Residual Monitoring of Safety Properties

Prove what you can and monitor the leftovers

Matthew Dwyer

joint work with Rahul Purandare,
Sebastian Elbaum, Madeline Diep, and Alex Kinneer

Department of Computer Science and Engineering

Reality Check

Static analysis and verification techniques have advanced significantly in the past decade

It is easy to find papers reporting on techniques that are able to confirm properties of real programs

Reality : most fault-free programs cannot be confirmed as such

In that case …

Analyze, rank, and investigate false error reports

Reality : for real development scenarios the number of false error reports is too large (~10k reports [ICSE'08])

Run a focused stronger type of static analysis

Reality : even rich staged analyses still leave lots of "potential errors" (e.g., 5k->500 [ISSTA'06])

Reality : You usually run out of resources (time, people, expertise) before confirming correctness

Reactions to these realities

Most practitioners

- Hopefully we'll find some more bugs

Many researchers

- Practitioners are only interested in finding bugs!
- lots of great work pursuant to this "cop out"

Reactions to these realities

A few researchers

How can analysis/verification be done at runtime?
(e.g., Havelund, Rosu, …)

A few practitioners

Build two systems and let monitoring drive failover to a certified core system (e.g., Aircraft Control)

Runtime V&V Challenges

- Overhead
  - very low (<5-10%) for all properties combined
- Precision
  - no false error reports
- Recall
  - don’t miss any executed errors

Controlling overhead

A common overhead reduction approach

selectively observe the system

Won't work for path properties `(a;b)*`

system behavior: `ababab` `abbab`

sampled behavior: `ab bab` `ab ab`

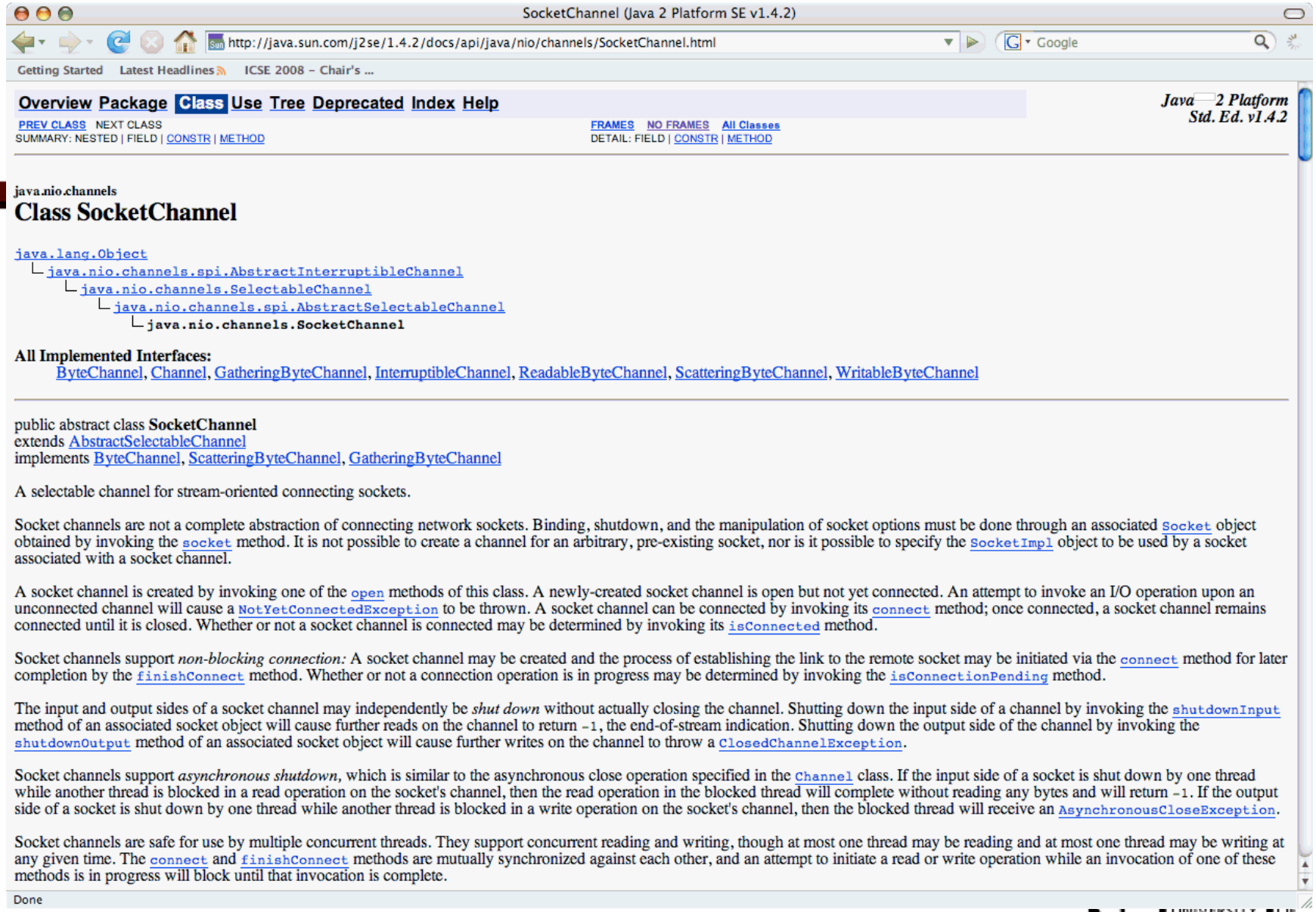
In this talk …

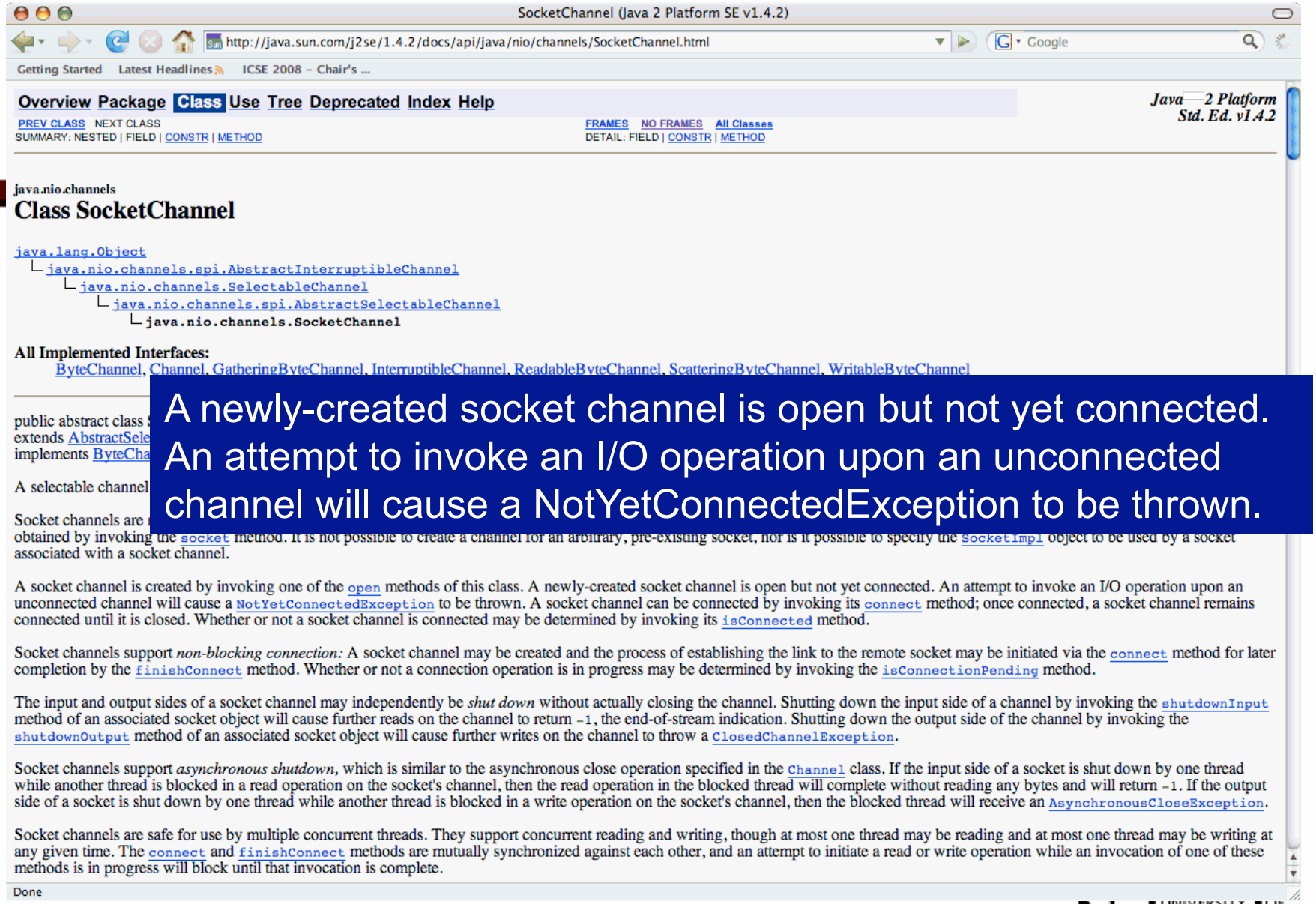
Reduce overhead without losing precision/recall

Two ideas …

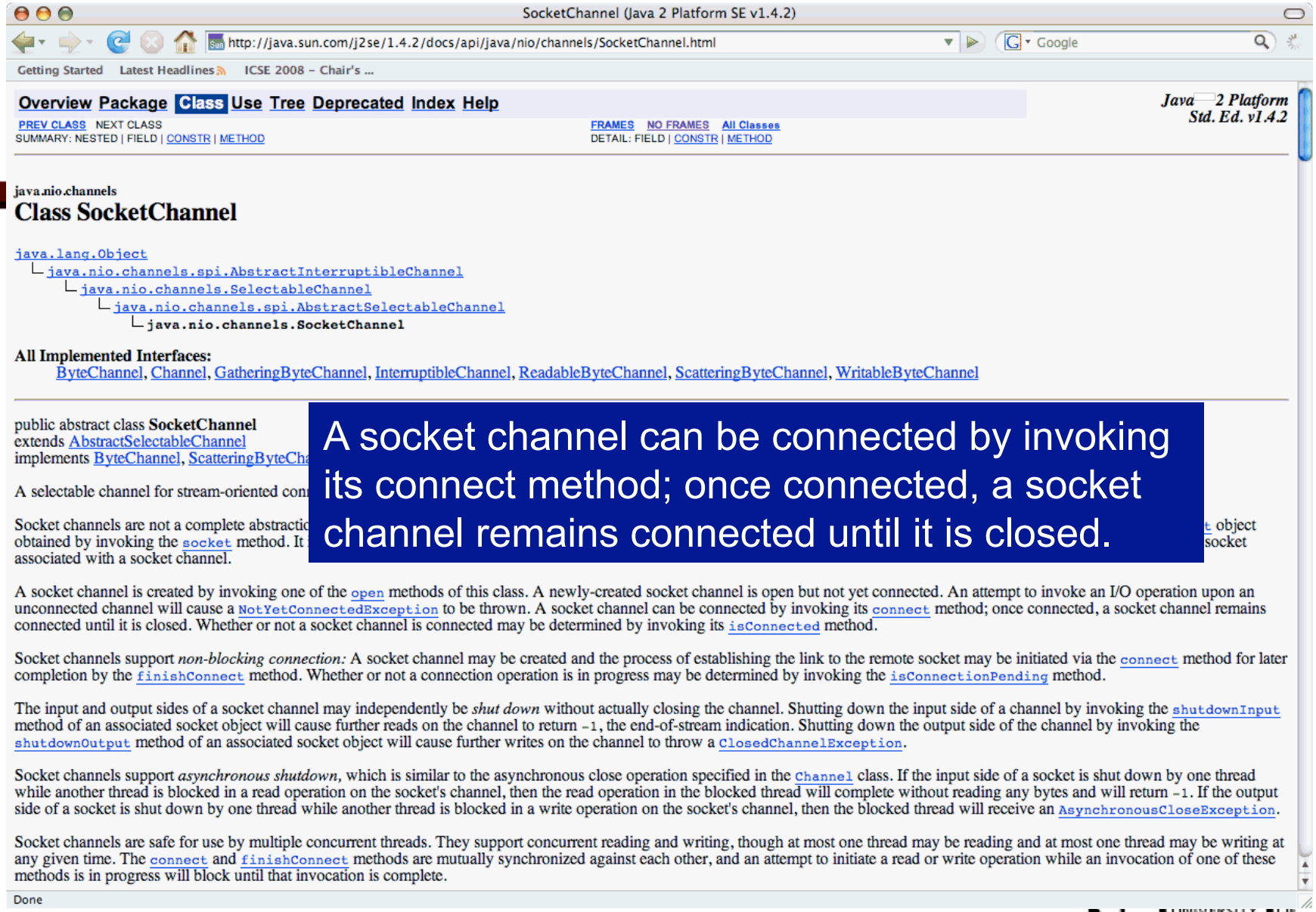
- Residual analysis : extend staged analysis to runtime
- Adaptive analysis : adjust monitoring to program state

Explained in terms of flow analysis for sequential programs


SocketChannel (Java 2 Platform SE v1.4.2)
http://java.sun.com/j2se/1.4.2/docs/api/java/nio/channels/SocketChannel.html
Google
Getting Started
Latest Headlines
ICSE 2008 – Chair's ...
Overview Package Class Use Tree Deprecated Index Help
Java 2 Platform Std. Ed. v1.4.2
PREV CLASS NEXT CLASS
SUMMARY: NESTED | FIELD | CONSTR | METHOD
FRAMES NO FRAMES All Classes
DETAIL: FIELD | CONSTR | METHOD
java.nio.channels
Class SocketChannel
java.lang.Object
java.nio.channels.spi.AbstractInterruptibleChannel
java.nio.channels.SelectableChannel
java.nio.channels.spi.AbstractSelectableChannel
java.nio.channels.SocketChannel
All Implemented Interfaces:
ByteChannel, Channel, GatheringByteChannel, InterruptibleChannel, ReadableByteChannel, ScatteringByteChannel, WritableByteChannel
public abstract class SocketChannel
extends AbstractSelectableChannel
implements ByteChannel, ScatteringByteChannel, GatheringByteChannel
A selectable channel for stream-oriented connecting sockets.
Socket channels are not a complete abstraction of connecting network sockets. Binding, shutdown, and the manipulation of socket options must be done through an associated Socket object obtained by invoking the socket method. It is not possible to create a channel for an arbitrary, pre-existing socket, nor is it possible to specify the SocketImpl object to be used by a socket associated with a socket channel.
A socket channel is created by invoking one of the open methods of this class. A newly-created socket channel is open but not yet connected. An attempt to invoke an I/O operation upon an unconnected channel will cause a NotYetConnectedException to be thrown. A socket channel can be connected by invoking its connect method; once connected, a socket channel remains connected until it is closed. Whether or not a socket channel is connected may be determined by invoking its isConnected method.
Socket channels support non-blocking connection: A socket channel may be created and the process of establishing the link to the remote socket may be initiated via the connect method for later completion by the finishConnect method. Whether or not a connection operation is in progress may be determined by invoking the isConnectionPending method.
The input and output sides of a socket channel may independently be shut down without actually closing the channel. Shutting down the input side of a channel by invoking the shutdownInput method of an associated socket object will cause further reads on the channel to return -1, the end-of-stream indication. Shutting down the output side of the channel by invoking the shutdownOutput method of an associated socket object will cause further writes on the channel to throw a ClosedChannelException.
Socket channels support asynchronous shutdown, which is similar to the asynchronous close operation specified in the Channel class. If the input side of a socket is shut down by one thread while another thread is blocked in a read operation on the socket's channel, then the read operation in the blocked thread will complete without reading any bytes and will return -1. If the output side of a socket is shut down by one thread while another thread is blocked in a write operation on the socket's channel, then the blocked thread will receive an AsynchronousCloseException.
Socket channels are safe for use by multiple concurrent threads. They support concurrent reading and writing, though at most one thread may be reading and at most one thread may be writing at any given time. The connect and finishConnect methods are mutually synchronized against each other, and an attempt to initiate a read or write operation while an invocation of one of these methods is in progress will block until that invocation is complete.
Done



A newly-created socket channel is open but not yet connected. An attempt to invoke an I/O operation upon an unconnected channel will cause a NotYetConnectedException to be thrown.


SocketChannel (Java 2 Platform SE v1.4.2)
http://java.sun.com/j2se/1.4.2/docs/api/java/nio/channels/SocketChannel.html
Google
Getting Started
Latest Headlines
ICSE 2008 – Chair's ...
Overview Package Class Use Tree Deprecated Index Help
Java 2 Platform Std. Ed. v1.4.2
PREV CLASS NEXT CLASS
SUMMARY: NESTED | FIELD | CONSTR | METHOD
FRAMES NO FRAMES All Classes
DETAIL: FIELD | CONSTR | METHOD
java.nio.channels
Class SocketChannel
java.lang.Object
java.nio.channels.spi.AbstractInterruptibleChannel
java.nio.channels.SelectableChannel
java.nio.channels.spi.AbstractSelectableChannel
java.nio.channels.SocketChannel
All Implemented Interfaces:
ByteChannel, Channel, GatheringByteChannel, InterruptibleChannel, ReadableByteChannel, ScatteringByteChannel, WritableByteChannel
public abstract class SocketChannel
extends AbstractSelectableChannel
implements ByteChannel, ScatteringByteCha
A selectable channel for stream-oriented con
Socket channels are not a complete abstracti
obtained by invoking the socket method. It
associated with a socket channel.
t object
socket
A socket channel is created by invoking one of the open methods of this class. A newly-created socket channel is open but not yet connected. An attempt to invoke an I/O operation upon an unconnected channel will cause a NotYetConnectedException to be thrown. A socket channel can be connected by invoking its connect method; once connected, a socket channel remains connected until it is closed. Whether or not a socket channel is connected may be determined by invoking its isConnected method.
Socket channels support non-blocking connection: A socket channel may be created and the process of establishing the link to the remote socket may be initiated via the connect method for later completion by the finishConnect method. Whether or not a connection operation is in progress may be determined by invoking the isConnectionPending method.
The input and output sides of a socket channel may independently be shut down without actually closing the channel. Shutting down the input side of a channel by invoking the shutdownInput method of an associated socket object will cause further reads on the channel to return -1, the end-of-stream indication. Shutting down the output side of the channel by invoking the shutdownOutput method of an associated socket object will cause further writes on the channel to throw a ClosedChannelException.
Socket channels support asynchronous shutdown, which is similar to the asynchronous close operation specified in the Channel class. If the input side of a socket is shut down by one thread while another thread is blocked in a read operation on the socket's channel, then the read operation in the blocked thread will complete without reading any bytes and will return -1. If the output side of a socket is shut down by one thread while another thread is blocked in a write operation on the socket's channel, then the blocked thread will receive an AsynchronousCloseException.
Socket channels are safe for use by multiple concurrent threads. They support concurrent reading and writing, though at most one thread may be reading and at most one thread may be writing at any given time. The connect and finishConnect methods are mutually synchronized against each other, and an attempt to initiate a read or write operation while an invocation of one of these methods is in progress will block until that invocation is complete.
Done
A socket channel can be connected by invoking its connect method; once connected, a socket channel remains connected until it is closed.

Selected SocketChannel Methods

static SocketChannel open() …

static SocketChannel open(SocketAddress …

SocketChannel connect(SocketAddress …

char read(ByteBuffer dst) …

int write(ByteBuffer src) …

final void close() …

…

Selected SocketChannel Methods

static SocketChannel open() …
static SocketChannel open(SocketAddress …
SocketChannel connect(SocketAddress …
char read(ByteBuffer dst) …
int write(ByteBuffer src) …
final void close() …
…

Constraints from Javadoc …

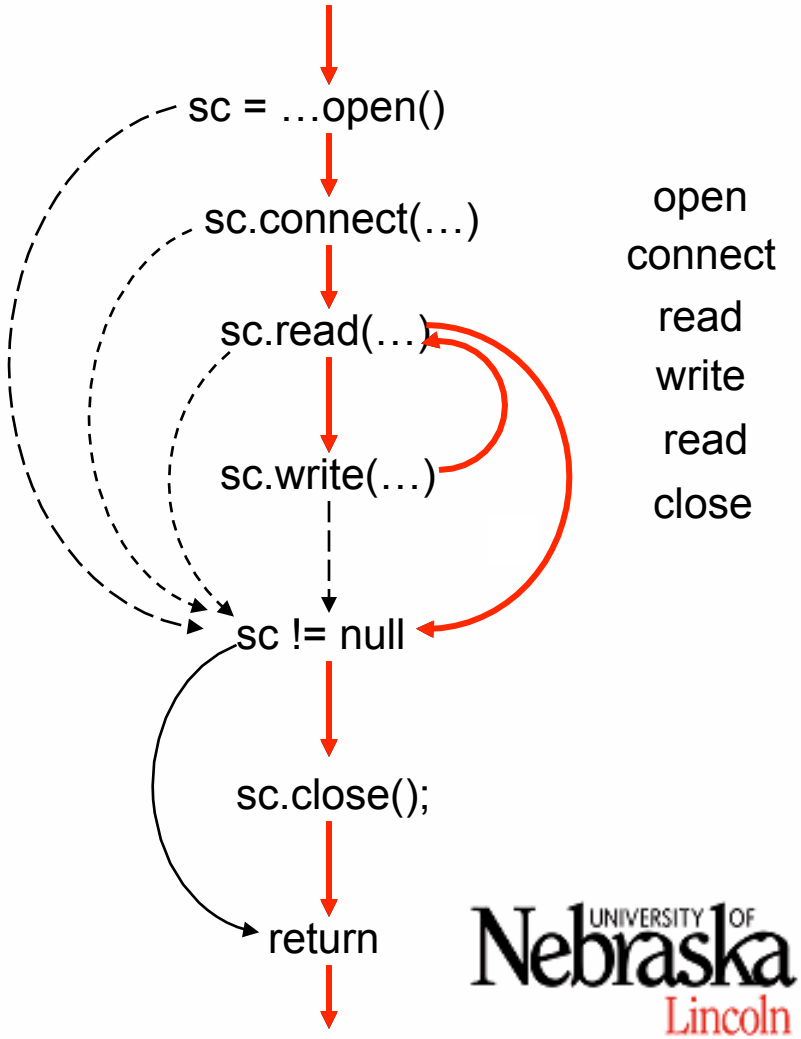
- open before connect
- connect before read/write
- close after open

SocketChannel API Example

```
public void transformData() {
        SocketChannel sc;
        ByteBuffer buf;
        try{
            sc = SocketChannel.open();
            sc.connect(new
            InetSocketAddress(…));
                while (sc.read(buf) != -1){
                    sc.write(buf);
                }
        } catch (Exception e){ …
        } finally {
                if (sc != null)
                    sc.close();
        }
}
```

SocketChannel API Example

```
public void transformData() {
        SocketChannel sc;
        ByteBuffer buf;
        try{
            sc = SocketChannel.open();
            sc.connect(new
            InetSocketAddress(…));
                while (sc.read(buf) != -1){
                    sc.write(buf);
                }
        } catch (Exception e){ …
        } finally {
                if (sc != null)
                    sc.close();
        }
}
```

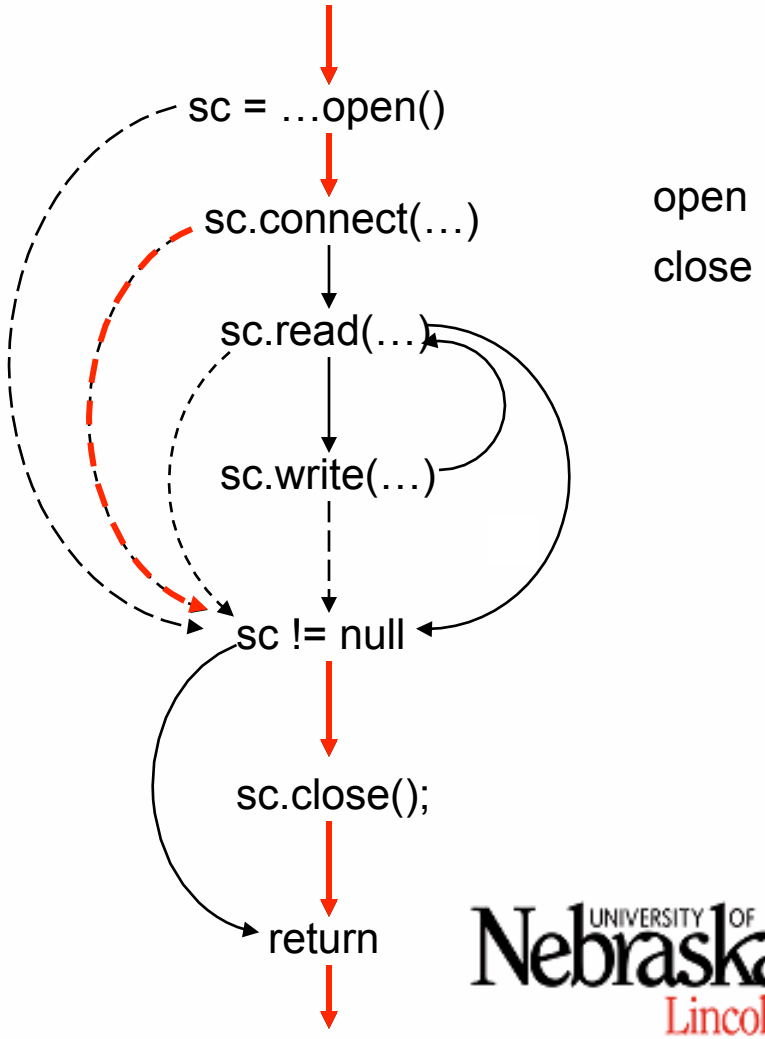


ESQuaReD

University of Nebraska Lincoln

SocketChannel API Example

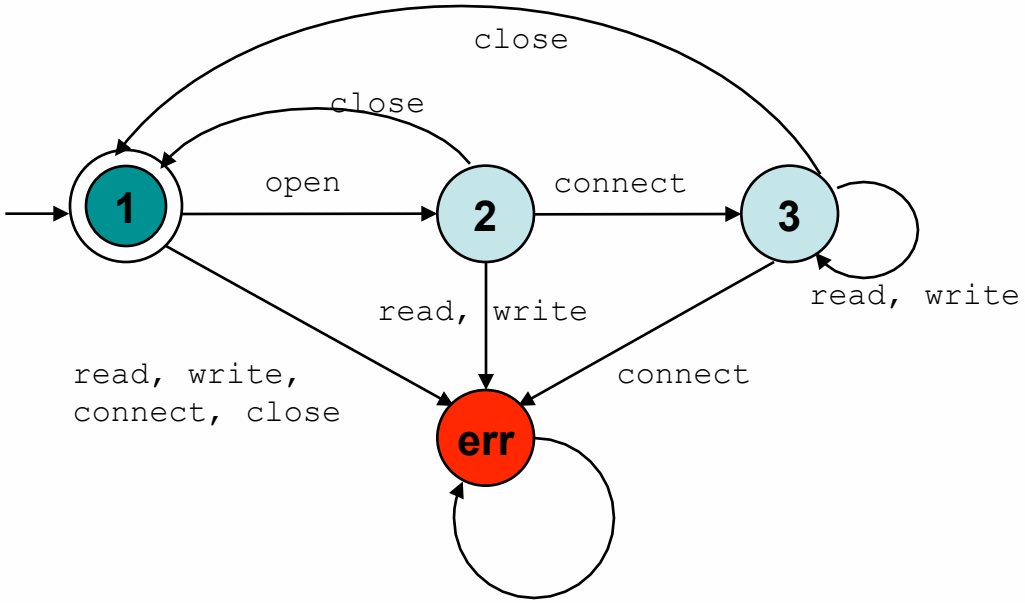
```
public void transformData() {
        SocketChannel sc;
        ByteBuffer buf;
        try{
            sc = SocketChannel.open();
            sc.connect(new
            InetSocketAddress(…));
                while (sc.read(buf) != -1){
                    sc.write(buf);
                }
        } catch (Exception e){ …
        } finally {
                if (sc != null)
                    sc.close();
        }
}
```



ESQuaReD

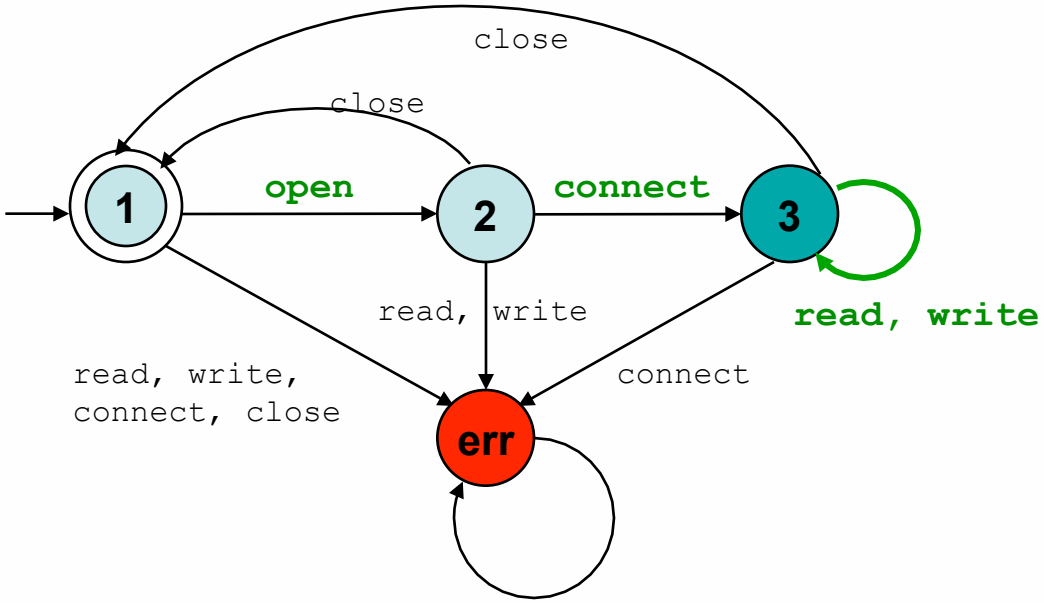
University of Nebraska Lincoln

SocketChannel API Typestate FSA



Strom & Yemeni,
TSE, 1986

SocketChannel API Typestate FSA



Strom & Yemeni,
TSE, 1986

ESQuaReD

University of Nebraska Lincoln

Typestate Analysis

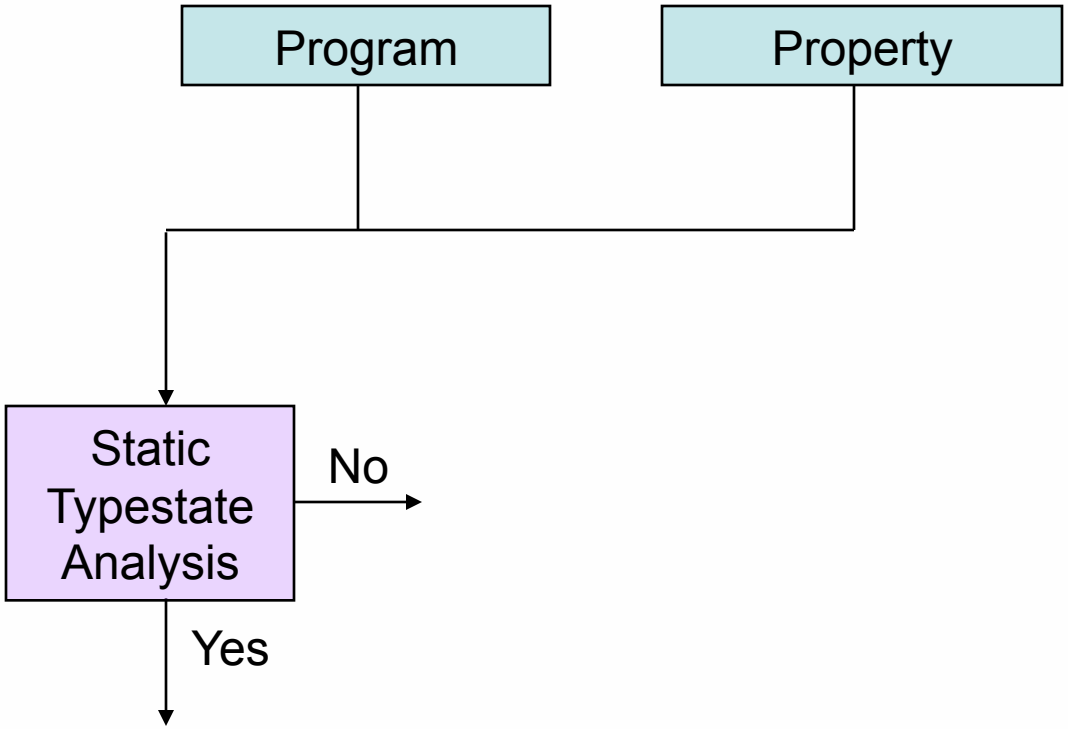
Static Typestate Analysis

- MSR ESP (PLDI'02,ISSTA'04), IBM SAFE (ISSTA'06)
- Data flow analysis to reason about path-property conformance
- Inherently flow and object-sensitive … precision is expensive

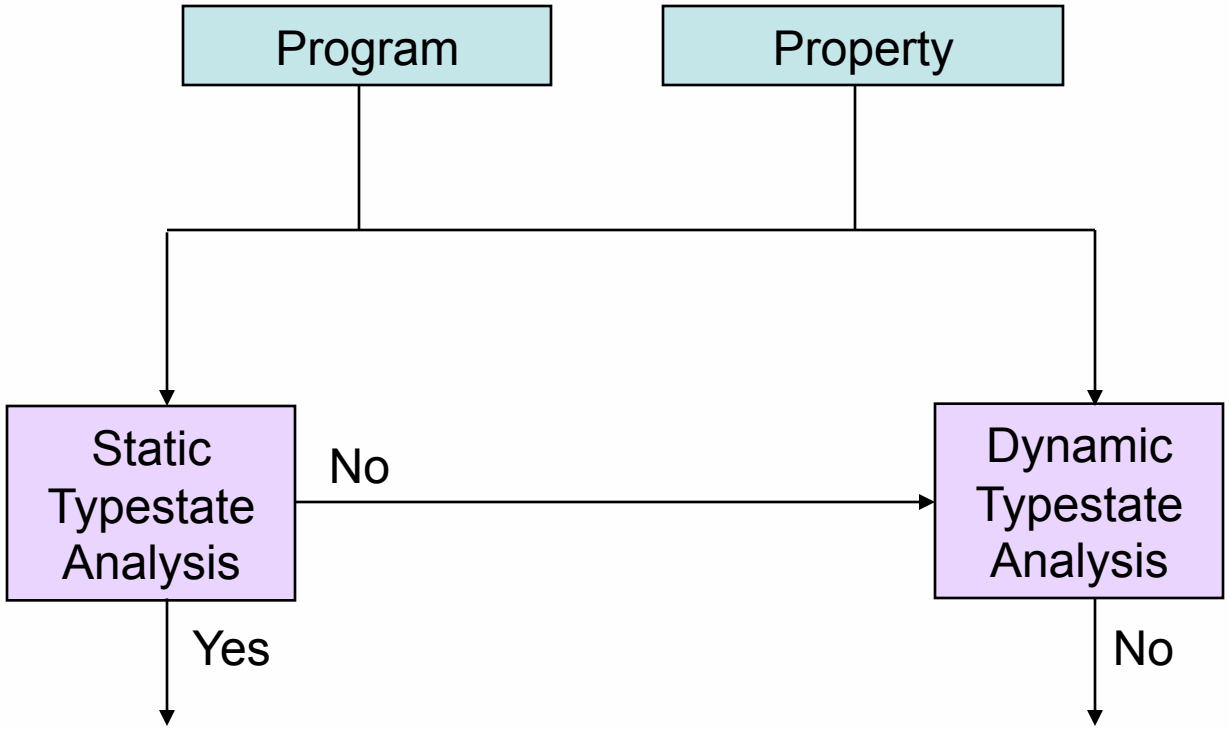
Dynamic Typestate Analysis

- UPenn MaC (FMSD'04), UIUC JavaMOP (OOPSLA'07), McGill/ Oxford Tracematches (ECOOP'07)
- Instrument program to monitor property conformance at run-time
- Can incur significant runtime overhead

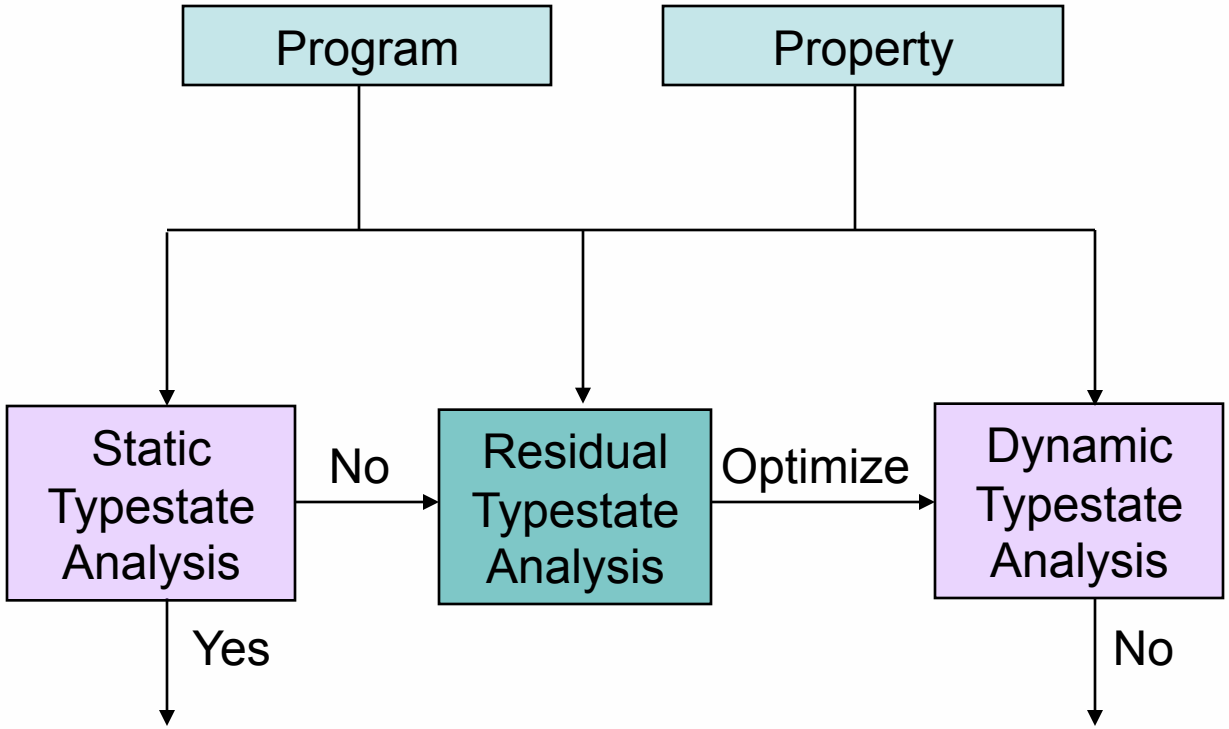
Typestate Analysis



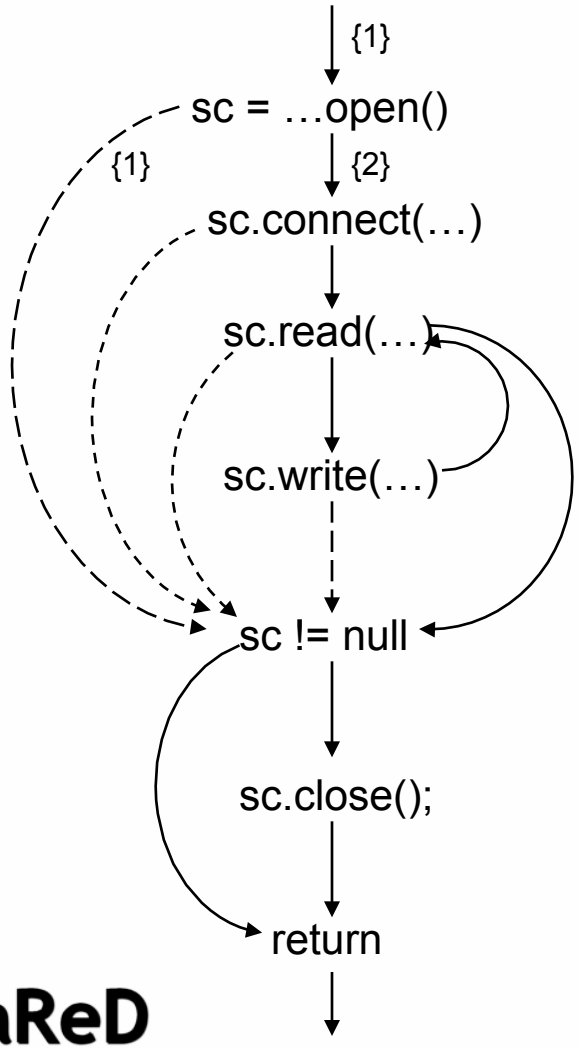
Residual Typestate Analysis

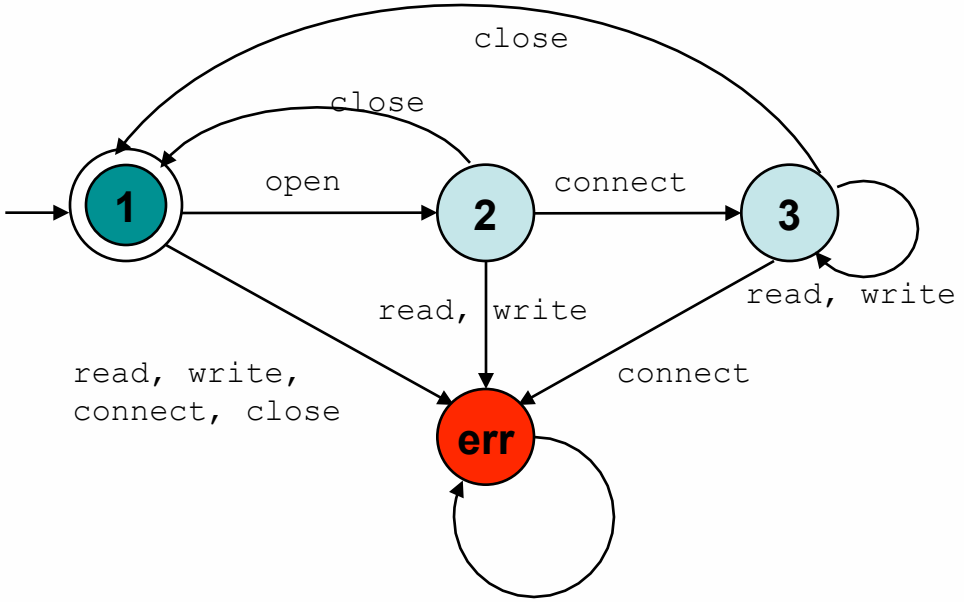


Residual Typestate Analysis [ASE'07]



Static Typestate Analysis

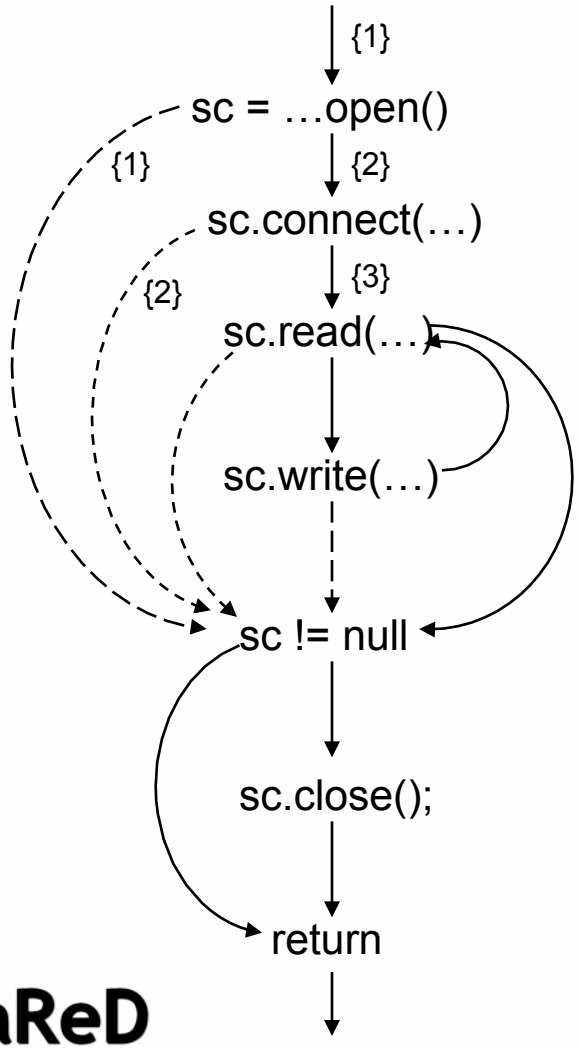


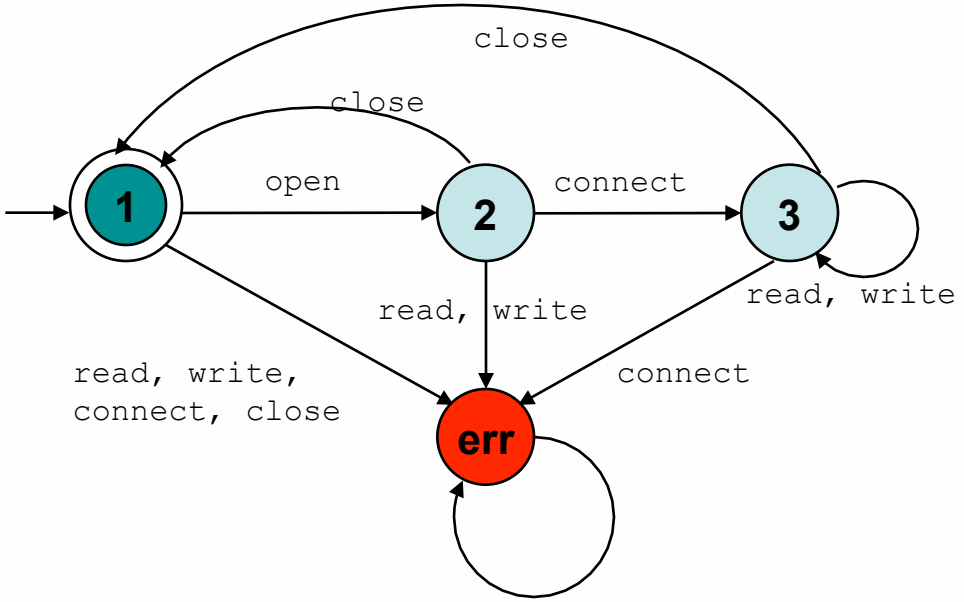


ESQuaReD

University of Nebraska Lincoln

Static Typestate Analysis

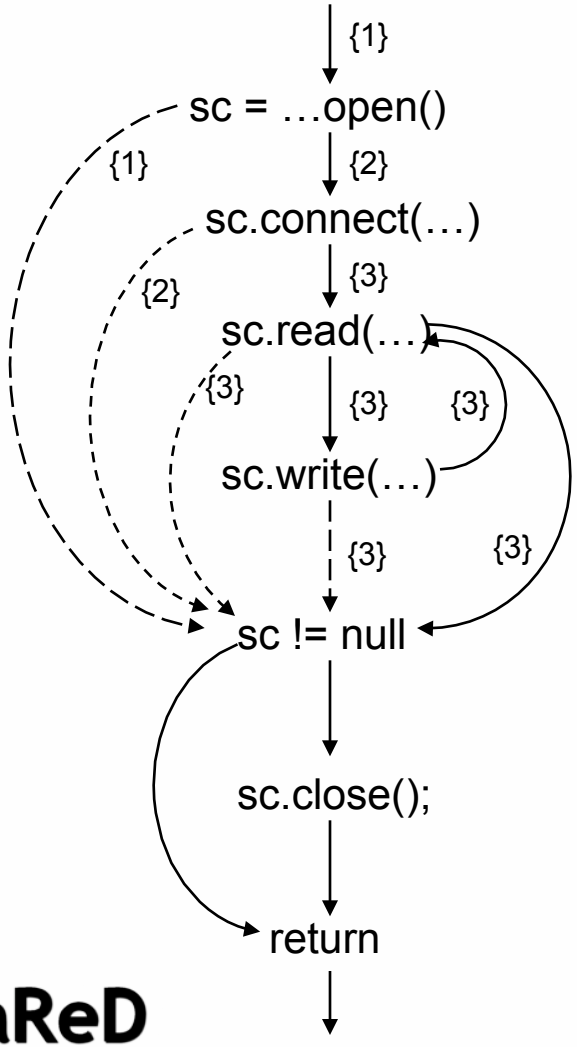


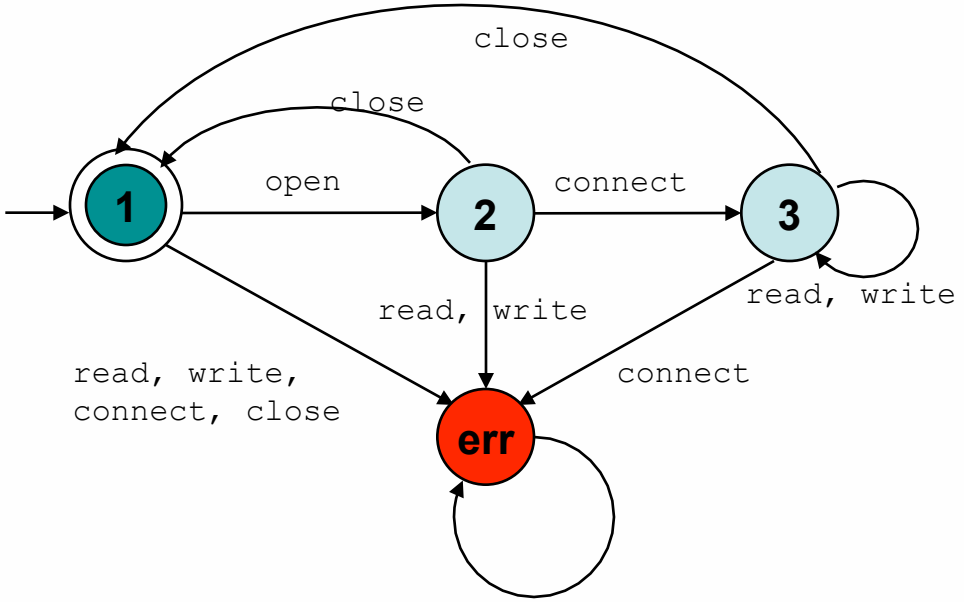


ESQuaReD

University of Nebraska Lincoln

Static Typestate Analysis

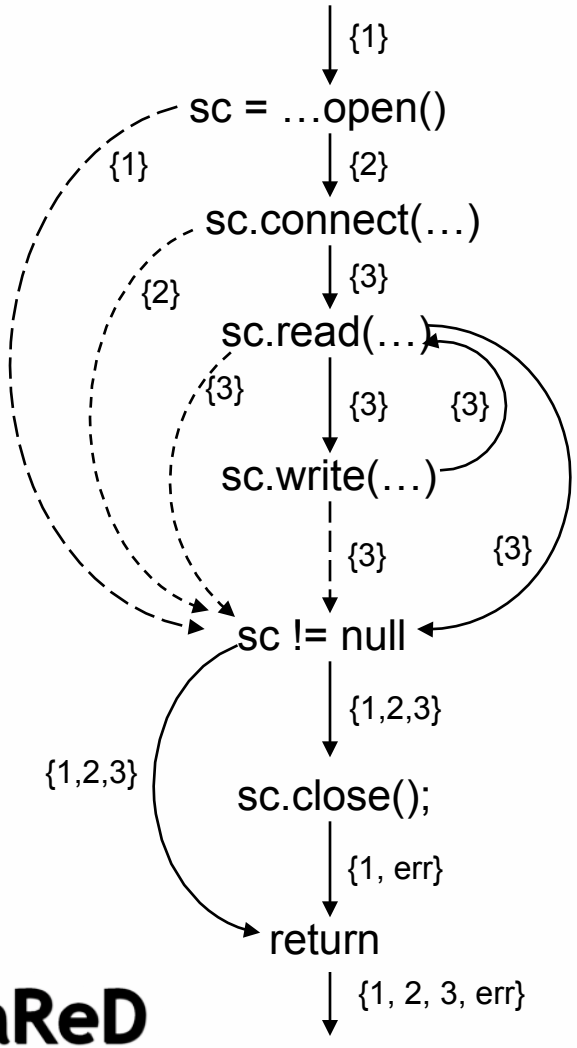


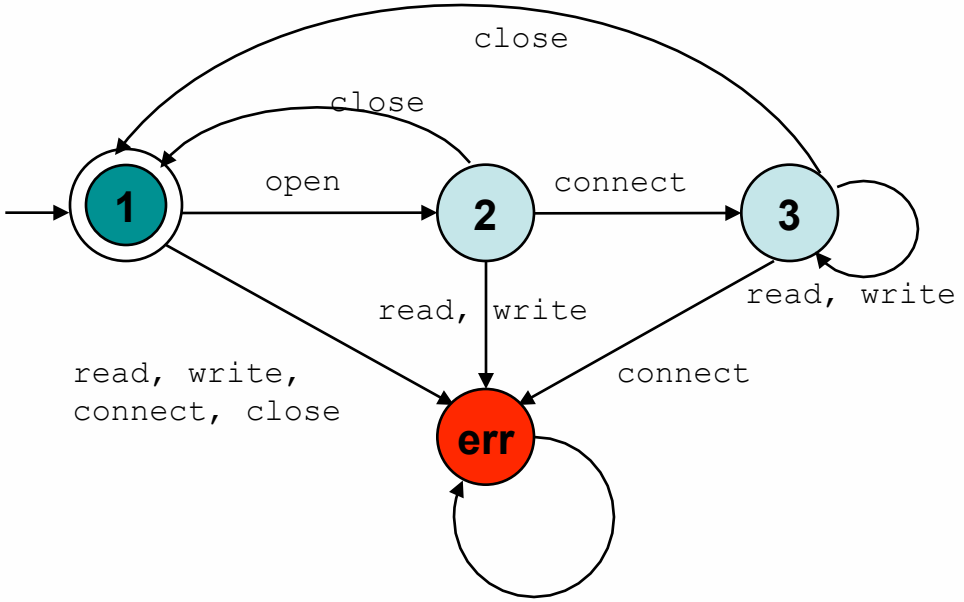


ESQuaReD

University of Nebraska Lincoln

Static Typestate Analysis

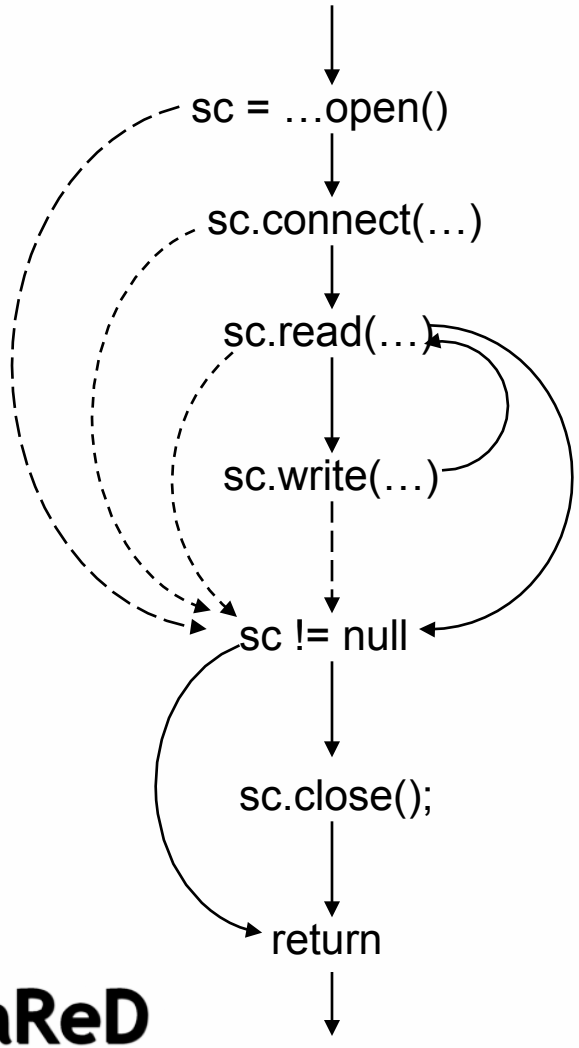


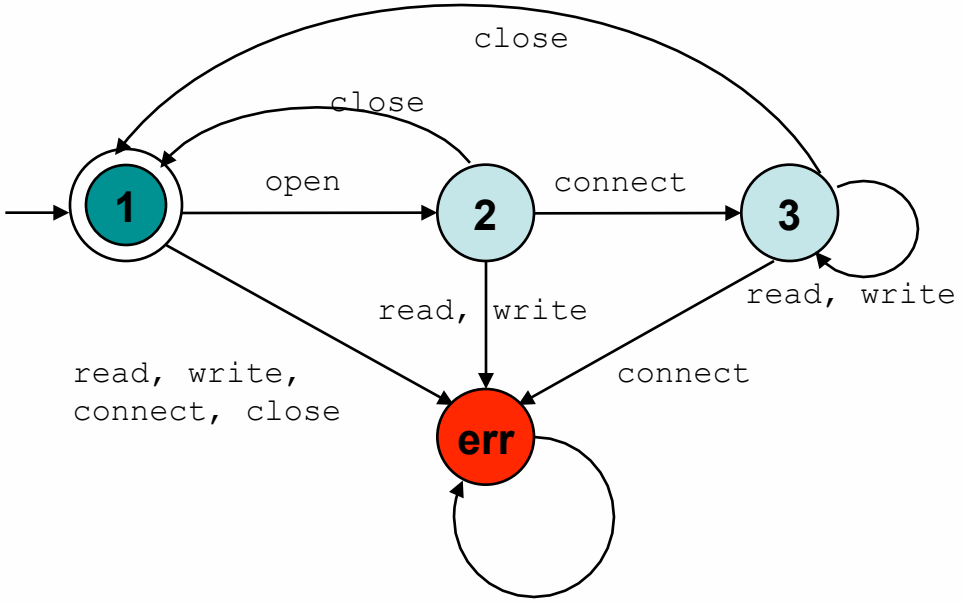


ESQuaReD

University of Nebraska Lincoln

Dynamic Typestate Analysis

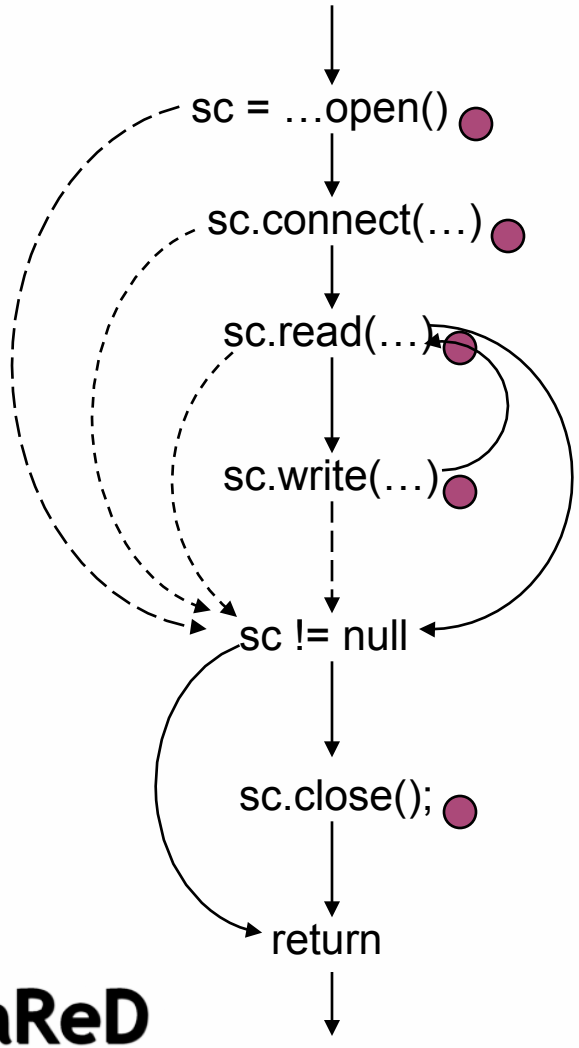


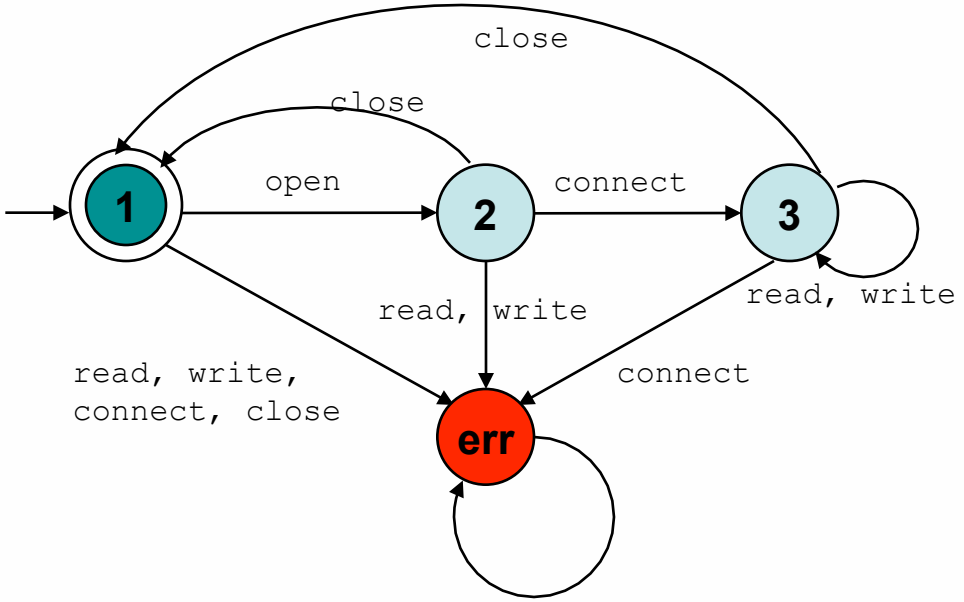


ESQuaReD

University of Nebraska Lincoln

Dynamic Typestate Analysis

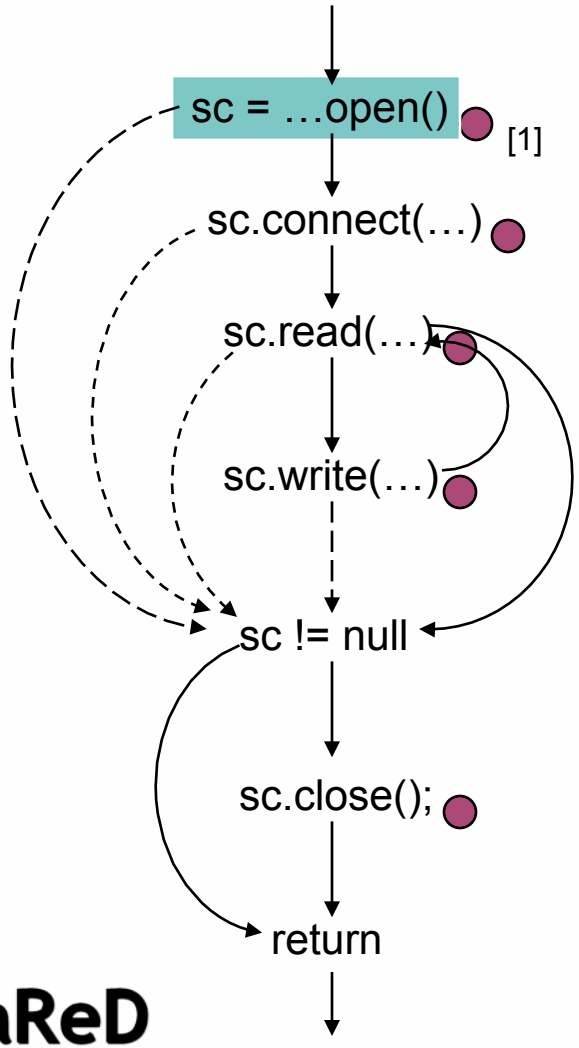


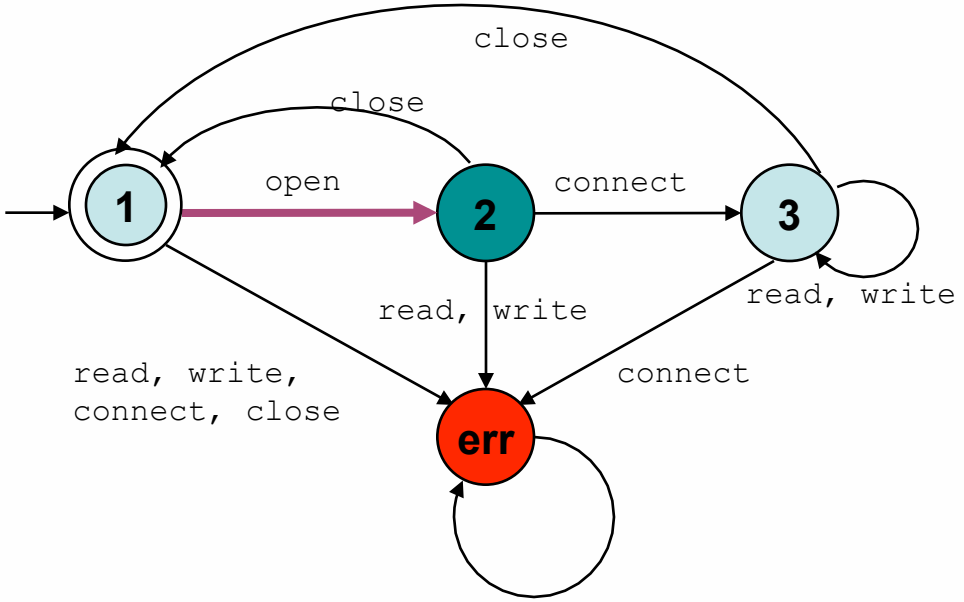


ESQuaReD

University of Nebraska Lincoln

Dynamic Typestate Analysis

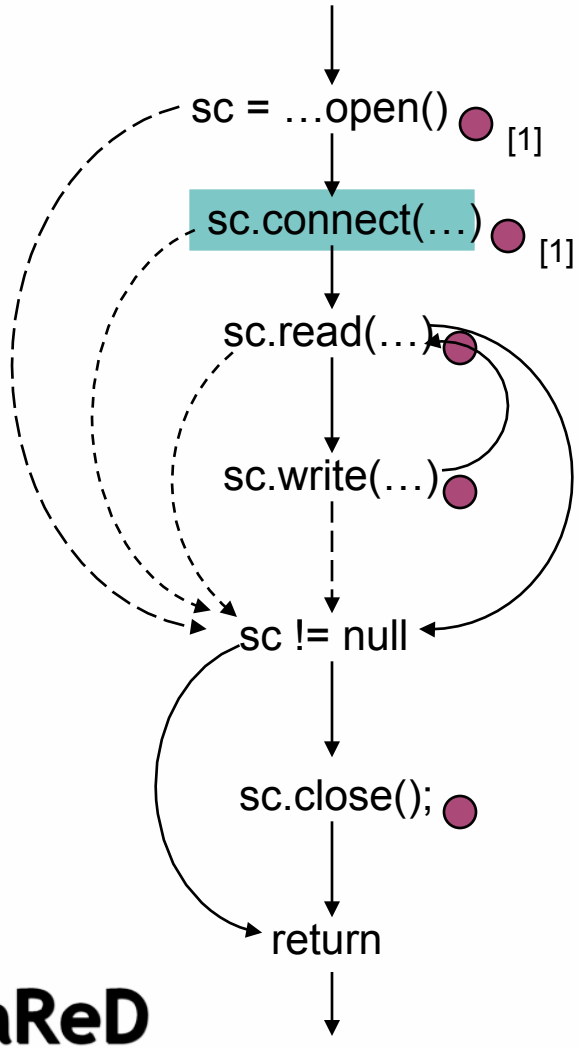


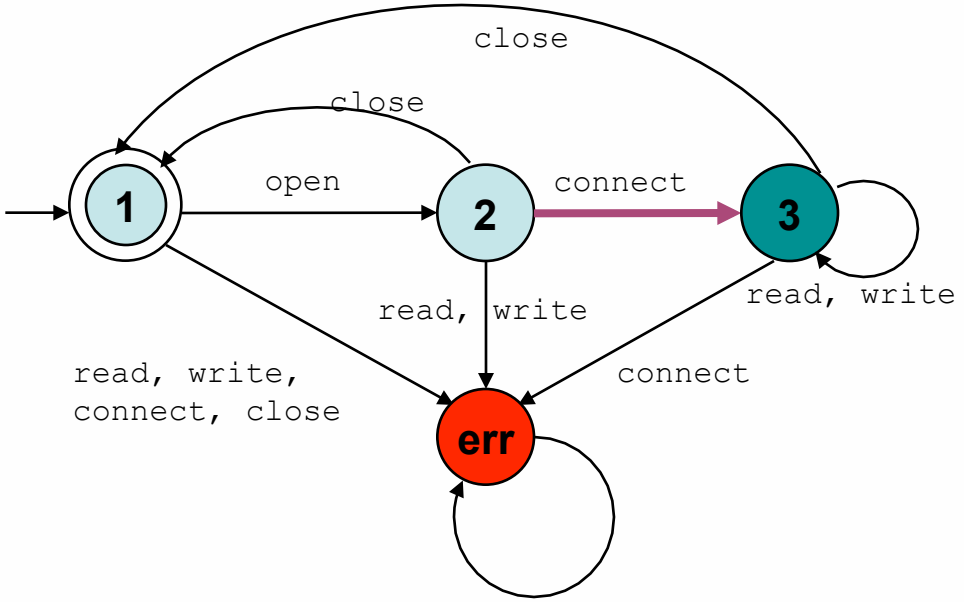


ESQuaReD

University of Nebraska Lincoln

Dynamic Typestate Analysis

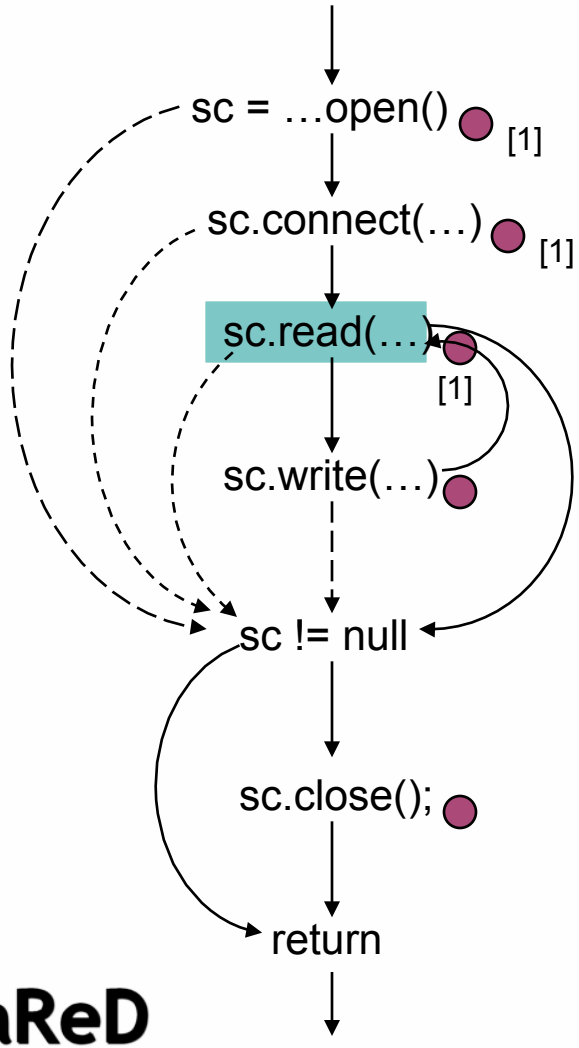


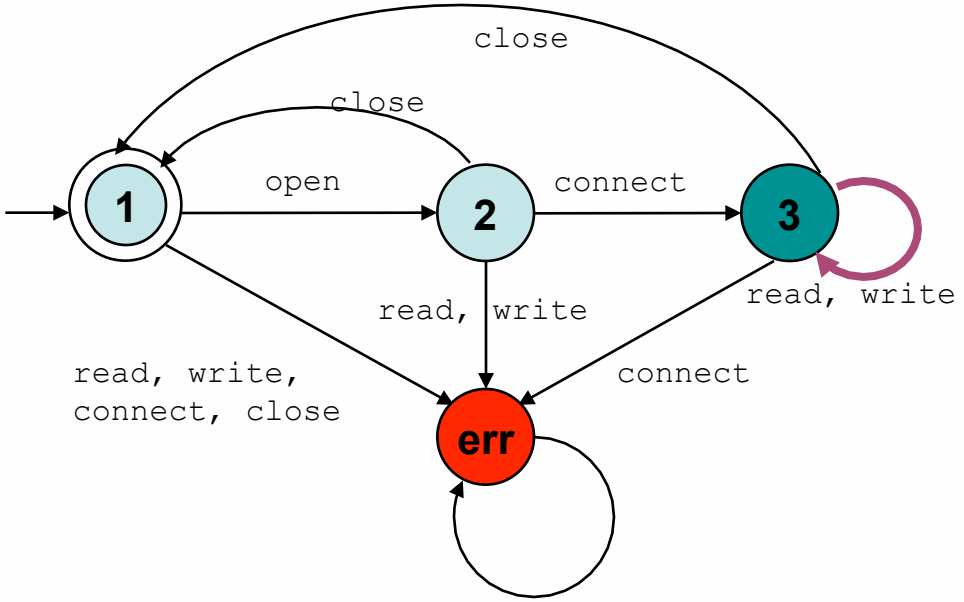


ESQuaReD

University of Nebraska Lincoln

Dynamic Typestate Analysis

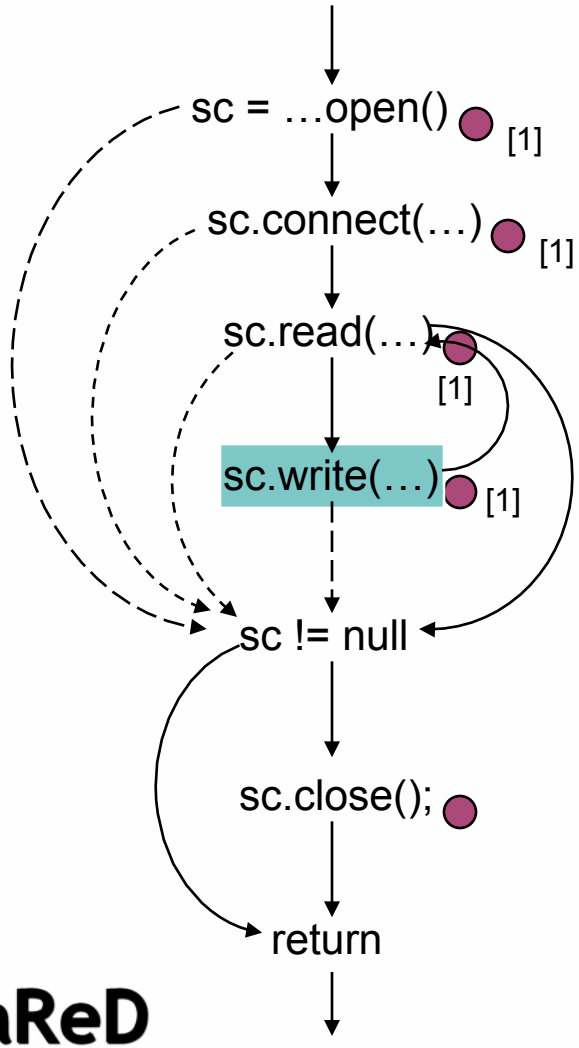


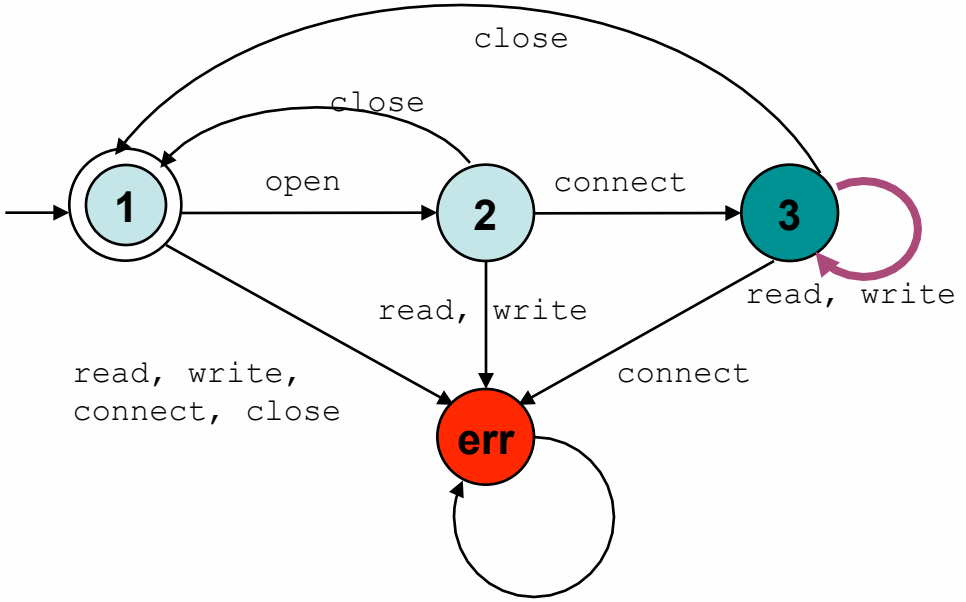


ESQuaReD

University of Nebraska Lincoln

Dynamic Typestate Analysis

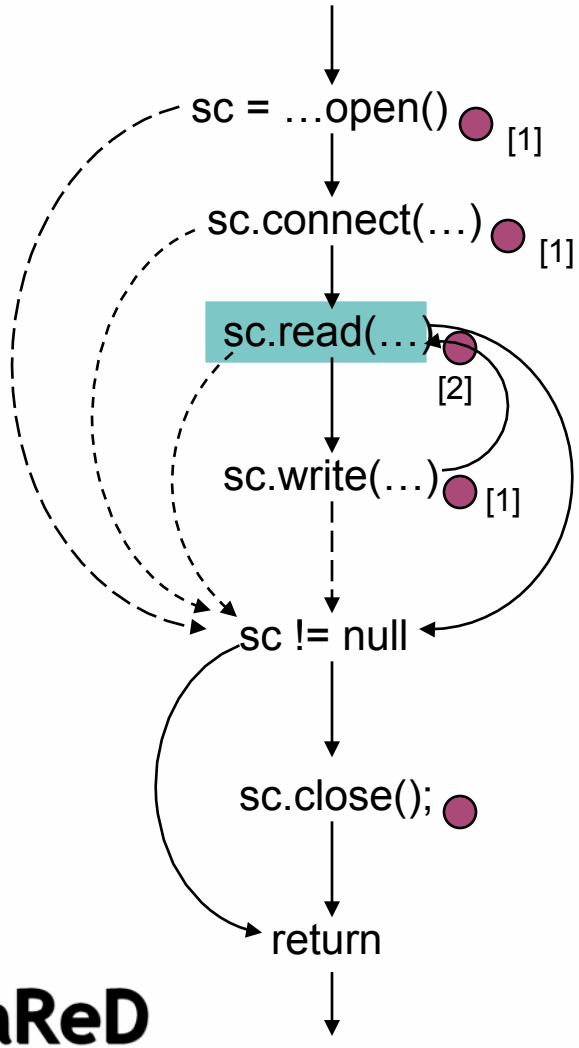


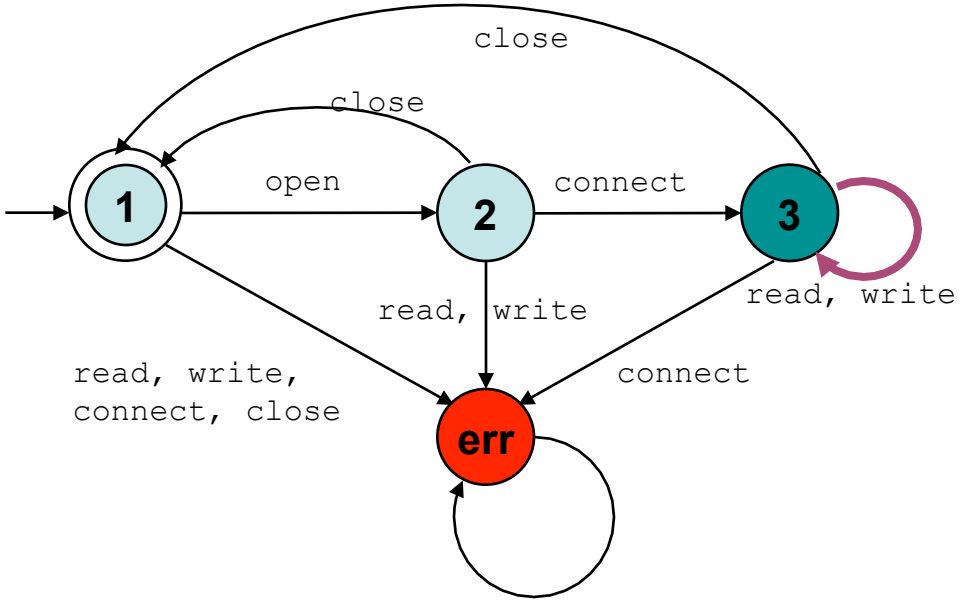


ESQuaReD

University of Nebraska Lincoln

Dynamic Typestate Analysis

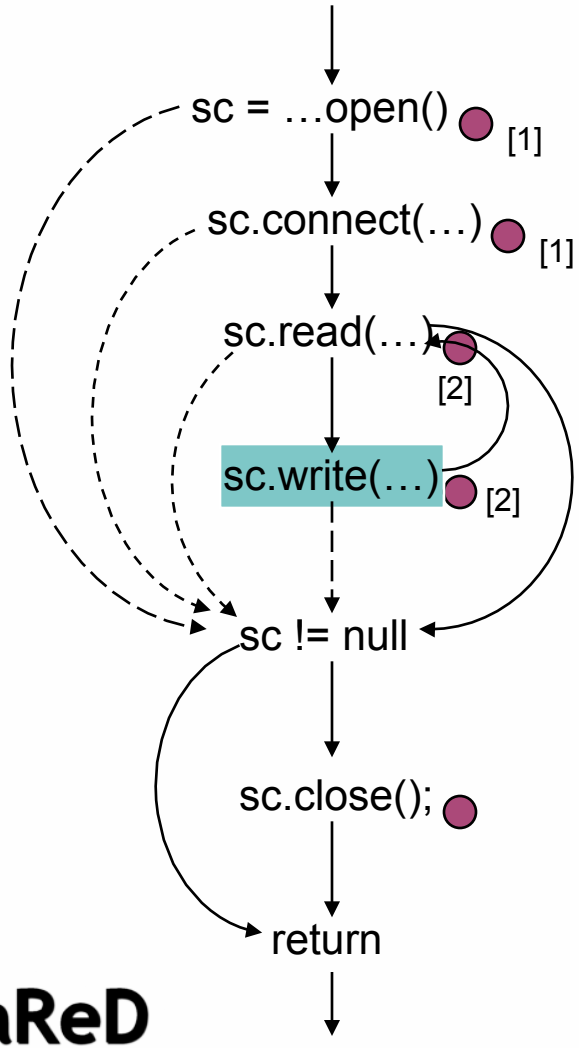


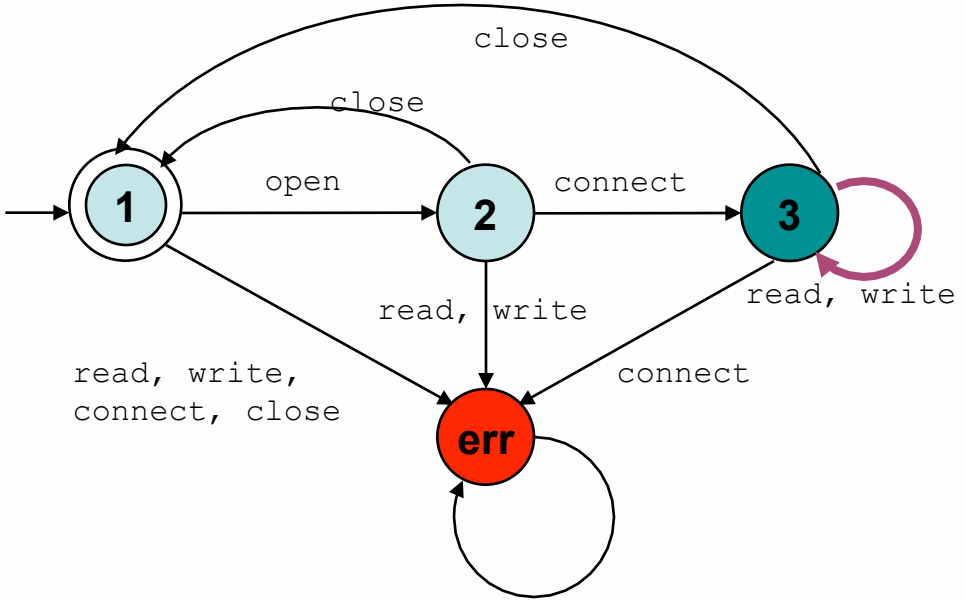


ESQuaReD

University of Nebraska Lincoln

Dynamic Typestate Analysis

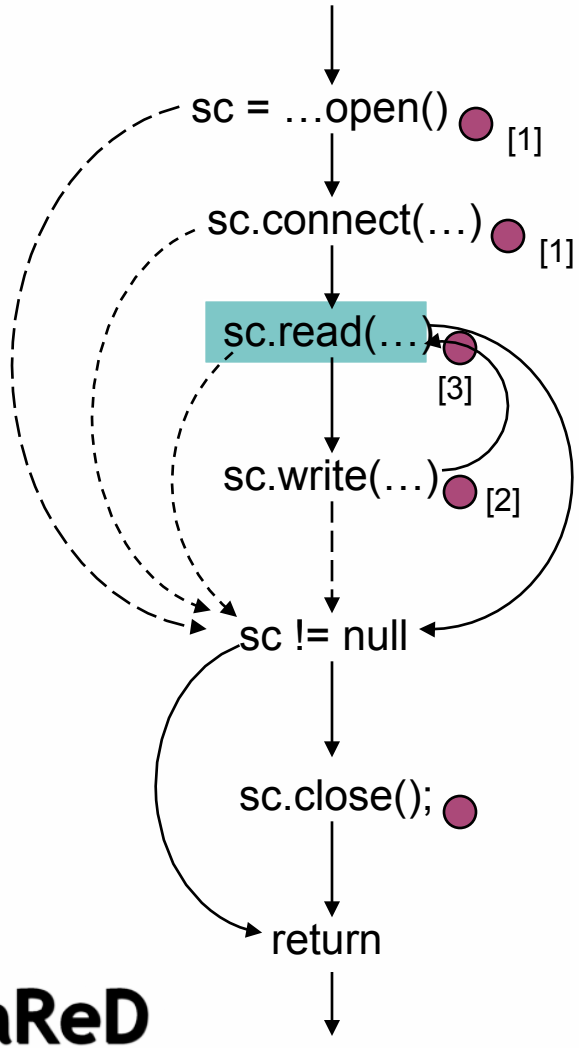


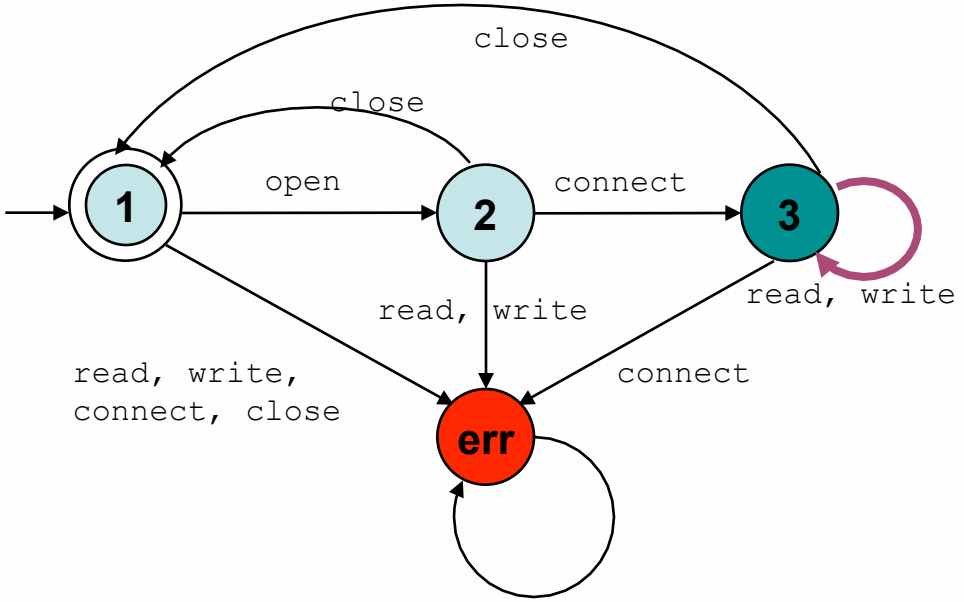


ESQuaReD

University of Nebraska Lincoln

Dynamic Typestate Analysis

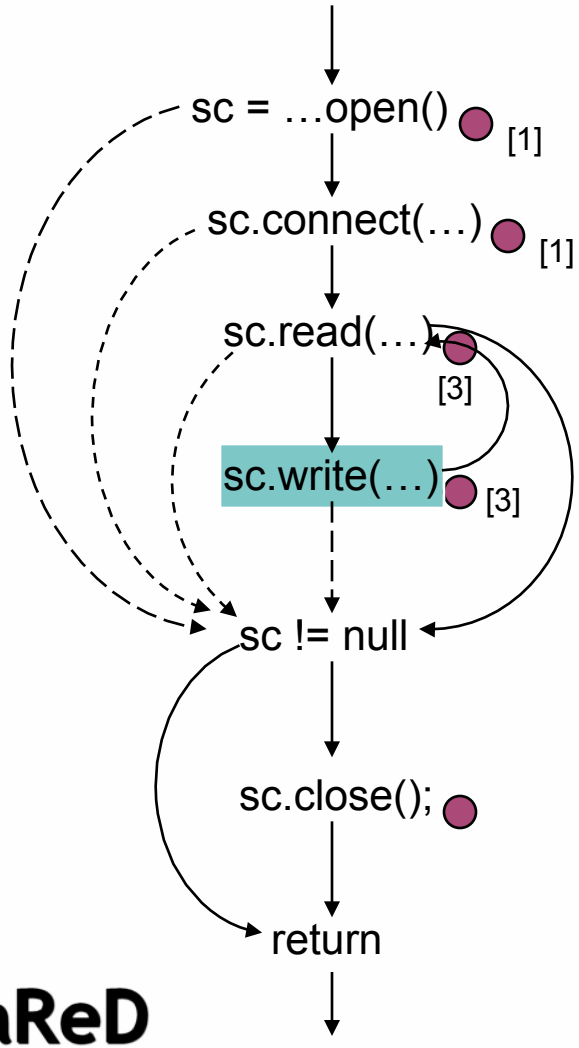


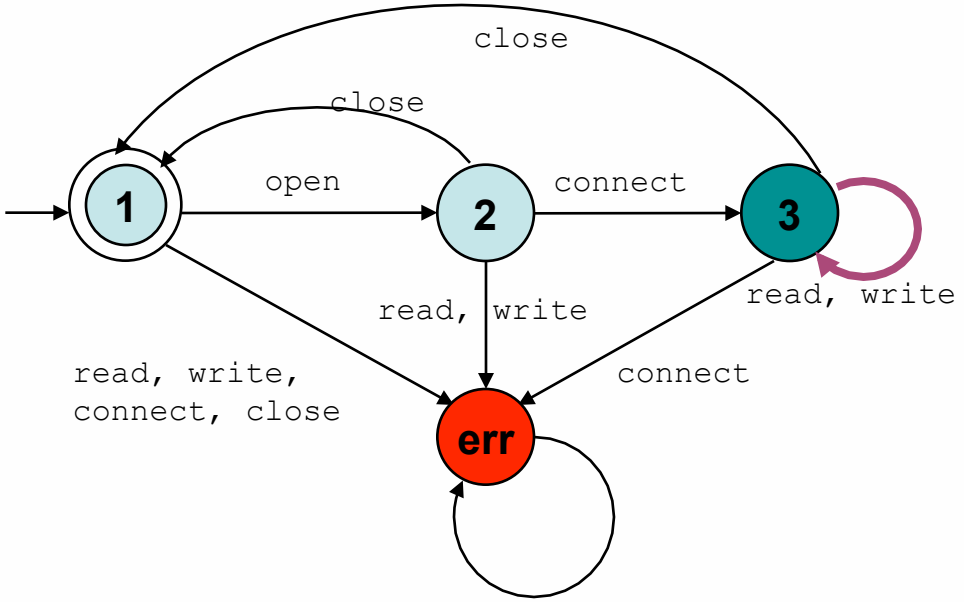


ESQuaReD

University of Nebraska Lincoln

Dynamic Typestate Analysis

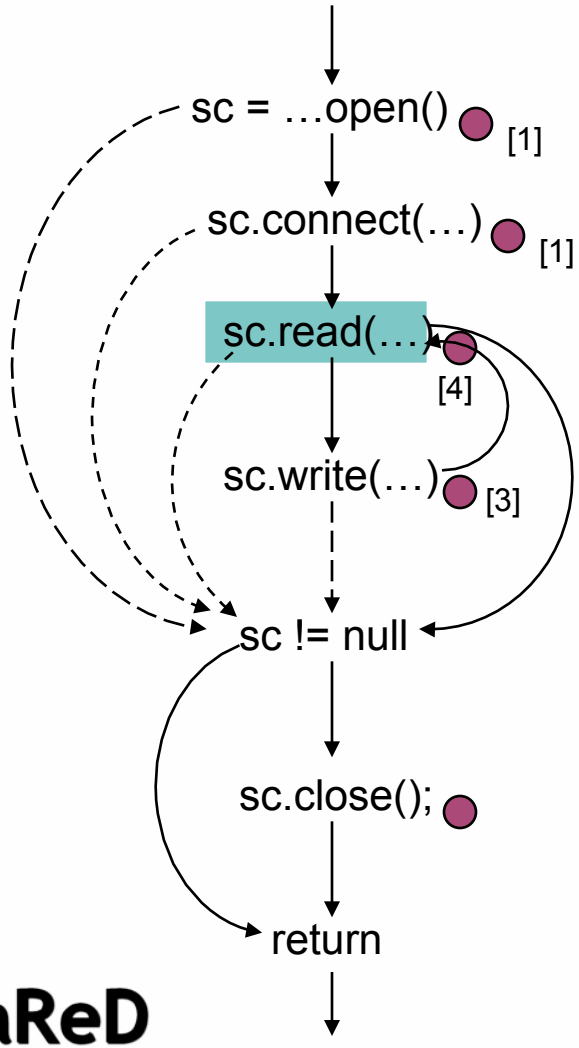


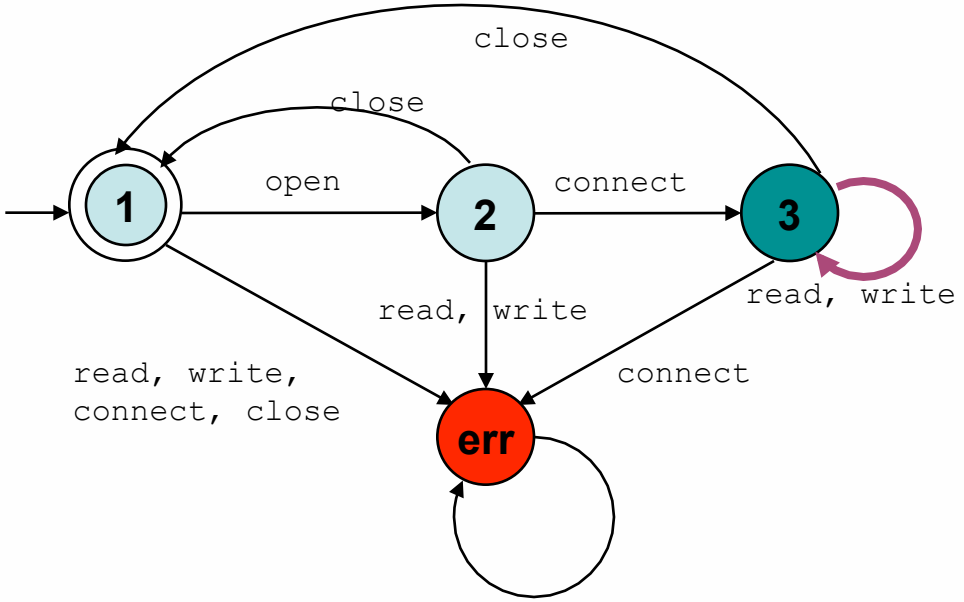


ESQuaReD

University of Nebraska Lincoln

Dynamic Typestate Analysis

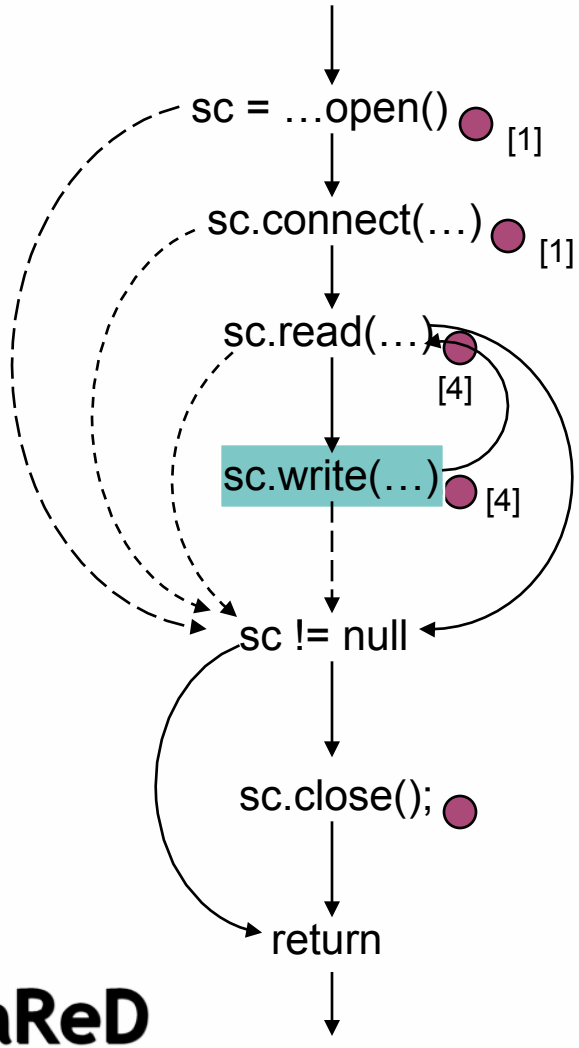


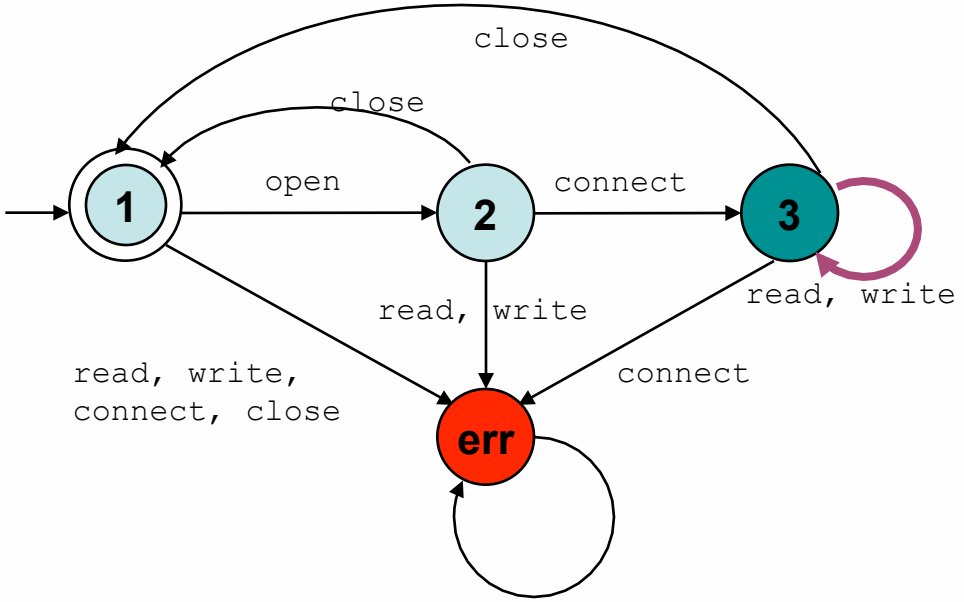


ESQuaReD

University of Nebraska Lincoln

Dynamic Typestate Analysis

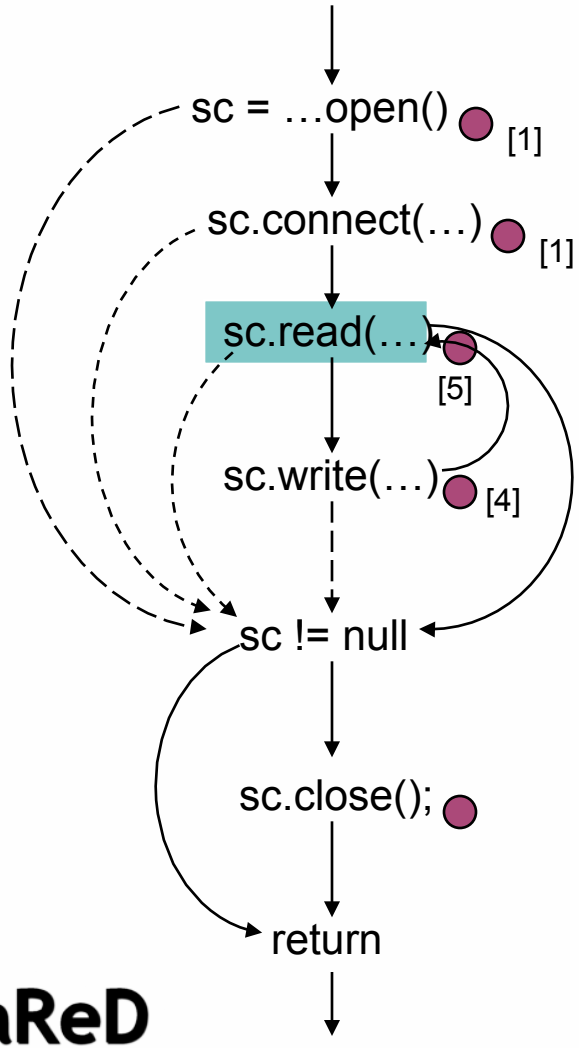


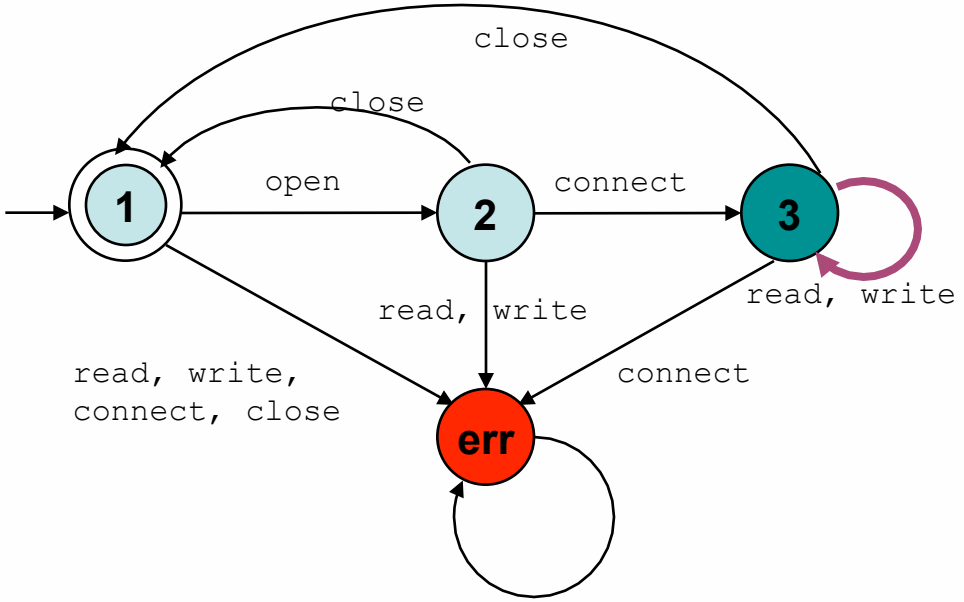


ESQuaReD

University of Nebraska Lincoln

Dynamic Typestate Analysis

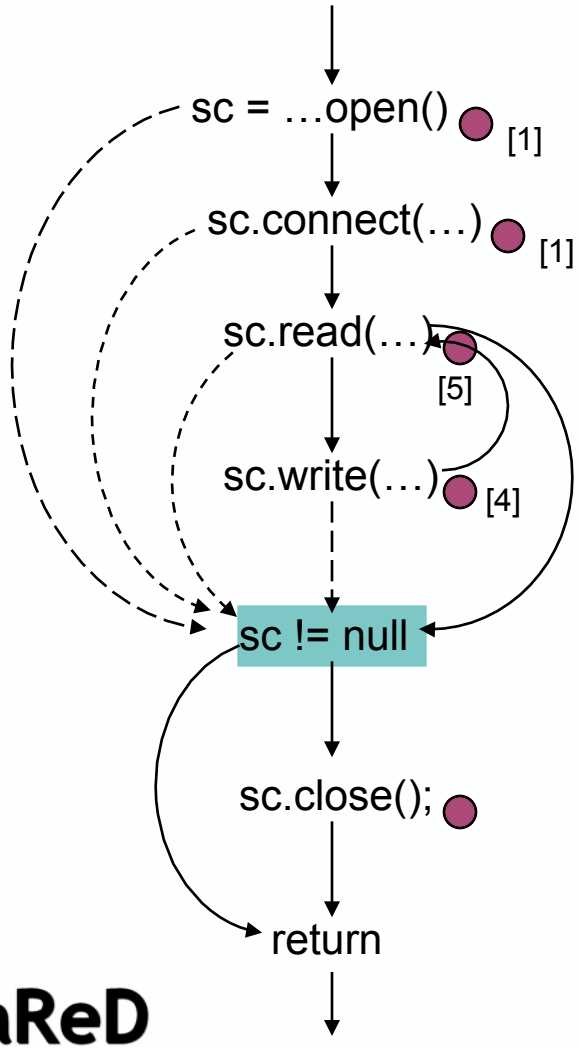


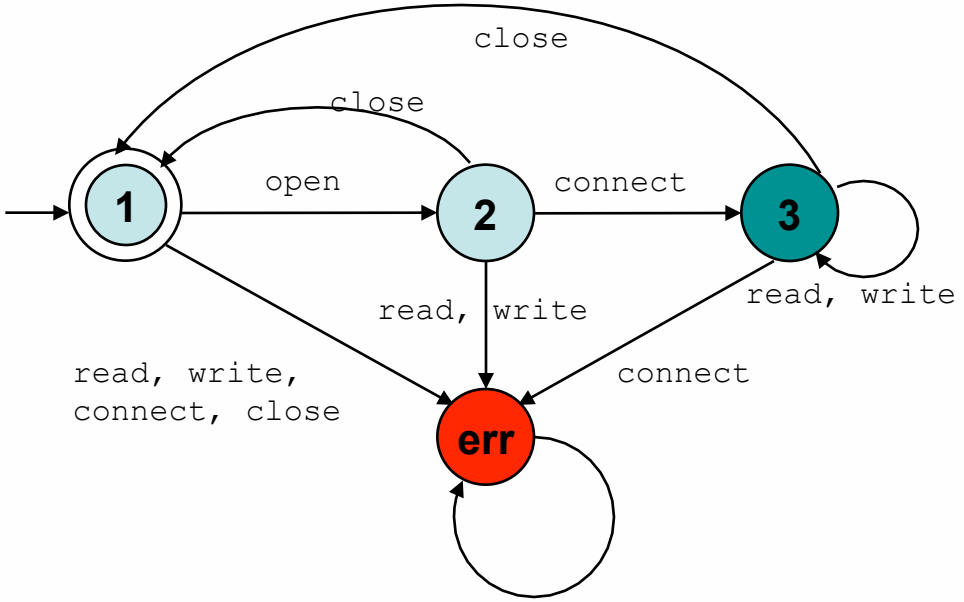


ESQuaReD

University of Nebraska Lincoln

Dynamic Typestate Analysis

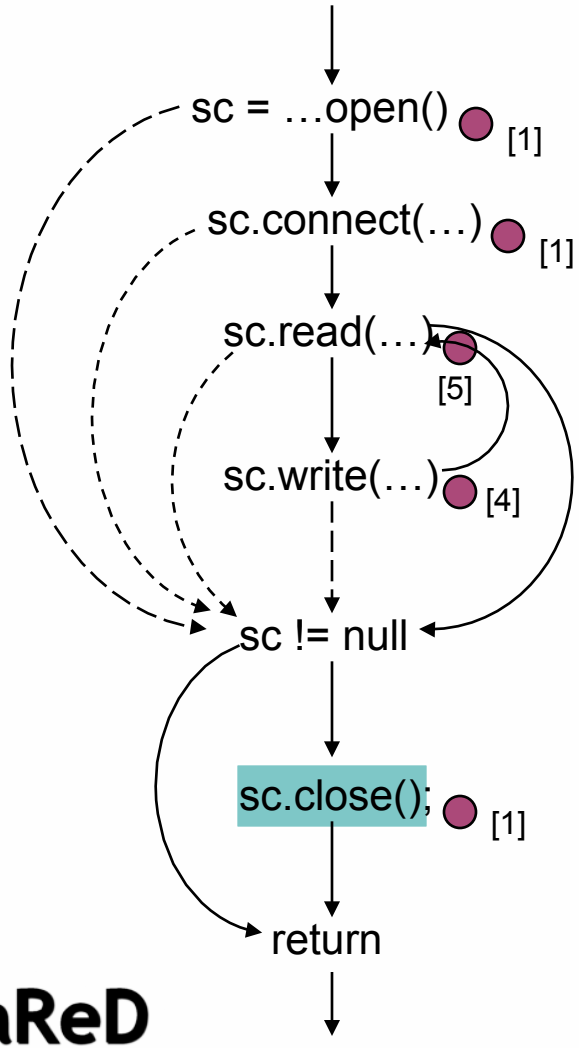


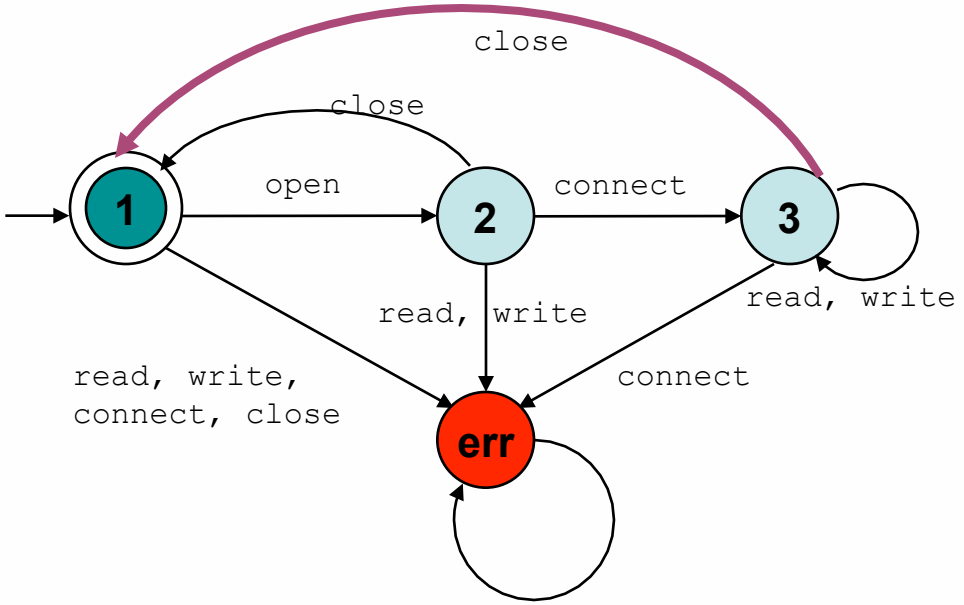


ESQuaReD

University of Nebraska Lincoln

Dynamic Typestate Analysis

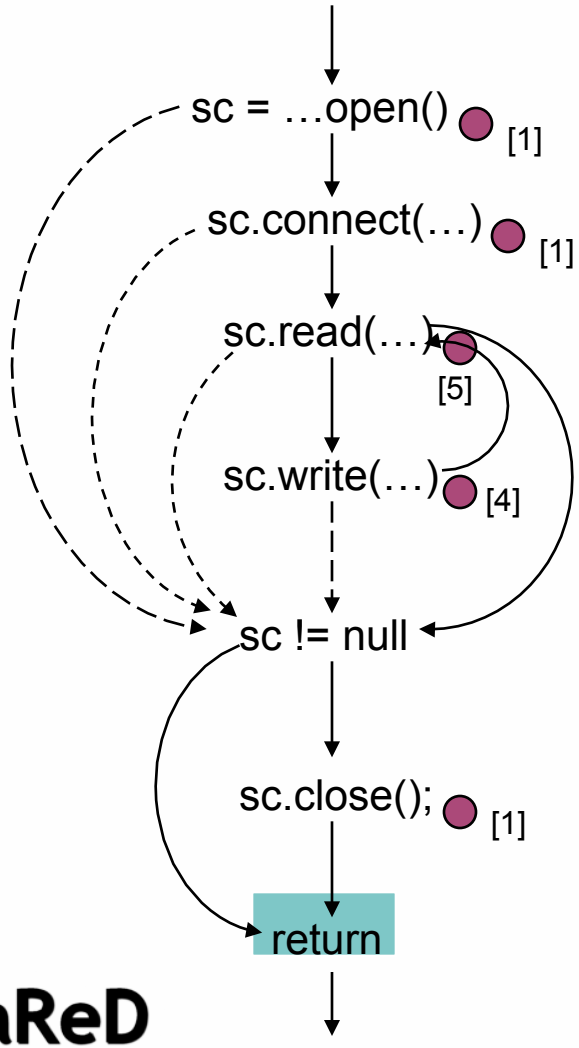


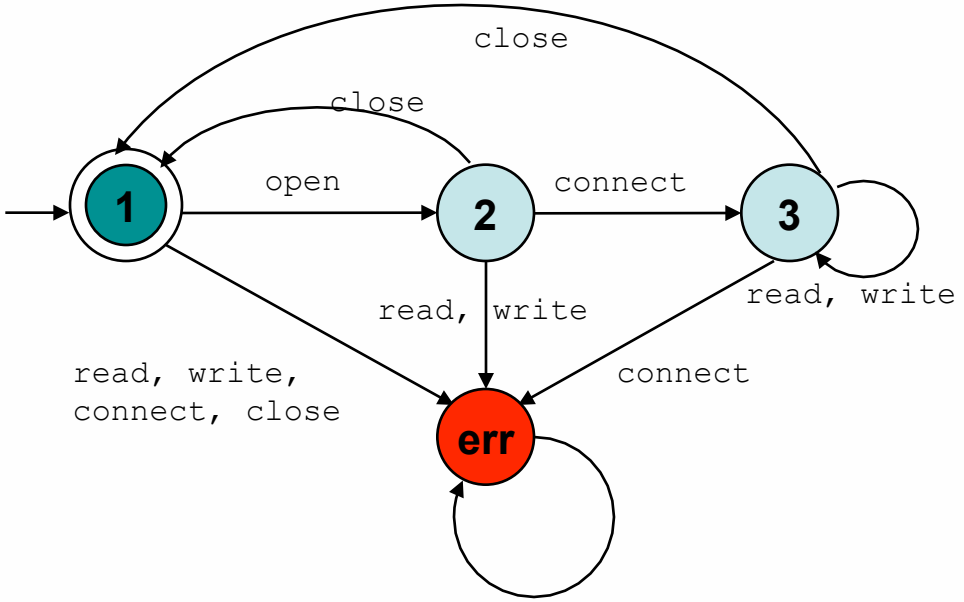


ESQuaReD

University of Nebraska Lincoln

Dynamic Typestate Analysis

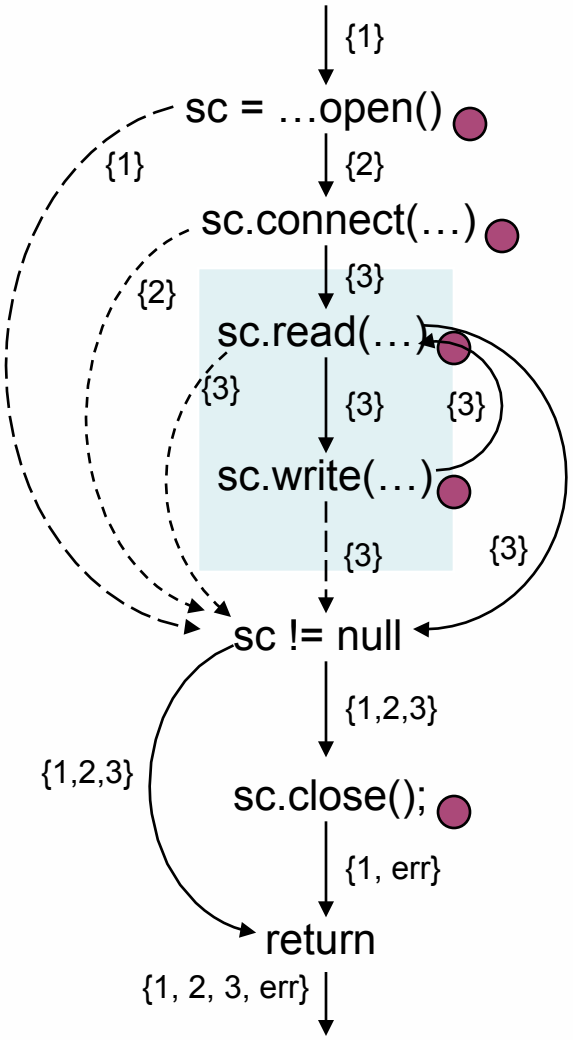




ESQuaReD

University of Nebraska Lincoln

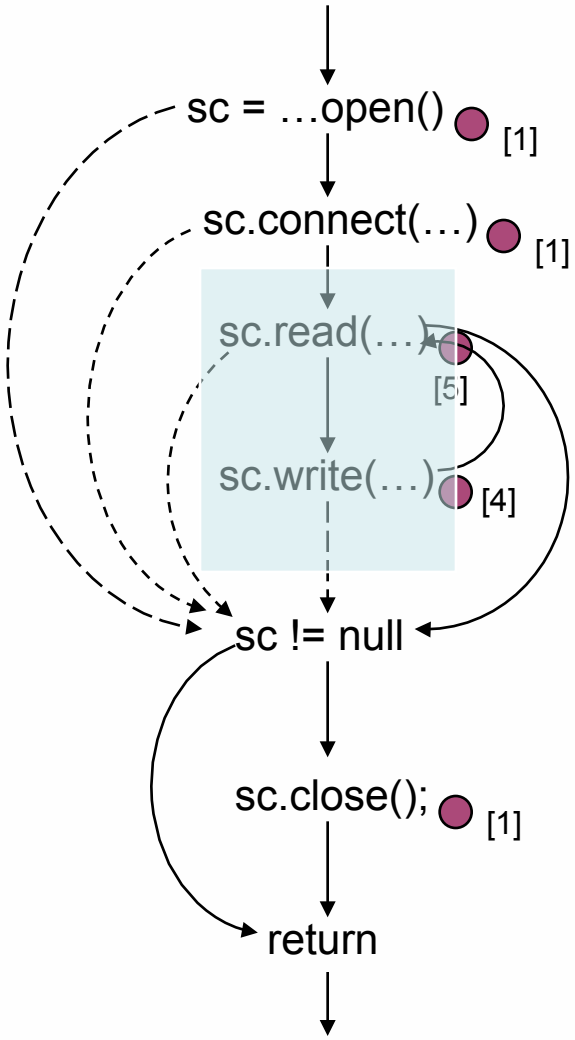
Leveraging Static Typestate Analysis



ESQuaReD

University of Nebraska Lincoln

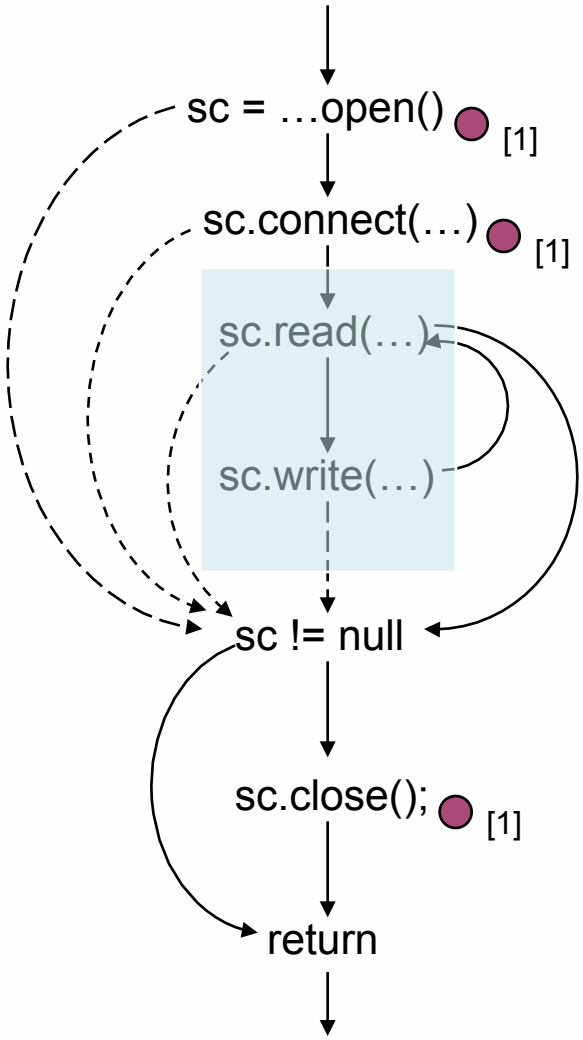
Leveraging Static Typestate Analysis



ESQuaReD

University of Nebraska Lincoln

Leveraging Static Typestate Analysis



ESQuaReD

University of Nebraska Lincoln

Safe Region

A single-entry region, r, of a control flow graph such that

$$\forall p, p' \in Paths(r)\ \forall s \in S - \{err\}:$$
$$\Delta(s, \sigma(p)) = \Delta(s, \sigma(p')) \wedge$$
$$\Delta(s, \sigma(p)) \neq err$$

where

$(S, \Sigma, \delta, s_o, A)$

$err \notin A \wedge \forall a \in \Sigma : \delta(err, a) = err$

$\Delta : S \times \Sigma^+ \mapsto S$

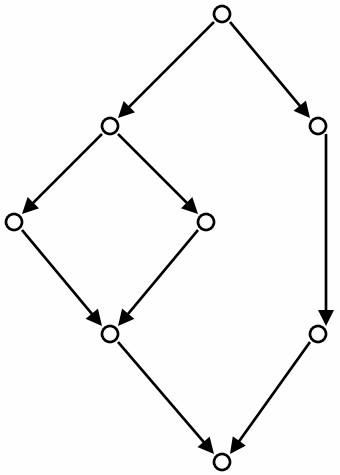
ESQuaReD

University of Nebraska Lincoln

Safe Region

A region of a control flow graph for which

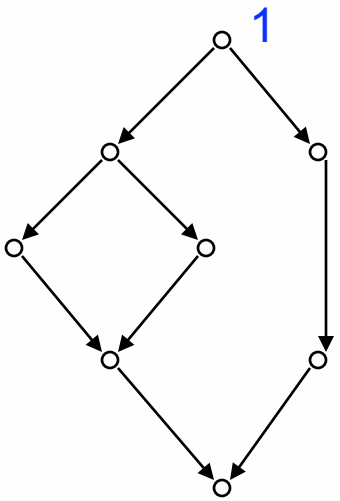
- all paths have the same cumulative effect on the typestate FSA
- no transitions from a non-error state to error state



Safe Region

A region of a control flow graph for which

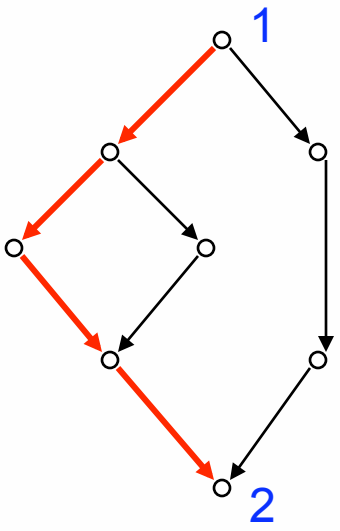
- all paths have the same cumulative effect on the typestate FSA
- no transitions from a non-error state to error state



Safe Region

A region of a control flow graph for which

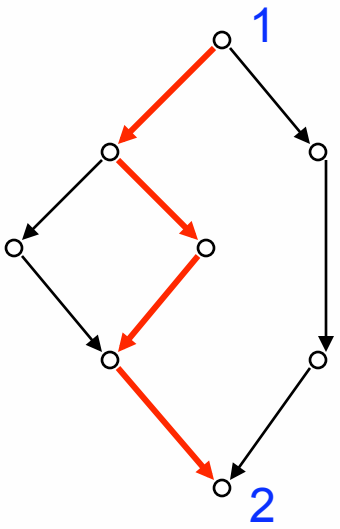
- all paths have the same cumulative effect on the typestate FSA
- no transitions from a non-error state to error state



Safe Region

A region of a control flow graph for which

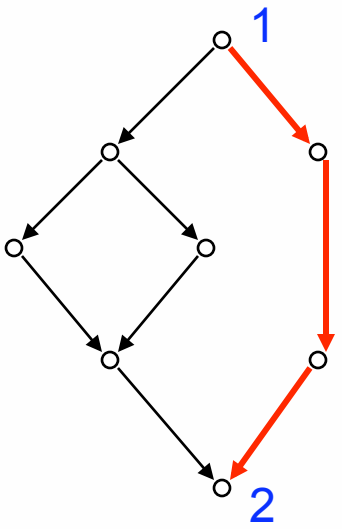
- all paths have the same cumulative effect on the typestate FSA
- no transitions from a non-error state to error state



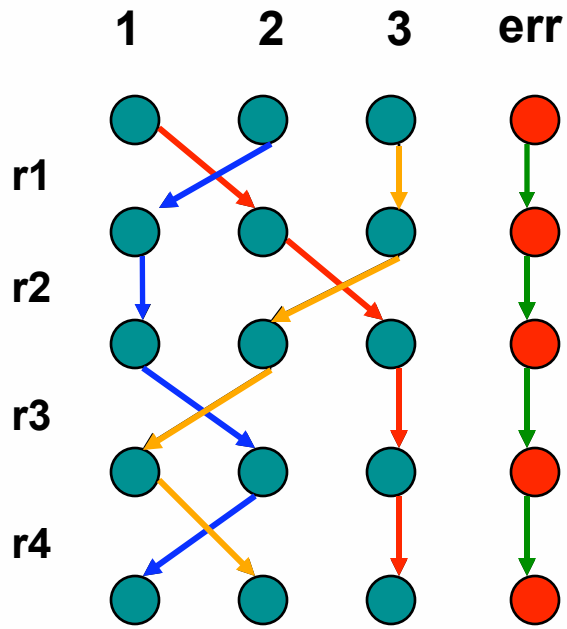
Safe Region

A region of a control flow graph for which

- all paths have the same cumulative effect on the typestate FSA
- no transitions from a non-error state to error state



Example of Safe Region



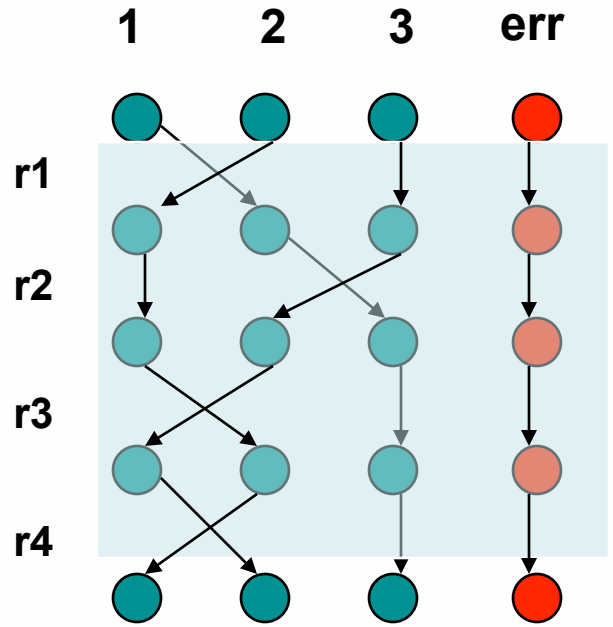
1 -> 3

2 -> 1

3 -> 2

err -> err

Example of Safe Region



Reachably Safe Region

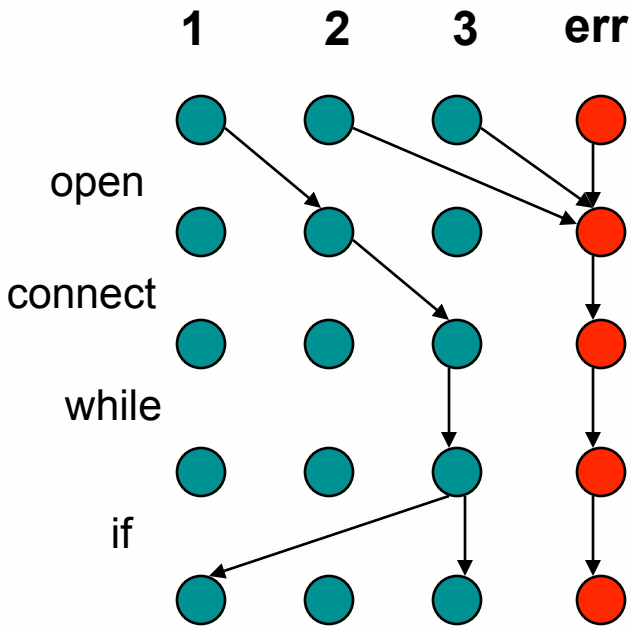
Not all states of a property reach all program points in a typestate analysis

Reachably Safe Region

– A region of a control flow graph which is safe relative to the subset of the typestates that may reach its entry

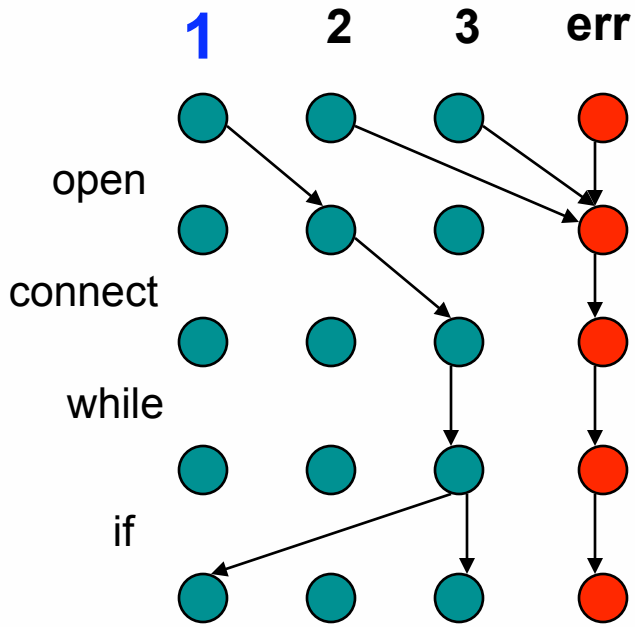
Example of Reachably Safe Region

```
public void simplifiedTransformData()
{
        SocketChannel sc;
        ByteBuffer buf;
        sc = SocketChannel.open();
        sc.connect(new
        InetSocketAddress(…));
        while (sc.read(buf) != -1){
              sc.write(buf);
        }
        if (sc != null)
              sc.close();
}
```



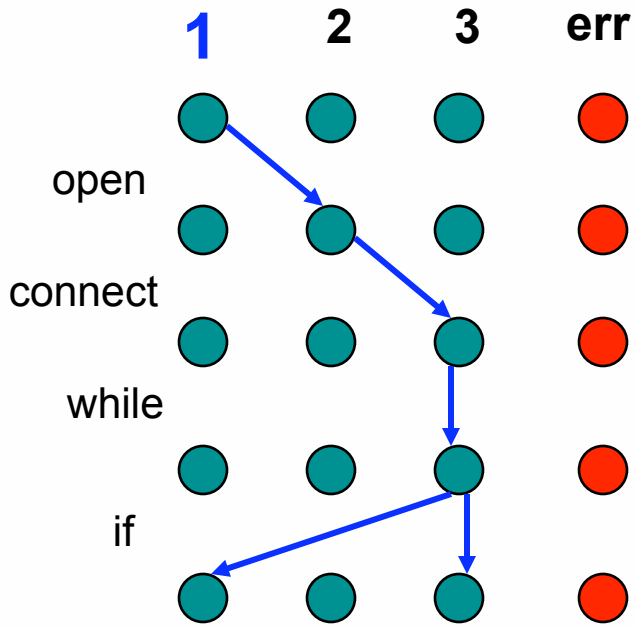
Example of Reachably Safe Region

```
public void simplifiedTransformData()
{
        SocketChannel sc;
        ByteBuffer buf;
        sc = SocketChannel.open();
        sc.connect(new
        InetSocketAddress(…));
        while (sc.read(buf) != -1){
              sc.write(buf);
        }
        if (sc != null)
              sc.close();
}
```



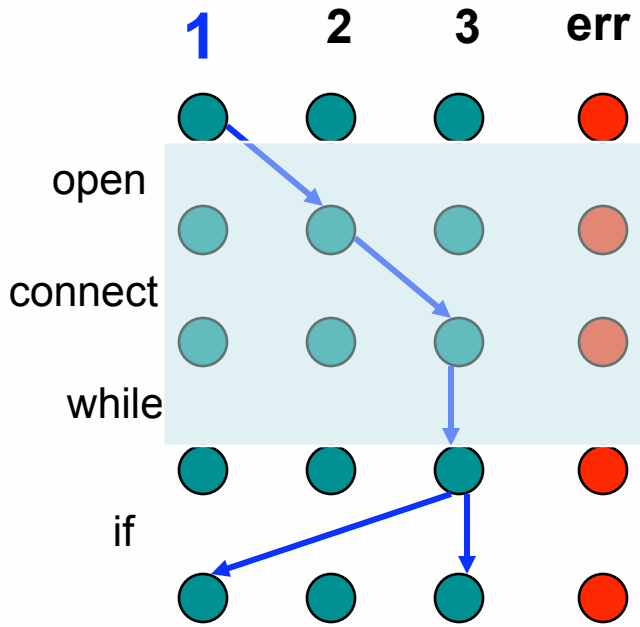
Example of Reachably Safe Region

```
public void simplifiedTransformData()
{
        SocketChannel sc;
        ByteBuffer buf;
        sc = SocketChannel.open();
        sc.connect(new
        InetSocketAddress(…));
        while (sc.read(buf) != -1){
              sc.write(buf);
        }
        if (sc != null)
              sc.close();
}
```



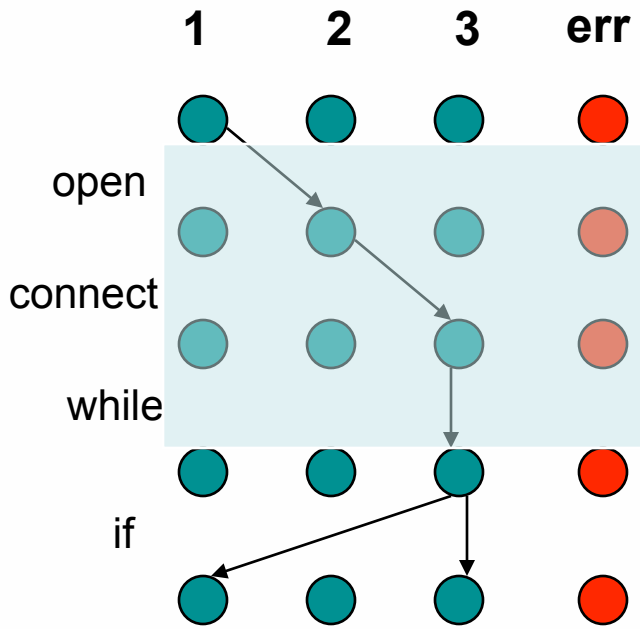
Example of Reachably Safe Region

```
public void simplifiedTransformData()
{
        SocketChannel sc;
        ByteBuffer buf;
        sc = SocketChannel.open();
        sc.connect(new
        InetSocketAddress(…));
        while (sc.read(buf) != -1){
              sc.write(buf);
        }
        if (sc != null)
              sc.close();
}
```



Example of Reachably Safe Region

```
public void simplifiedTransformData()
{
        SocketChannel sc;
        ByteBuffer buf;
        sc = SocketChannel.open();
        sc.connect(new
        InetSocketAddress(…));
        while (sc.read(buf) != -1){
              sc.write(buf);
        }
        if (sc != null)
              sc.close();
}
```



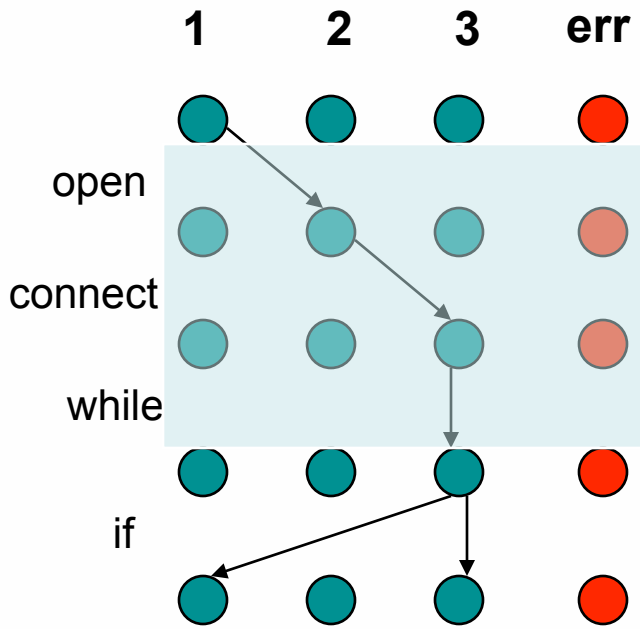
Identity Safe Region

- Identity Safe Region
  - A special case of a (reachably) safe region that yields identity summary on the subset of the typestates that may reach its entry

$$\forall p \in Paths(r)\ \forall s \in S :$$
$$\Delta(s, \sigma(p)) = s$$

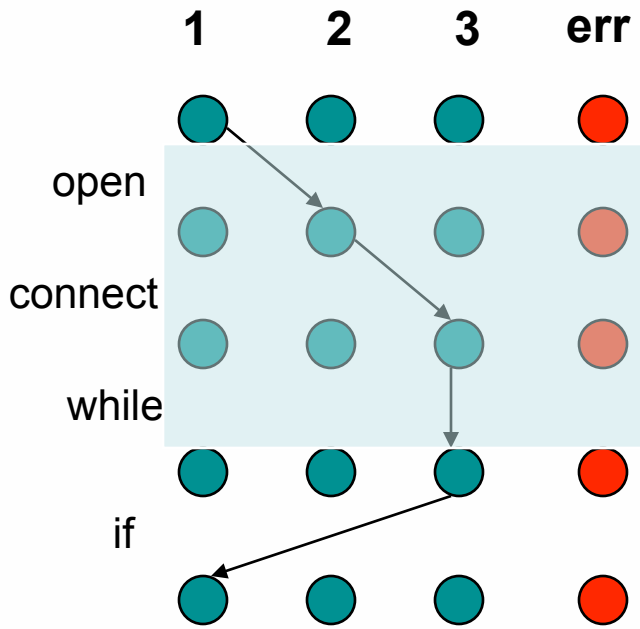
Example of Identity Safe Region

```
public void simplifiedTransformData()
{
        SocketChannel sc;
        ByteBuffer buf;
        sc = SocketChannel.open();
        sc.connect(new
        InetSocketAddress(…));
        while (sc.read(buf) != -1){
              sc.write(buf);
        }
        if (sc != null)
              sc.close();
}
```



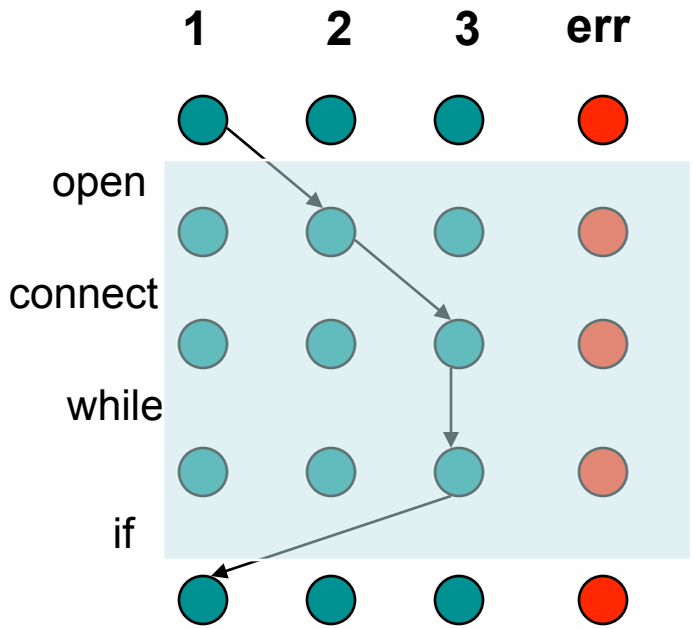
Example of Identity Safe Region

```
public void simplifiedTransformData()
{
        SocketChannel sc;
        ByteBuffer buf;
        sc = SocketChannel.open();
        sc.connect(new
        InetSocketAddress(…));
        while (sc.read(buf) != -1){
              sc.write(buf);
        }
        if (sc != null)
              sc.close();
}
```



Example of Identity Safe Region

```
public void simplifiedTransformData()
{
        SocketChannel sc;
        ByteBuffer buf;
        sc = SocketChannel.open();
        sc.connect(new
        InetSocketAddress(…));
        while (sc.read(buf) != -1){
              sc.write(buf);
        }
        if (sc != null)
              sc.close();
}
```



Algorithm

Basic Steps

1. Reduces a control flow graph region to a sequence of single entry regions
2. Calls static typestate analysis to calculate functional summaries of reachable program regions
3. Within a region, identify candidate safe regions by marking boundaries that cannot be crossed by any safe region
4. Identify safe regions inside candidate safe regions
5. Drop (and if required, add) FSA transitions for safe regions
6. Repeat the steps for all regions that lie outside safe regions

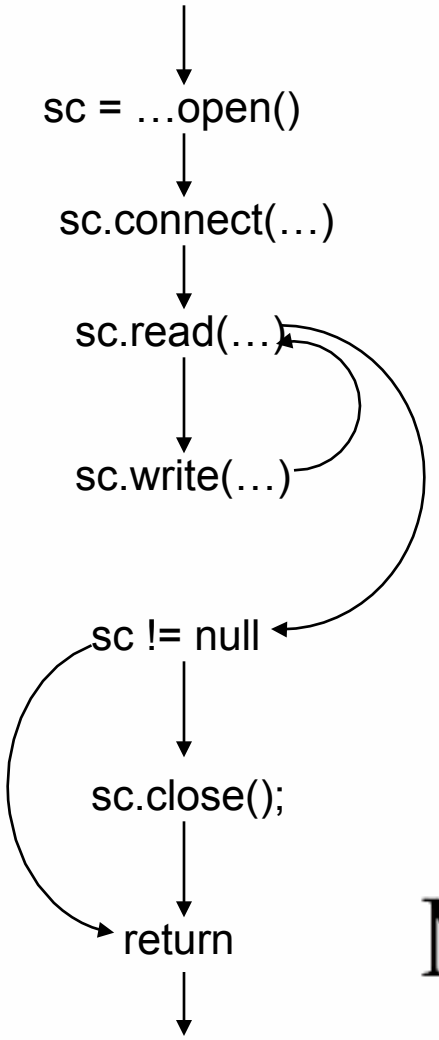
Algorithm

Basic Steps

1. Reduces a control flow graph region to a sequence of single entry regions
2. Calls static typestate analysis to calculate functional summaries of reachable program regions
3. Within a region, identify candidate safe regions by marking boundaries that cannot be crossed by any safe region
4. Identify safe regions inside candidate safe regions
5. Drop (and if required, add) FSA transitions for safe regions
6. Repeat the steps for all regions that lie outside safe regions

Reduce Control Flow Graph

```
public void simplifiedTransformData()
{
        SocketChannel sc;
        ByteBuffer buf;
        sc = SocketChannel.open();
        sc.connect(new
        InetSocketAddress(…));
        while (sc.read(buf) != -1){
              sc.write(buf);
        }
        if (sc != null)
              sc.close();
}
```

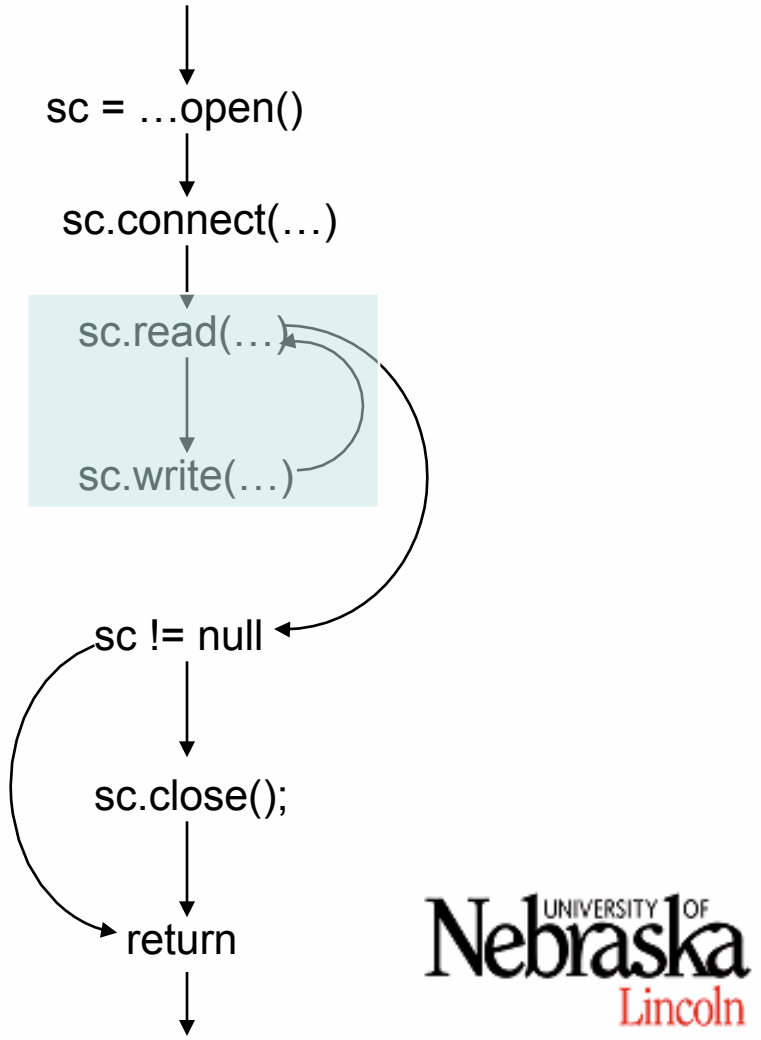


ESQuaReD

University of Nebraska Lincoln

Reduce Control Flow Graph

```
public void simplifiedTransformData()
{
        SocketChannel sc;
        ByteBuffer buf;
        sc = SocketChannel.open();
        sc.connect(new
        InetSocketAddress(…));
        while (sc.read(buf) != -1){
              sc.write(buf);
        }
        if (sc != null)
              sc.close();
}
```

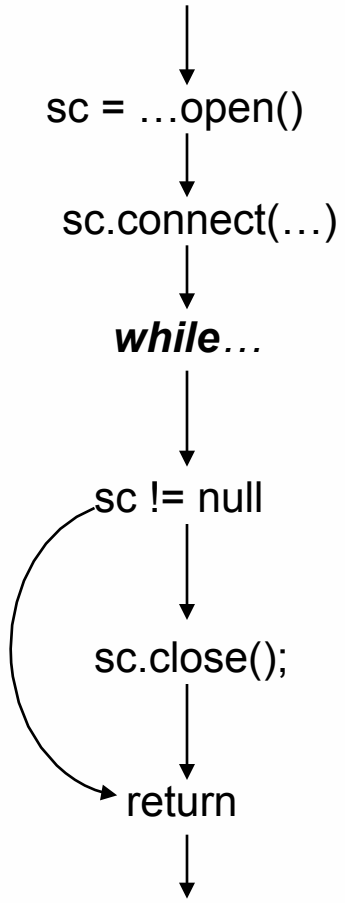


ESQuaReD

University of Nebraska Lincoln

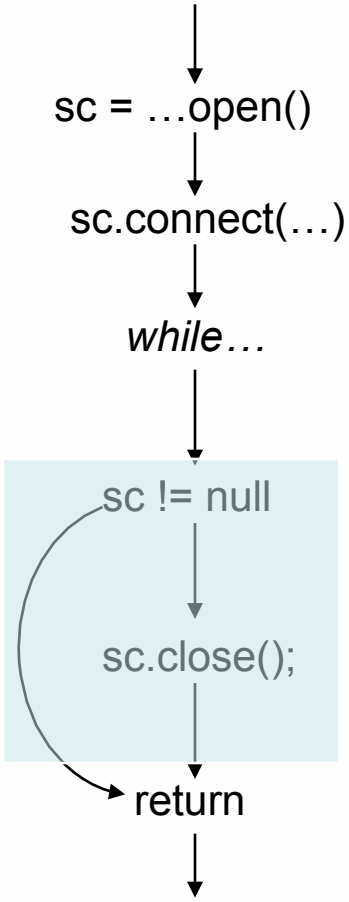
Reduce Control Flow Graph

```
public void simplifiedTransformData()
{
        SocketChannel sc;
        ByteBuffer buf;
        sc = SocketChannel.open();
        sc.connect(new
        InetSocketAddress(…));
        while (sc.read(buf) != -1){
                sc.write(buf);
        }
        if (sc != null)
                sc.close();
}
```



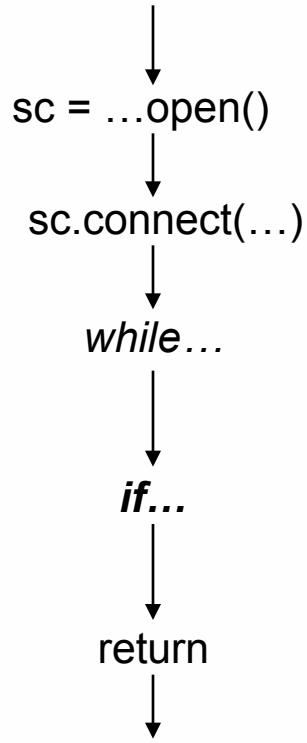
Reduce Control Flow Graph

```
public void simplifiedTransformData()
{
        SocketChannel sc;
        ByteBuffer buf;
        sc = SocketChannel.open();
        sc.connect(new
        InetSocketAddress(…));
        while (sc.read(buf) != -1){
                sc.write(buf);
        }
        if (sc != null)
                sc.close();
}
```



Reduce Control Flow Graph

```
public void simplifiedTransformData()
{
        SocketChannel sc;
        ByteBuffer buf;
        sc = SocketChannel.open();
        sc.connect(new
        InetSocketAddress(…));
        while (sc.read(buf) != -1){
                sc.write(buf);
        }
        if (sc != null)
                sc.close();
}
```



ESQuaReD

University of Nebraska Lincoln

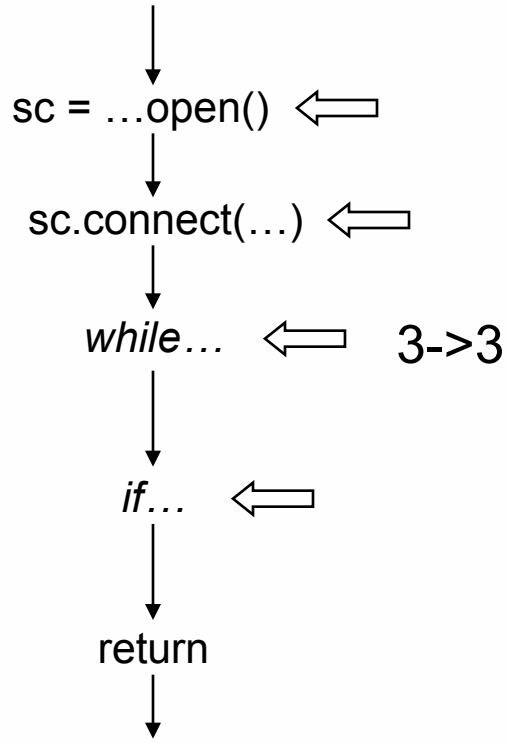
Algorithm

Basic Steps

1. Reduces a control flow graph region to a sequence of single entry and single exit regions
2. Calls static typestate analysis to calculate functional summaries of reachable program regions
3. Within a region, identify candidate safe regions by marking boundaries that cannot be crossed by any safe region
4. Identify safe regions inside candidate safe regions
5. Drop (and if required, add) FSA transitions for safe regions
6. Repeat the steps for all regions that lie outside safe regions

Calculate Functional Summary

```
public void simplifiedTransformData()
{
        SocketChannel sc;
        ByteBuffer buf;
        sc = SocketChannel.open();
        sc.connect(new
        InetSocketAddress(…));
        while (sc.read(buf) != -1){
                sc.write(buf);
        }
        if (sc != null)
                sc.close();
}
```



ESQuaReD

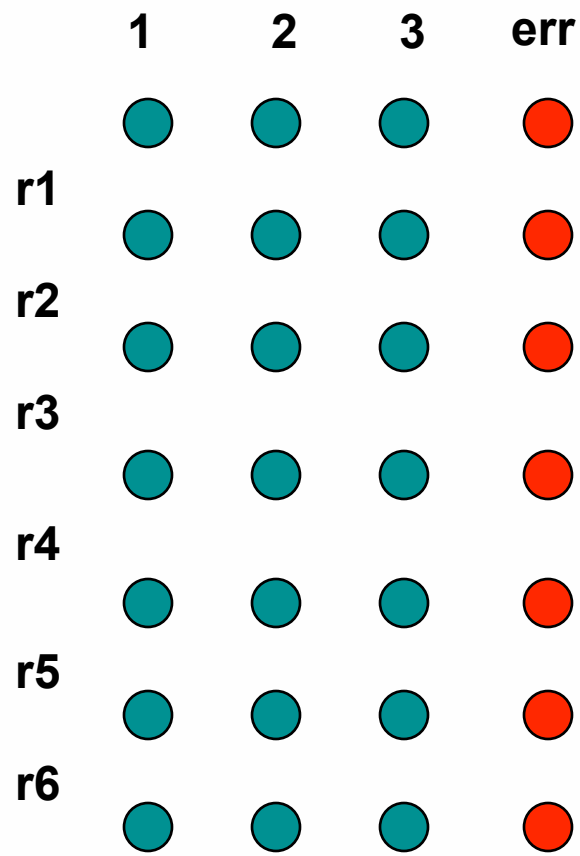
University of Nebraska Lincoln

Algorithm

Basic Steps

1. Reduces a control flow graph region to a sequence of single entry regions
2. Calls static typestate analysis to calculate functional summaries of reachable program regions
3. Within a region, identify candidate safe regions by marking boundaries that cannot be crossed by any safe region
4. Identify safe regions inside candidate safe regions
5. Drop (and if required, add) FSA transitions for safe regions
6. Repeat the steps for all regions that lie outside safe regions

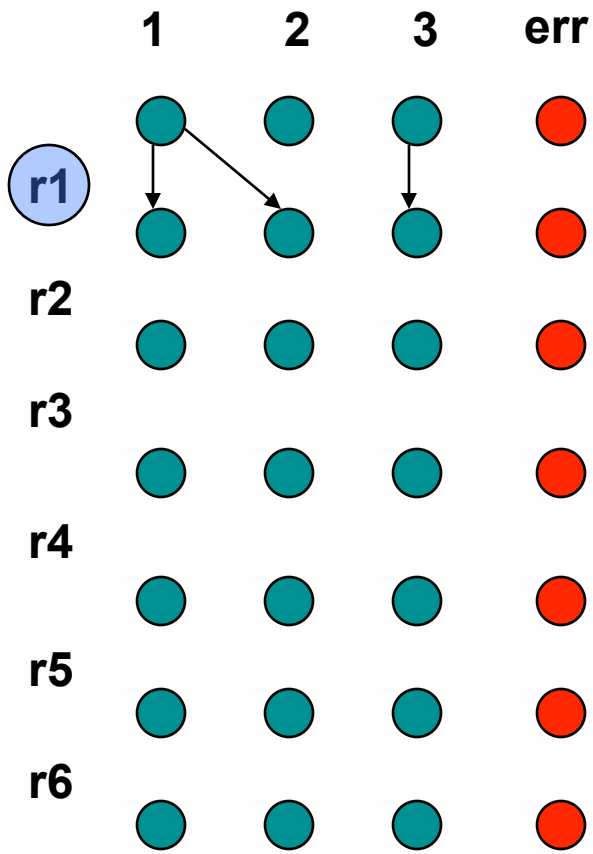
Identify Candidate Safe Regions



ESQuaReD

University of Nebraska Lincoln

Identify Candidate Safe Regions



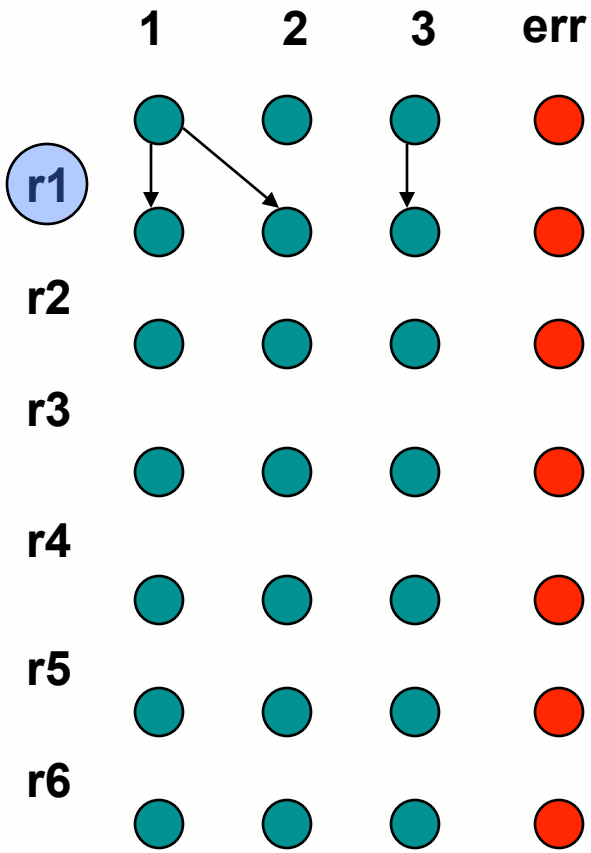
ESQuaReD

University of Nebraska Lincoln

Identify Candidate Safe Regions

	r1
r1	1->{1,2} 3->{3}

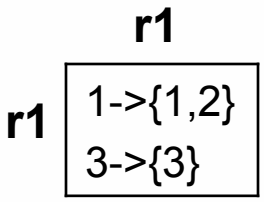
Region Matrix



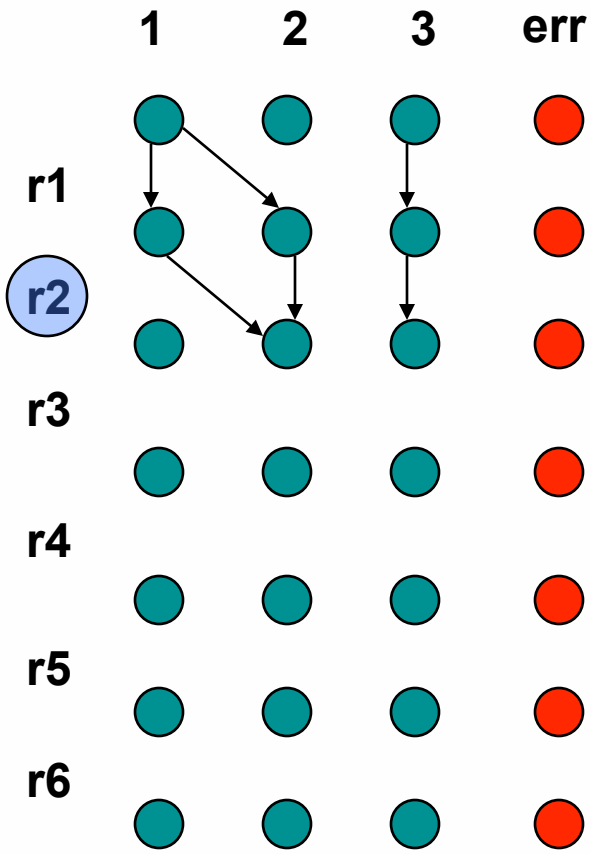
ESQuaReD

University of Nebraska Lincoln

Identify Candidate Safe Regions



Region Matrix



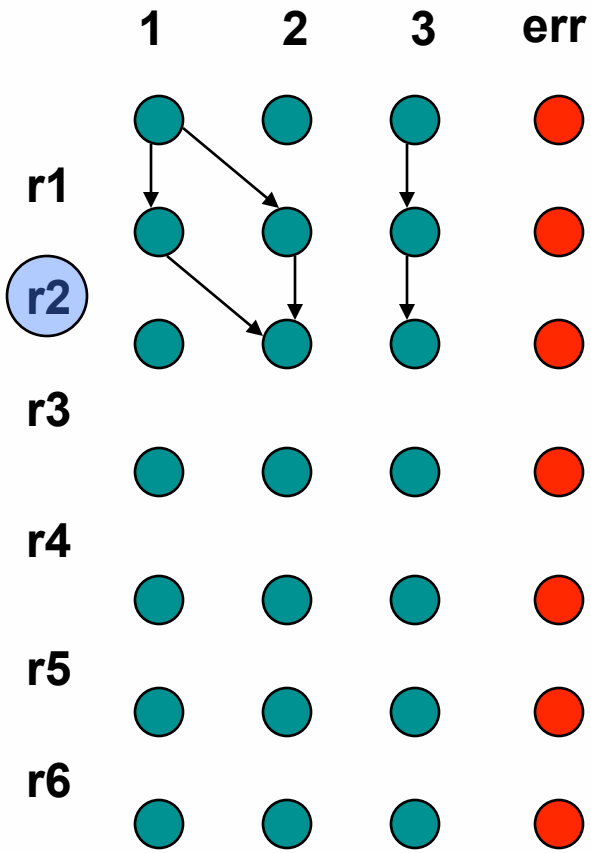
ESQuaReD

University of Nebraska Lincoln

Identify Candidate Safe Regions

	r1	r2
r1	1->{1,2} 3->{3}	1->{2} 3->{3}
r2		1->{2} 2->{2} 3->{3}

Region Matrix



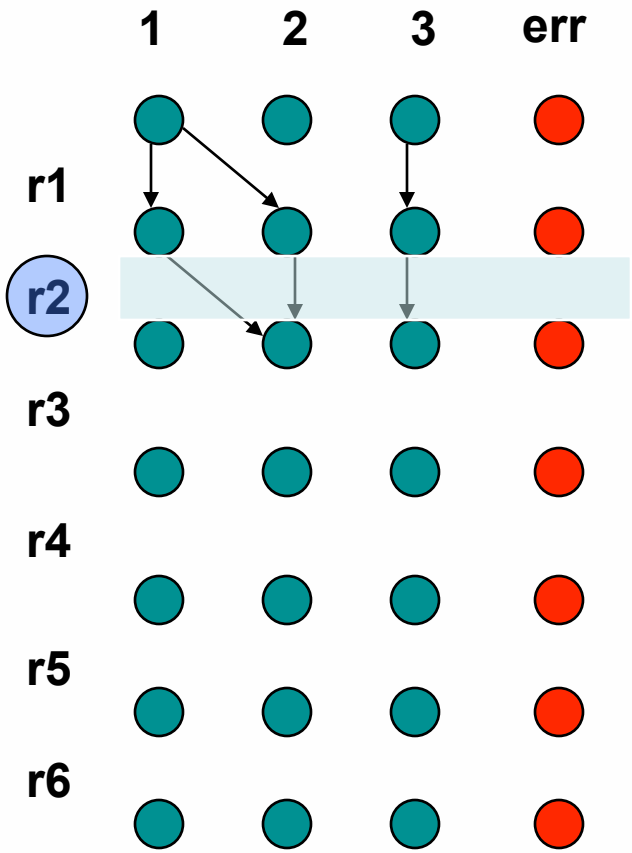
ESQuaReD

University of Nebraska Lincoln

Identify Candidate Safe Regions

	r1	r2
r1	1->{1,2} 3->{3}	1->{2} 3->{3}
r2		1->{2} 2->{2} 3->{3}

Region Matrix



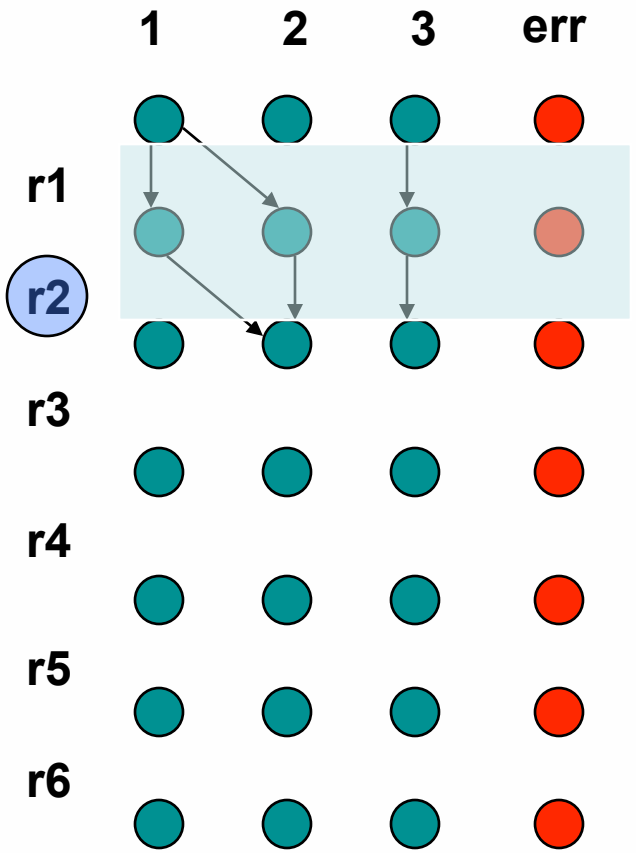
ESQuaReD

University of Nebraska Lincoln

Identify Candidate Safe Regions

	r1	r2
r1	1->{1,2} 3->{3}	1->{2} 3->{3}
r2		1->{2} 2->{2} 3->{3}

Region Matrix



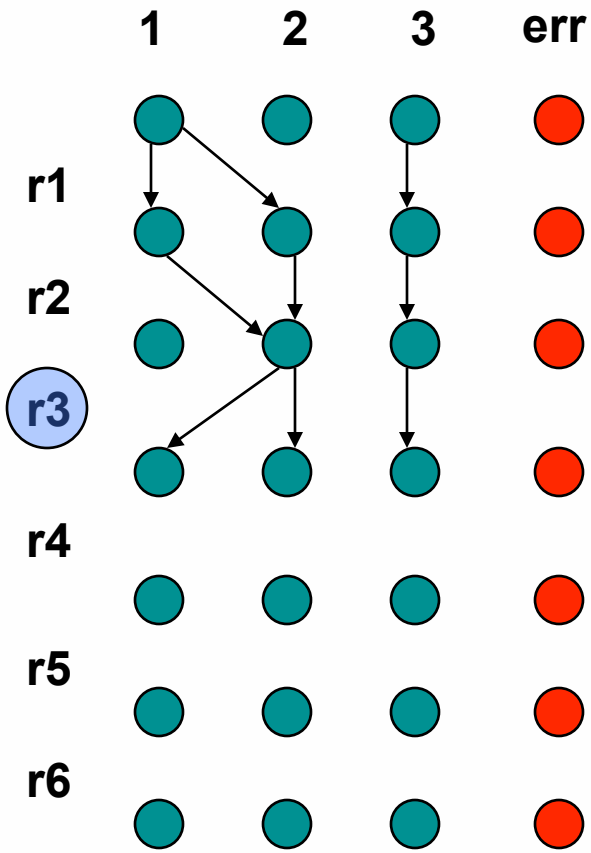
ESQuaReD

University of Nebraska Lincoln

Identify Candidate Safe Regions

	r1	r2	r3
r1	1->{1,2} 3->{3}	1->{2} 3->{3}	1->{1,2} 3->{3}
r2		1->{2} 2->{2} 3->{3}	1->{1,2} 2->{1,2} 3->{3}
r3			2->{1,2} 3->{3}

Region Matrix



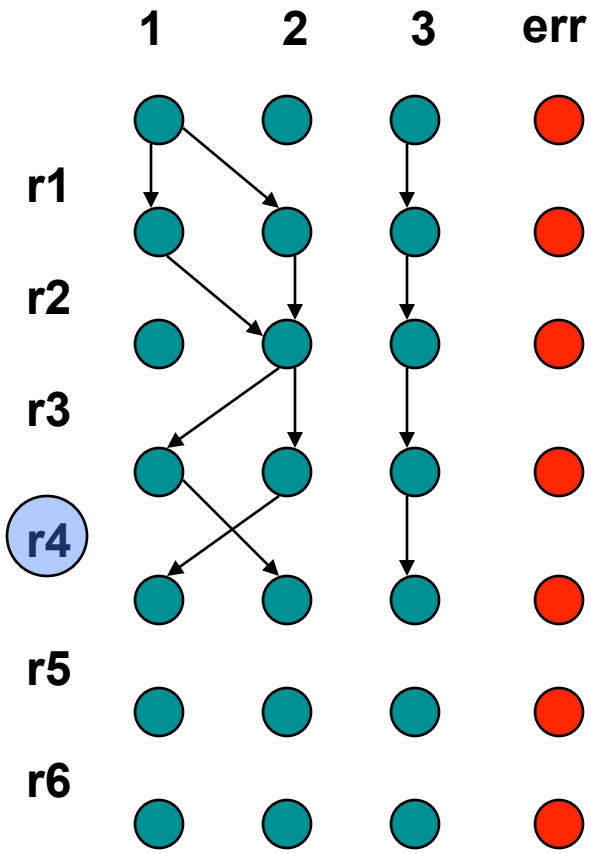
ESQuaReD

University of Nebraska Lincoln

Identify Candidate Safe Regions

	r1	r2	r3
r1	1->{1,2} 3->{3}	1->{2} 3->{3}	1->{1,2} 3->{3}
r2		1->{2} 2->{2} 3->{3}	1->{1,2} 2->{1,2} 3->{3}
r3			2->{1,2} 3->{3}

Region Matrix



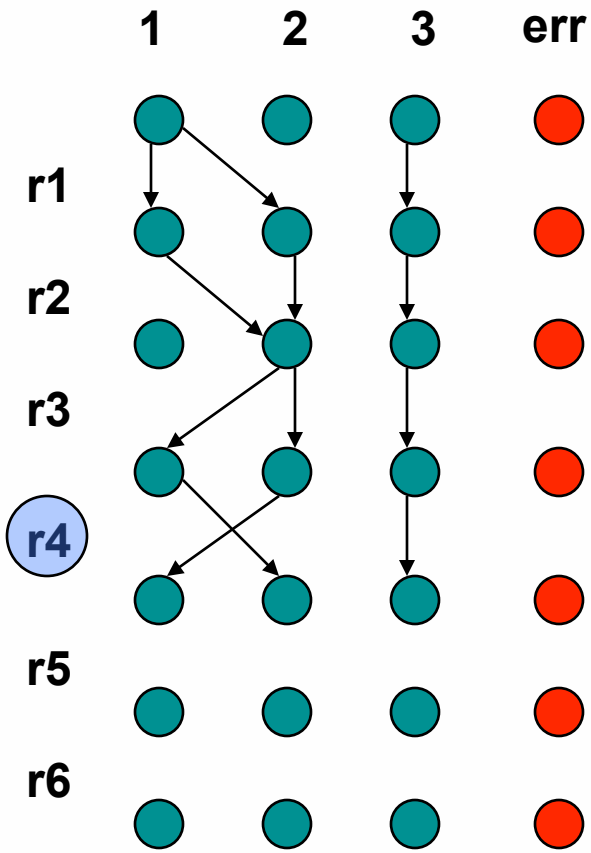
ESQuaReD

University of Nebraska Lincoln

Identify Candidate Safe Regions

	r1	r2	r3	r4
r1	1->{1,2} 3->{3}	1->{2} 3->{3}	1->{1,2} 3->{3}	1->{1,2} 3->{3}
r2		1->{2} 2->{2} 3->{3}	1->{1,2} 2->{1,2} 3->{3}	1->{1,2} 2->{1,2} 3->{3}
r3			2->{1,2} 3->{3}	2->{1,2} 3->{3}
r4				1->{2} 2->{1} 3->{3}

Region Matrix



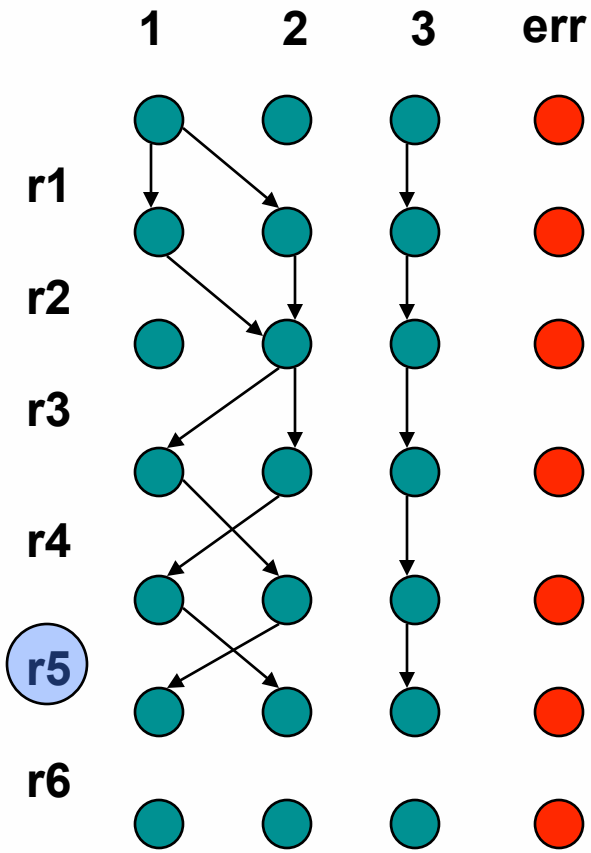
ESQuaReD

University of Nebraska Lincoln

Identify Candidate Safe Regions

	r1	r2	r3	r4
r1	1->{1,2} 3->{3}	1->{2} 3->{3}	1->{1,2} 3->{3}	1->{1,2} 3->{3}
r2		1->{2} 2->{2} 3->{3}	1->{1,2} 2->{1,2} 3->{3}	1->{1,2} 2->{1,2} 3->{3}
r3			2->{1,2} 3->{3}	2->{1,2} 3->{3}
r4				1->{2} 2->{1} 3->{3}

Region Matrix



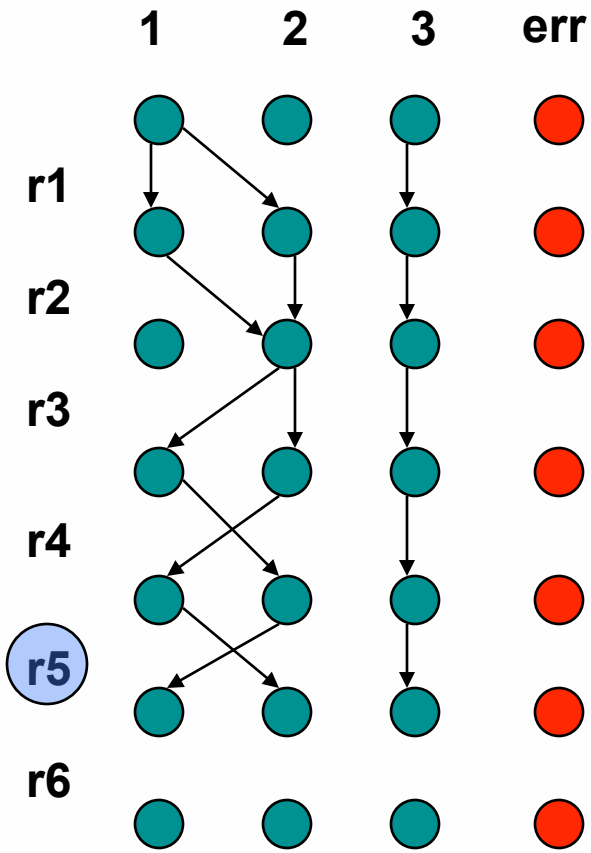
ESQuaReD

University of Nebraska Lincoln

Identify Candidate Safe Regions

	r1	r2	r3	r4	r5
r1	1->{1,2} 3->{3}	1->{2} 3->{3}	1->{1,2} 3->{3}	1->{1,2} 3->{3}	1->{1,2} 3->{3}
r2		1->{2} 2->{2} 3->{3}	1->{1,2} 2->{1,2} 3->{3}	1->{1,2} 2->{1,2} 3->{3}	1->{1,2} 2->{1,2} 3->{3}
r3			2->{1,2} 3->{3}	2->{1,2} 3->{3}	2->{1,2} 3->{3}
r4				1->{2} 2->{1} 3->{3}	1->{1} 2->{2} 3->{3}
r5					1->{2} 2->{1} 3->{3}

Region Matrix



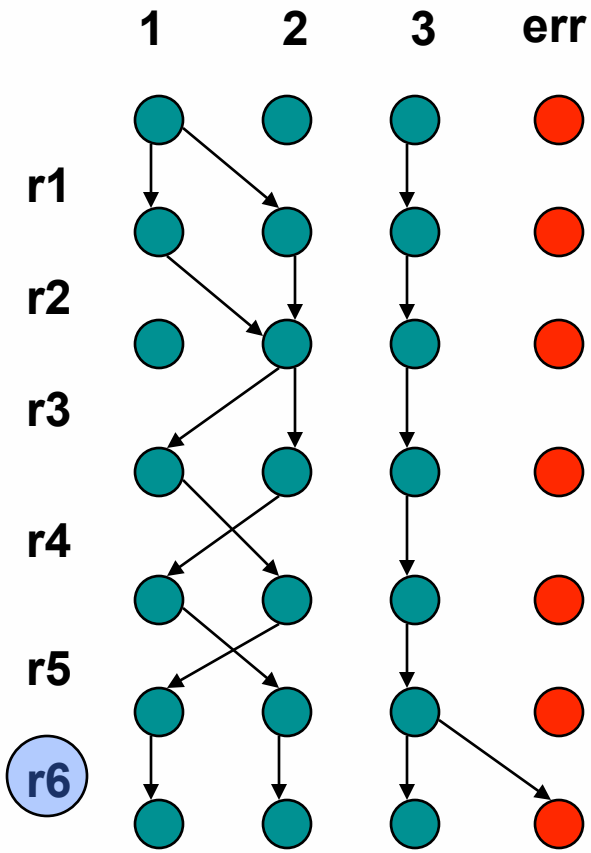
ESQuaReD

University of Nebraska Lincoln

Identify Candidate Safe Regions

	r1	r2	r3	r4	r5
r1	1->{1,2} 3->{3}	1->{2} 3->{3}	1->{1,2} 3->{3}	1->{1,2} 3->{3}	1->{1,2} 3->{3}
r2		1->{2} 2->{2} 3->{3}	1->{1,2} 2->{1,2} 3->{3}	1->{1,2} 2->{1,2} 3->{3}	1->{1,2} 2->{1,2} 3->{3}
r3			2->{1,2} 3->{3}	2->{1,2} 3->{3}	2->{1,2} 3->{3}
r4				1->{2} 2->{1} 3->{3}	1->{1} 2->{2} 3->{3}
r5					1->{2} 2->{1} 3->{3}

Region Matrix



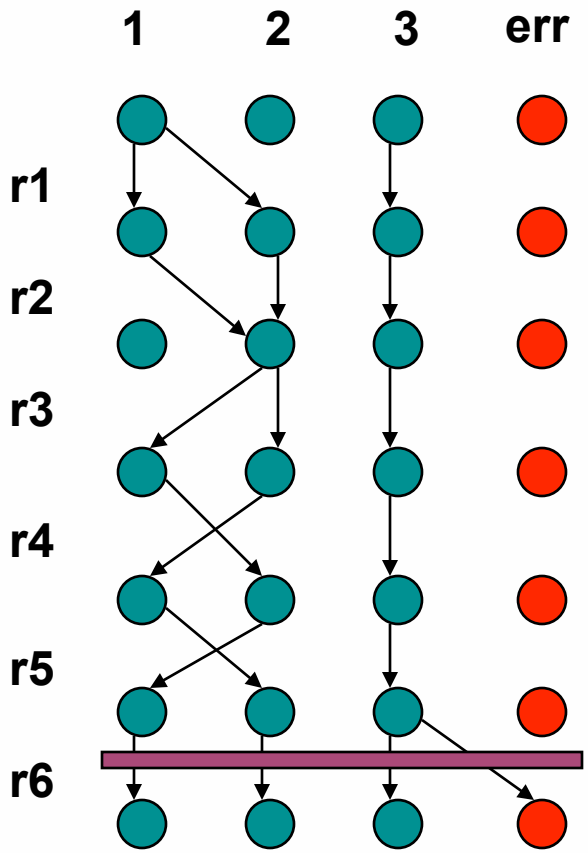
ESQuaReD

University of Nebraska Lincoln

Identify Candidate Safe Regions

	r1	r2	r3	r4	r5
r1	1->{1,2} 3->{3}	1->{2} 3->{3}	1->{1,2} 3->{3}	1->{1,2} 3->{3}	1->{1,2} 3->{3}
r2		1->{2} 2->{2} 3->{3}	1->{1,2} 2->{1,2} 3->{3}	1->{1,2} 2->{1,2} 3->{3}	1->{1,2} 2->{1,2} 3->{3}
r3			2->{1,2} 3->{3}	2->{1,2} 3->{3}	2->{1,2} 3->{3}
r4				1->{2} 2->{1} 3->{3}	1->{1} 2->{2} 3->{3}
r5					1->{2} 2->{1} 3->{3}

Region Matrix



ESQuaReD

University of Nebraska Lincoln

Algorithm

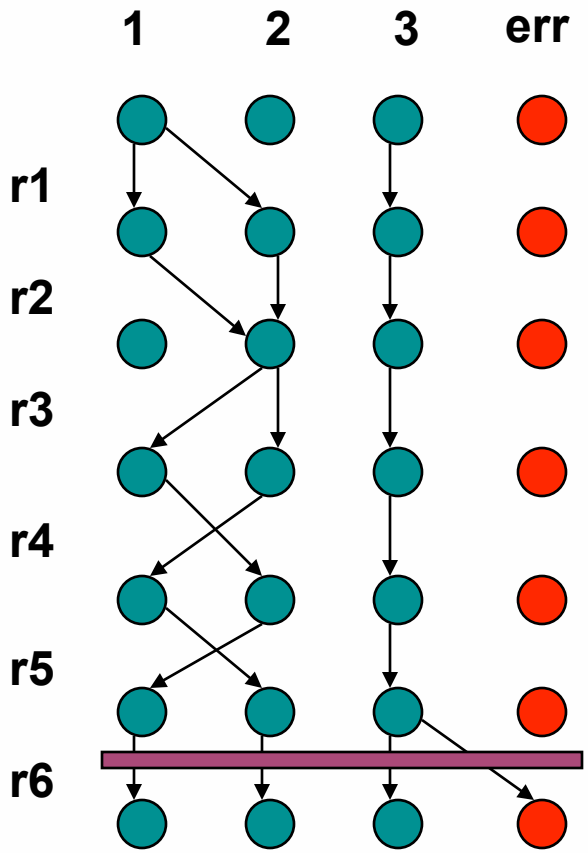
Basic Steps

1. Reduces a control flow graph region to a sequence of single entry regions
2. Calls static typestate analysis to calculate functional summaries of reachable program regions
3. Within a region, identify candidate safe regions by marking boundaries that cannot be crossed by any safe region
4. Identify safe regions inside candidate safe regions
5. Drop (and if required, add) FSA transitions for safe regions
6. Repeat the steps for all regions that lie outside safe regions

Identify Safe Regions

	r1	r2	r3	r4	r5
r1	1->{1,2} 3->{3}	1->{2} 3->{3}	1->{1,2} 3->{3}	1->{1,2} 3->{3}	1->{1,2} 3->{3}
r2		1->{2} 2->{2} 3->{3}	1->{1,2} 2->{1,2} 3->{3}	1->{1,2} 2->{1,2} 3->{3}	1->{1,2} 2->{1,2} 3->{3}
r3			2->{1,2} 3->{3}	2->{1,2} 3->{3}	2->{1,2} 3->{3}
r4				1->{2} 2->{1} 3->{3}	1->{1} 2->{2} 3->{3}
r5					1->{2} 2->{1} 3->{3}

Region Matrix



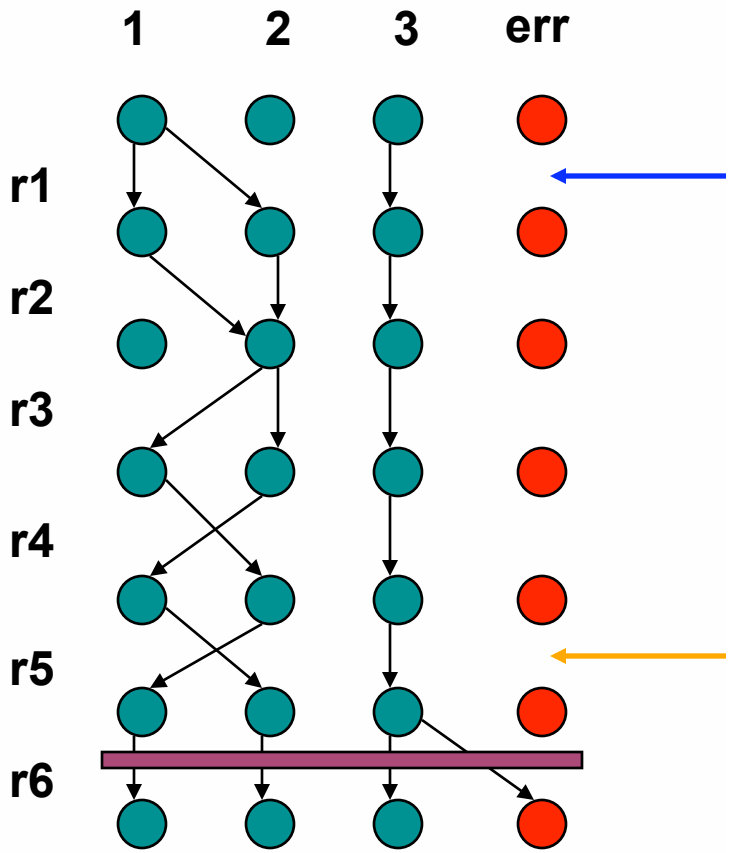
ESQuaReD

University of Nebraska Lincoln

Identify Safe Regions

	r1	r2	r3	r4	r5
r1	1->{1,2} 3->{3}	1->{2} 3->{3}	1->{1,2} 3->{3}	1->{1,2} 3->{3}	1->{1,2} 3->{3}
r2		1->{2} 2->{2} 3->{3}	1->{1,2} 2->{1,2} 3->{3}	1->{1,2} 2->{1,2} 3->{3}	1->{1,2} 2->{1,2} 3->{3}
r3			2->{1,2} 3->{3}	2->{1,2} 3->{3}	2->{1,2} 3->{3}
r4				1->{2} 2->{1} 3->{3}	1->{1} 2->{2} 3->{3}
r5					1->{2} 2->{1} 3->{3}

Region Matrix



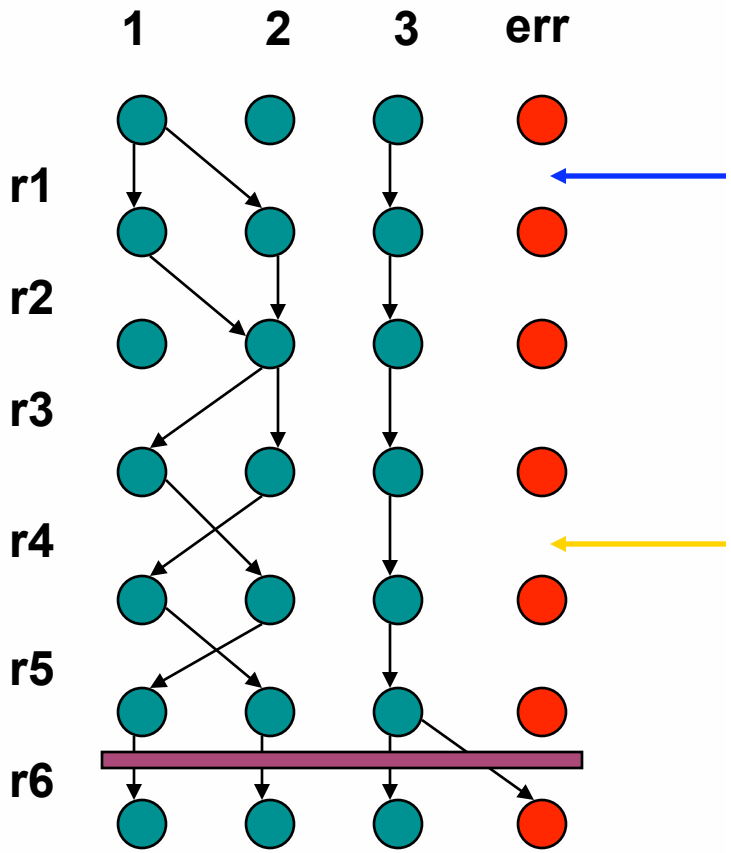
ESQuaReD

University of Nebraska Lincoln

Identify Safe Regions

	r1	r2	r3	r4	r5
r1	1->{1,2} 3->{3}	1->{2} 3->{3}	1->{1,2} 3->{3}	1->{1,2} 3->{3}	1->{1,2} 3->{3}
r2		1->{2} 2->{2} 3->{3}	1->{1,2} 2->{1,2} 3->{3}	1->{1,2} 2->{1,2} 3->{3}	1->{1,2} 2->{1,2} 3->{3}
r3			2->{1,2} 3->{3}	2->{1,2} 3->{3}	2->{1,2} 3->{3}
r4				1->{2} 2->{1} 3->{3}	1->{1} 2->{2} 3->{3}
r5					1->{2} 2->{1} 3->{3}

Region Matrix



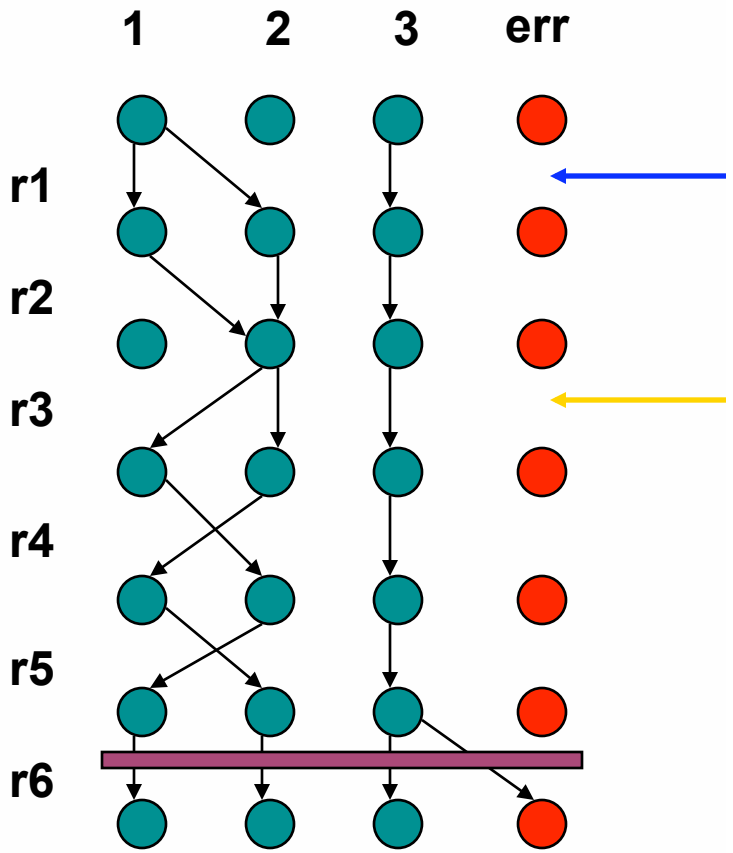
ESQuaReD

University of Nebraska Lincoln

Identify Safe Regions

	r1	r2	r3	r4	r5
r1	1->{1,2} 3->{3}	1->{2} 3->{3}	1->{1,2} 3->{3}	1->{1,2} 3->{3}	1->{1,2} 3->{3}
r2		1->{2} 2->{2} 3->{3}	1->{1,2} 2->{1,2} 3->{3}	1->{1,2} 2->{1,2} 3->{3}	1->{1,2} 2->{1,2} 3->{3}
r3			2->{1,2} 3->{3}	2->{1,2} 3->{3}	2->{1,2} 3->{3}
r4				1->{2} 2->{1} 3->{3}	1->{1} 2->{2} 3->{3}
r5					1->{2} 2->{1} 3->{3}

Region Matrix



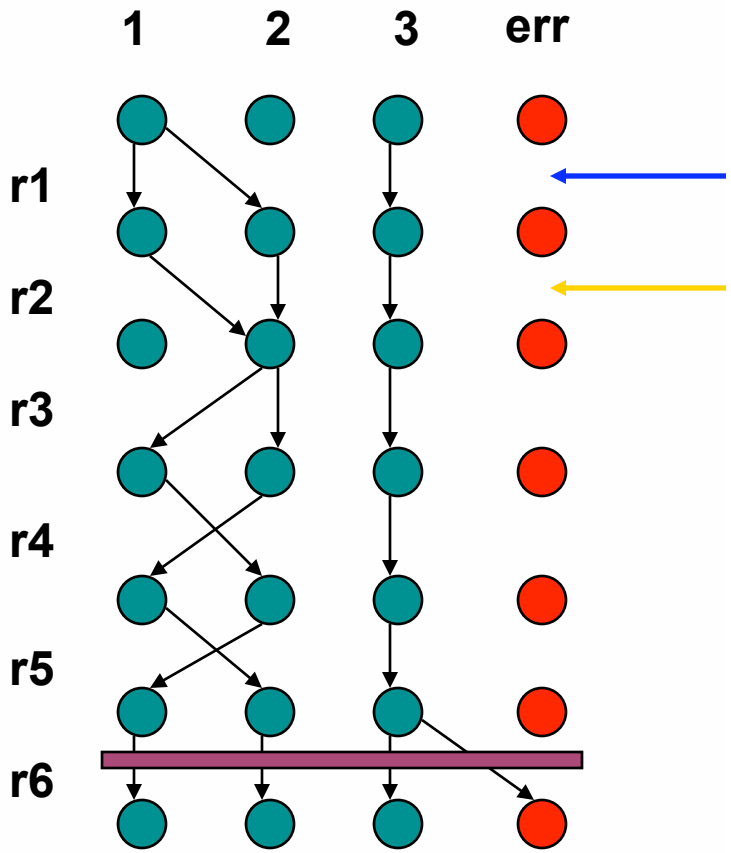
ESQuaReD

University of Nebraska Lincoln

Identify Safe Regions

	r1	r2	r3	r4	r5
r1	1->{1,2} 3->{3}	1->{2} 3->{3}	1->{1,2} 3->{3}	1->{1,2} 3->{3}	1->{1,2} 3->{3}
r2		1->{2} 2->{2} 3->{3}	1->{1,2} 2->{1,2} 3->{3}	1->{1,2} 2->{1,2} 3->{3}	1->{1,2} 2->{1,2} 3->{3}
r3			2->{1,2} 3->{3}	2->{1,2} 3->{3}	2->{1,2} 3->{3}
r4				1->{2} 2->{1} 3->{3}	1->{1} 2->{2} 3->{3}
r5					1->{2} 2->{1} 3->{3}

Region Matrix



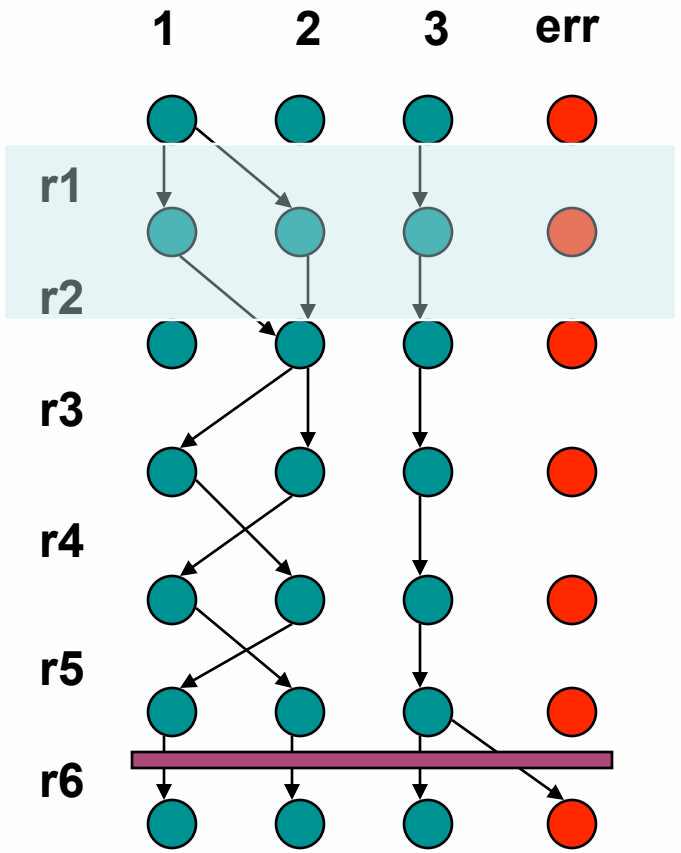
ESQuaReD

University of Nebraska Lincoln

Identify Safe Regions

	r1	r2	r3	r4	r5
r1	1->{1,2} 3->{3}	1->{2} 3->{3}	1->{1,2} 3->{3}	1->{1,2} 3->{3}	1->{1,2} 3->{3}
r2		1->{2} 2->{2} 3->{3}	1->{1,2} 2->{1,2} 3->{3}	1->{1,2} 2->{1,2} 3->{3}	1->{1,2} 2->{1,2} 3->{3}
r3			2->{1,2} 3->{3}	2->{1,2} 3->{3}	2->{1,2} 3->{3}
r4				1->{2} 2->{1} 3->{3}	1->{1} 2->{2} 3->{3}
r5					1->{2} 2->{1} 3->{3}

Region Matrix



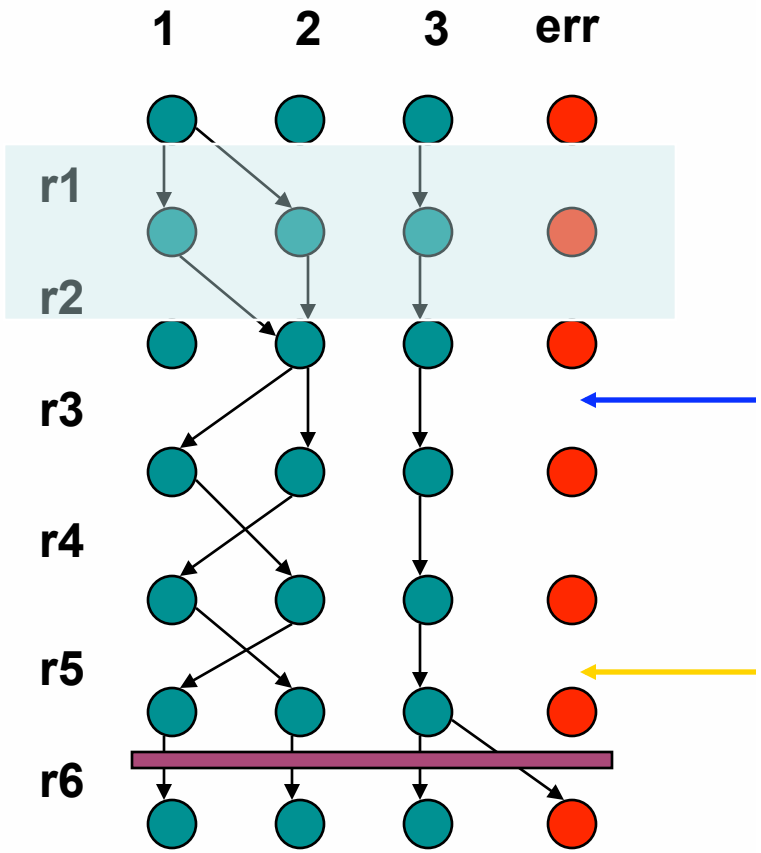
ESQuaReD

University of Nebraska Lincoln

Identify Safe Regions

	r1	r2	r3	r4	r5
r1	1->{1,2} 3->{3}	1->{2} 3->{3}	1->{1,2} 3->{3}	1->{1,2} 3->{3}	1->{1,2} 3->{3}
r2		1->{2} 2->{2} 3->{3}	1->{1,2} 2->{1,2} 3->{3}	1->{1,2} 2->{1,2} 3->{3}	1->{1,2} 2->{1,2} 3->{3}
r3			2->{1,2} 3->{3}	2->{1,2} 3->{3}	2->{1,2} 3->{3}
r4				1->{2} 2->{1} 3->{3}	1->{1} 2->{2} 3->{3}
r5					1->{2} 2->{1} 3->{3}

Region Matrix



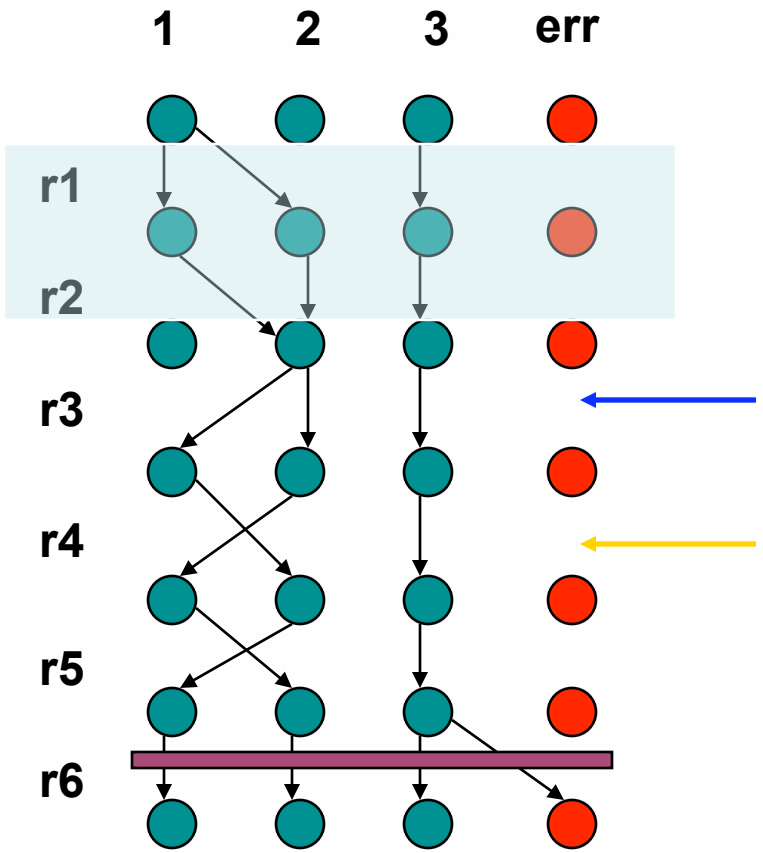
ESQuaReD

University of Nebraska Lincoln

Identify Safe Regions

	r1	r2	r3	r4	r5
r1	1->{1,2} 3->{3}	1->{2} 3->{3}	1->{1,2} 3->{3}	1->{1,2} 3->{3}	1->{1,2} 3->{3}
r2		1->{2} 2->{2} 3->{3}	1->{1,2} 2->{1,2} 3->{3}	1->{1,2} 2->{1,2} 3->{3}	1->{1,2} 2->{1,2} 3->{3}
r3			2->{1,2} 3->{3}	2->{1,2} 3->{3}	2->{1,2} 3->{3}
r4				1->{2} 2->{1} 3->{3}	1->{1} 2->{2} 3->{3}
r5					1->{2} 2->{1} 3->{3}

Region Matrix



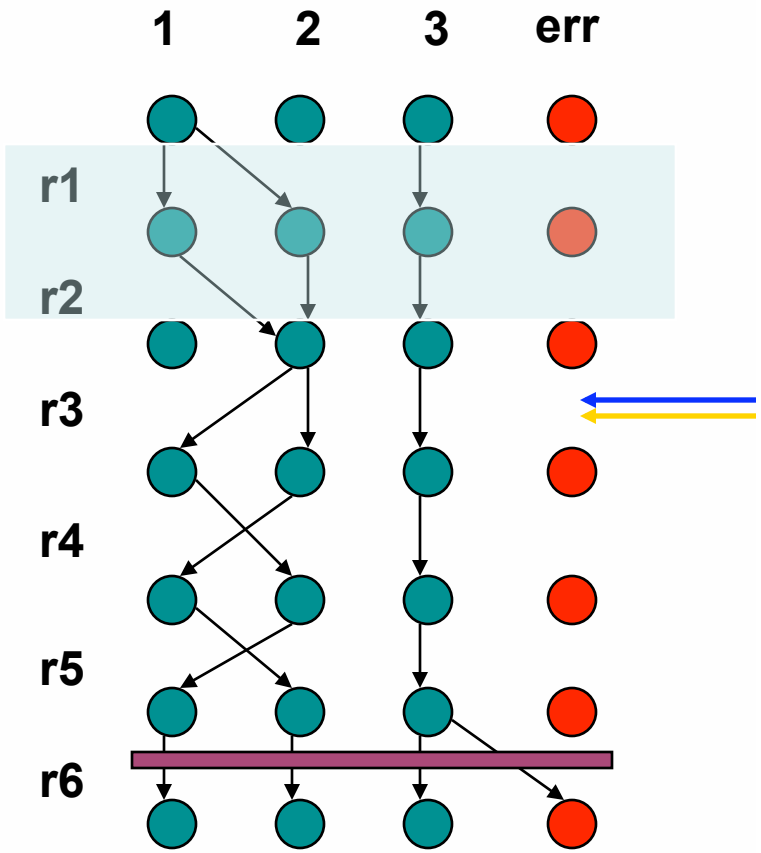
ESQuaReD

University of Nebraska Lincoln

Identify Safe Regions

	r1	r2	r3	r4	r5
r1	1->{1,2} 3->{3}	1->{2} 3->{3}	1->{1,2} 3->{3}	1->{1,2} 3->{3}	1->{1,2} 3->{3}
r2		1->{2} 2->{2} 3->{3}	1->{1,2} 2->{1,2} 3->{3}	1->{1,2} 2->{1,2} 3->{3}	1->{1,2} 2->{1,2} 3->{3}
r3			2->{1,2} 3->{3}	2->{1,2} 3->{3}	2->{1,2} 3->{3}
r4				1->{2} 2->{1} 3->{3}	1->{1} 2->{2} 3->{3}
r5					1->{2} 2->{1} 3->{3}

Region Matrix



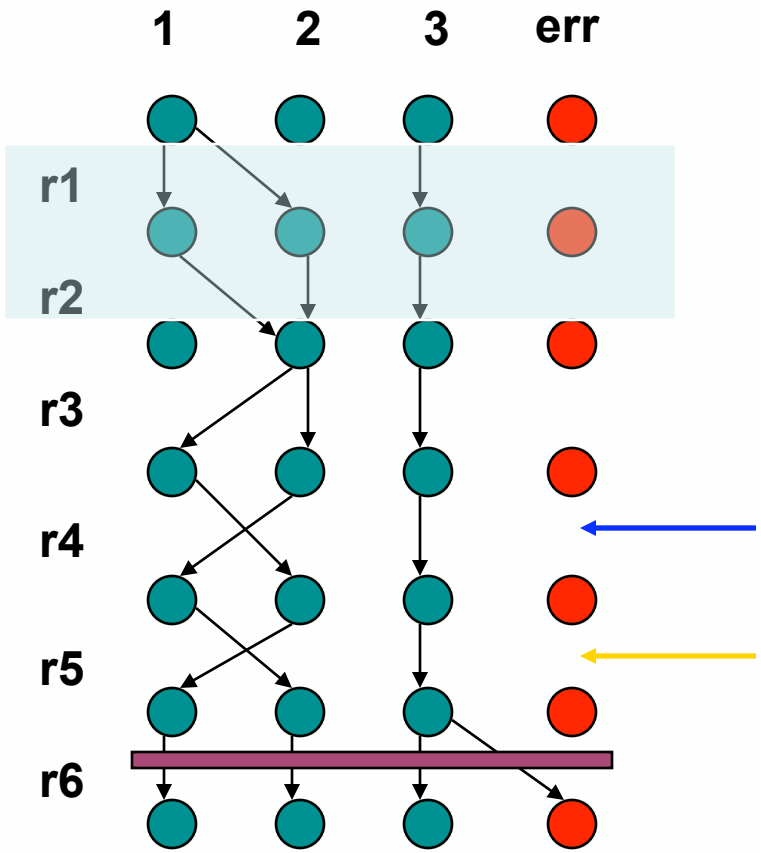
ESQuaReD

University of Nebraska Lincoln

Identify Safe Regions

	r1	r2	r3	r4	r5
r1	1->{1,2} 3->{3}	1->{2} 3->{3}	1->{1,2} 3->{3}	1->{1,2} 3->{3}	1->{1,2} 3->{3}
r2		1->{2} 2->{2} 3->{3}	1->{1,2} 2->{1,2} 3->{3}	1->{1,2} 2->{1,2} 3->{3}	1->{1,2} 2->{1,2} 3->{3}
r3			2->{1,2} 3->{3}	2->{1,2} 3->{3}	2->{1,2} 3->{3}
r4				1->{2} 2->{1} 3->{3}	1->{1} 2->{2} 3->{3}
r5					1->{2} 2->{1} 3->{3}

Region Matrix



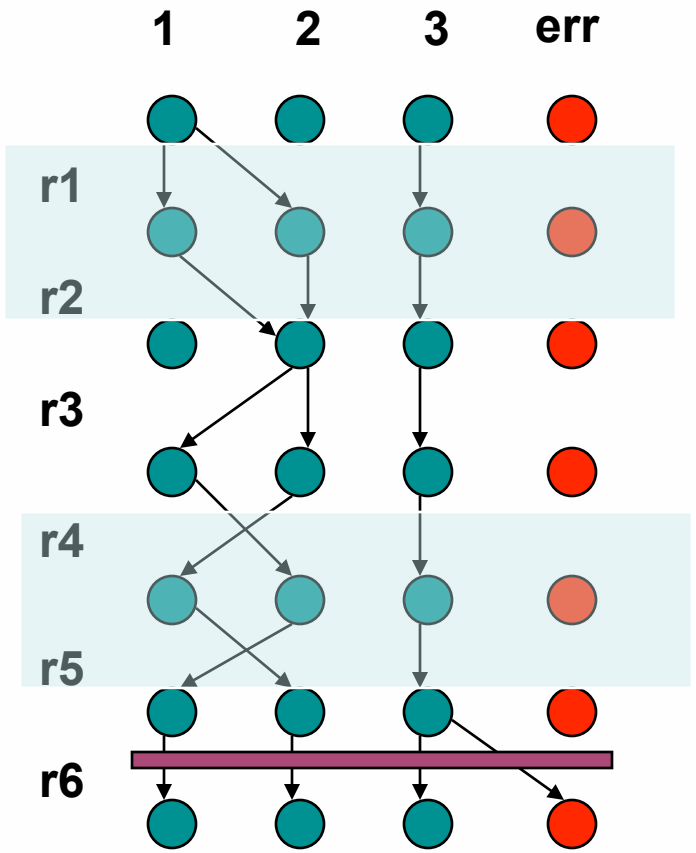
ESQuaReD

University of Nebraska Lincoln

Identify Safe Regions

	r1	r2	r3	r4	r5
r1	1->{1,2} 3->{3}	1->{2} 3->{3}	1->{1,2} 3->{3}	1->{1,2} 3->{3}	1->{1,2} 3->{3}
r2		1->{2} 2->{2} 3->{3}	1->{1,2} 2->{1,2} 3->{3}	1->{1,2} 2->{1,2} 3->{3}	1->{1,2} 2->{1,2} 3->{3}
r3			2->{1,2} 3->{3}	2->{1,2} 3->{3}	2->{1,2} 3->{3}
r4				1->{2} 2->{1} 3->{3}	1->{1} 2->{2} 3->{3}
r5					1->{2} 2->{1} 3->{3}

Region Matrix



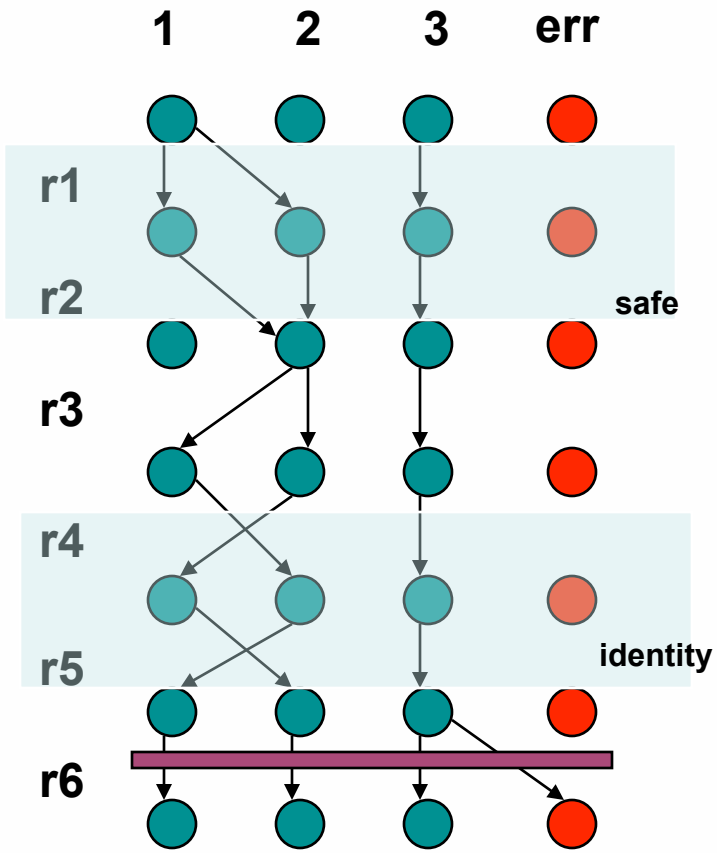
ESQuaReD

University of Nebraska Lincoln

Identify Safe Regions

	r1	r2	r3	r4	r5
r1	1->{1,2} 3->{3}	1->{2} 3->{3}	1->{1,2} 3->{3}	1->{1,2} 3->{3}	1->{1,2} 3->{3}
r2		1->{2} 2->{2} 3->{3}	1->{1,2} 2->{1,2} 3->{3}	1->{1,2} 2->{1,2} 3->{3}	1->{1,2} 2->{1,2} 3->{3}
r3			2->{1,2} 3->{3}	2->{1,2} 3->{3}	2->{1,2} 3->{3}
r4				1->{2} 2->{1} 3->{3}	1->{1} 2->{2} 3->{3}
r5					1->{2} 2->{1} 3->{3}

Region Matrix



ESQuaReD

University of Nebraska Lincoln

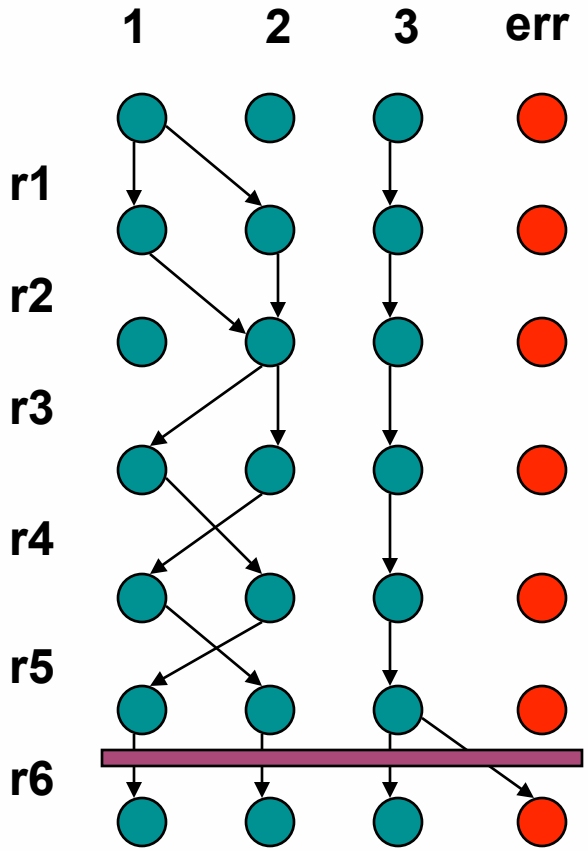
Algorithm

Basic Steps

1. Reduces a control flow graph region to a sequence of single entry regions
2. Calls static typestate analysis to calculate functional summaries of reachable program regions
3. Within a region, identify candidate safe regions by marking boundaries that cannot be crossed by any safe region
4. Identify safe regions inside candidate safe regions
5. Drop (and if required, add) FSA transitions for safe regions
6. Repeat the steps for all regions that lie outside safe regions

Adding and Dropping Transitions

```
if(!(t instanceof ClassType)){
  Expression e = ((ExpressionStatement)s).getExpression();
  if(e instanceof Assignment){
    Expression rhs = ((Assignment)e).getRightHandSide();
    rhs.accept(v);
  }
  else if(e instanceof MethodInvocation)
    e.accept(v);
}
while(sit.hasNext()){
  var = sit.next();
  Iterator <Pair<String, ASTNode>> sait = ssa.iterator();
  while(sait.hasNext()){
    p = sait.next();
    if(p.first.equals(var) && p.second instanceof VariableDeclarationStatement)
      sp.add(new Pair <ASTNode, String>(s, var));
  }
}
if(lhsType != tf.Int){
  Expression e = ((DoStatement)s).getExpression();
  e.accept(v);
}
if(!classMap.containsKey(className)){
  Expression e = ((ForStatement)s).getExpression();
  if(e != null)
    e.accept(v);
}
if(s instanceof WhileStatement){
  Expression e = ((WhileStatement)s).getExpression();
  e.accept(v);
}
```

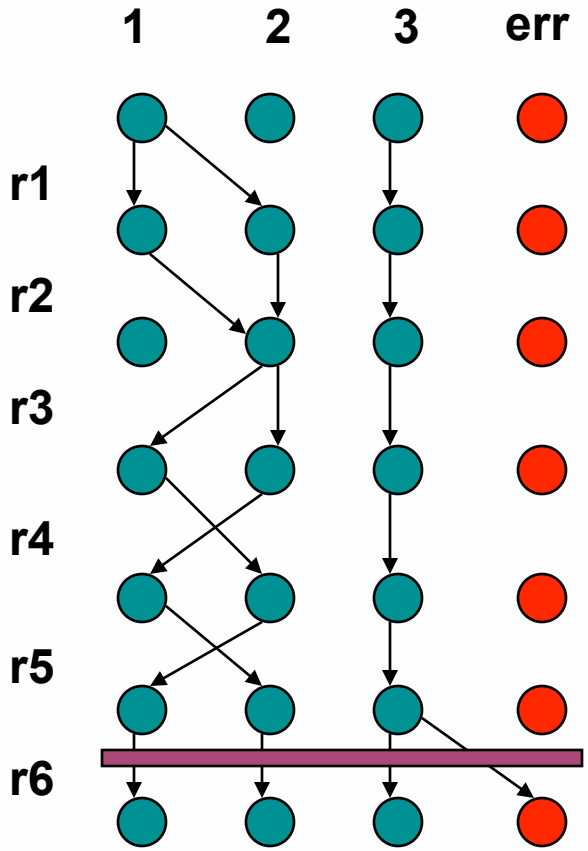


ESQuaReD

University of Nebraska Lincoln

Adding and Dropping Transitions

```
r1  if(!(t instanceof ClassType)){
      Expression e = ((ExpressionStatement)s).getExpression();
      if(e instanceof Assignment){
        Expression rhs = ((Assignment)e).getRightHandSide();
        rhs.accept(v);
      }
      else if(e instanceof MethodInvocation)
        e.accept(v);
    }
r2  while(sit.hasNext()){
      var = sit.next();
      Iterator <Pair<String, ASTNode>> sait = ssa.iterator();
      while(sait.hasNext()){
        p = sait.next();
        if(p.first.equals(var) && p.second instanceof VariableDeclarationStatement)
          sp.add(new Pair <ASTNode, String>(s, var));
      }
    }
r3  if(lhsType != tf.Int){
      Expression e = ((DoStatement)s).getExpression();
      e.accept(v);
    }
r4  if(!classMap.containsKey(className)){
      Expression e = ((ForStatement)s).getExpression();
      if(e != null)
        e.accept(v);
    }
r5  if(s instanceof WhileStatement){
      Expression e = ((WhileStatement)s).getExpression();
      e.accept(v);
    }
```

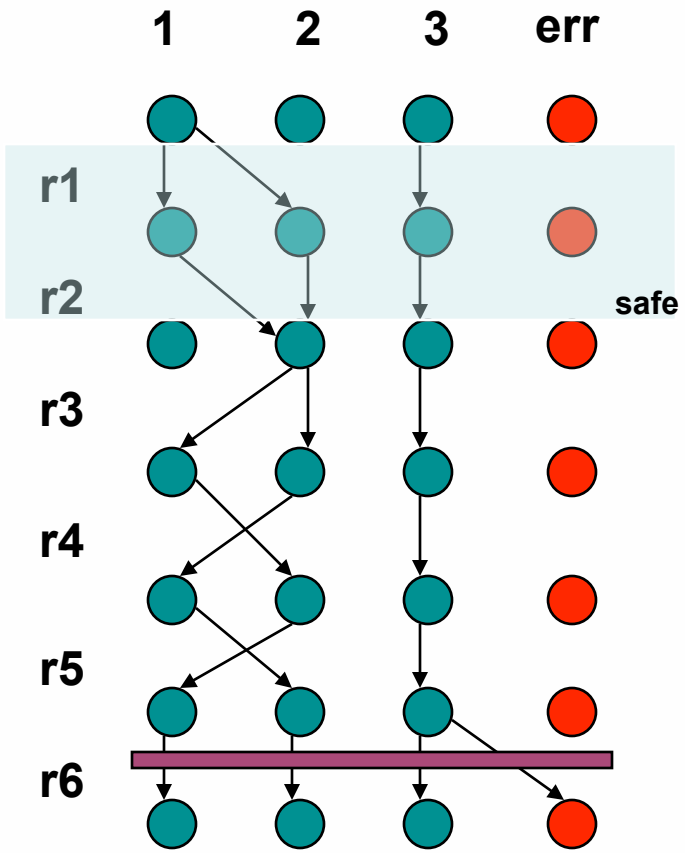


ESQuaReD

University of Nebraska Lincoln

Adding and Dropping Transitions

```
r1  if(!(t instanceof ClassType)){
      Expression e = ((ExpressionStatement)s).getExpression();
      if(e instanceof Assignment){
        Expression rhs = ((Assignment)e).getRightHandSide();
        rhs.accept(v);
      }
      else if(e instanceof MethodInvocation)
        e.accept(v);
    }
r2  while(sit.hasNext()){
      var = sit.next();
      Iterator <Pair<String, ASTNode>> sait = ssa.iterator();
      while(sait.hasNext()){
        p = sait.next();
        if(p.first.equals(var) && p.second instanceof VariableDeclarationStatement)
          sp.add(new Pair <ASTNode, String>(s, var));
      }
    }
r3  if(lhsType != tf.Int){
      Expression e = ((DoStatement)s).getExpression();
      e.accept(v);
    }
r4  if(!classMap.containsKey(className)){
      Expression e = ((ForStatement)s).getExpression();
      if(e != null)
        e.accept(v);
    }
r5  if(s instanceof WhileStatement){
      Expression e = ((WhileStatement)s).getExpression();
      e.accept(v);
    }
```

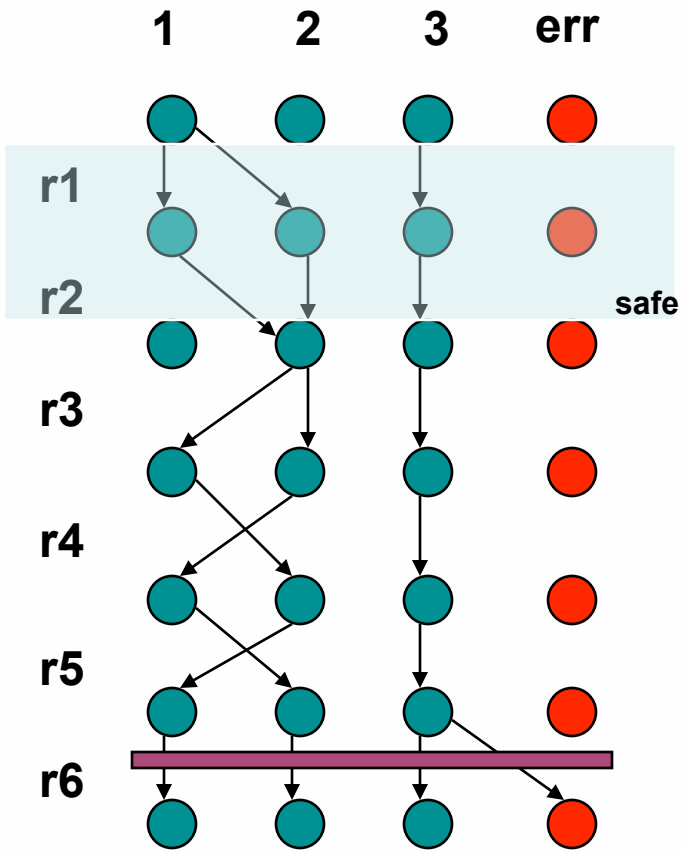


ESQuaReD

University of Nebraska Lincoln

Adding and Dropping Transitions

```
r1  if(!(t instanceof ClassType)){
      Expression e = ((ExpressionStatement)s).getExpression();
      if(e instanceof Assignment){
        Expression rhs = ((Assignment)e).getRightHandSide();
        rhs.accept(v);
      }
      else if(e instanceof MethodInvocation)
        e.accept(v);
    }
r2  while(sit.hasNext()){
      var = sit.next();
      Iterator <Pair<String, ASTNode>> sait = ssa.iterator();
      while(sait.hasNext()){
        p = sait.next();
        if(p.first.equals(var) && p.second instanceof VariableDeclarationStatement)
          sp.add(new Pair <ASTNode, String>(s, var));
      }
    }
r3  if(lhsType != tf.Int){
      Expression e = ((DoStatement)s).getExpression();
      e.accept(v);
    }
r4  if(!classMap.containsKey(className)){
      Expression e = ((ForStatement)s).getExpression();
      if(e != null)
        e.accept(v);
    }
r5  if(s instanceof WhileStatement){
      Expression e = ((WhileStatement)s).getExpression();
      e.accept(v);
    }
```



ESQuaReD

University of Nebraska Lincoln

Adding and Dropping Transitions

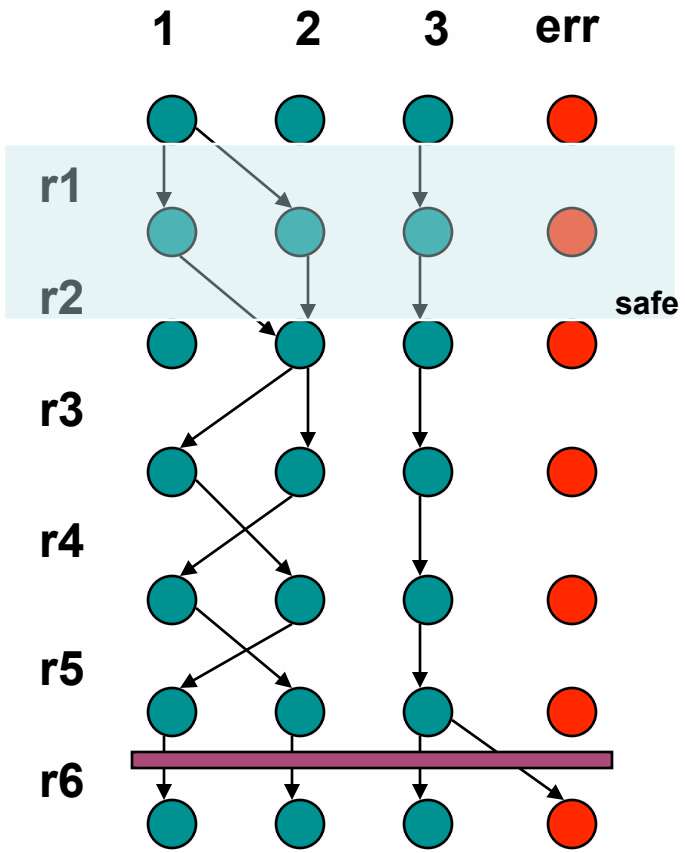
r1
```
if(!(t instanceof ClassType)){
  Expression e = ((ExpressionStatement)s).getExpression();
  if(e instanceof Assignment){
    Expression rhs = ((Assignment)e).getRightHandSide();
    rhs.accept(v);
  }
  else if(e instanceof MethodInvocation)
    e.accept(v);
}
```

r2
```
while(sit.hasNext()){
  var = sit.next();
  Iterator <Pair<String, ASTNode>> sait = ssa.iterator();
  while(sait.hasNext()){
    p = sait.next();
    if(p.first.equals(var) && p.second instanceof VariableDeclarationStatement)
      sp.add(new Pair <ASTNode, String>(s, var));
  }
}
```

r3
```
if(lhsType != tf.Int){
  Expression e = ((DoStatement)s).getExpression();
  e.accept(v);
}
```

r4
```
if(!classMap.containsKey(className)){
  Expression e = ((ForStatement)s).getExpression();
  if(e != null)
    e.accept(v);
}
```

r5
```
if(s instanceof WhileStatement){
  Expression e = ((WhileStatement)s).getExpression();
  e.accept(v);
}
```



ESQuaReD

University of Nebraska Lincoln

Adding and Dropping Transitions

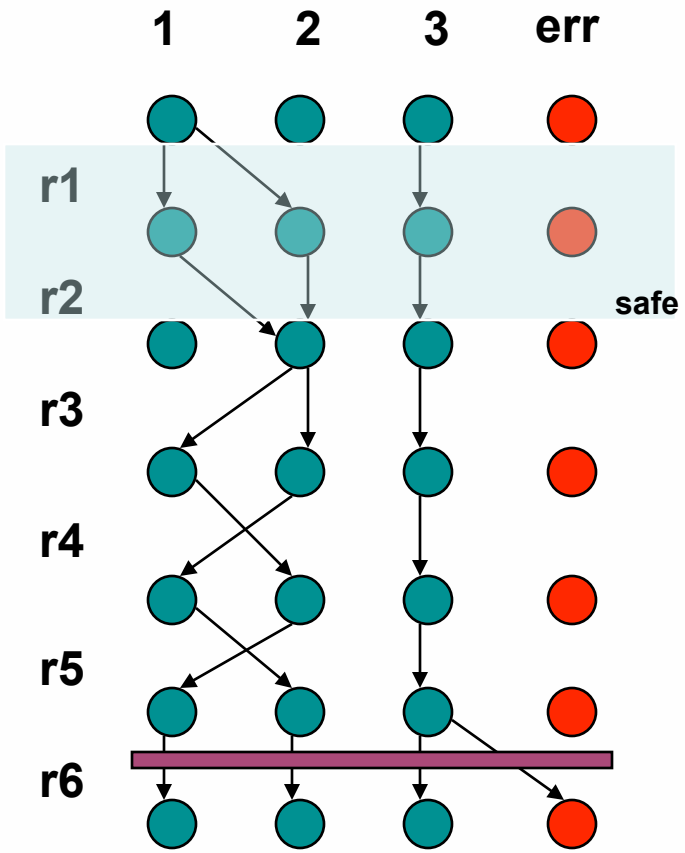
r1
```
if(!(t instanceof ClassType)){
  Expression e = ((ExpressionStatement)s).getExpression();
  if(e instanceof Assignment){
    Expression rhs = ((Assignment)e).getRightHandSide();
    rhs.accept(v);
  }
  else if(e instanceof MethodInvocation)
    e.accept(v);
}
```

r2
```
while(sit.hasNext()){
  var = sit.next();
  Iterator <Pair<String, ASTNode>> sait = ssa.iterator();
  while(sait.hasNext()){
    p = sait.next();
    if(p.first.equals(var) && p.second instanceof VariableDeclarationStatement)
      sp.add(new Pair <ASTNode, String>(s, var));
  }
}
```

r3
```
if(lhsType != tf.Int){
  Expression e = ((DoStatement)s).getExpression();
  e.accept(v);
}
```

r4
```
if(!classMap.containsKey(className)){
  Expression e = ((ForStatement)s).getExpression();
  if(e != null)
    e.accept(v);
}
```

r5
```
if(s instanceof WhileStatement){
  Expression e = ((WhileStatement)s).getExpression();
  e.accept(v);
}
```



ESQuaReD

University of Nebraska Lincoln

Adding and Dropping Transitions

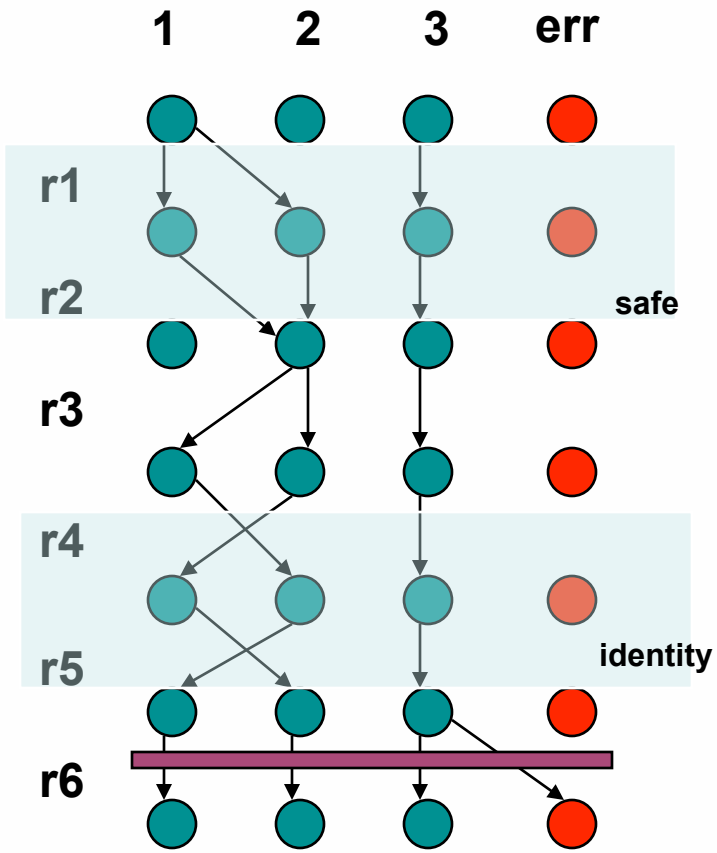
r1
```
if(!(t instanceof ClassType)){
  Expression e = ((ExpressionStatement)s).getExpression();
  if(e instanceof Assignment){
    Expression rhs = ((Assignment)e).getRightHandSide();
    rhs.accept(v);
  }
  else if(e instanceof MethodInvocation)
    e.accept(v);
}
```

r2
```
while(sit.hasNext()){
  var = sit.next();
  Iterator <Pair<String, ASTNode>> sait = ssa.iterator();
  while(sait.hasNext()){
    p = sait.next();
    if(p.first.equals(var) && p.second instanceof VariableDeclarationStatement)
      sp.add(new Pair <ASTNode, String>(s, var));
  }
}
```

r3
```
if(lhsType != tf.Int){
  Expression e = ((DoStatement)s).getExpression();
  e.accept(v);
}
```

r4
```
if(!classMap.containsKey(className)){
  Expression e = ((ForStatement)s).getExpression();
  if(e != null)
    e.accept(v);
}
```

r5
```
if(s instanceof WhileStatement){
  Expression e = ((WhileStatement)s).getExpression();
  e.accept(v);
}
```



ESQuaReD

University of Nebraska Lincoln

Adding and Dropping Transitions

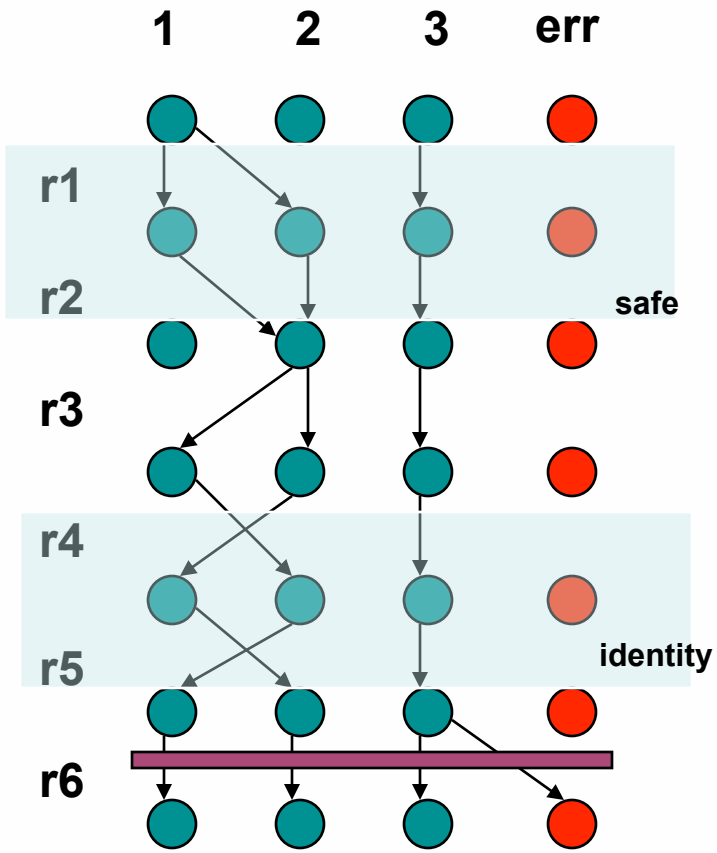
r1
```
if(!(t instanceof ClassType)){
  Expression e = ((ExpressionStatement)s).getExpression();
  if(e instanceof Assignment){
    Expression rhs = ((Assignment)e).getRightHandSide();
    rhs.accept(v);
  }
  else if(e instanceof MethodInvocation)
    e.accept(v);
}
```

r2
```
while(sit.hasNext()){
  var = sit.next();
  Iterator <Pair<String, ASTNode>> sait = ssa.iterator();
  while(sait.hasNext()){
    p = sait.next();
    if(p.first.equals(var) && p.second instanceof VariableDeclarationStatement)
      sp.add(new Pair <ASTNode, String>(s, var));
  }
}
```

r3
```
if(lhsType != tf.Int){
  Expression e = ((DoStatement)s).getExpression();
  e.accept(v);
}
```

r4
```
if(!classMap.containsKey(className)){
  Expression e = ((ForStatement)s).getExpression();
  if(e != null)
    e.accept(v);
}
```

r5
```
if(s instanceof WhileStatement){
  Expression e = ((WhileStatement)s).getExpression();
  e.accept(v);
}
```



ESQuaReD

University of Nebraska Lincoln

Adding and Dropping Transitions

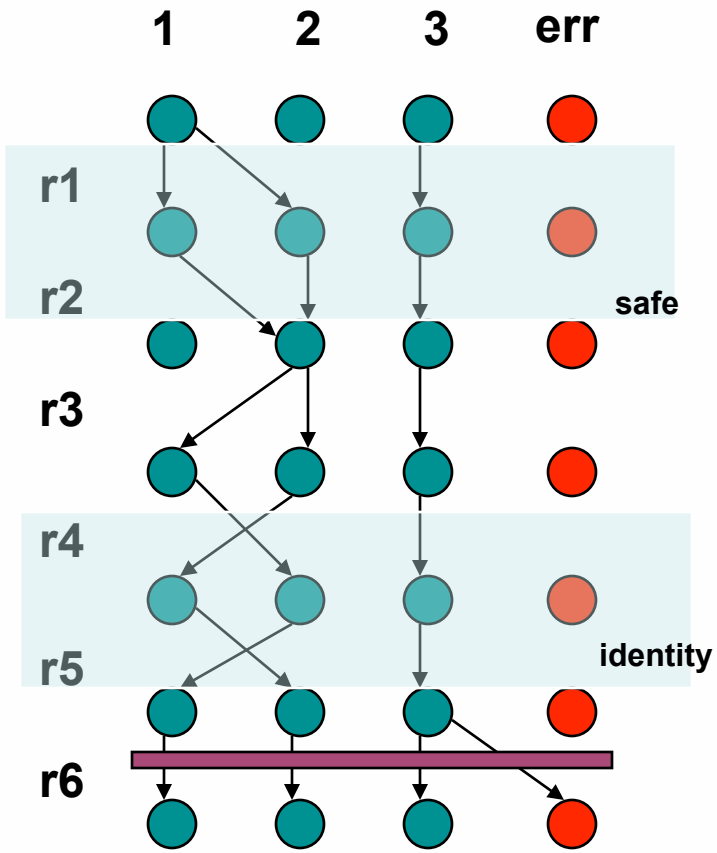
r1
```
if(!(t instanceof ClassType)){
  Expression e = ((ExpressionStatement)s).getExpression();
  if(e instanceof Assignment){
    Expression rhs = ((Assignment)e).getRightHandSide();
    rhs.accept(v);
  }
  else if(e instanceof MethodInvocation)
    e.accept(v);
}
```

r2
```
while(sit.hasNext()){
  var = sit.next();
  Iterator <Pair<String, ASTNode>> sait = ssa.iterator();
  while(sait.hasNext()){
    p = sait.next();
    if(p.first.equals(var) && p.second instanceof VariableDeclarationStatement)
      sp.add(new Pair <ASTNode, String>(s, var));
  }
}
```

r3
```
if(lhsType != tf.Int){
  Expression e = ((DoStatement)s).getExpression();
  e.accept(v);
}
```

r4
```
if(!classMap.containsKey(className)){
  Expression e = ((ForStatement)s).getExpression();
  if(e != null)
    e.accept(v);
}
```

r5
```
if(s instanceof WhileStatement){
  Expression e = ((WhileStatement)s).getExpression();
  e.accept(v);
}
```

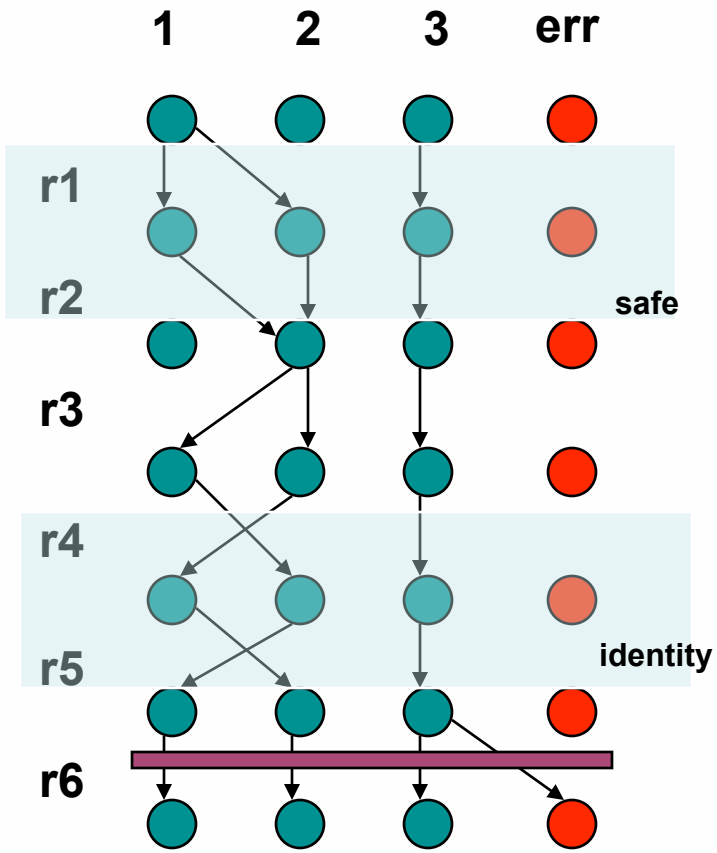


ESQuaReD

University of Nebraska Lincoln

Adding and Dropping Transitions

```
r1  if(!(t instanceof ClassType)){
      Expression e = ((ExpressionStatement)s).getExpression();
      if(e instanceof Assignment){
        Expression rhs = ((Assignment)e).getRightHandSide();
        rhs.accept(v);
      }
      else if(e instanceof MethodInvocation)
        e.accept(v);
    }
r2  while(sit.hasNext()){
      var = sit.next();
      Iterator <Pair<String, ASTNode>> sait = ssa.iterator();
      while(sait.hasNext()){
        p = sait.next();
        if(p.first.equals(var) && p.second instanceof VariableDeclarationStatement)
          sp.add(new Pair <ASTNode, String>(s, var));
      }
    }
r3  if(lhsType != tf.Int){
      Expression e = ((DoStatement)s).getExpression();
      e.accept(v);
    }
r4  if(!classMap.containsKey(className)){
      Expression e = ((ForStatement)s).getExpression();
      if(e != null)
        e.accept(v);
    }
r5  if(s instanceof WhileStatement){
      Expression e = ((WhileStatement)s).getExpression();
      e.accept(v);
    }
```



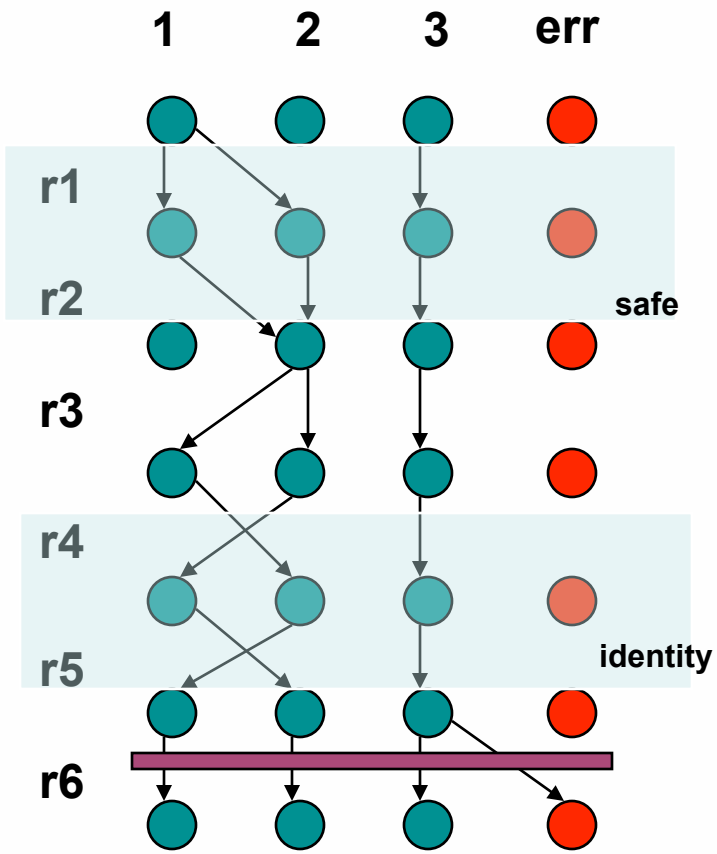
ESQuaReD

University of Nebraska Lincoln

Adding and Dropping Transitions

	r1	r2	r3	r4	r5
r1		1->{2} 3->{3}			
r2					
r3					
r4					
r5					

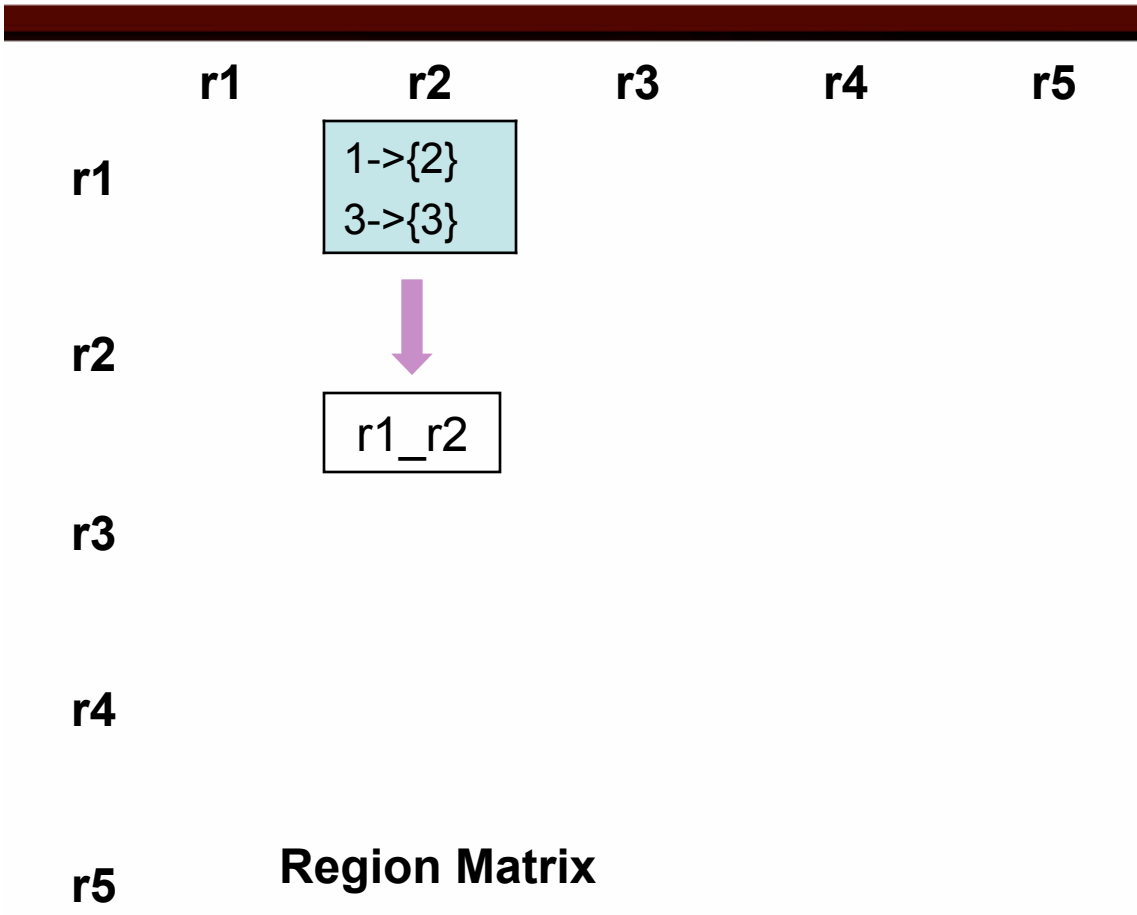
Region Matrix

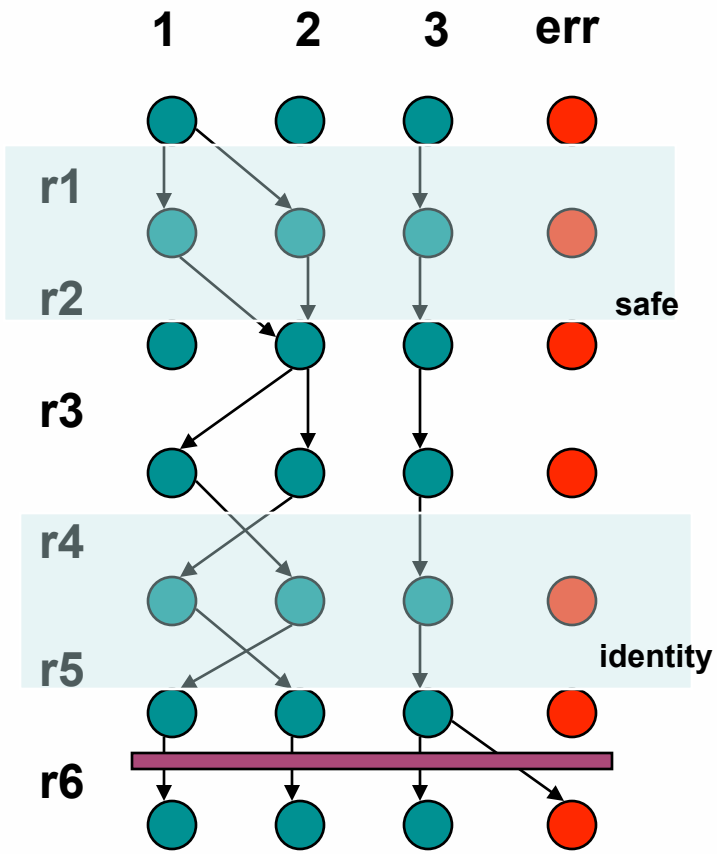


ESQuaReD

University of Nebraska Lincoln

Adding and Dropping Transitions

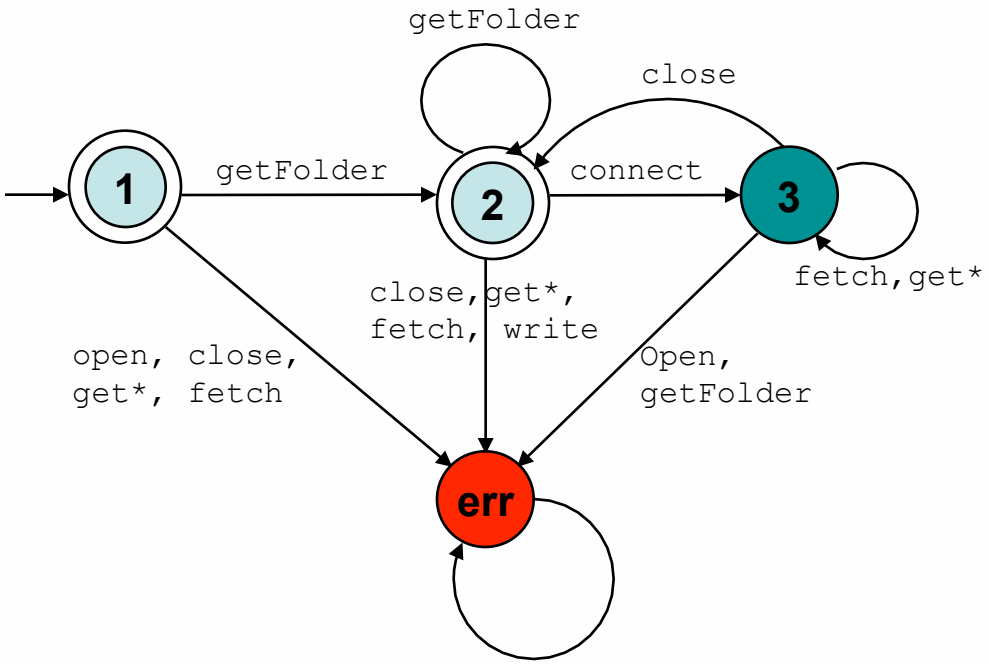


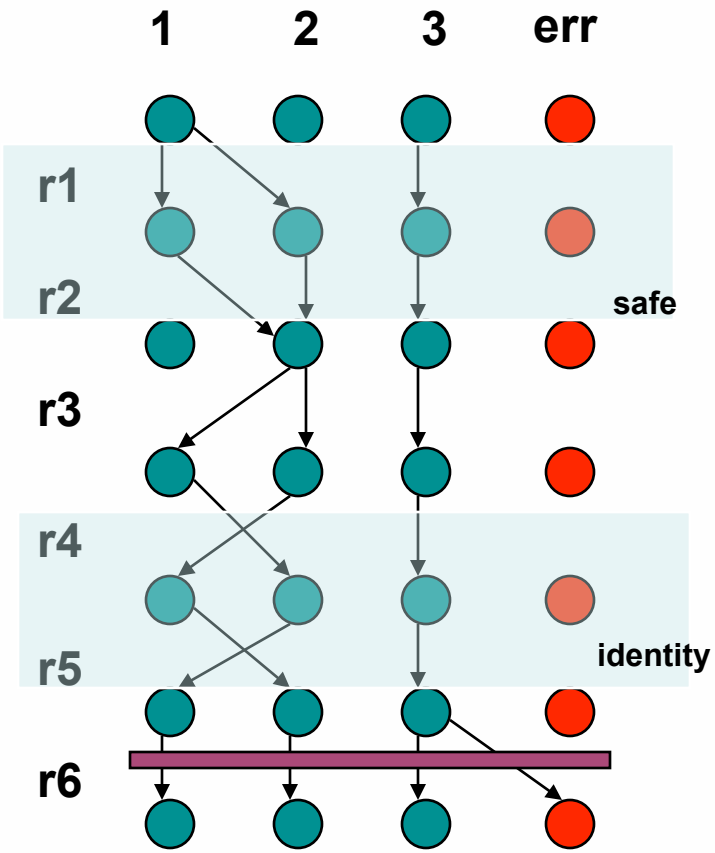


ESQuaReD

University of Nebraska Lincoln

Adding and Dropping Transitions

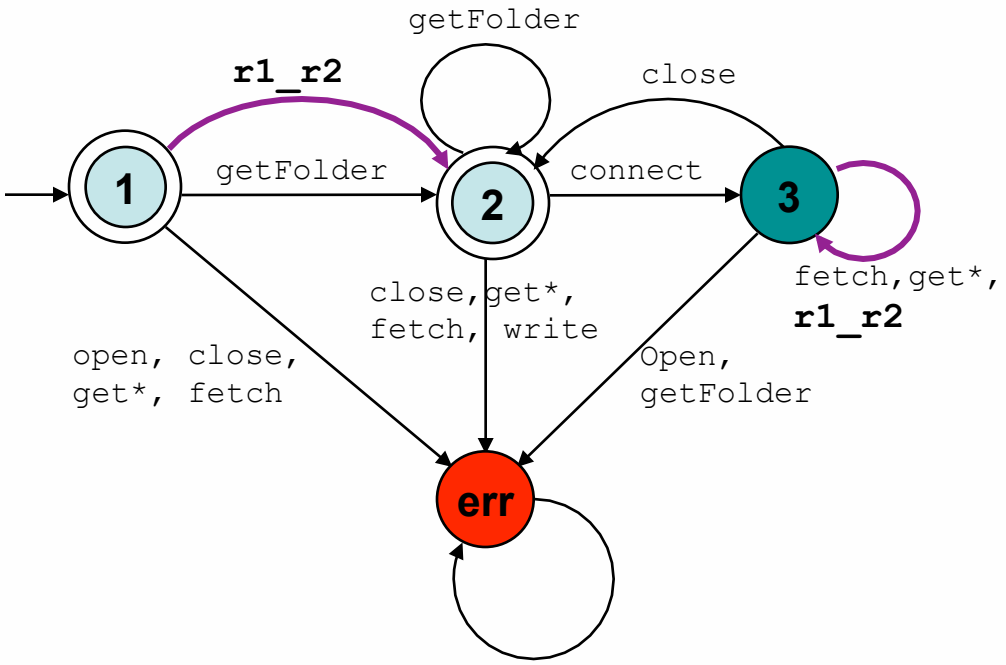


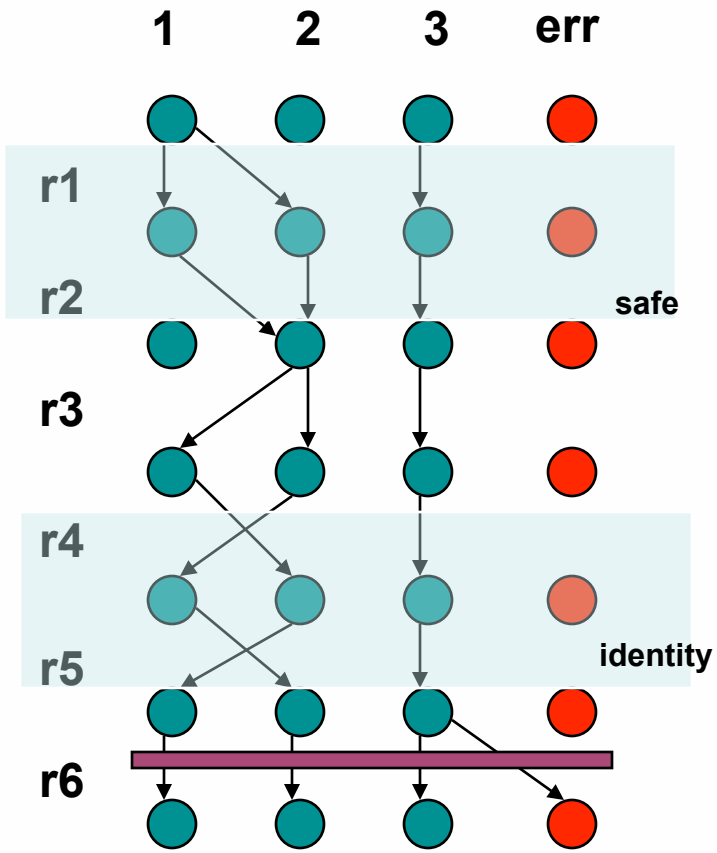


ESQuaReD

University of Nebraska Lincoln

Adding and Dropping Transitions





ESQuaReD

University of Nebraska Lincoln

Algorithm

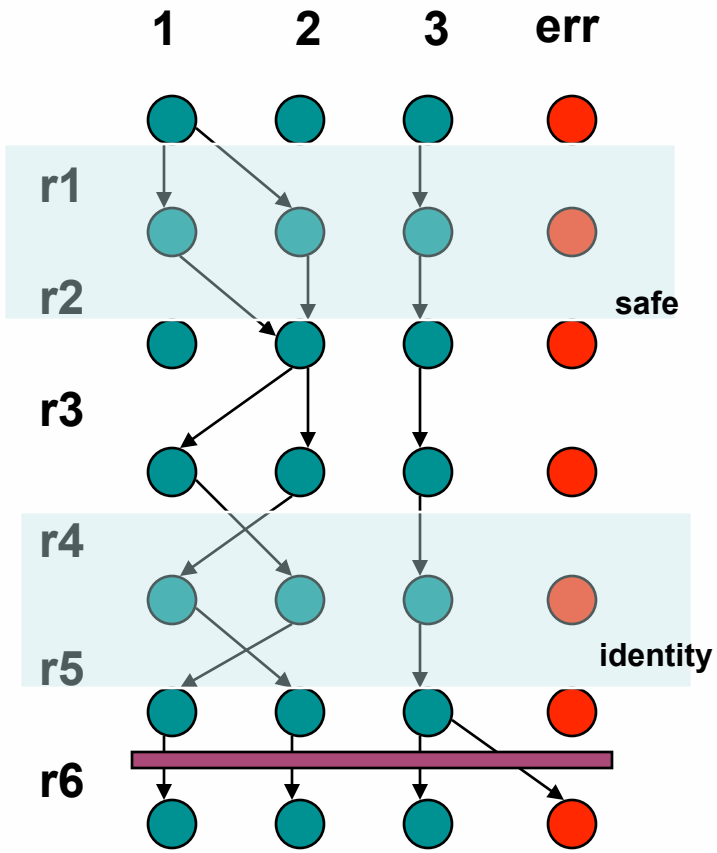
Basic Steps

1. Reduces a control flow graph region to a sequence of single entry regions
2. Calls static typestate analysis to calculate functional summaries of reachable program regions
3. Within a region, identify candidate safe regions by marking boundaries that cannot be crossed by any safe region
4. Identify safe regions inside candidate safe regions
5. Calculate (or remove) FSA transitions for safe regions
6. Repeat the steps for all regions that lie outside safe regions

Expand “Unsafe” Regions

```
if(!(t instanceof ClassType)){
  Expression e = ((ExpressionStatement)s).getExpression();
  if(e instanceof Assignment){
    Expression rhs = ((Assignment)e).getRightHandSide();
    rhs.accept(v);
  }
  else if(e instanceof MethodInvocation)
    e.accept(v);
}
while(sit.hasNext()){
  var = sit.next();
  Iterator <Pair<String, ASTNode>> sait = ssa.iterator();
  while(sait.hasNext()){
    p = sait.next();
    if(p.first.equals(var) && p.second instanceof VariableDeclarationStatement)
      sp.add(new Pair <ASTNode, String>(s, var));
  }
}

if(lhsType != tf.Int){
  Expression e = ((DoStatement)s).getExpression();
  e.accept(v);
}
if(!classMap.containsKey(className)){
  Expression e = ((ForStatement)s).getExpression();
  if(e != null)
    e.accept(v);
}
if(s instanceof WhileStatement){
  Expression e = ((WhileStatement)s).getExpression();
  e.accept(v);
}
```



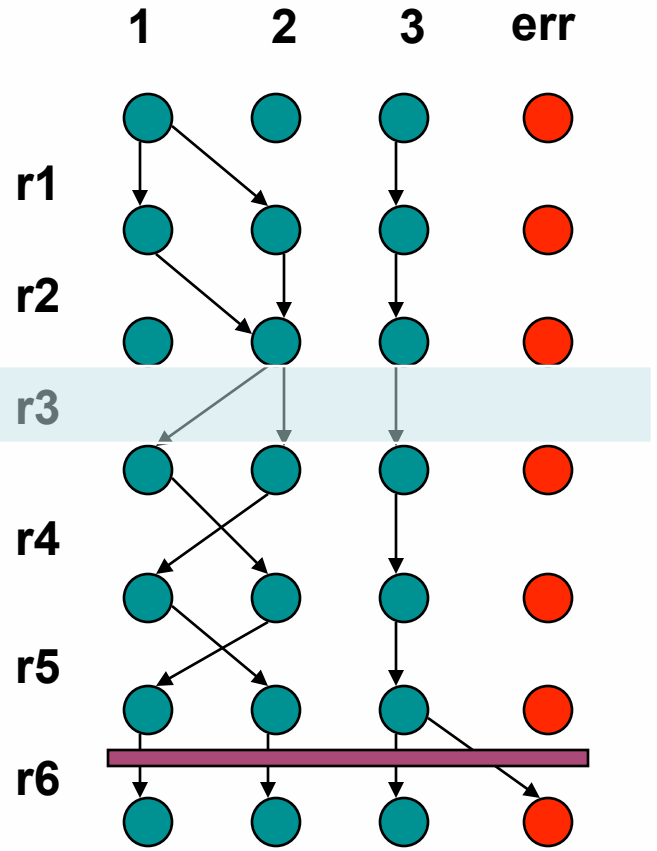
ESQuaReD

University of Nebraska Lincoln

Expand “Unsafe” Regions

```
if(!(t instanceof ClassType)){
  Expression e = ((ExpressionStatement)s).getExpression();
  if(e instanceof Assignment){
    Expression rhs = ((Assignment)e).getRightHandSide();
    rhs.accept(v);
  }
  else if(e instanceof MethodInvocation)
    e.accept(v);
}
while(sit.hasNext()){
  var = sit.next();
  Iterator <Pair<String, ASTNode>> sait = ssa.iterator();
  while(sait.hasNext()){
    p = sait.next();
    if(p.first.equals(var) && p.second instanceof VariableDeclarationStatement)
      sp.add(new Pair <ASTNode, String>(s, var));
  }
}

if(lhsType != tf.Int){
  Expression e = ((DoStatement)s).getExpression();
  e.accept(v);
}
if(!classMap.containsKey(className)){
  Expression e = ((ForStatement)s).getExpression();
  if(e != null)
    e.accept(v);
}
if(s instanceof WhileStatement){
  Expression e = ((WhileStatement)s).getExpression();
  e.accept(v);
}
```



ESQuaReD

University of Nebraska Lincoln

Experience

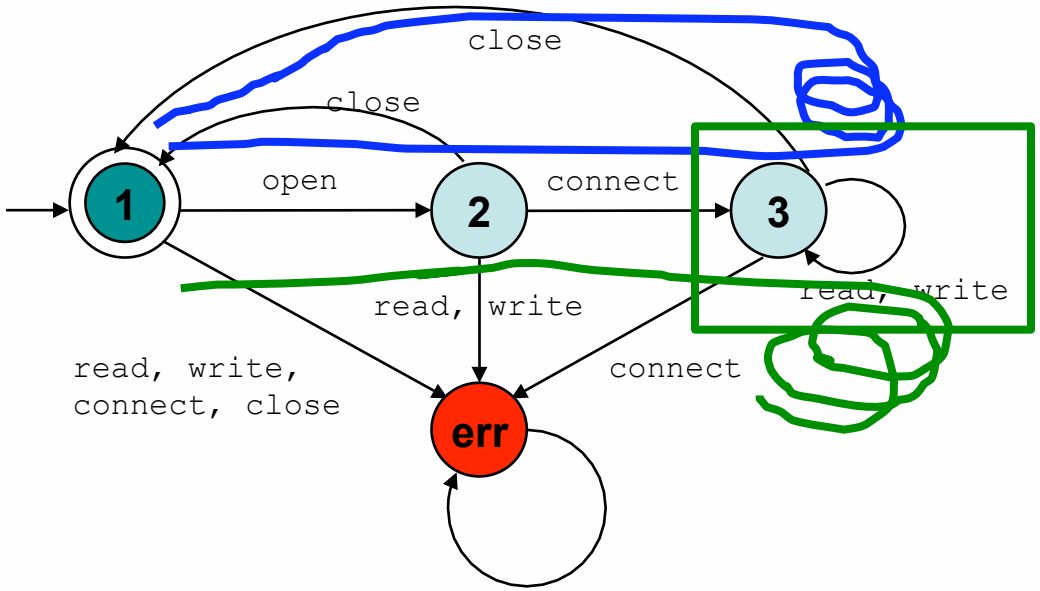
- Prototype Implementation
  - Inter-procedural static typestate analysis based on Soot (McGill)
  - Configurable dynamic typestate analysis using Sofya (UNL)
  - Residual typestate analysis algorithm connecting static and dynamic typestate analyses
- Sample Test Programs
  - TimeQuery
    - Application that uses SocketChannels to connect to NTP time servers
    - Input varies in the number of servers
    - About 100 lines of Java code
  - Gmail POP3
    - Application that implements a command-line interface to access Gmail
    - Input varies in number of commands
    - About 500 lines of Java code

Experience

Sample Programs and Residual Analysis Results based on Prototype Implementation

Program	OSFSA		Residual OSFSA	
	# Runtime Observations	% Runtime Overhead	# Runtime Observations	% Runtime Overhead
TimeQuery v1	400	69	200	37
TimeQuery v2	420	73	100	21
TimeQuery v2	800	33	0	1
Gmail POP3	43	46	3	9
Gmail POP3	203	51	3	6

Another Optimization



Consider a control flow region with two paths

this region is unsafe

`open`	`open`
`connect`	`connect`
`read*`	`write*`
`close`	

`close`

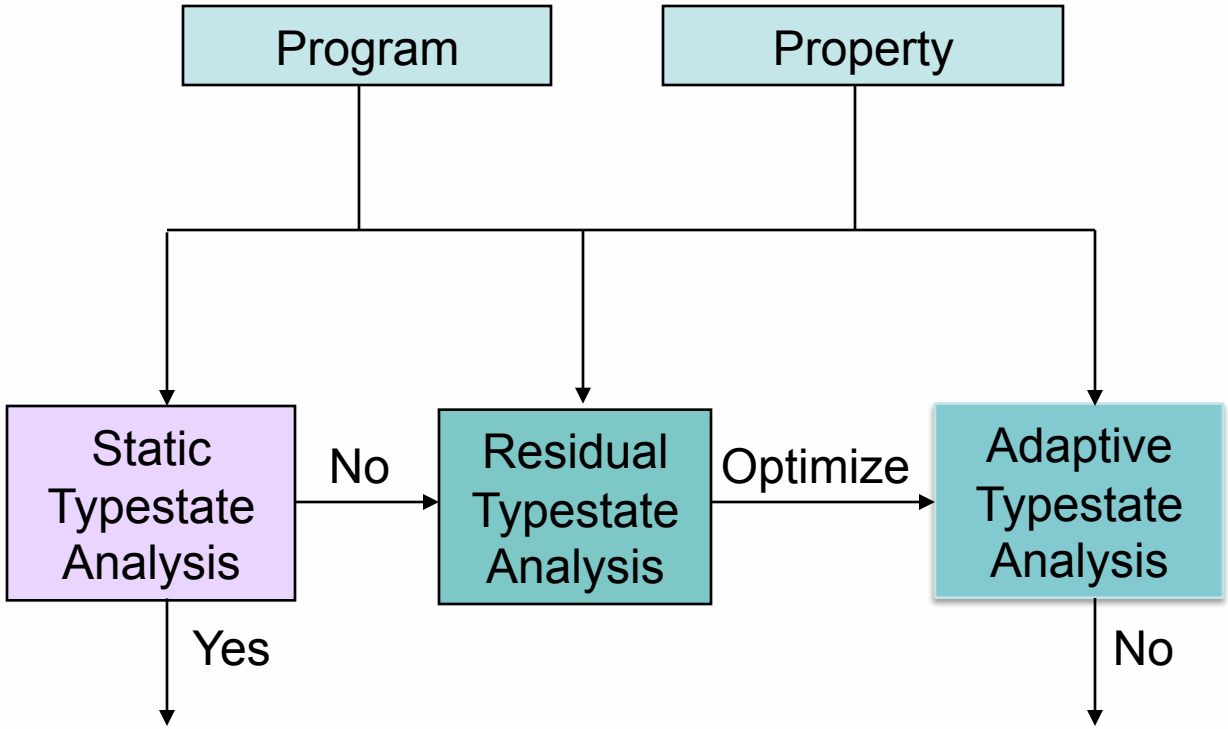
Neither path is a violation

Neither `read` nor `write` help detect violations in state 3

ESQuaReD

University of Nebraska Lincoln

Adaptive Typestate Analysis [ICSE'07]



Per-state Progress Symbols

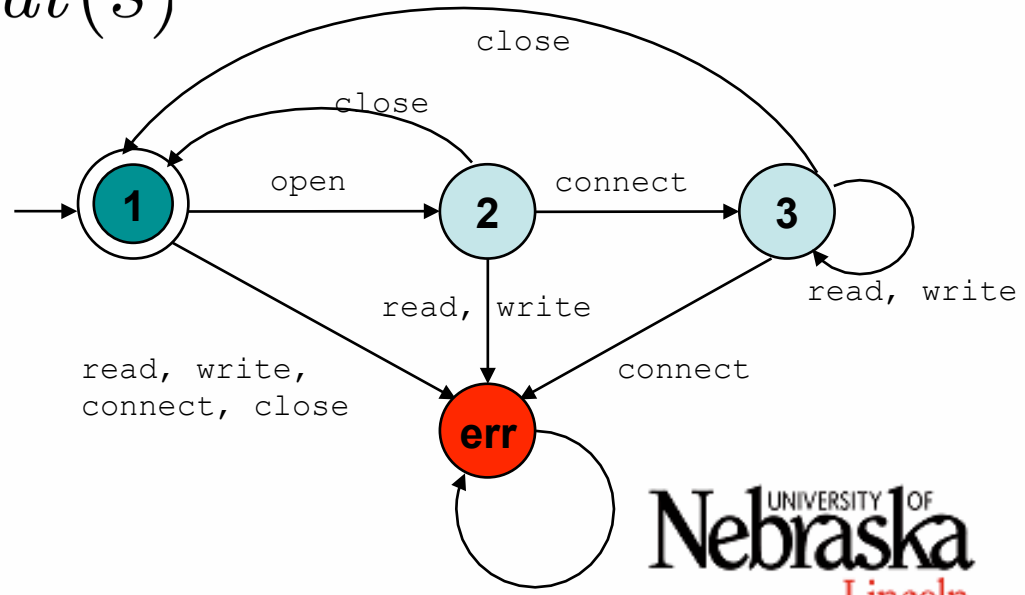
Set of symbols that transition between states

$$out(s) = \{a | a \in \Sigma \wedge \delta(s, a) \neq s\}$$

for convenience

$$self(s) = \Sigma - out(s)$$

State	Self Symbols	Outgoing Symbols
1	{}	Σ
2	{}	Σ
3	{read, write}	{open, connect, close}
err	Σ	{}



ESQuaReD

University of Nebraska Lincoln

Ignoring Self-symbols

Basic Monitor

`Init:`

$s_{cur} = s_0$

`TransitionOn(symbol a):`

$s_{cur} = \delta(s_{cur}, a)$

Adaptive Monitor

`Init:`

$s_{cur} = s_0$

$enable(out(s_{cur}))$

`TransitionOn(symbol a):`

$s_{next} = \delta(s_{cur}, a)$

$enable(out(s_{next}) - out(s_{cur}))$

$disable(self(s_{next}) - self(s_{cur}))$

$s_{cur} = s_{next}$

preserves
fault detection

ESQuaReD

University of Nebraska Lincoln

Checking Multiple Properties

Consider $(S, \Sigma, \delta, s_0, A), (S', \Sigma', \delta', {s_0}', A')$

where $\Sigma \cap \Sigma' \neq \emptyset$

Problem: monitor interference

May reach a program point, $a \in \Sigma \cap \Sigma'$, where

$$a \in self(s_{cur}) \wedge a \notin self(s'_{cur})$$

Solution: reference counting

- count # of FSAs requiring each observable
- enable observable as long as count > 0

ESQuaReD

University of Nebraska Lincoln

Experience

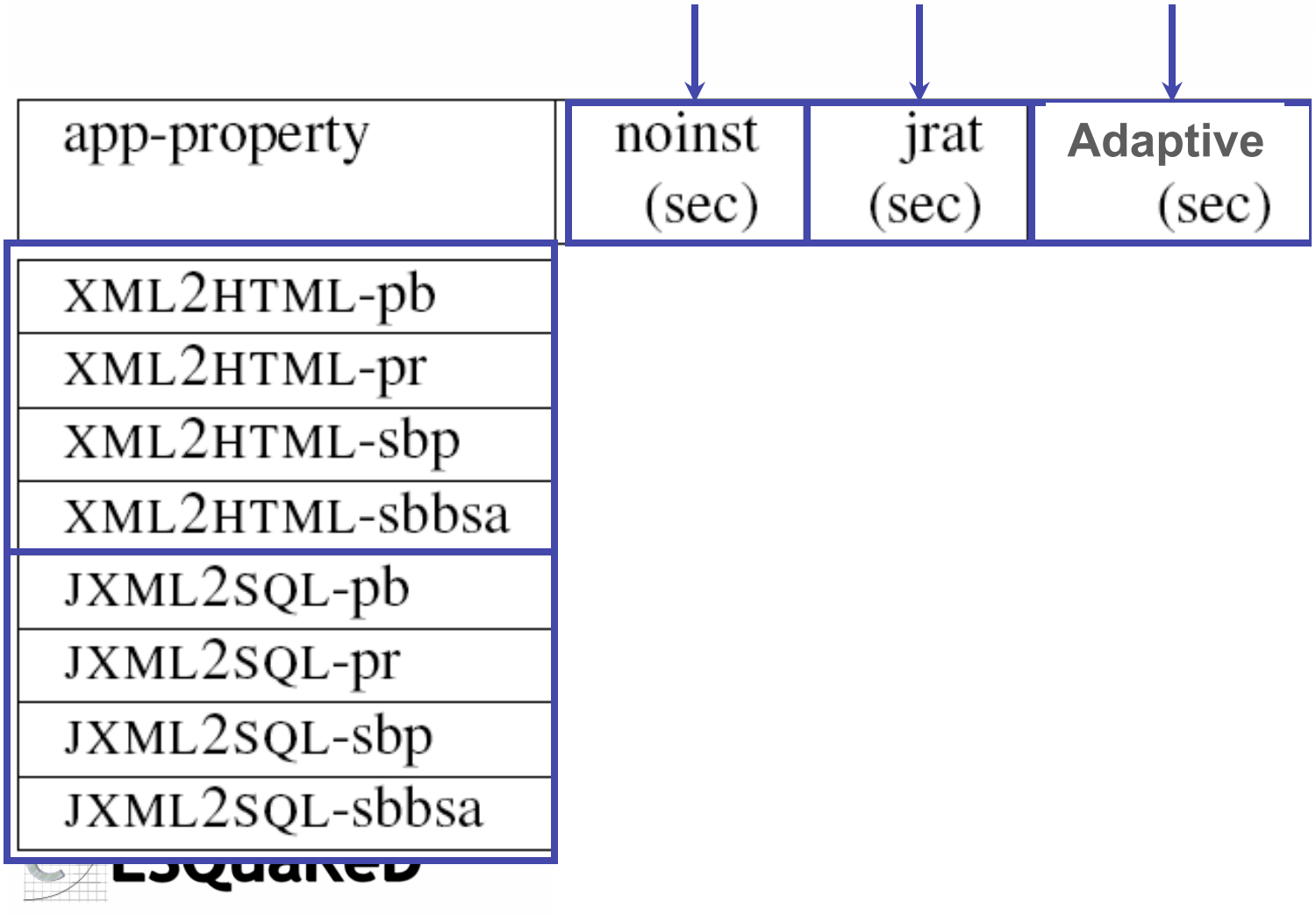
- NanoXML, xml processing library (SIR)
  - XML2HTML and JXML2SQL
- Multiple properties
  - Set Reader Before Parse (sbp)

```
for events {"IXMLParser:parse", "IXMLParser:setReader" }
            ~["parse"]* |("setReader";.*)
```

ESQuaReD

University of Nebraska Lincoln

Experience

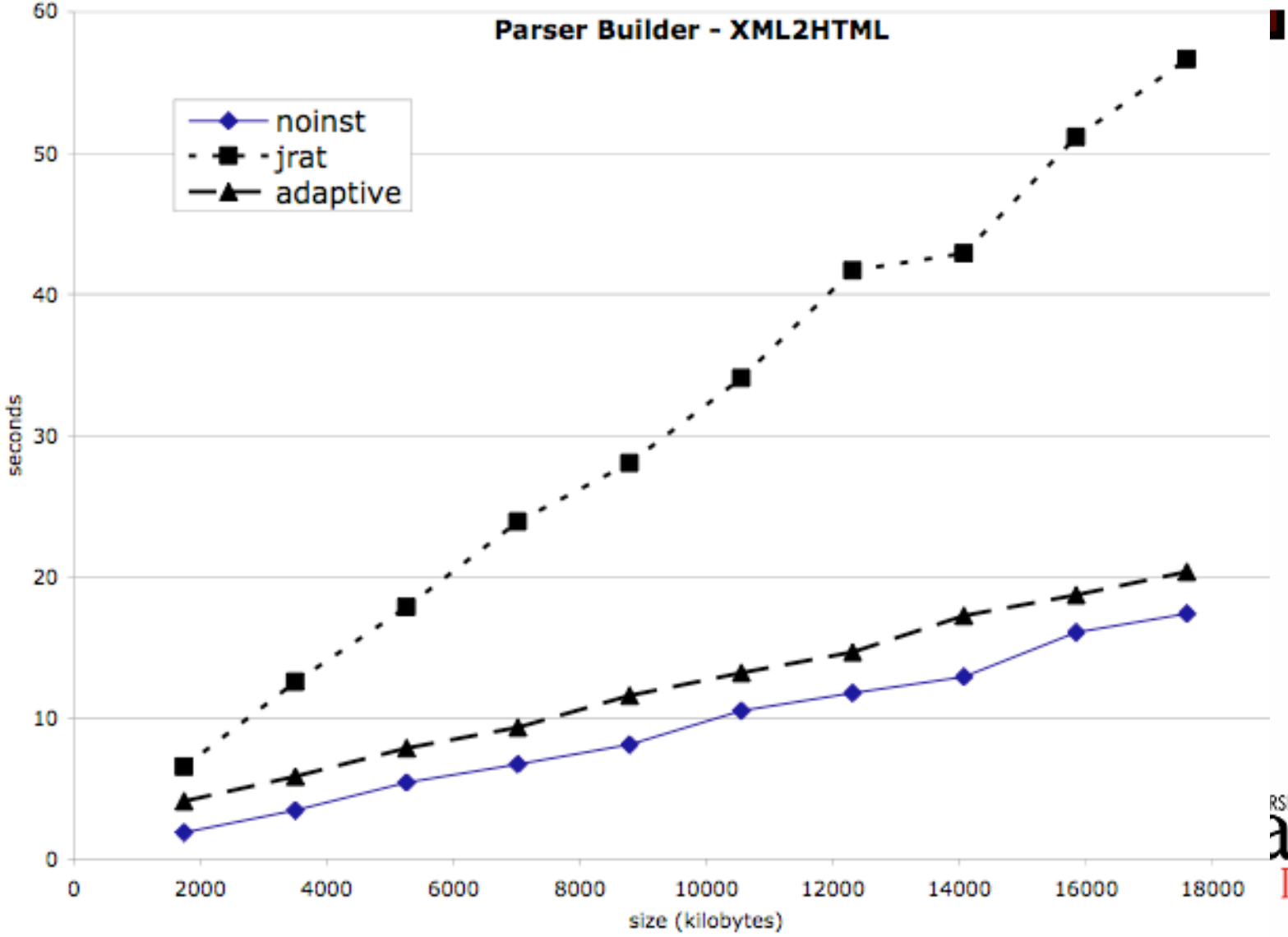


app-property	noinst (sec)	jrat (sec)	Adaptive (sec)
XML2HTML-pb			
XML2HTML-pr			
XML2HTML-sbp			
XML2HTML-sbbsa			
JXML2SQL-pb			
JXML2SQL-pr			
JXML2SQL-sbp			
JXML2SQL-sbbsa			

Experience

app-property	noinst (sec)	jrat (sec)	Adaptive (sec)	Adaptive %overhead
XML2HTML-pb	10.5	34.1	13.2	25.7%
XML2HTML-pr	10.5	271.0	13.1	24.8%
XML2HTML-sbp	10.5	14.5	13.0	23.8%
XML2HTML-sbbsa	10.5	26.4	12.9	22.9%
JXML2SQL-pb	12.6	30.1	16.8	33.3%
JXML2SQL-pr	12.6	277.4	16.5	31.0%
JXML2SQL-sbp	12.6	15.5	16.3	29.4%
JXML2SQL-sbbsa	12.6	24.8	16.6	31.7%

Experience


Parser Builder - XML2HTML
noinst
jrat
adaptive
seconds
size (kilobytes)
0
10
20
30
40
50
60
0
2000
4000
6000
8000
10000
12000
14000
16000
18000

Related Work

- Staged analysis
  - Loop dependence testing (see Wolfe book)
  - IBM SAFE (ISSTA'06)
- Overhead reduction for state properties
  - Up-front analysis to improve probe placement
    - Dominators, weighted edges, payload
  - Run-time sampling
    - Over time, events, user population
  - Adjusting probes during execution
    - Jfluid, JVM 1.5

Our ongoing work

Empirical study to understand the cost-effectiveness of this approach

- Large complex widely-used APIs (e.g., Hibernate)
- Real code bases

Sampling for path properties

- No false error reports, but may miss errors [ASE'08]
- Exploit feedback from deployed execution

Monitoring Infrastructure

Sofya an open source framework for developing dynamic analyses techniques for Java

- Mechanisms to add/remove probes efficiently
- Reference counting support
- Encoding and handling of FSAs
  - Event Description Language
  - Tune event capture to requirements of analyses
  - Components to split/filter streams (multiple FSAs)

http://sofya.unl.edu

e² ESQuaReD

University of Nebraska Lincoln

For you to do

Investigate the benefits of using more precise path-sensitive static analysis to obtain more or bigger safe regions

What information can be mined from an inconclusive model check?

- with abstractions (see Lal et al. SAS'07)
- with resource bounds

Related Work : Runtime Monitoring

- Eagle, Java PathExplorer (Klaus & friends)
- MOP
- MAC
- Tracematches (ECOOP'07, OOPSLA'7, ISSTA'08)
- Sampling for state properties (RV'07, ASE'08,OOPSLA'08)