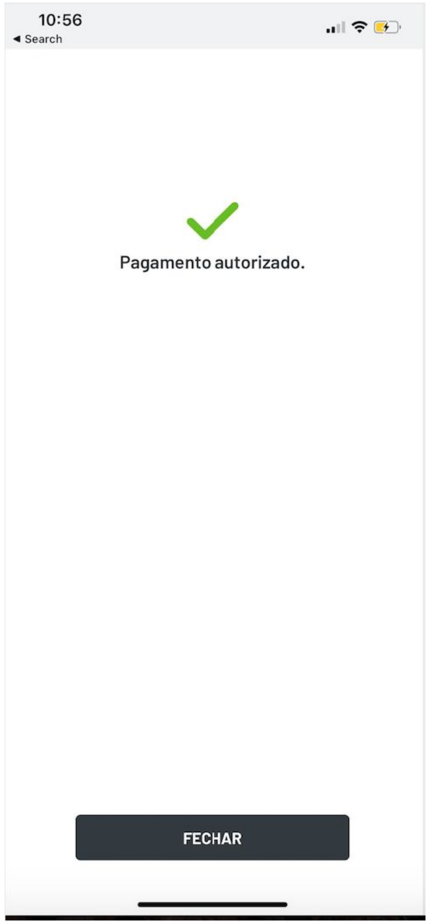
Protocols are common

- Example: MBWay
- Payment flow:
    - Confirm the payment in the MBWay app…

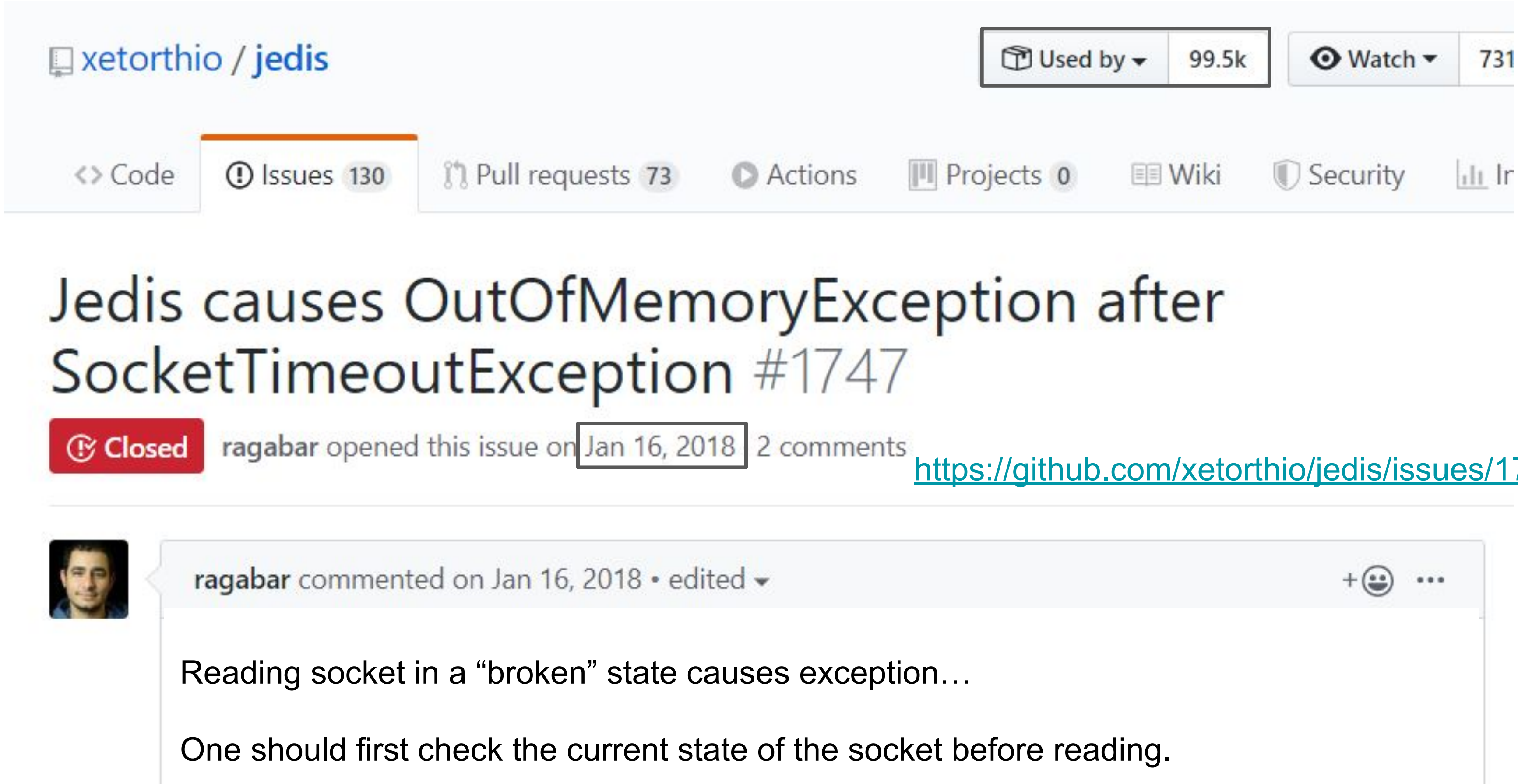


Protocols are common

- Example: MBWay
- Payment flow:
  - Payment is successful!



Development is hard



Jedis causes OutOfMemoryException after SocketTimeoutException #1747

Closed ragabar opened this issue on Jan 16, 2018 · 2 comments

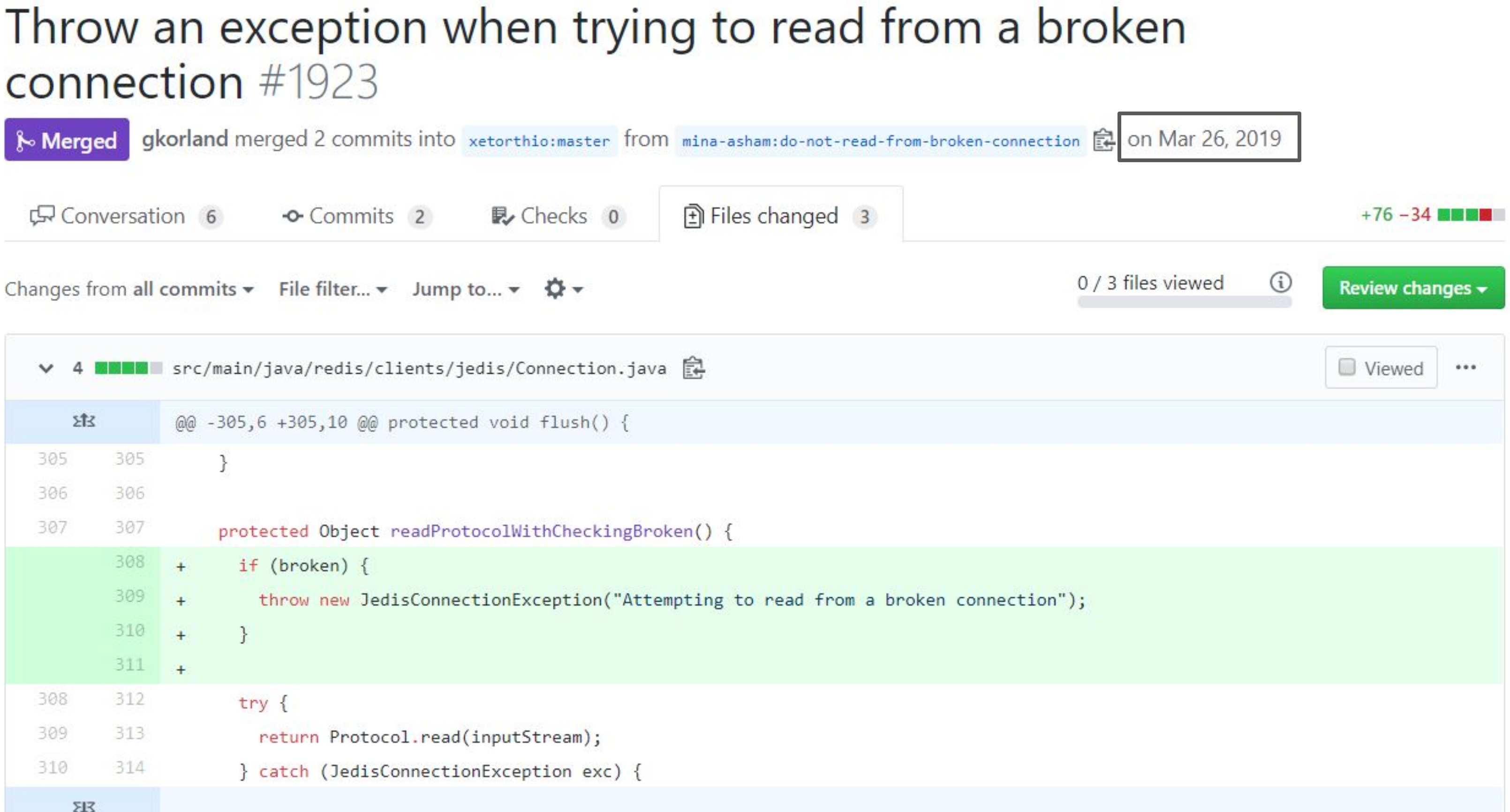
https://github.com/xetorthio/jedis/issues/1747

ragabar commented on Jan 16, 2018 • edited

Reading socket in a “broken” state causes exception…

One should first check the current state of the socket before reading.

Development is hard



Throw an exception when trying to read from a broken connection #1923

Merged gkorland merged 2 commits into `xetorthio:master` from `mina-asham:do-not-read-from-broken-connection` on Mar 26, 2019

4 src/main/java/redis/clients/jedis/Connection.java

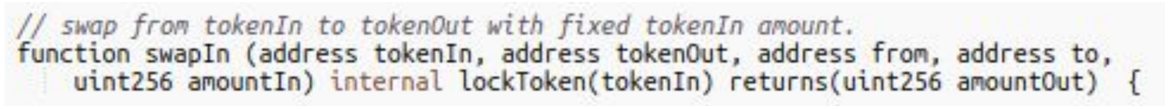
```
@@ -305,6 +305,10 @@ protected void flush() {
   }

   protected Object readProtocolWithCheckingBroken() {
+    if (broken) {
+      throw new JedisConnectionException("Attempting to read from a broken connection");
+    }
+
     try {
       return Protocol.read(inputStream);
     } catch (JedisConnectionException exc) {
```

Development is hard

December 2, 2021:

Really Stupid 'Smart Contract' Bug Let Hackers Steal $31 Million In Digital Coin

User could send tokens to themselves and increase their balance!



```
// swap from tokenIn to tokenOut with fixed tokenIn amount.
function swapIn (address tokenIn, address tokenOut, address from, address to,
    uint256 amountIn) internal lockToken(tokenIn) returns(uint256 amountOut)  {
```

Someone forgot an if statement: *tokenIn != tokenOut*

Only incompetent programmers do this, right?

What to do?

“Program testing can be used to show the presence of bugs, but never to show their absence!”

Edsger W. Dijkstra

Turing award in 1972: “The humble programmer”

“If you want more effective programmers, you will discover that they should not waste their time debugging, they should not introduce the bugs to start with.”

What to do?

The verified software initiative

C.A.R. Hoare*, Jayadev Misra, Gary T. Leavens, and Natarajan Shankar

*Turing award in 1980: “The Emperor's Old Clothes”

The opening of the Manifesto: “We propose an ambitious and long-term research program toward the construction of error-free software systems.”

Science fiction?

What to do?

Google announces KataOS

October 14, 2022 on Google Open Source Blog

“a **provably secure** platform that's optimized for embedded devices that run ML applications.”

Based on Rust, on top of seL4:

- Rust: compile-time memory and thread safety
  (no null-deref, use-after-free, double-free; no races)
- seL4: “The world's most highly assured OS kernel.”

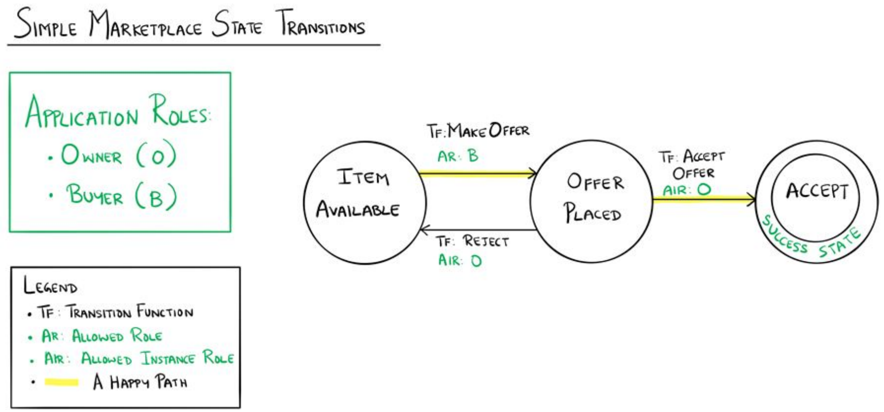
The issue

Detecting errors and vulnerabilities in software is crucial for the industry.

It is not enough to rule out **data-errors** (i.e. if types are compatible).

We also need the **behavior of programs to be correct**!

Marketplace Smart Contract



Example from Microsoft Azure's GitHub:
github.com/Azure-Samples/blockchain

- Protocols are usually described:
  - in natural languages;
  - or drawn as state machines;
- Code requires **defensive programming** to check the current state.

```
function AcceptOffer() public {
    if ( msg.sender != InstanceOwner ) { revert(); }
    State = StateType.Accepted;
}
```

Very prone to bugs: AcceptOffer does not check the state!

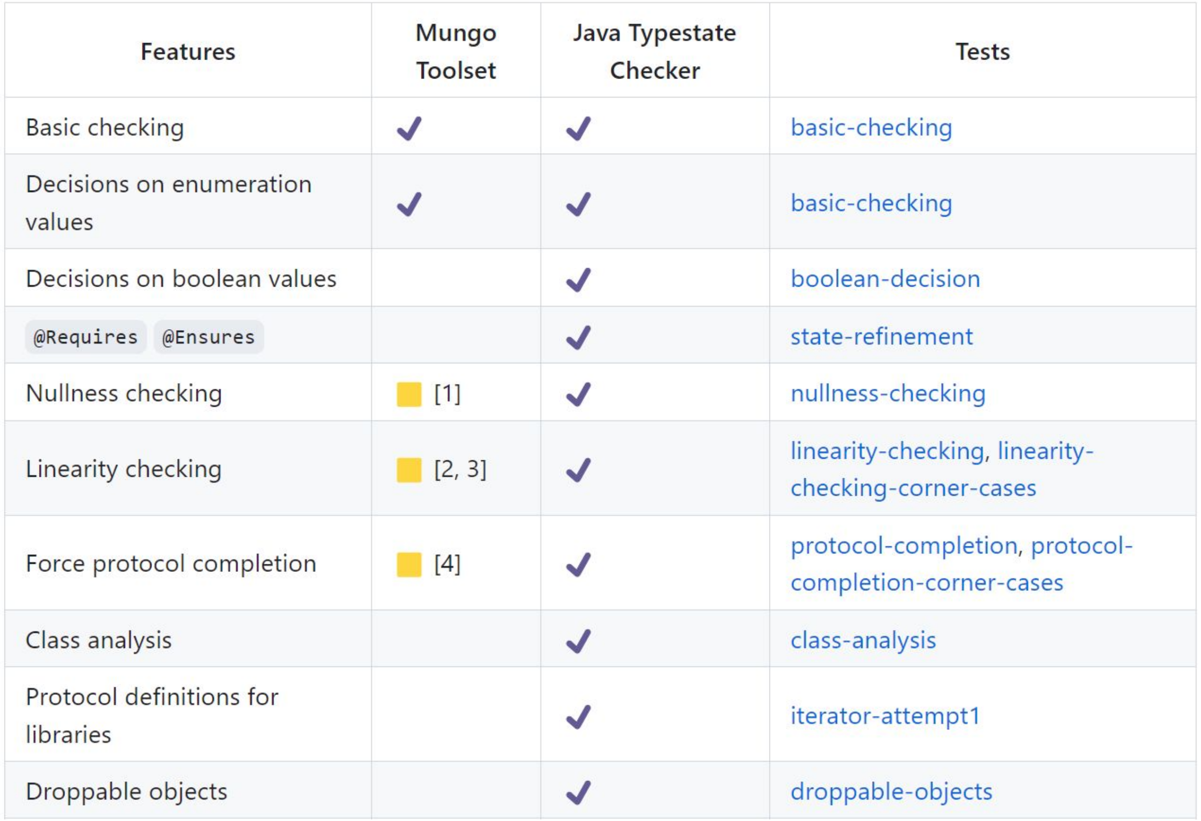
Java Typestate Checker

- **Statically** checks Java code where objects are associated with **typestates**;
- **Typestates** describe the methods available in each protocol state;
- Built on top of the **Checker Framework**.
- Guarantees:
    - Protocol **compliance**;
    - Protocol **completion** (assuming that program terminates);
    - Null pointer exception absence;
    - Subclasses' instances respect the protocol of their superclasses.

github.com/jdmota/java-typestate-checker

Reimplementation of Mungo: **www.dcs.gla.ac.uk/research/mungo/**

Comparison table: **github.com/jdmota/java-typestate-checker/wiki/Mungo-comparison**



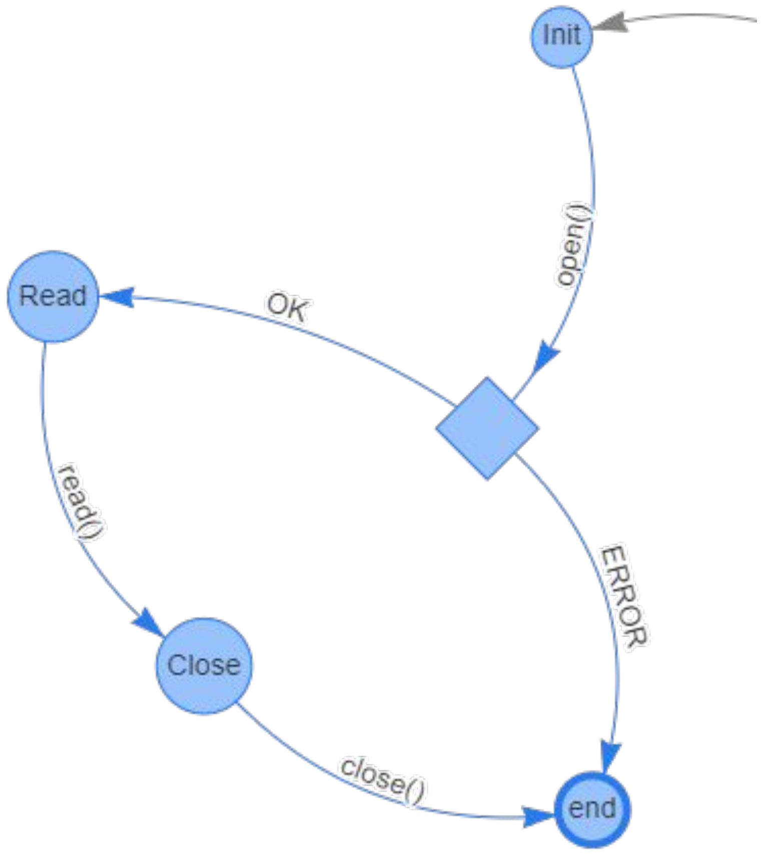
Features	Mungo Toolset	Java Typestate Checker	Tests
Basic checking	✓	✓	basic-checking
Decisions on enumeration values	✓	✓	basic-checking
Decisions on boolean values		✓	boolean-decision
`@Requires` `@Ensures`		✓	state-refinement
Nullness checking	🟨 [1]	✓	nullness-checking
Linearity checking	🟨 [2, 3]	✓	linearity-checking, linearity-checking-corner-cases
Force protocol completion	🟨 [4]	✓	protocol-completion, protocol-completion-corner-cases
Class analysis		✓	class-analysis
Protocol definitions for libraries		✓	iterator-attempt1
Droppable objects		✓	droppable-objects

Deterministic Object Automata

Simple file reader protocol's happy-path:

`open()` returning `OK`
`read()`
`close()`

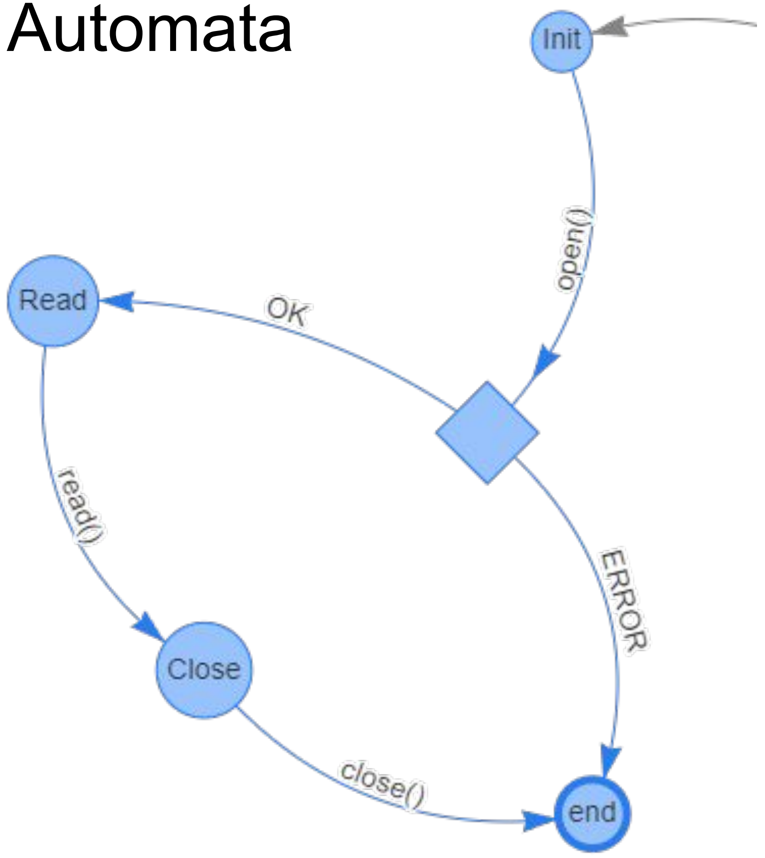
typestate-editor.github.io



Typestate vs Deterministic Object Automata

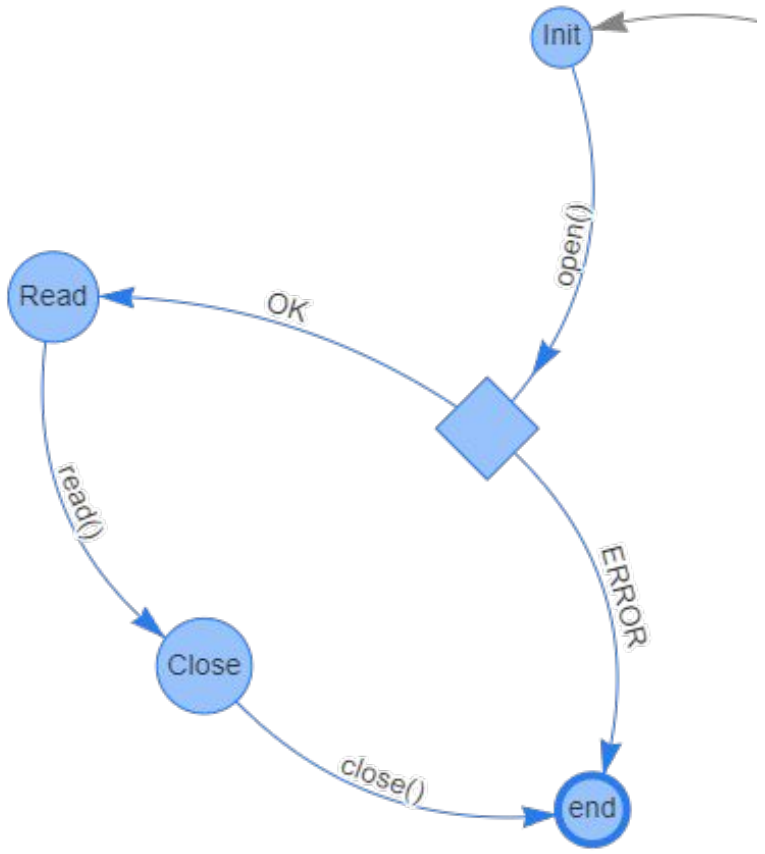
```
typestate FileProtocol {
  Init = {
    FileStatus open(): <OK: Read, ERROR: end>
  }
  Read = {
    String read(): Close
  }
  Close = {
    void close(): end
  }
}
```

typestate-editor.github.io



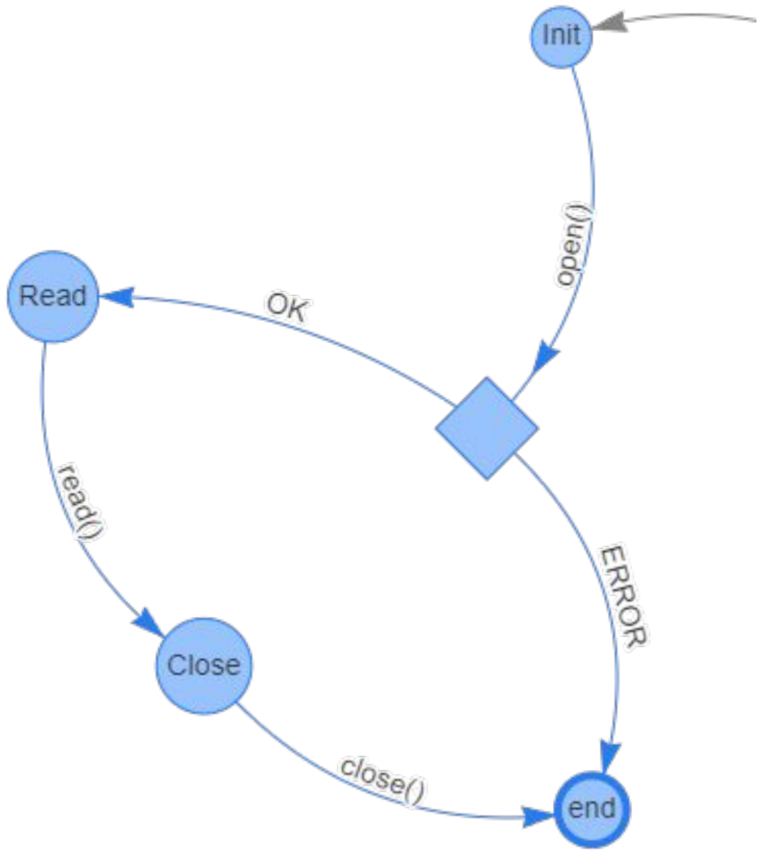
Protocol compliance

```
File f = new File();

System.out.println(f.read());
// error: Cannot call [read] on State{File, Init}
```



Protocol compliance

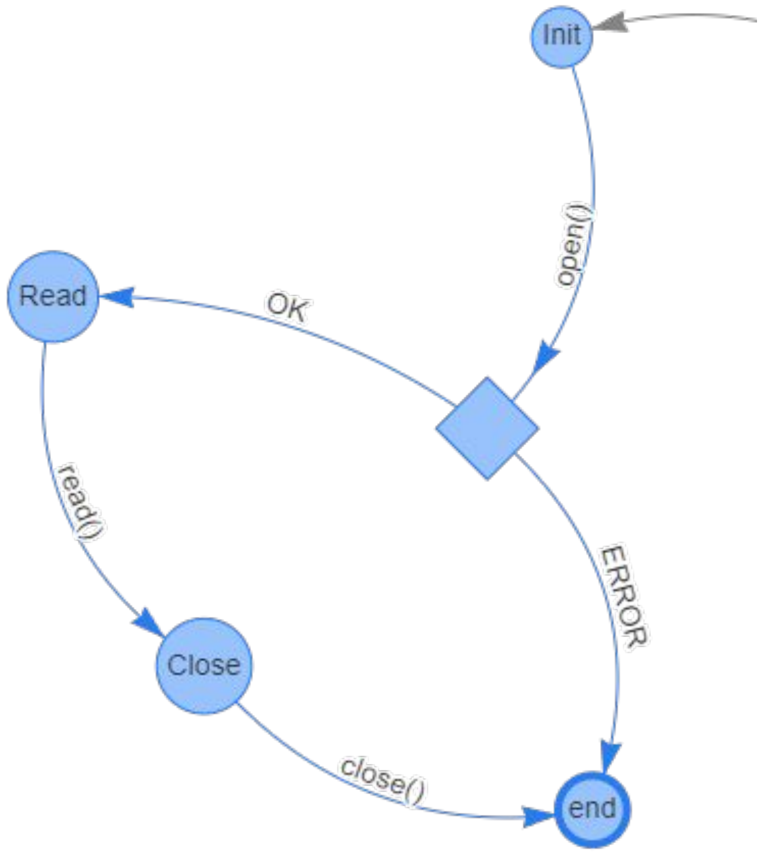
```
   File f = new File();

🟩 switch (f.open()) {
🟩   case OK:
       System.out.println(f.read());
🟩     break;
🟩   case ERROR:
🟩     break;
🟩 }
```



Protocol completion

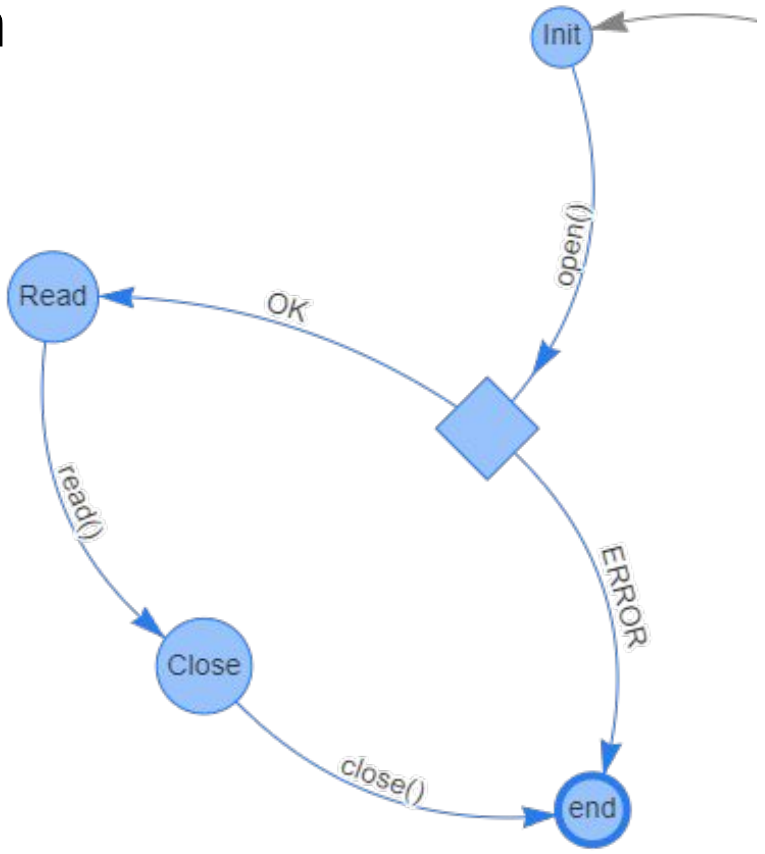
```
  File f = new File();

🟩 switch (f.open()) {
🟩   case OK:
       System.out.println(f.read());
🟩     break;
🟩   case ERROR:
🟩     break;
🟩 }
```

```
// error: [f] did not complete its protocol
(found: State{File, Close} | State{File, end})
```



Protocol compliance & completion

```
File f = new File();

switch (f.open()) {
  case OK:
    System.out.println(f.read());
🟩  f.close();
    break;
  case ERROR:
    break;
}

// OK!
```



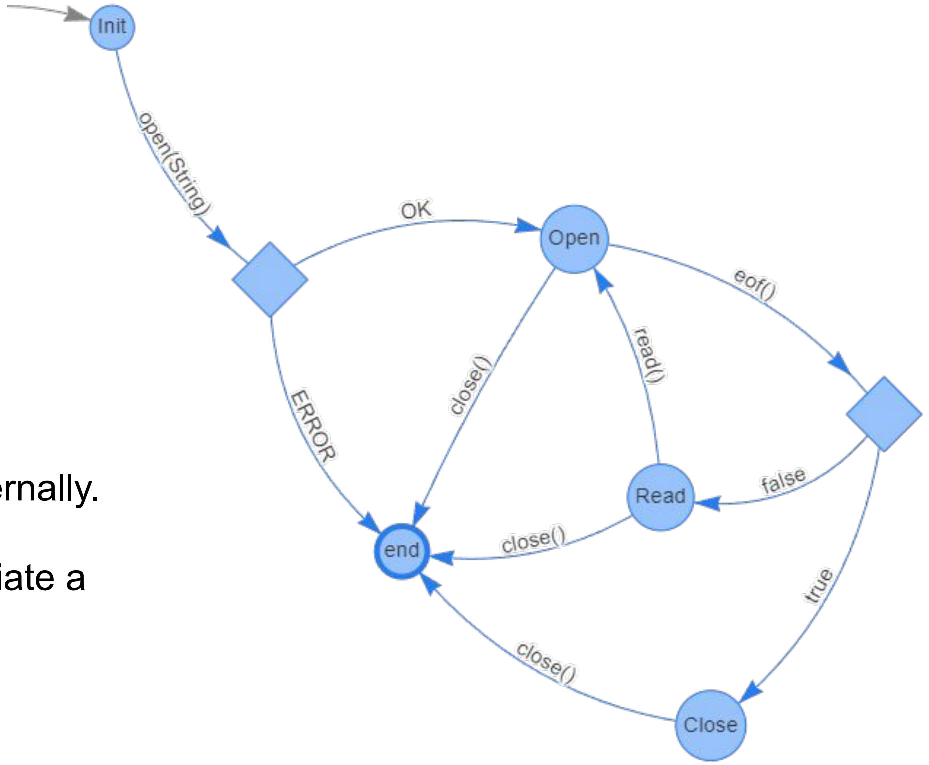
Line Reader

Protocol's happy-path:

`open()` returning `OK`
`read()` while `!eof()`
`close()`

The `LineReader` uses `java.io.FileReader` internally.

To better check its implementation, we can associate a protocol with `java.io.FileReader` as well!

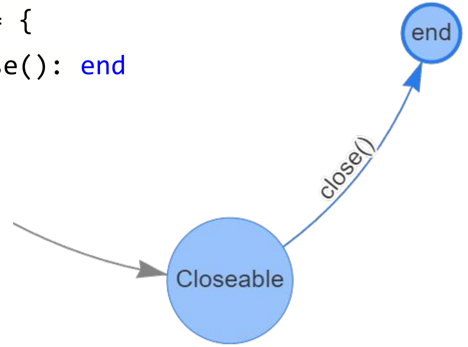


Protocols for library classes

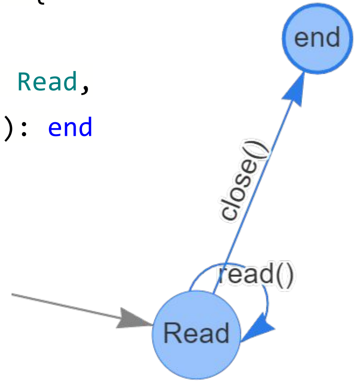
Configuration file

```
java.lang.AutoCloseable=AutoCloseable.protocol
java.io.Reader=Reader.protocol
```

```
typestate AutoCloseable {
  Closeable = {
    void close(): end
  }
}
```



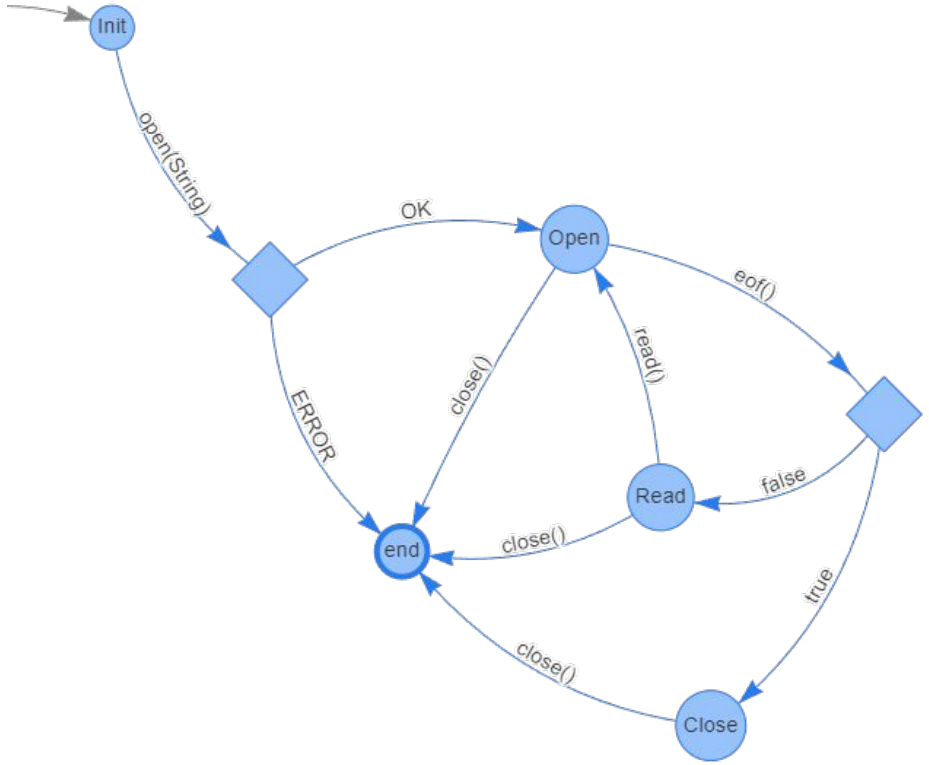
```
typestate Reader {
  Read = {
    int read(): Read,
    void close(): end
  }
}
```



Since `java.io.FileReader` extends `java.io.Reader` (which implements `java.lang.AutoCloseable`), it will inherit the protocol of `java.io.Reader`

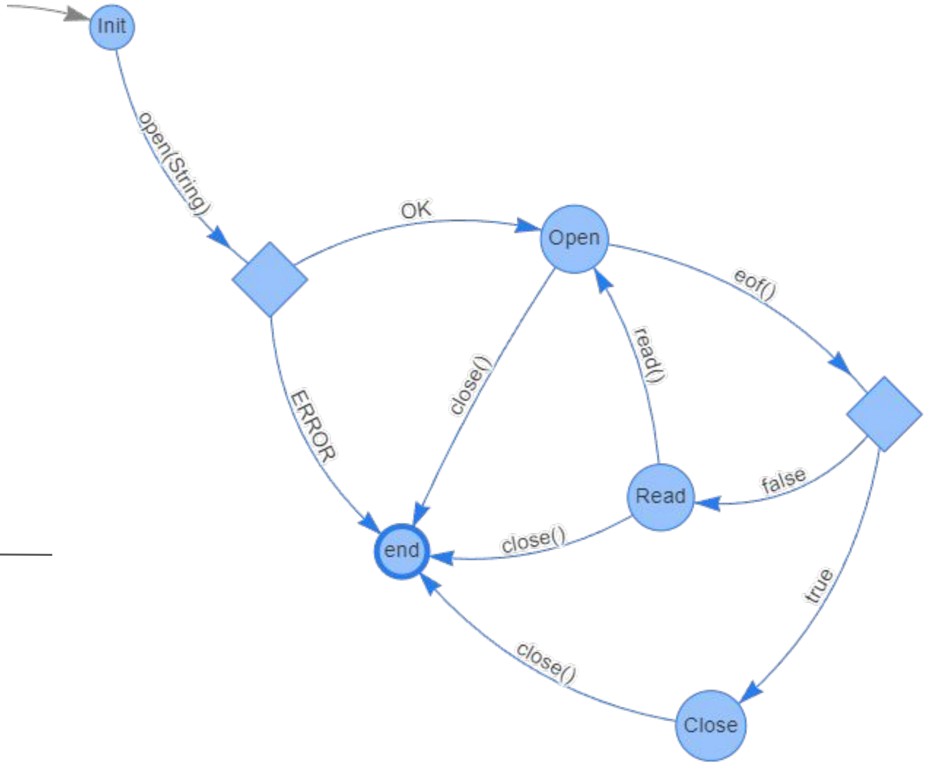
Line Reader

```
@Typestate("LineReader")
public class LineReader {

  private @Nullable FileReader file;
  private int curr;
  public Status open(String filename) {
    /* ... */
    curr = 0; /* ... */
  }
  public String read() {
    /* ... */ curr = file.read(); /* ... */
  }

  public boolean eof() { return curr == -1; }
  public void close() {}
}
```



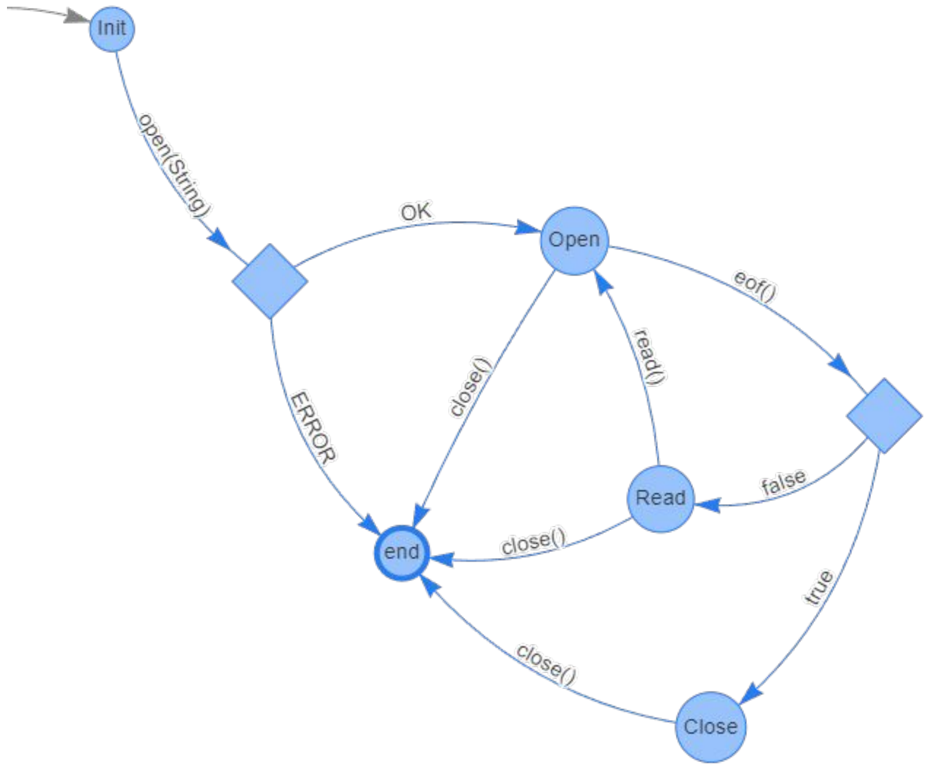
Line Reader

```
@Typestate("LineReader")
public class LineReader {

  private @Nullable FileReader file;
  private int curr;
  public Status open(String filename) {
    /* ... */
    curr = 0; /* ... */
  }
  public String read() {
    /* ... */ curr = file.read(); /* ... */
    // error: Cannot call read on null
  }
  public boolean eof() { return curr == -1; }
  public void close() {}
}
```



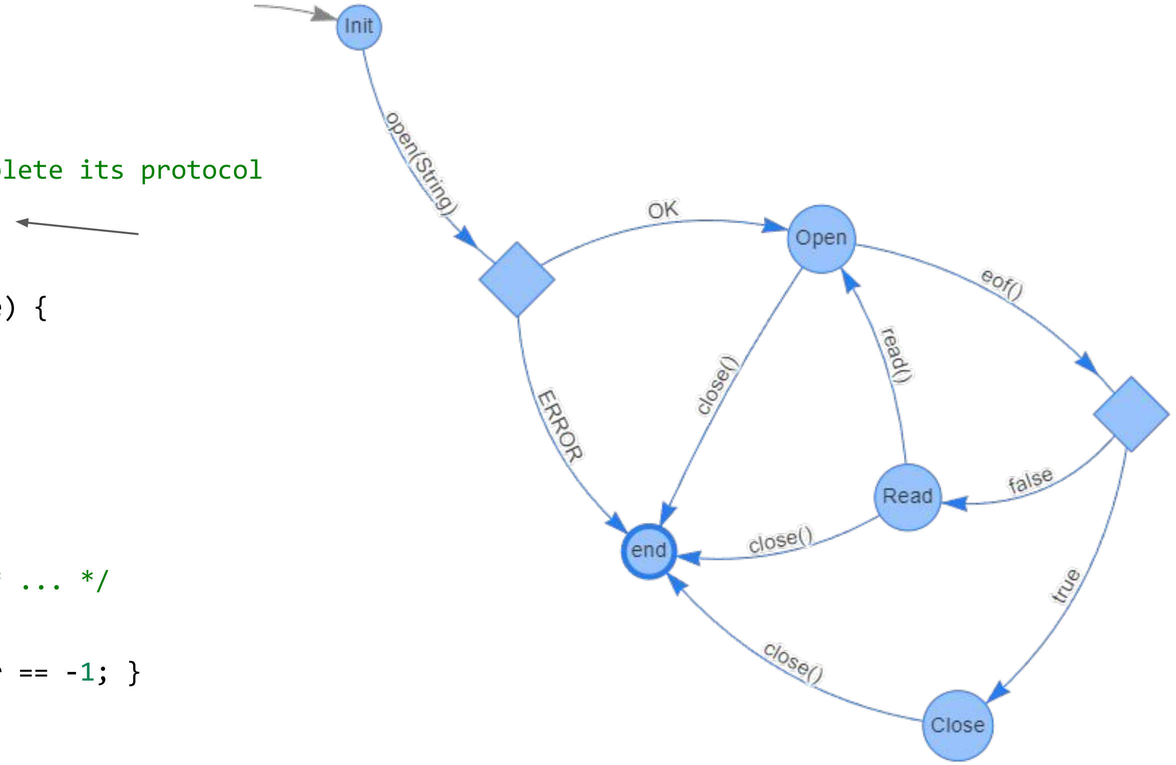
Line Reader

```
@Typestate("LineReader")
public class LineReader {

  private @Nullable FileReader file;
  private int curr;
  public Status open(String filename) {
    /* ... */
🟩    file = new FileReader(filename);
🟩    curr = file.read(); /* ... */
  }
  public String read() {
    /* ... */ curr = file.read(); /* ... */
  }
  public boolean eof() { return curr == -1; }
  public void close() {}
}
```



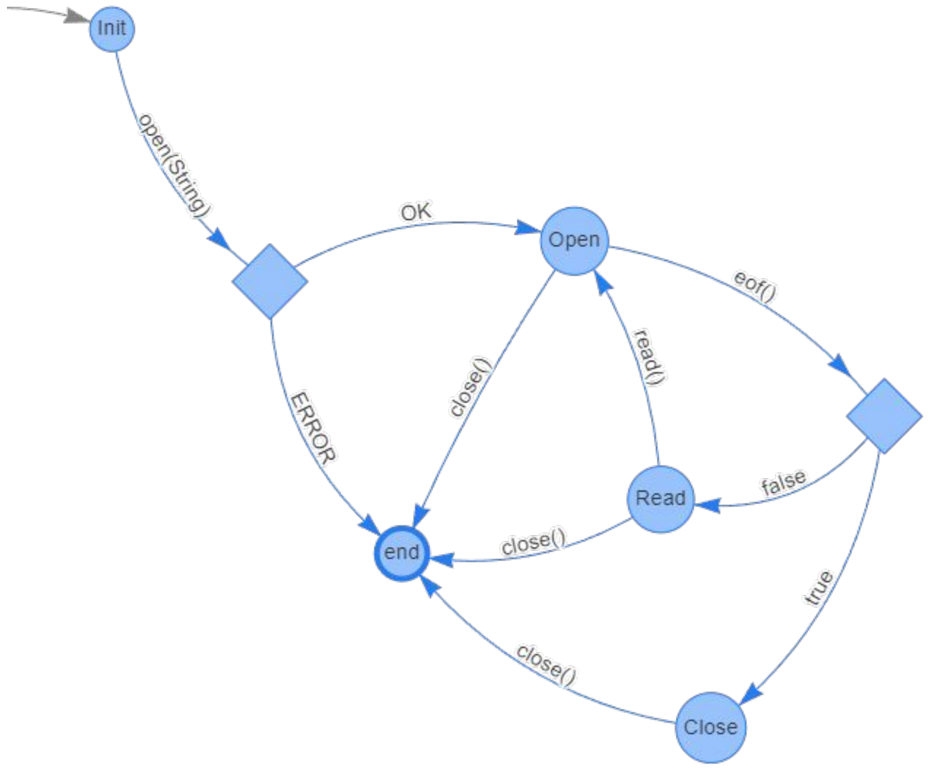
Line Reader

```
@Typestate("LineReader")
public class LineReader {
  // error: [this.file] did not complete its protocol
  private @Nullable FileReader file;
  private int curr;
  public Status open(String filename) {
    /* ... */
    file = new FileReader(filename);
    curr = file.read(); /* ... */
  }
  public String read() {
    /* ... */ curr = file.read(); /* ... */
  }
  public boolean eof() { return curr == -1; }
  public void close() {}
}
```



Line Reader

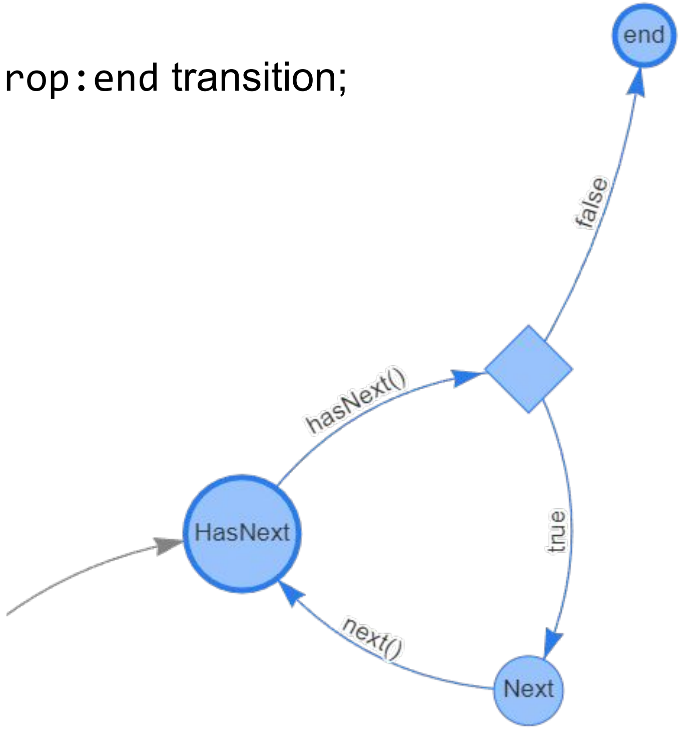
```
@Typestate("LineReader")
public class LineReader {

  private @Nullable FileReader file;
  private int curr;
  public Status open(String filename) {
    /* ... */
    file = new FileReader(filename);
    curr = file.read(); /* ... */
  }
  public String read() {
    /* ... */ curr = file.read(); /* ... */
  }
  public boolean eof() { return curr == -1; }
🟩 public void close() { file.close(); }
}
```



Droppable states

- One can mark other states as final with the special `drop:end` transition;
- For example, the `HasNext` state is final.

```
typestate Iterator {
  HasNext = {
    boolean hasNext(): <true: Next, false: end>,
🟩    drop: end
  }

  Next = {
    Object next(): HasNext
  }
}
```



Digital Locker

From Azure-Samples: *The Digital Locker application expresses a workflow of sharing digitally locked files where the owner of the files controls the access to these files.*

Roles:

- Owner: The owner of the digital asset.
- BankAgent: The keeper of the digital asset.
- ThirdPartyRequestor: A person requesting access to the digital asset.

github.com/Azure-Samples/blockchain

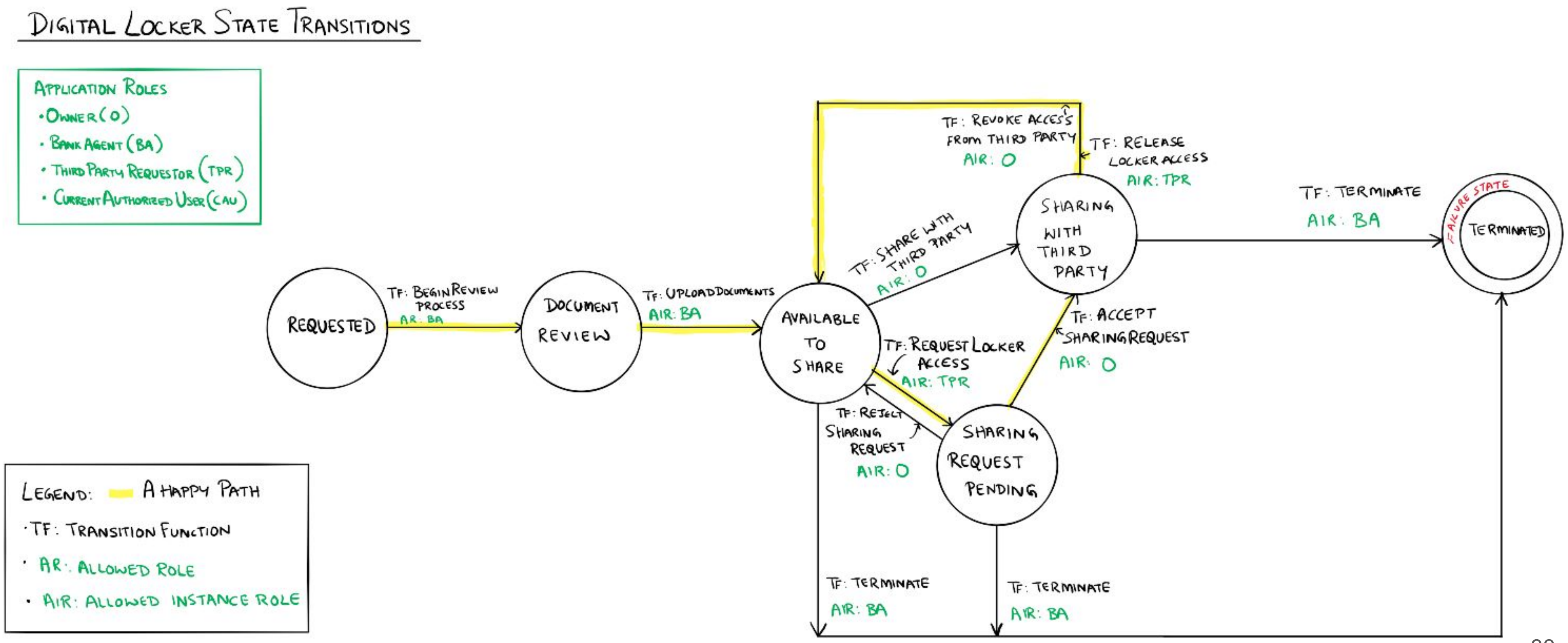
Digital Locker

From Azure-Samples: *The Digital Locker application expresses a workflow of sharing digitally locked files where the owner of the files controls the access to these files.*

States:

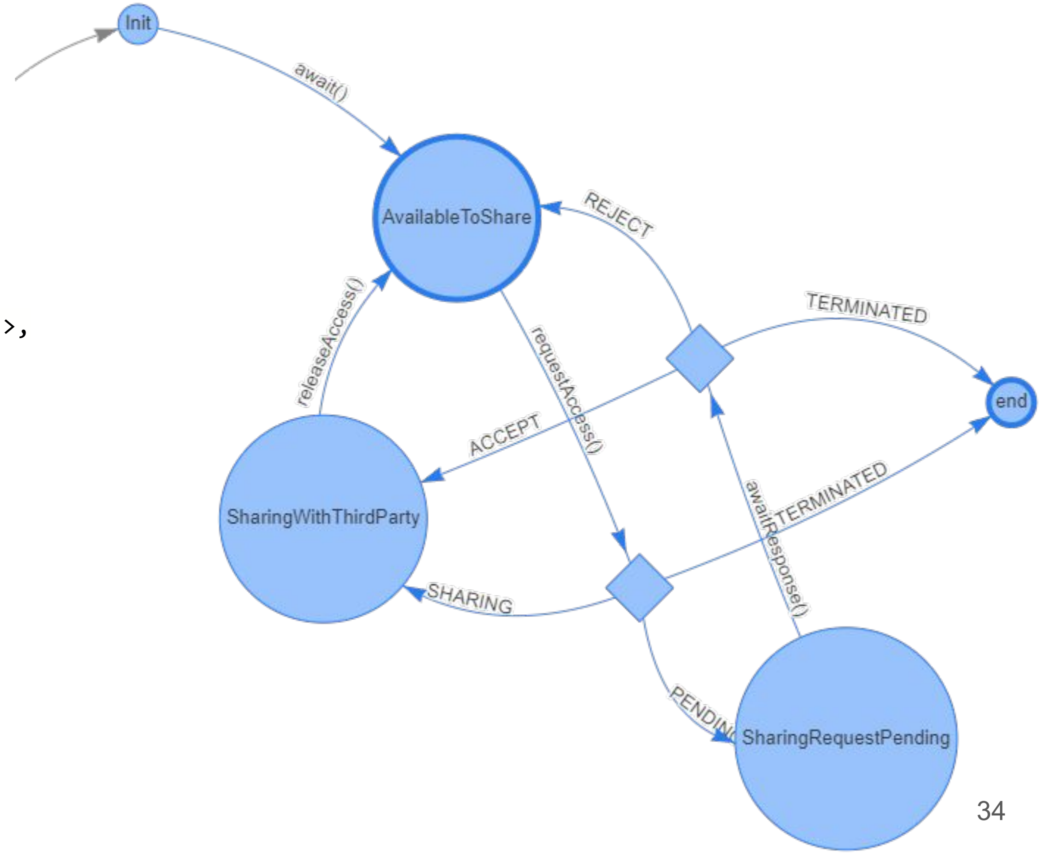
- Requested: Initial state;
- DocumentReview: The bank agent has reviewed the owner's request;
- AvailableToShare: The bank agent has uploaded the digital asset and the digital asset is available for sharing;
- SharingRequestPending: The owner is reviewing a third party's request to access the digital asset;
- SharingWithThirdParty: The third party is accessing the asset.

Digital Locker



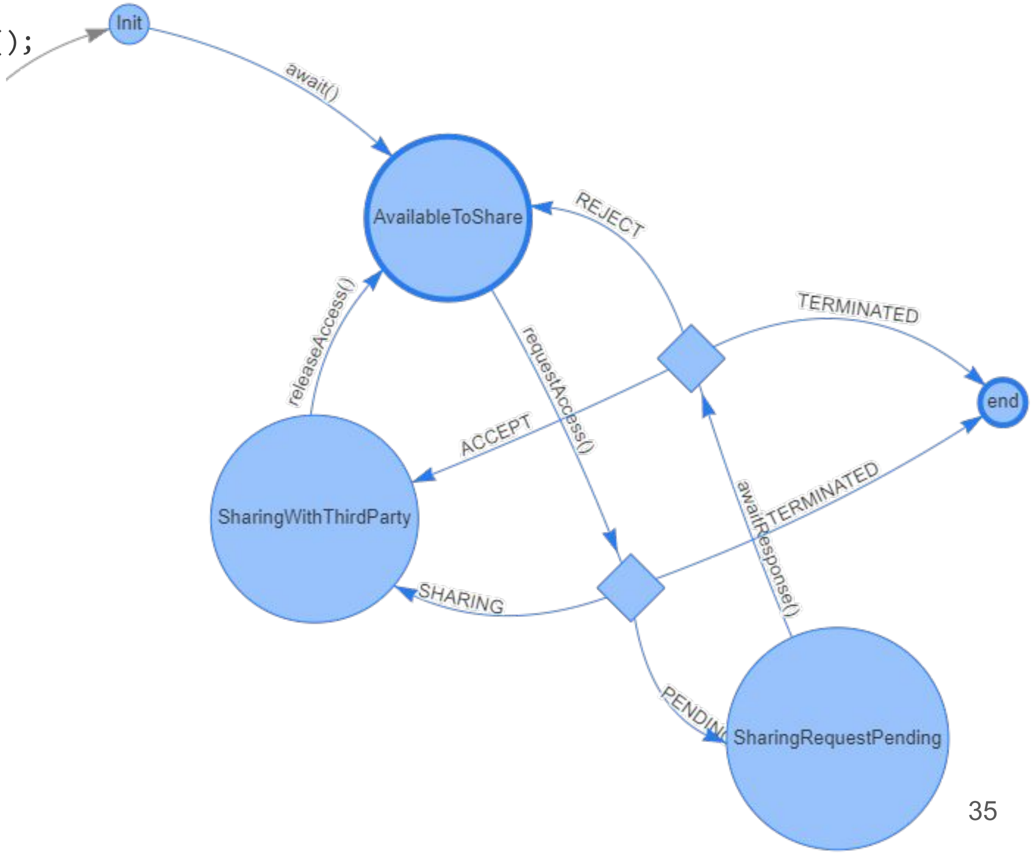
Digital Locker: ThirdPartyRequestor

```
typestate ThirdPartyRequestor {
  Init = {
    void await(): AvailableToShare
  }
  AvailableToShare = {
    SharingState requestAccess():
      <PENDING: SharingRequestPending,
       SHARING: SharingWithThirdParty, TERMINATED: end>,
    drop: end
  }
  SharingRequestPending = {
    OwnerResponse awaitResponse():
      <ACCEPT: SharingWithThirdParty,
       REJECT: AvailableToShare, TERMINATED: end>
  }
  SharingWithThirdParty = {
    void releaseAccess(): AvailableToShare
  }
}
```



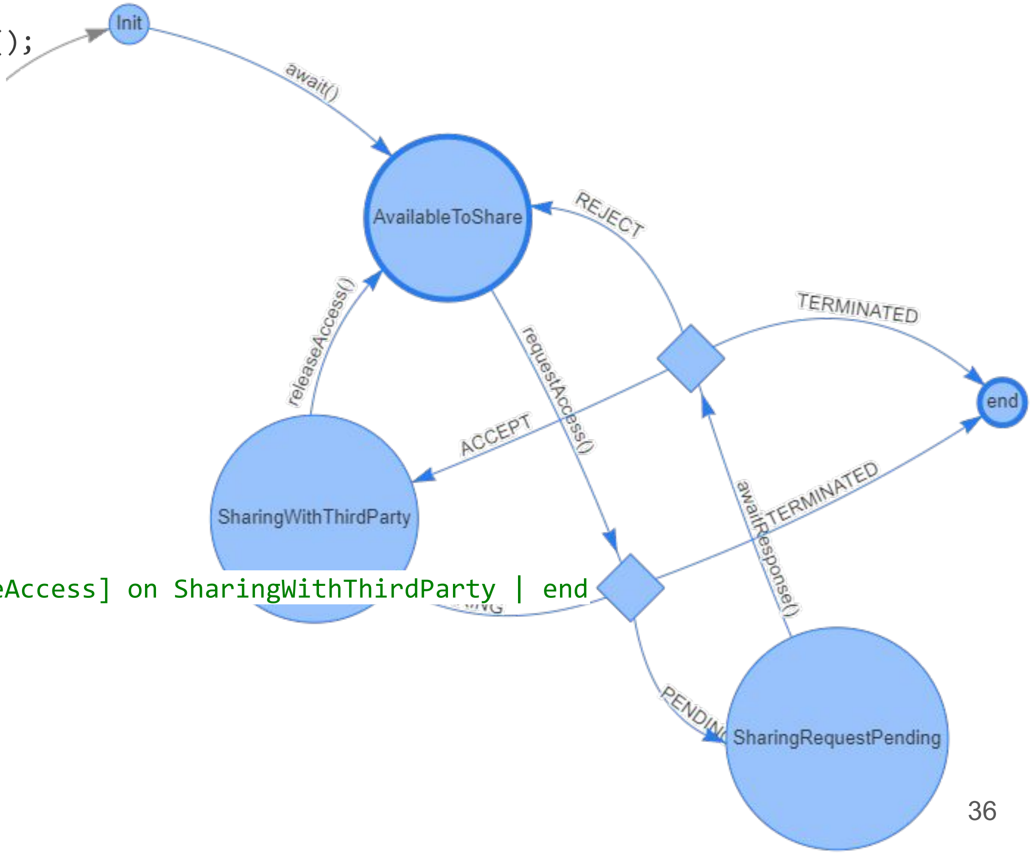
Digital Locker: ThirdPartyRequestor

```
ThirdPartyRequestor t = new ThirdPartyRequestor();
t.await();
loop: while (true) {
  switch (t.requestAccess()) {
    case PENDING:
      switch (t.awaitResponse()) {
        case REJECT:
        case TERMINATED:
          break;
        case ACCEPT:
      }
    case SHARING:
      t.releaseAccess();
      break;
    case TERMINATED:
      return;
  }
}
```



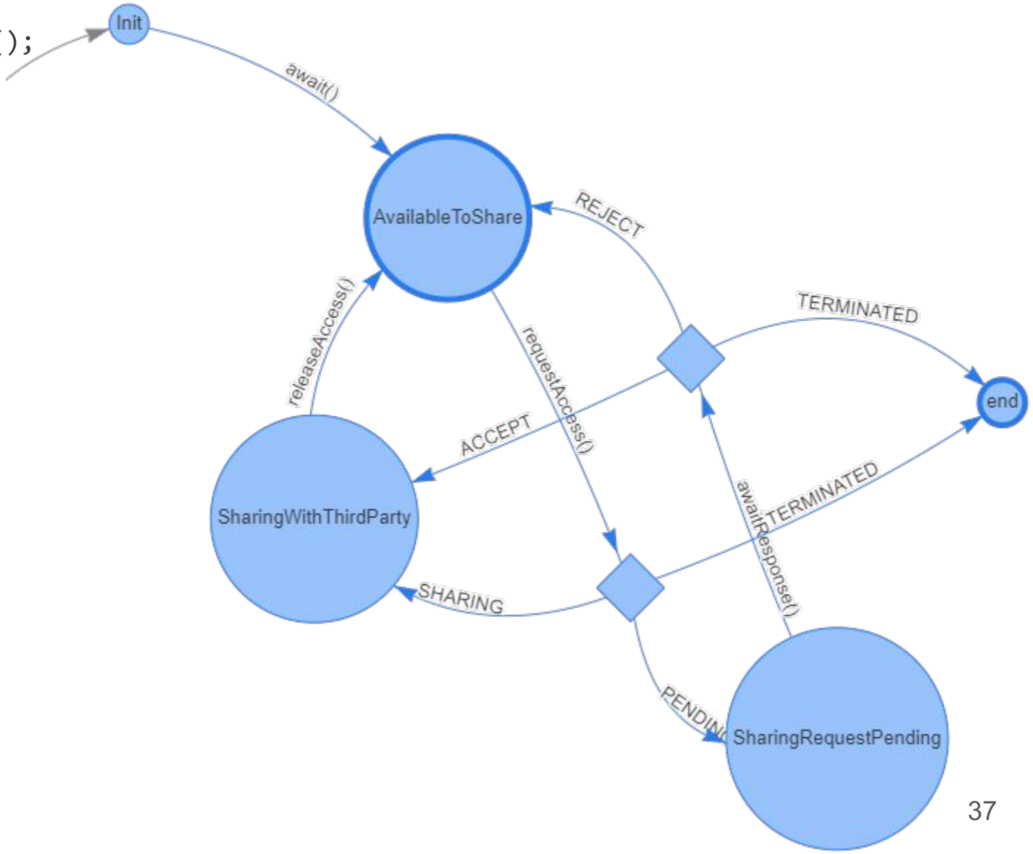
Digital Locker: ThirdPartyRequestor

```
ThirdPartyRequestor t = new ThirdPartyRequestor();
t.await();
loop: while (true) {
  switch (t.requestAccess()) {
    case PENDING:
      switch (t.awaitResponse()) {
        case REJECT:
        case TERMINATED:
          break;
        case ACCEPT:
      }
    case SHARING:
      t.releaseAccess(); // Cannot call [releaseAccess] on SharingWithThirdParty | end
      break;
    case TERMINATED:
      return;
  }
}
```



Digital Locker: ThirdPartyRequestor

```
ThirdPartyRequestor t = new ThirdPartyRequestor();
t.await();
loop: while (true) {
  switch (t.requestAccess()) {
    case PENDING:
      switch (t.awaitResponse()) {
        case REJECT:
        case TERMINATED:
          break loop;
        case ACCEPT:
      }
    case SHARING:
      t.releaseAccess(); // OK
      break;
    case TERMINATED:
      return;
  }
}
```



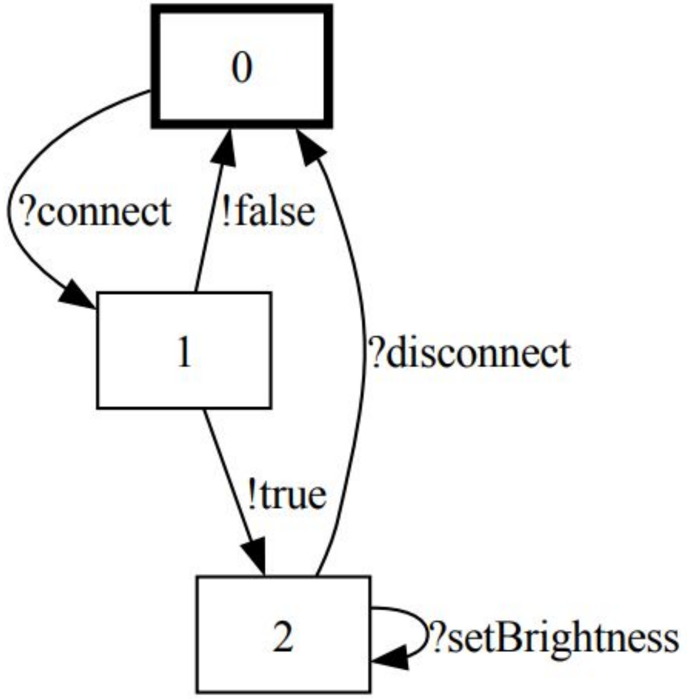
Subtyping

- We leverage on the **synchronous subtyping algorithm for session types** by Gay and Hole.
- One can check if one session type is a subtype of another using the *Session Subtyping Tool*: Lorenzo Bacchiani, Mario Bravetti, Julien Lange, and Gianluigi Zavattaro (2021).

github.com/LBacchiani/session-subtyping-tool

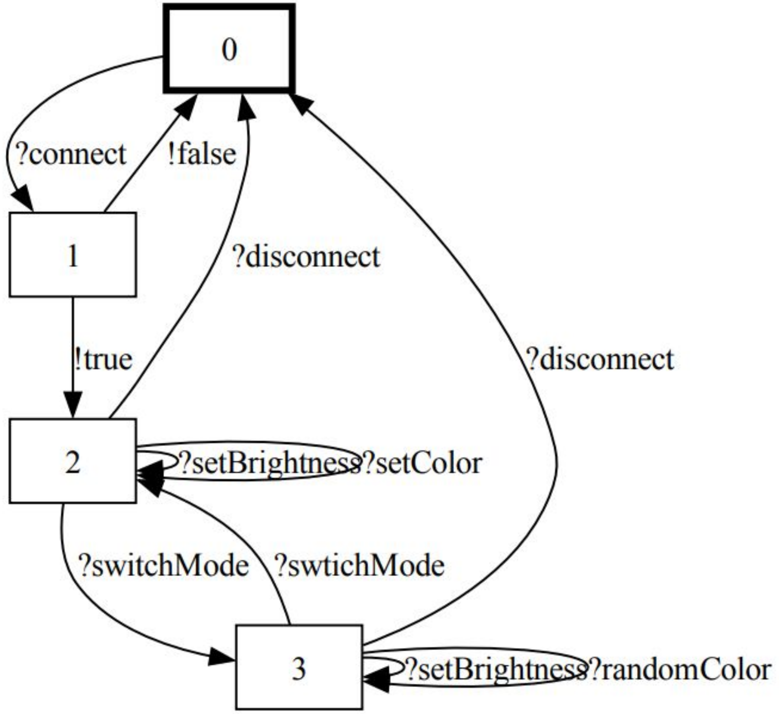
Bulb

```
typestate Bulb {
  DISCONN = {
    boolean connect(): <true: CONN, false: DISCONN>,
    drop: end
  }

  CONN = {
    void disconnect(): DISCONN,
    void setBrightness(int): CONN
  }
}
```

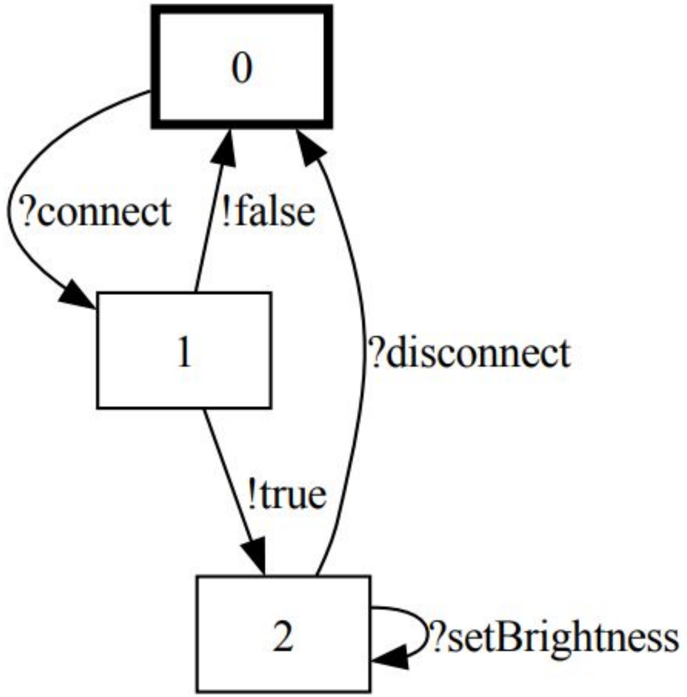


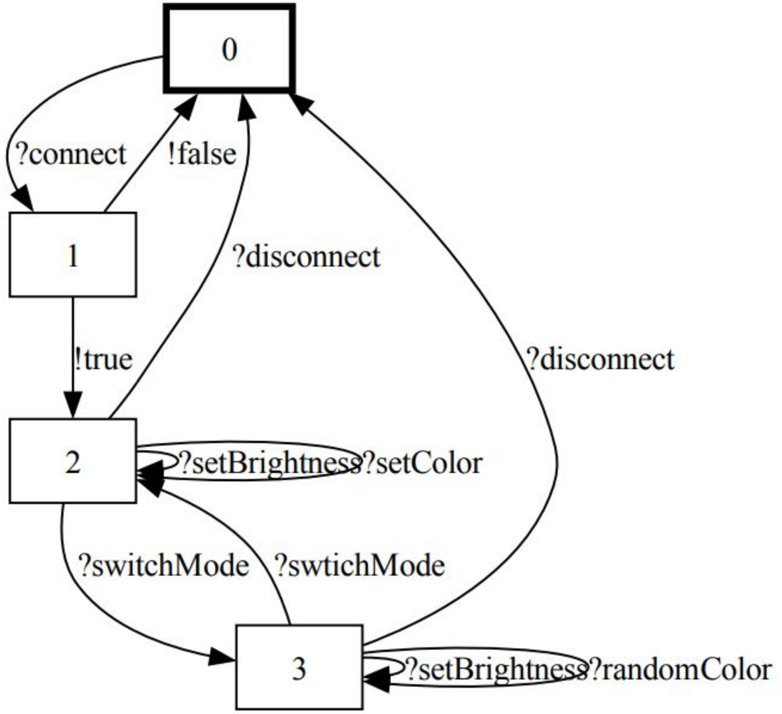
FunnyBulb

```
typestate FunnyBulb {
  DISCONN = {
    boolean connect(): <true: STD_CONN, false: DISCONN>,
    drop: end
  }
  STD_CONN = {
    void disconnect(): DISCONN,
    void setBrightness(int): STD_CONN,
    Mode switchMode(): <RND: RND_CONN, STD: STD_CONN>,
    void setColor(String): STD_CONN
  }
  RND_CONN = {
    void disconnect(): DISCONN,
    void setBrightness(int): RND_CONN,
    Mode switchMode(): <RND: RND_CONN, STD: STD_CONN>,
    void randomColor(): RND_CONN
  }
}
```



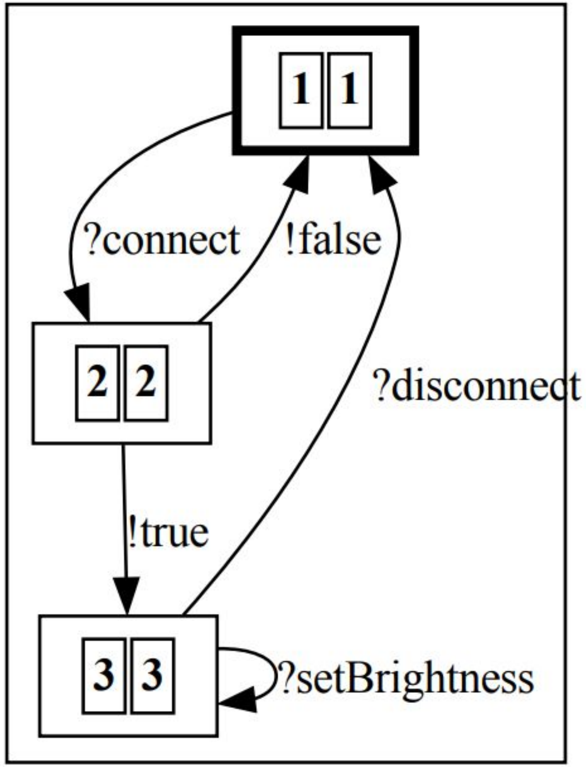
FunnyBulb extends Bulb





github.com/LBacchiani/session-subtyping-tool

FunnyBulb extends Bulb



Note: We can execute Gay and Hole's algorithm on any pair of states to check for subtyping. To check class compatibility, we run the algorithm on the initial states, as seen in the image.

github.com/LBacchiani/session-subtyping-tool

Polymorphic code

```
import jatyc.lib.Requires;

public class ClientCode {
  public static void example() {
    FunnyBulb f = new FunnyBulb(); // DISCONN
    while (!f.connect()) {} // STD_CONN
    f.switchMode(); // STD_CONN | RND_CONN
    setBrightness(f);
  }

  private static void setBrightness(@Requires("CONN") Bulb b) {
    if (b instanceof FunnyBulb && ((FunnyBulb) b).switchMode() == Mode.RND) {
      ((FunnyBulb) b).randomColor(); // RND_CONN
    }
    b.setBrightness(10); // CONN
    b.disconnect(); // end
  }
}
```

Limitations & Future work

- Objects with protocol must be used in a linear way;
- No overall support for generics;
- No support for dealing with state changes in the presence of exceptions;
- No support for multiple inheritance;
- No functional verification.

github.com/jdmota/java-typestate-checker

Conclusion

- API’s naturally have **protocols**;
- Traditional type systems **do not verify protocols** requiring:
  - Defensive programming;
  - The programmer imagining the protocol;
- Solution: associate a **protocol with the type** of the objects;
- **JaTyC** statically ensures:
  - Protocol **compliance** and **completion;**
  - Null pointer exception absence;
  - Subclasses' instances respect the protocol of their superclasses.

github.com/jdmota/java-typestate-checker

Thank you!

Any questions?

github.com/jdmota/java-typestate-checker