Algorithmic Verification of Procedural Programs in the Presence of Code Variability

Siavash Soleimanifard Dilian Gurov

School of Computer Science and Communication
KTH Royal Institute of Technology
Stockholm

FACS 2014, Bertinoro, Italy
September 2014

Algorithmic Verification of Procedural Programs in the Presence of Code Variability

Siavash Soleimanifard Dilian Gurov

School of Computer Science and Communication
KTH Royal Institute of Technology
Stockholm

FACS 2014, Bertinoro, Italy
September 2014

Algorithmic Verification of Procedural Programs in the Presence of Code Variability

Siavash Soleimanifard Dilian Gurov

School of Computer Science and Communication
KTH Royal Institute of Technology
Stockholm

FACS 2014, Bertinoro, Italy
September 2014

Algorithmic Verification of Procedural Programs in the Presence of Code Variability

Siavash Soleimanifard Dilian Gurov

School of Computer Science and Communication
KTH Royal Institute of Technology
Stockholm

FACS 2014, Bertinoro, Italy
September 2014

Variability Scenarios

Incomplete programs

- E.g., open protocols

Mobile Code

- E.g., add-ons, extensions

Code evolution

- E.g., application updates, self adaptive systems

Multiple implementations

- E.g., product families

Variability

```
import java.io.*;
import javax.servlet.ServletException;

import vle.*;
//import utils.*;

public class ContainerModel {

    public void dispatcher(String arg) {
        // modeling the request by the input argument
        String request = arg;
        /* modeling the instantiation of the container by
           creating objects of servlets
           The mapping is extracted from web.xml file
        */
        try {
            VLEGetData vlegetdata = new VLEGetData();
            VLEPostData vlepostdata = new VLEPostData();
            VLEPostJournalData vlepostjournaldata = new VLEPostJournalData();
            VLEGetJournalData vlegetjournaldata = new VLEGetJournalData();
            VLEGetAnnotations vlegetannotations = new VLEGetAnnotations();
            VLEPostAnnotations vlepostannotations = new VLEPostAnnotations();
            VLEGetFlag vlegetflag = new VLEGetFlag();
            VLEPostFlag vlepostflag = new VLEPostFlag();
            VLEView vleview = new VLEView();
            VLEConfig vleconfig = new VLEConfig();
            VLEGetUser vlegetuser = new VLEGetUser();

            // these are constructors of utils classes
            /*EchoPostData echopostdata = new EchoPostData();
            FileManager filemanager = new FileManager();
            TTS tts = new TTS(""); */

            /* modeling the container calls by a while loop
               useful when there is request dispatching and
               forwarding
            */
            while (true) {
                if (request.equals("vlegetdata")) { vlegetdata.doGet(null, null); }
                if (request.equals("vlepostdata")) { vlepostdata.doPost(null, null); }
                if (request.equals("vlegetjournaldata")) { vlegetjournaldata.doGet(null, null); }
                if (request.equals("vlepostjournaldata")) { vlepostjournaldata.doPost(null, null); }
                if (request.equals("vlegetannotations")) { vlegetannotations.doGet(null, null); }
                if (request.equals("vlepostannotations")) { vlepostannotations.doPost(null, null); }
                if (request.equals("vlegetflag")) { vlegetflag.doGet(null, null); }
                if (request.equals("vlepostflag")) { vlepostflag.doPost(null, null); }
                if (request.equals("vleview")) { vleview.doGet(null, null); }
                /* if (request.equals("vleconfig")) {
                    vleconfig.doGet(null, null);
                    //vleconfig.doPost(null, null);
                }
                if (request.equals("vleconfig")) {
                    vlegetuser.doGet(null, null);
                    //vlegetuser.doPost(null, null);
                } */

                // these are calls to utils classes
                /* if (request.equals("tts")) { tts.saveToFile("file"); }
                if (request.equals("echopostdata")) {
                    echopostdata.doGet(null, null);
                    echopostdata.doPost(null, null);
                    } */
                /* if (request.equals("filemanager")) {
                    filemanager.doGet(null, null);
                    filemanager.doPost(null, null);
                    } */

                // this is to break the loop if the request is not going be dispatched
                if (! request.equals("forward")) { break; }
            }

            /* modeling the container calls by a if conditions
               useful when there is no request dispatching and
```

```
*/
public class FileManager extends HttpServlet implements Servlet{
  static final long serialVersionUID = 1L;

  private final static String COMMAND = "command";

  private final static String PARAM1 = "param1";

  private final static String PARAM2 = "param2";

  private final static String PARAM3 = "param3";

  private final static String PARAM4 = "param4";

  private final static String PROJECT_PATHS = "projectPaths";

  private final static String HOSTED_PROJECT_PATHS = "hostedProjectPaths";

  private final static String ZIP_DIRECTORY = "archives";

      /* (non-Java-doc)
       * @see javax.servlet.http.HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)
       */
      protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
            String command = request.getParameter(COMMAND);
            if(command.equals("retrieveFile")){
                  response.getWriter().write(this.retrieveFile(request));
            }
      }

      /* (non-Java-doc)
       * @see javax.servlet.http.HttpServlet#doPost(HttpServletRequest request, HttpServletResponse response)
       */
      protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
            String command = request.getParameter(COMMAND);

            if(command!=null){
                  if(command.equals("createProject")){
                        response.getWriter().write(this.createProject(request));
                  } else if(command.equals("projectList")){
                        response.getWriter().write(this.getProjectList(request));
                  } else if(command.equals("hostedProjectList")){
                        response.getWriter().write(this.getHostedProjectList(request));
                  } else if(command.equals("retrieveFile")){
                        response.getWriter().write(this.retrieveFile(request));
                  } else if(command.equals("updateFile")){
                        response.getWriter().write(this.updateFile(request));
                  } else if(command.equals("createNode")){
                        response.getWriter().write(this.createNode(request));
                  } else if(command.equals("createSequence")){
                        response.getWriter().write(this.createSequence(request));
                  } else if(command.equals("exportProject")){
                        this.exportProject(request, response);
                  } else if(command.equals("removeFile")){
                        response.getWriter().write(this.removeFile(request));
                  } else if(command.equals("updateAudioFiles")) {
                        response.getWriter().write(this.updateAudioFiles(request, response));
                  } else if(command.equals("special")){
                        this.processSpecial(request, response);
                  } else if(command.equals("specialToo")){
                        this.specialToo(request, response);
                  } else {
                        throw new ServletException("This servlet does not understand this command: " + command);
                  }
            } else if(ServletFileUpload.isMultipartContent(request)){
                  response.setContentType("text/html; charset=UTF-8");
                  try{
                        this.importProject(request);
                        response.getWriter().print("success");
                  } catch(Exception e){
                        e.printStackTrace();
                        response.getWriter().write("failed");
                  }
            } else {
```

$: \theta$

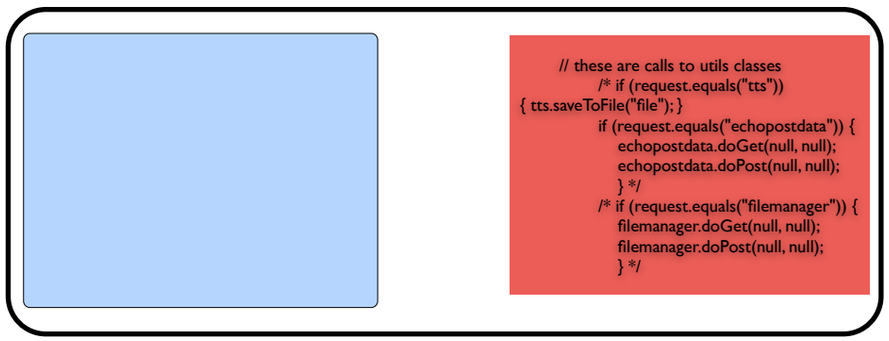
Variability

```
import java.io.*;
import javax.servlet.ServletException;

import vle.*;
//import utils.*;

public class ContainerModel {

    public void dispatcher(String arg) {
        // modeling the request by the input argument
        String request = arg;
        /* modeling the instantiation of the container by
           creating objects of servlets
           The mapping is extracted from web.xml file
        */
        try {
            VLEGetData vlegetdata = new VLEGetData();
            VLEPostData vlepostdata = new VLEPostData();
            VLEPostJournalData vlepostjournaldata = new VLEPostJournalData();
            VLEGetJournalData vlegetjournaldata = new VLEGetJournalData();
            VLEGetAnnotations vlegetannotations = new VLEGetAnnotations();
            VLEPostAnnotations vlepostannotations = new VLEPostAnnotations();
            VLEGetFlag vlegetflag = new VLEGetFlag();
            VLEPostFlag vlepostflag = new VLEPostFlag();
            VLEView vleview = new VLEView();
            VLEConfig vleconfig = new VLEConfig();
            VLEGetUser vlegetuser = new VLEGetUser();

            // these are constructors of utils classes
            /*EchoPostData echopostdata = new EchoPostData();
            FileManager filemanager = new FileManager();
            TTS tts = new TTS(""); */

            /* modeling the container calls by a while loop
               useful when there is request dispatching and
               forwarding
            */
            while (true) {
                if (request.equals("vlegetdata")) { vlegetdata.doGet(null, null); }
                if (request.equals("vlepostdata")) { vlepostdata.doPost(null, null); }
                if (request.equals("vlegetjournaldata")) { vlegetjournaldata.doGet(null, null); }
                if (request.equals("vlepostjournaldata")) { vlepostjournaldata.doPost(null, null); }
                if (request.equals("vlegetannotations")) { vlegetannotations.doGet(null, null); }
                if (request.equals("vlepostannotations")) { vlepostannotations.doPost(null, null); }
                if (request.equals("vlegetflag")) { vlegetflag.doGet(null, null); }
                if (request.equals("vlepostflag")) { vlepostflag.doPost(null, null); }
                if (request.equals("vleview")) { vleview.doGet(null, null); }
                /* if (request.equals("vleconfig")) {
                    vleconfig.doGet(null, null);
                    //vleconfig.doPost(null, null);
                }
                if (request.equals("vleconfig")) {
                    vlegetuser.doGet(null, null);
                    //vlegetuser.doPost(null, null);
                } */

                    // these are calls to utils classes
                            /* if (request.equals("tts"))                    }
                { tts.saveToFile("file"); }
                            if (request.equals("echopostdata")) {
                                echopostdata.doGet(null, null);
                                echopostdata.doPost(null, null);
                                } */
                            /* if (request.equals("filemanager")) {
                                filemanager.doGet(null, null);
                                filemanager.doPost(null, null);
                                } */
                                                                  oing be dispatched
                if (! request.equals("forward")) { break; }
            }

            /* modeling the container calls by a if conditions
               useful when there is no request dispatching and
```

```
*/
 public class FileManager extends HttpServlet implements Servlet{
   static final long serialVersionUID = 1L;

   private final static String COMMAND = "command";

   private final static String PARAM1 = "param1";

   private final static String PARAM2 = "param2";

   private final static String PARAM3 = "param3";

   private final static String PARAM4 = "param4";

   private final static String PROJECT_PATHS = "projectPaths";

   private final static String HOSTED_PROJECT_PATHS = "hostedProjectPaths";

   private final static String ZIP_DIRECTORY = "archives";

       /* (non-Java-doc)
        * @see javax.servlet.http.HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)
        */
       protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
               String command = request.getParameter(COMMAND);
               if(command.equals("retrieveFile")){
                       response.getWriter().write(this.retrieveFile(request));
               }
       }

       /* (non-Java-doc)
        * @see javax.servlet.http.HttpServlet#doPost(HttpServletRequest request, HttpServletResponse response)
        */
       protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
               String command = request.getParameter(COMMAND);

               if(command!=null){
                       if(command.equals("createProject")){
                               response.getWriter().write(this.createProject(request));
                       } else if(command.equals("projectList")){
                               response.getWriter().write(this.getProjectList(request));
                       } else if(command.equals("hostedProjectList")){
                               response.getWriter().write(this.getHostedProjectList(request));
                       } else if(command.equals("retrieveFile")){
                               response.getWriter().write(this.retrieveFile(request));
                       } else if(command.equals("updateFile")){
                               response.getWriter().write(this.updateFile(request));
                       } else if(command.equals("createNode")){
                               response.getWriter().write(this.createNode(request));
                       } else if(command.equals("createSequence")){
                               response.getWriter().write(this.createSequence(request));
                       } else if(command.equals("exportProject")){
                               this.exportProject(request, response);
                       } else if(command.equals("removeFile")){
                               response.getWriter().write(this.removeFile(request));
                       } else if(command.equals("updateAudioFiles")) {
                               response.getWriter().write(this.updateAudioFiles(request, response));
                       } else if(command.equals("special")){
                               this.processSpecial(request, response);
                       } else if(command.equals("specialToo")){
                               this.specialToo(request, response);
                       } else {
                               throw new ServletException("This servlet does not understand this command: " + command);
                       }
               } else if(ServletFileUpload.isMultipartContent(request)){
                       response.setContentType("text/html; charset=UTF-8");
                       try{
                               this.importProject(request);
                               response.getWriter().print("success");
                       } catch(Exception e){
                               e.printStackTrace();
                               response.getWriter().write("failed");
                       }
               } else {
```

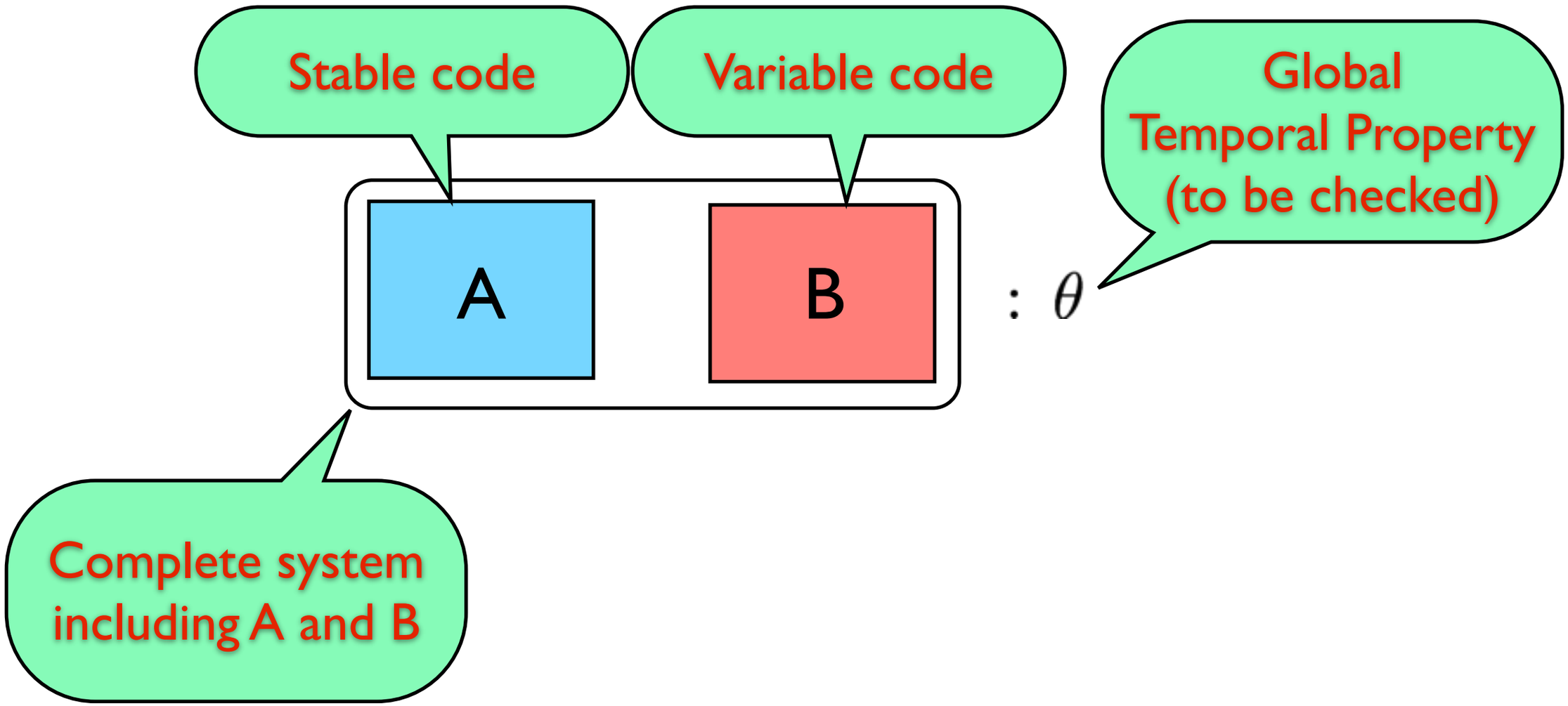
$: \theta$

Variability

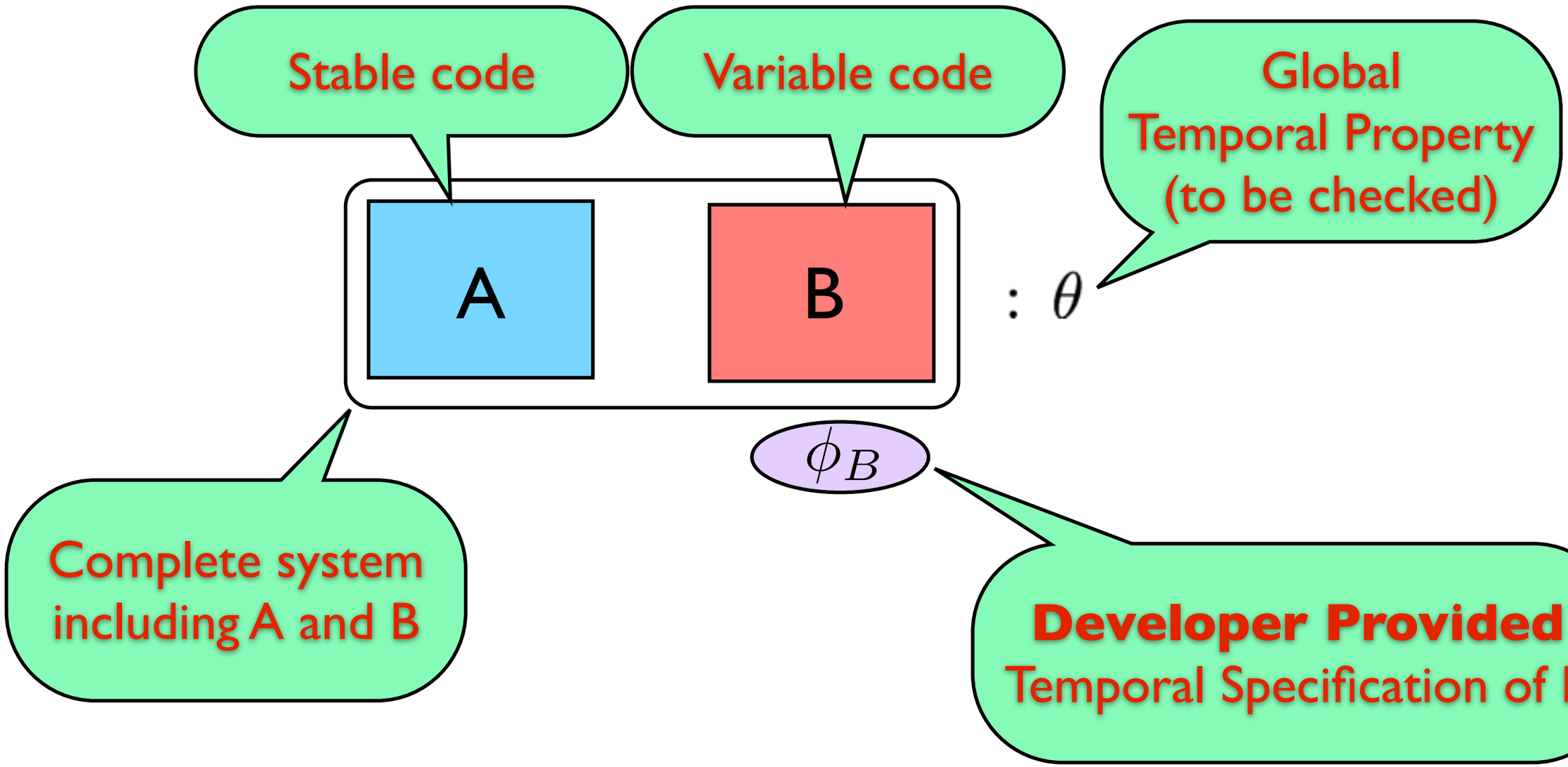


$: \theta$

Variable Systems



Verification Setup



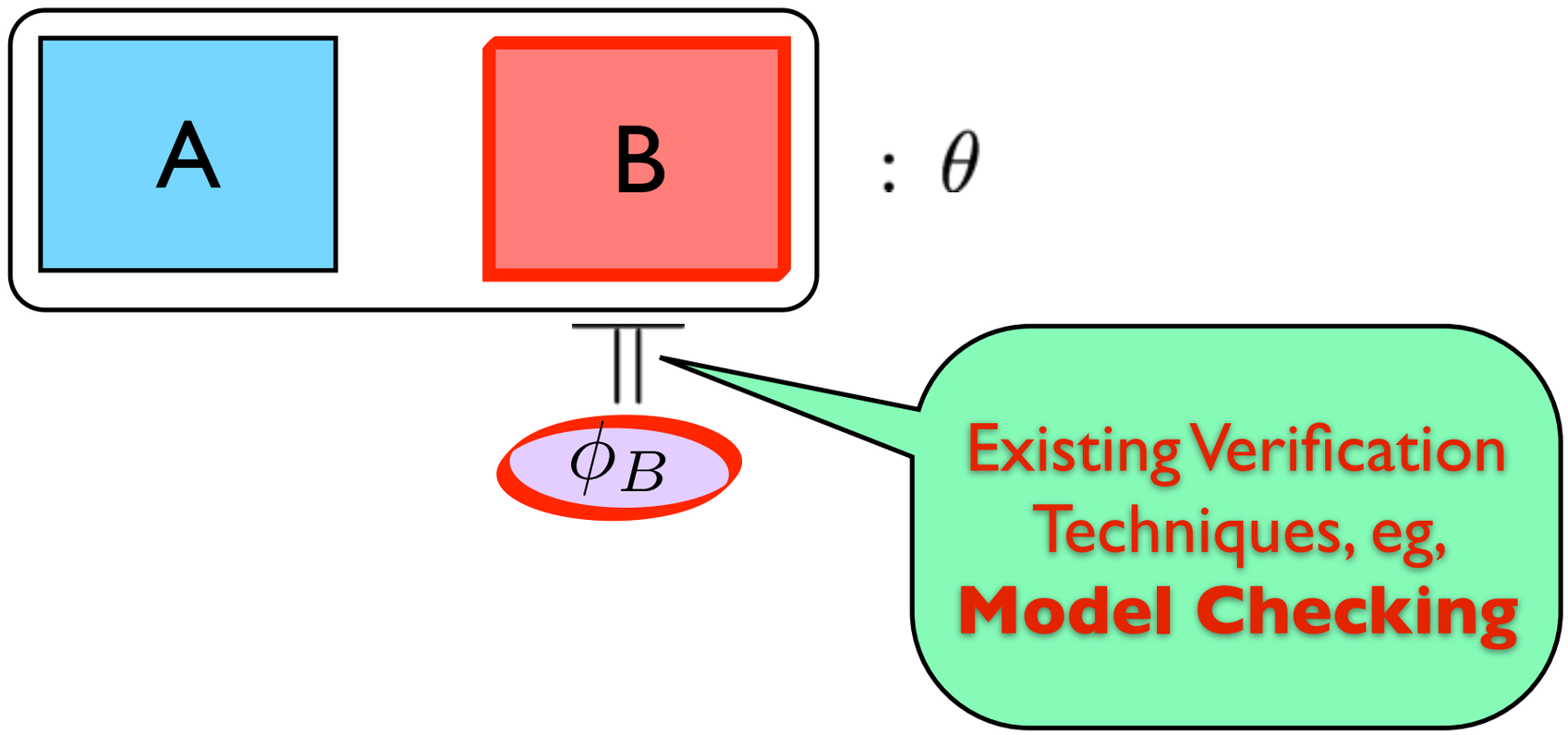
Modular Verification

Verification Subtasks

I. check that each variable component satisfies its local specification

II. check that the **composition** of the specification of variable components together with the implementation of the stable ones satisfies the global property

Task I: Local Check



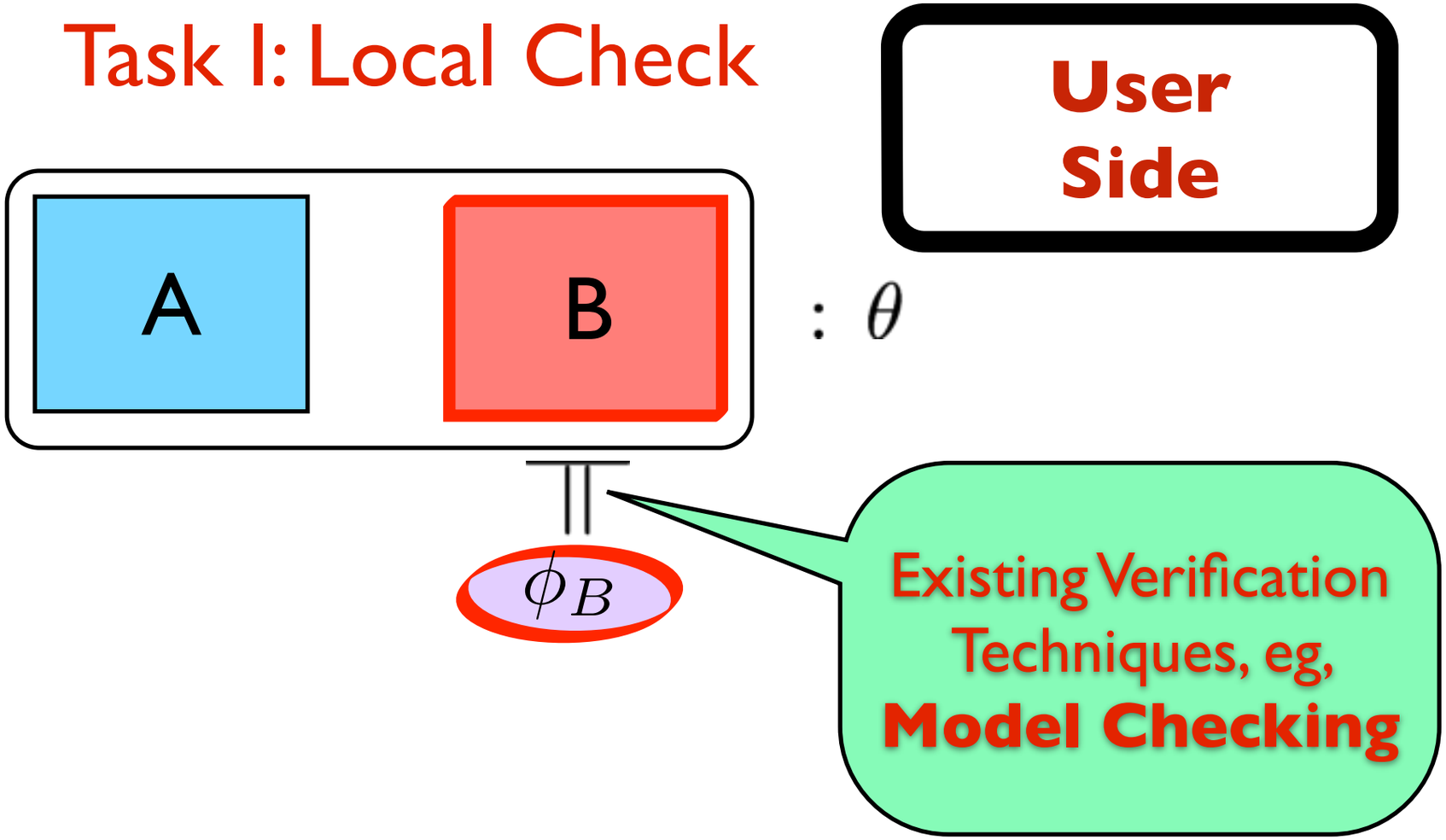
Modular Verification

Verification Subtasks

I. check that each variable component satisfies its local specification

II. check that the **composition** of the specification of variable components together with the implementation of the stable ones satisfies the global property

Task I: Local Check



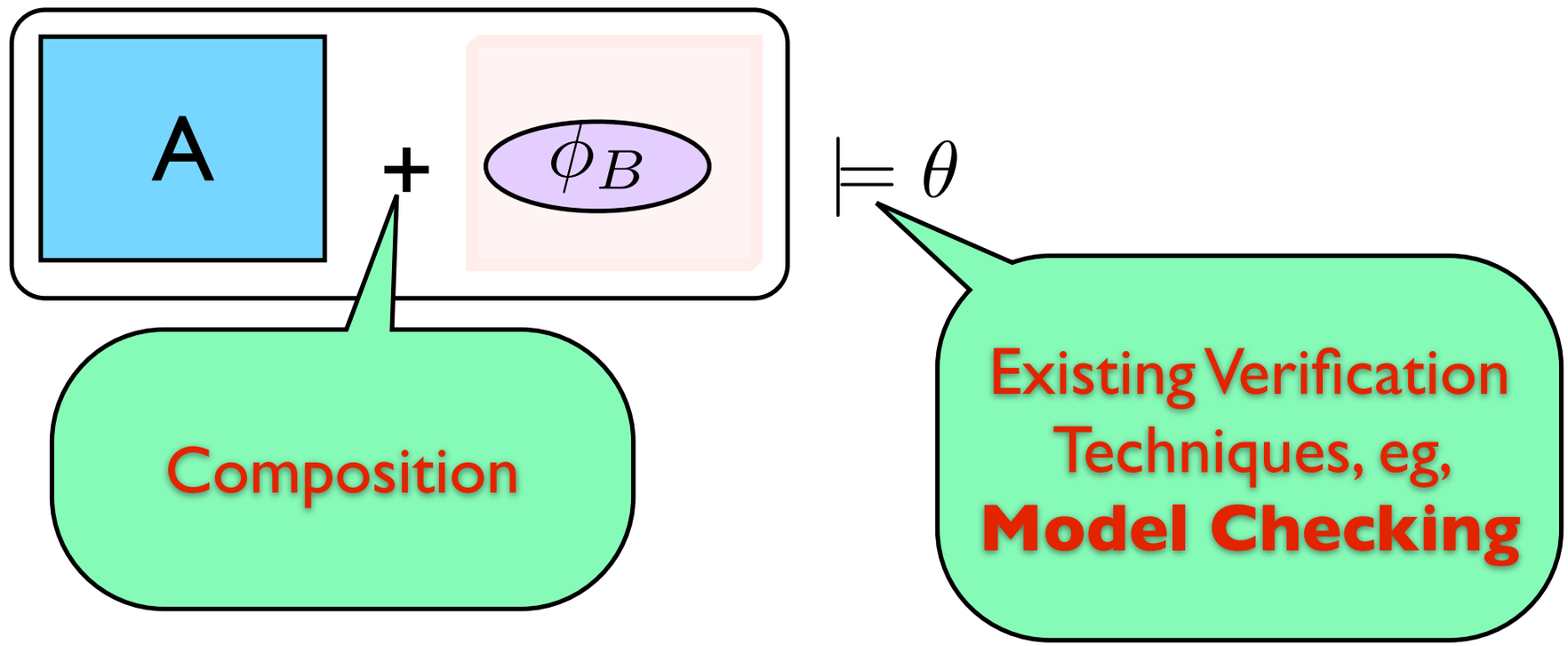
Modular Verification

Verification Subtasks

I. check that each **variable component** satisfies **its local specification**

II. check that the **composition** of the specification of variable components together with the implementation of the stable ones satisfies the global property

Task II: Global Check



Modular Verification

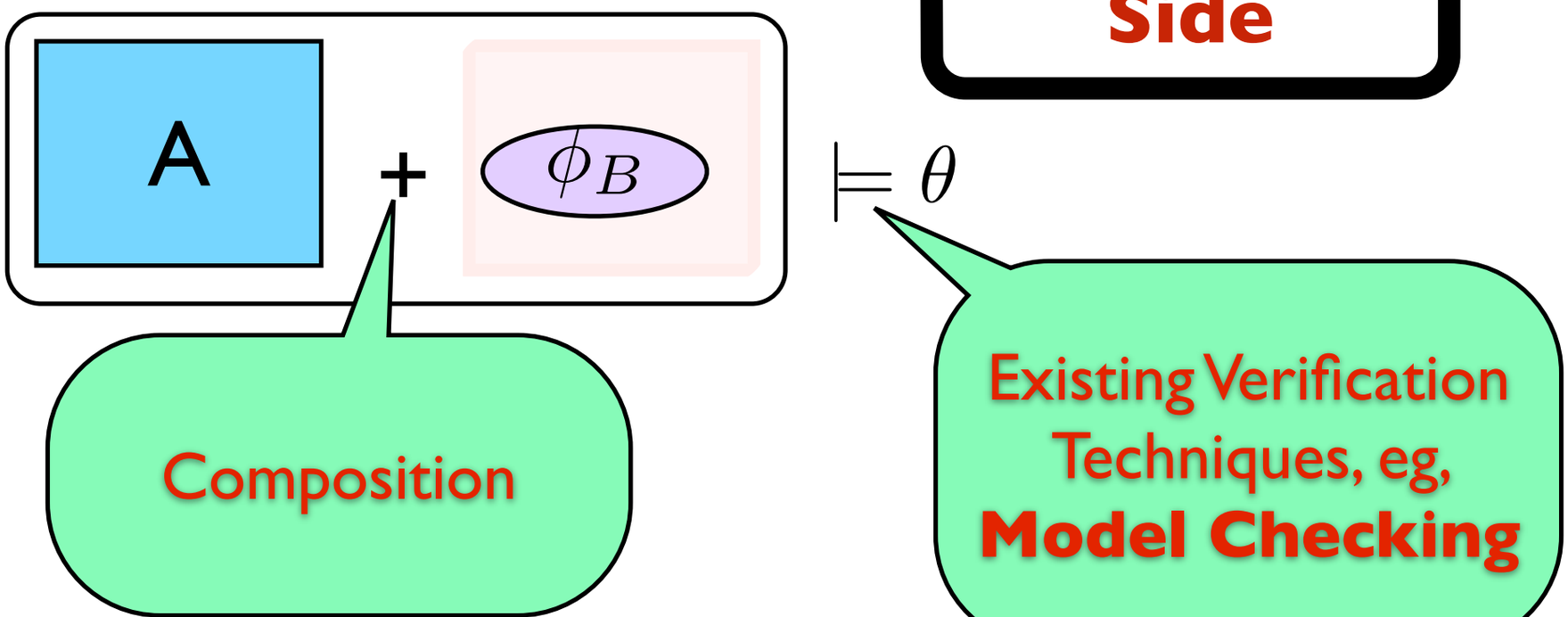
Verification Subtasks

I. check that each **variable component** satisfies **its local specification**

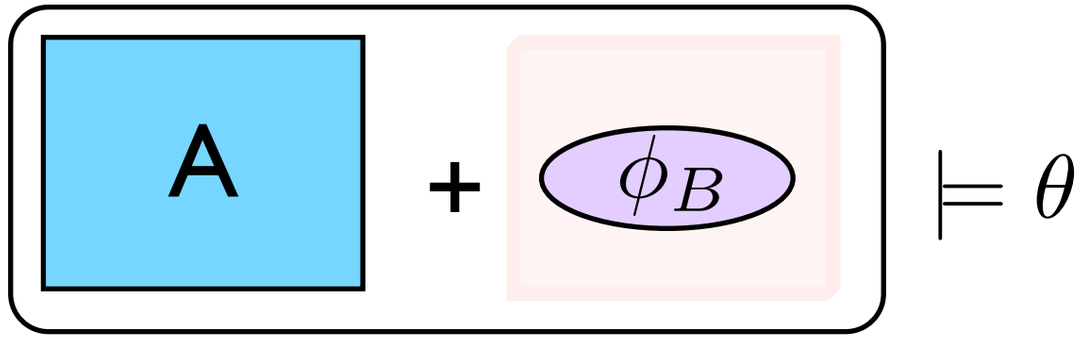
II. check that the **composition** of the specification of variable components together with the implementation of the stable ones satisfies the global property

Task II: Global Check

Developer Side



Modular Verification



Existing Approaches

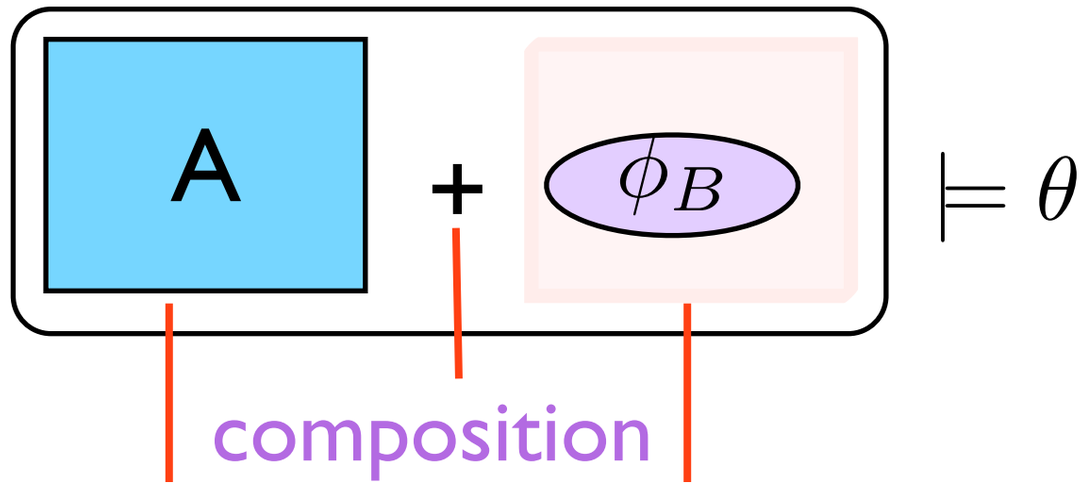
- Modular verification of procedural programs:
  "built-in" for **Hoare-logic** based approaches

Different Properties

- Modular model checking:
  based on **maximal model construction**
  Grumberg & Long 1994: ACTL
  Kupferman & Vardi 2000: ACTL*

Finite Systems (not procedural programs)

Verification based on Maximal Models

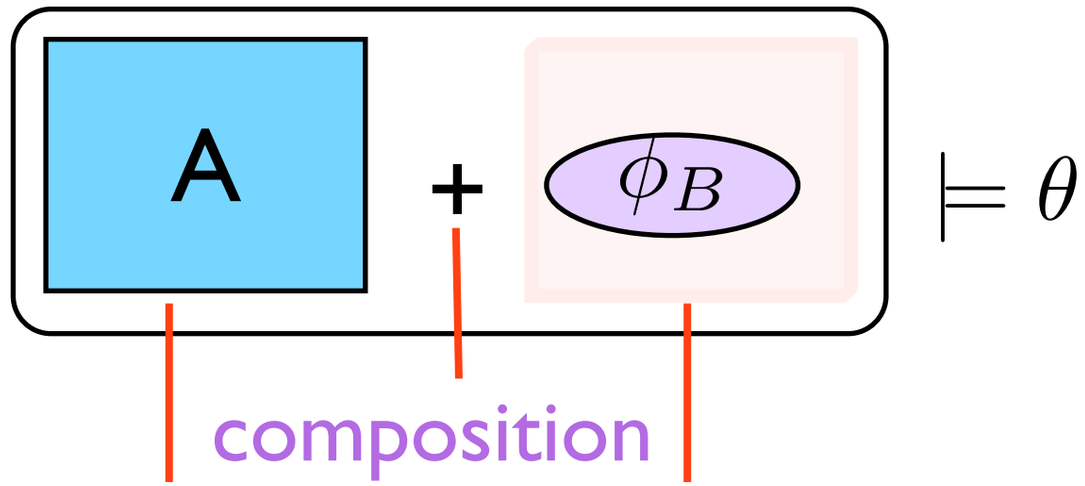


Program Model Maximal Model $\models \theta$

Maximal Models

- A maximal model for ϕ_B is the most general model that satisfies it
  - represents all models that satisfy ϕ_B
  - models represent code, thus a maximal model for ϕ_B represents any code that its model satisfies ϕ_B

Framework



Program Model Maximal Model $\models \theta$

Challenges

- Algorithmic solution (procedural programs, i.e., infinite pushdown systems)
- Practical
  - Task I (user side task) efficient and with minimal manual effort
  - complexity of the maximal model construction
  - difficulty and complexity of specifying specifications and models
  - Independent from programming languages

Framework

Induce

Structure

- Finite state flow graph
- **Symbolic data**
- Maximal Models

Behavior

- Infinite state, pushdown behavior
- **Concrete data values** from (abstract) finite domains

Framework

Code

Induce

Structure

- Finite state flow graph
- **Symbolic data**
- Maximal Models

Behavior

- Infinite state, pushdown behavior
- **Concrete data values** from (abstract) finite domains

Framework

Code

Execution

Induce

Structure

- Finite state flow graph
- **Symbolic data**
- Maximal Models

Behavior

- Infinite state, pushdown behavior
- **Concrete data values** from (abstract) finite domains

Framework

Code

Execution

Induce

Structure

- Finite state flow graph
- **Symbolic data**
- Maximal Models

Behavior

- Infinite state, pushdown behavior
- **Concrete data values** from (abstract) finite domains

Practical Specification

Framework

Code

Execution

Induce

Structure

- Finite state flow graph
- **Symbolic data**
- Maximal Models

Behavior

- Infinite state, pushdown behavior
- **Concrete data values** from (abstract) finite domains

Practical Specification

Completeness

Overview

User Tasks

1. Specification: Local specification (variable) & Global property
2. Define observed instructions

Task I

1. Model check the code of the variable components against their local specifications

Task II

1. Model extraction from stable code
2. Maximal model construction from local specification
3. Compose models and induce the behavior of the system
4. Model check the behavior

Pointer Language

```
decl x = null;
decl y = null;

Main() {
   while(*) {
      new x;
      y = x;
      Foo();
      delete x;
   }
}

Foo() {
  .....
}
```

Pointer Language

```
decl x = null;
decl y = null;

Main() {
   while(*) {
      new x;
      y = x;
      Foo();
      delete x;
   }
}

Foo() {
  .....
}
```

Specification

```
decl x = null;
decl y = null;

Main() {
   while(*) {
     new x;
     y = x;
     Foo();
     delete x;
   }
}
```

```
Foo() {
   .....
}
```

Specification

```
decl x = null;
decl y = null;

Main() {
   while(*) {
     new x;
     y = x;
     Foo();
     delete x;
   }
}
```

```
Foo() {
   .....
}
```

Global **Behavioral** Property

"always a **delete** between two **new**"

Specification

```
decl x = null;
decl y = null;

Main() {
   while(*) {
     new x;
     y = x;
     Foo();
     delete x;
   }
}
```

```
Foo() {
  new x;
}
```

Global **Behavioral** Property

"always a **delete** between two **new**"

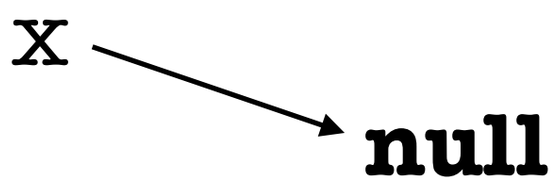
Specification

```
decl x = null;
decl y = null;

Main() {
   while(*) {
     new x;
     y = x;
     Foo();
     delete x;
   }
}
```

```
Foo() {
  new x;
}
```

Global **Behavioral** Property

"always a **delete** between two **new**"



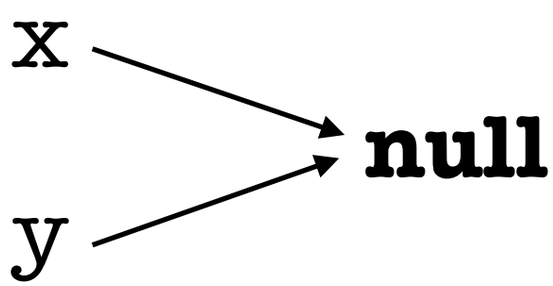
Specification

```
decl x = null;
decl y = null;

Main() {
   while(*) {
     new x;
     y = x;
     Foo();
     delete x;
   }
}
```

```
Foo() {
  new x;
}
```

Global **Behavioral** Property

"always a **delete** between two **new**"



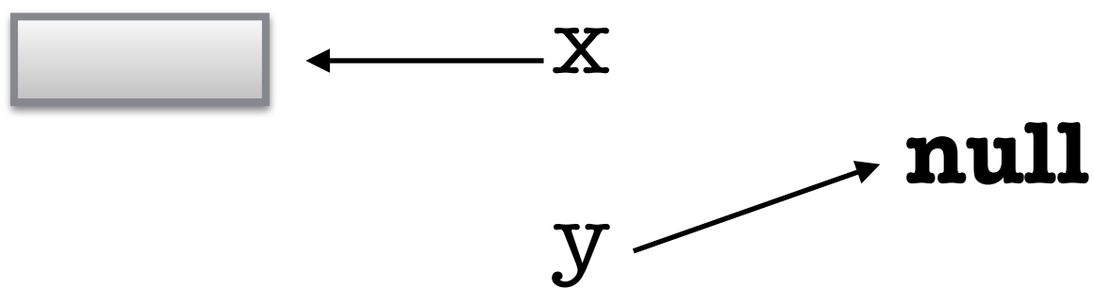
Specification

```
decl x = null;
decl y = null;

Main() {
   while(*) {
     new x;
     y = x;
     Foo();
     delete x;
   }
}
```

```
Foo() {
  new x;
}
```

Global **Behavioral** Property

"always a **delete** between two **new**"



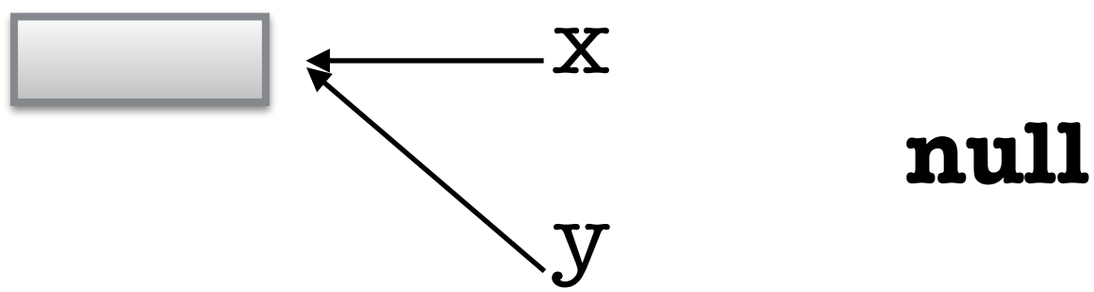
Specification

```
decl x = null;
decl y = null;

Main() {
   while(*) {
     new x;
     y = x;
     Foo();
     delete x;
   }
}
```

```
Foo() {
  new x;
}
```

Global **Behavioral** Property

"always a **delete** between two **new**"



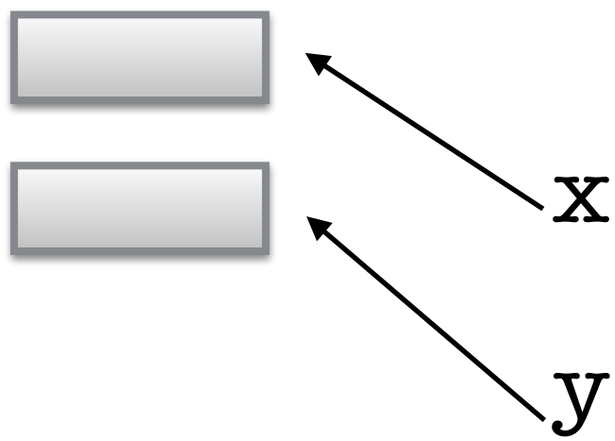
Specification

```
decl x = null;
decl y = null;

Main() {
   while(*) {
     new x;
     y = x;
     Foo();
     delete x;
   }
}
```

```
Foo() {
  new x;
}
```

Global **Behavioral** Property

"always a **delete** between two **new**"



null

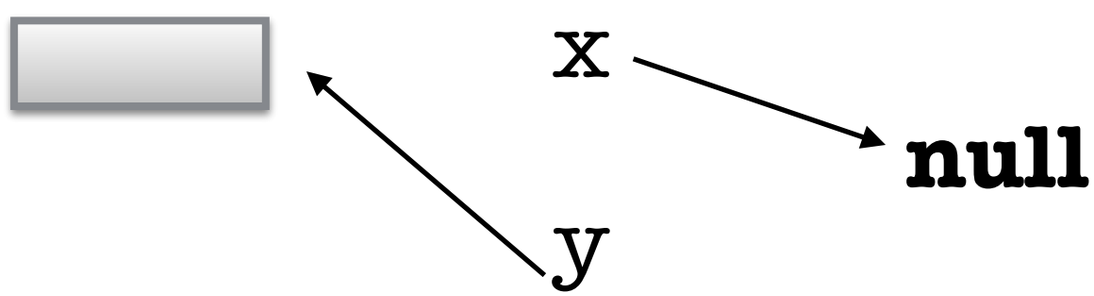
Specification

```
decl x = null;
decl y = null;

Main() {
   while(*) {
     new x;
     y = x;
     Foo();
     delete x;
   }
}
```

```
Foo() {
  new x;
}
```

Global **Behavioral** Property

"always a **delete** between two **new**"



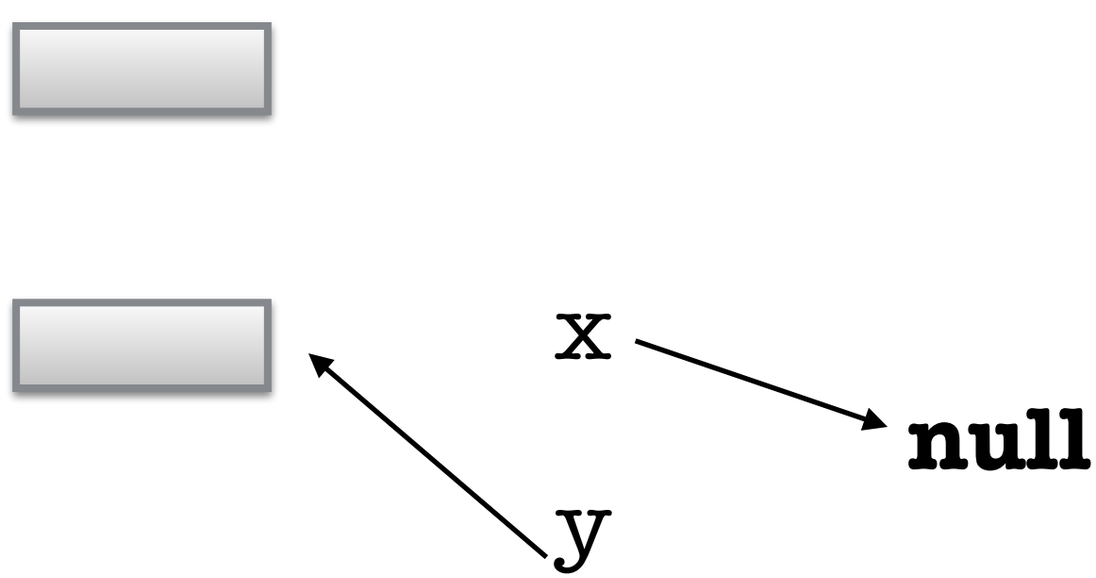
Specification

```
decl x = null;
decl y = null;

Main() {
   while(*) {
     new x;
     y = x;
     Foo();
     delete x;
   }
}
```

```
Foo() {
  new x;
}
```

Global **Behavioral** Property

"always a **delete** between two **new**"



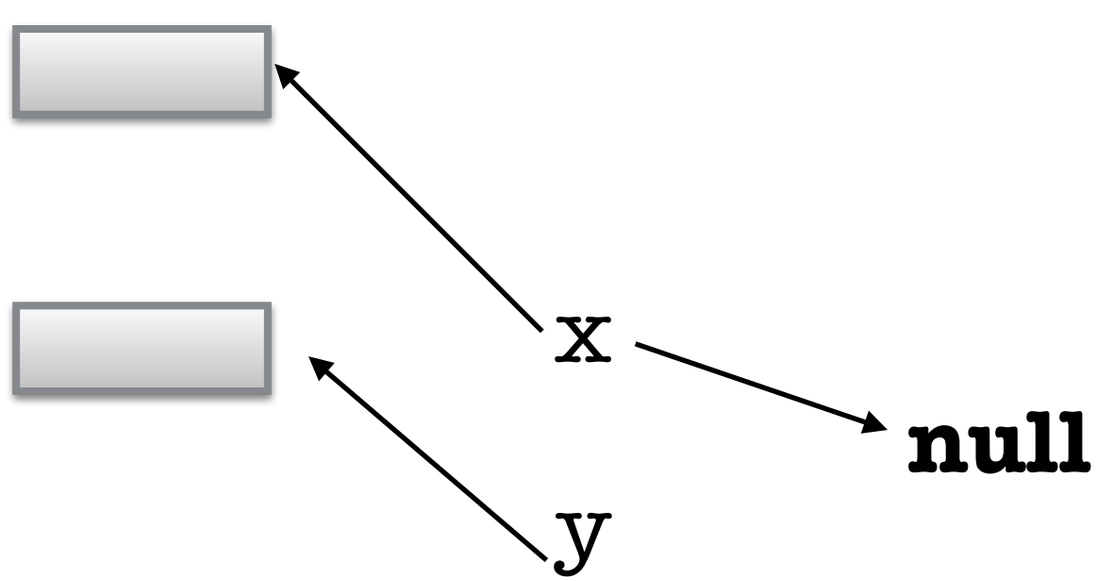
Specification

```
decl x = null;
decl y = null;

Main() {
   while(*) {
     new x;
     y = x;
     Foo();
     delete x;
   }
}
```

```
Foo() {
  new x;
}
```

Global **Behavioral** Property

"always a **delete** between two **new**"



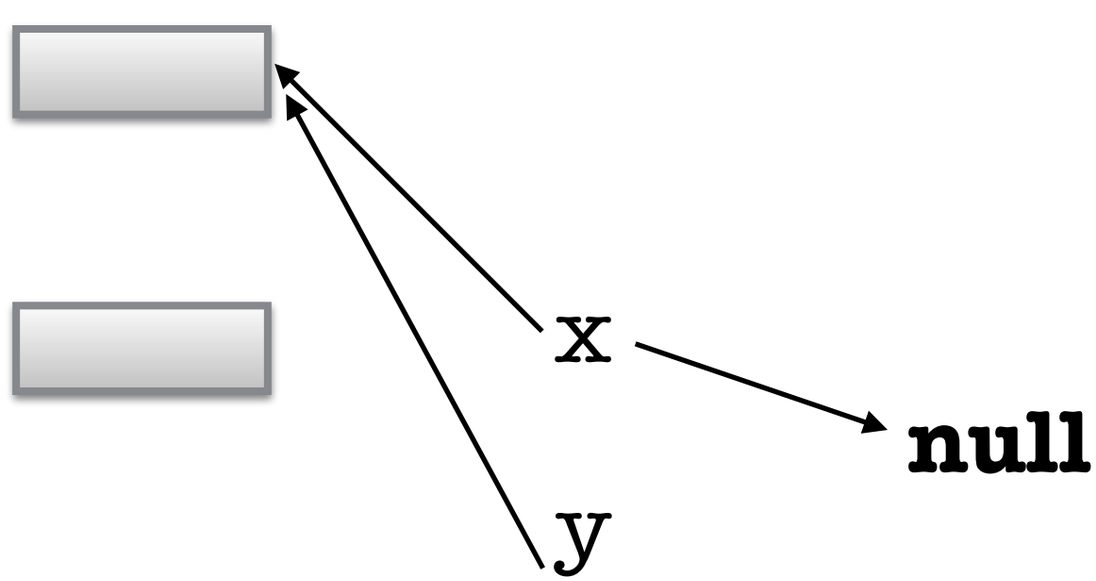
Specification

```
decl x = null;
decl y = null;

Main() {
   while(*) {
     new x;
     y = x;
     Foo();
     delete x;
   }
}
```

```
Foo() {
  new x;
}
```

Global **Behavioral** Property

"always a **delete** between two **new**"



Specification

```
decl x = null;
decl y = null;

Main() {
   while(*) {
     new x;
     y = x;
     Foo();
     delete x;
   }
}
```

```
Foo() {
   .....
}
```

Global **Behavioral** Property

"always a **delete** between two **new**"

Local **Structural** Specification of Foo

"no **new** statement"

Specification

```
decl x = null;
decl y = null;

Main() {
   while(*) {
     new x;
     y = x;
     Foo();
     delete x;
   }
}
```

```
Foo() {
   .....
}
```

Global **Behavioral** Property

"always a **delete** between two **new**"

Local **Structural** Specification of Foo

"no **new** statement"

Observed Instructions

- Observed instructions: **new**, **delete**
- Capture the effect of other instructions through logical conditions of the form
  $v = v' \quad v \neq v'$

Overview

User Tasks

1. Specification: Local specification (variable) & Global property
2. Define observed instructions

Task I

1. Model check the code of the variable components against their local specifications

Task II

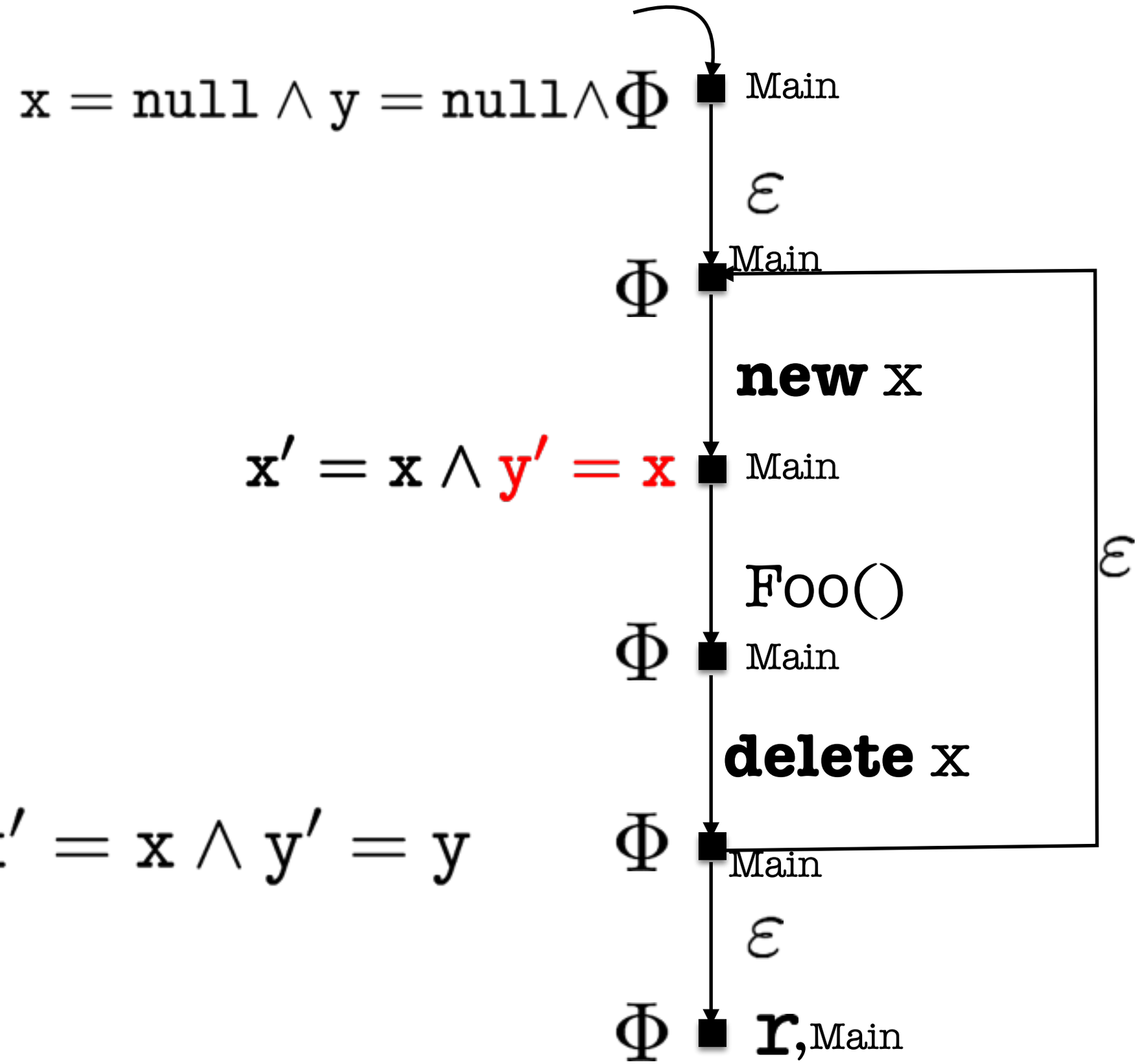
1. Model extraction from stable code
2. Maximal model construction from local specification
3. Compose models and induce the behavior of the system
4. Model check the behavior

Flow Graphs

```
decl x = null;
decl y = null;

Main() {
   while(*) {
     new x;
     y = x;
     Foo();
     delete x;
   }
}
```

```
Foo() {
   .....
}
```

Φ is $\mathtt{x}' = \mathtt{x} \wedge \mathtt{y}' = \mathtt{y}$

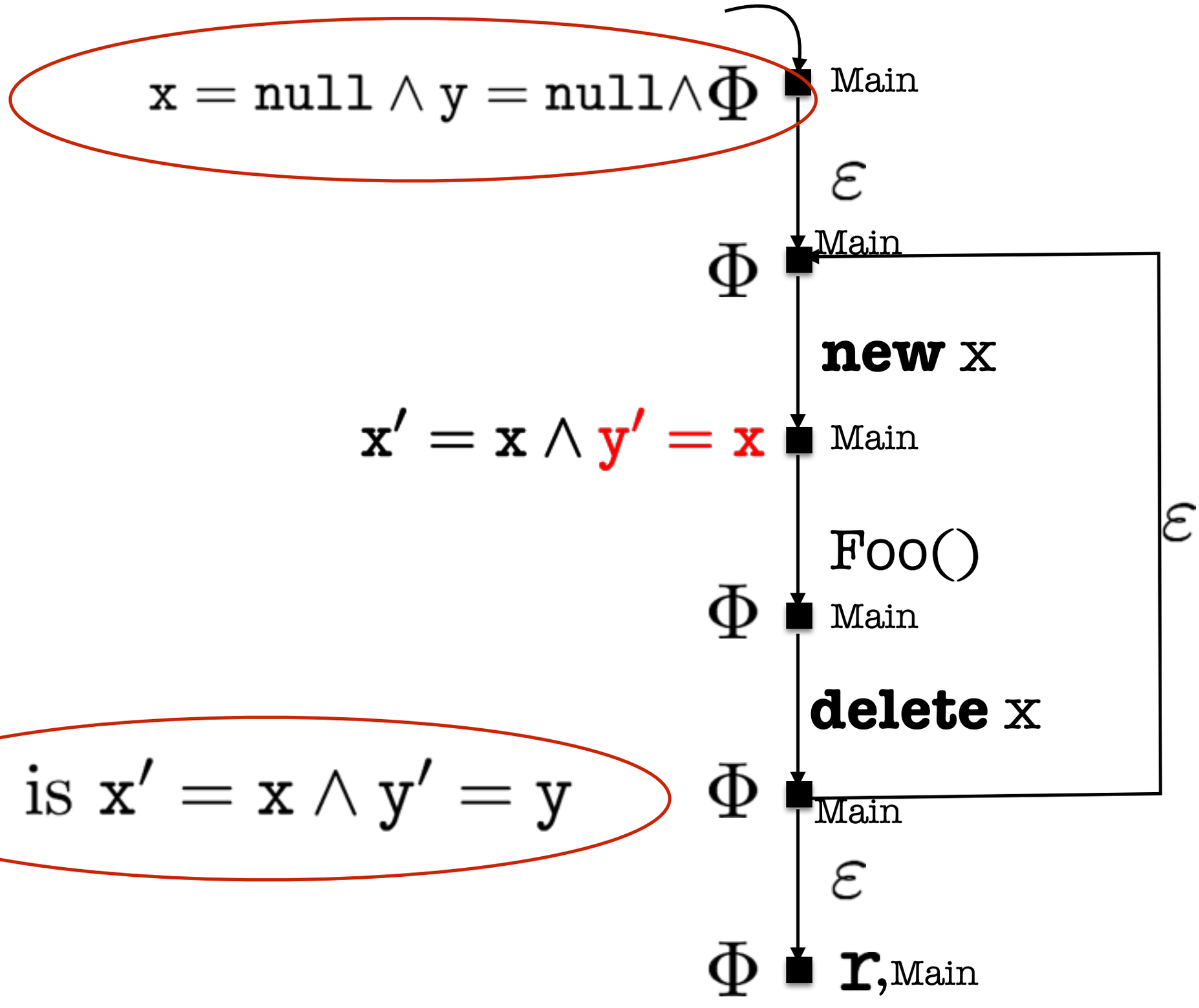


Flow Graphs

```
decl x = null;
decl y = null;

Main() {
    while(*) {
        new x;
        y = x;
        Foo();
        delete x;
    }
}
```

```
Foo() {
    .....
}
```

Φ is $x' = x \wedge y' = y$

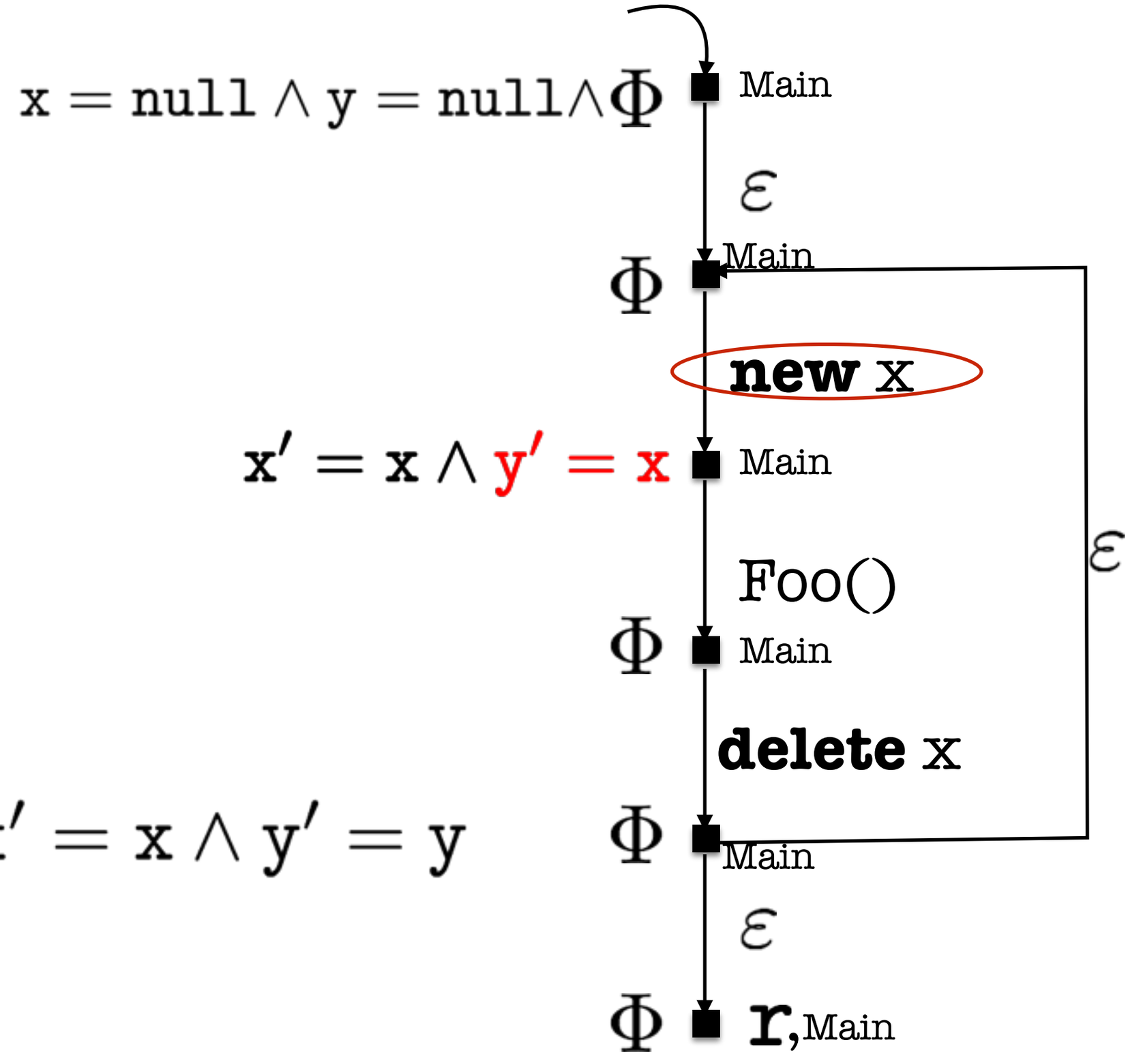


Flow Graphs

```
decl x = null;
decl y = null;

Main() {
   while(*) {
     new x;
     y = x;
     Foo();
     delete x;
   }
}
```

```
Foo() {
   .....
}
```

Φ is $x' = x \wedge y' = y$

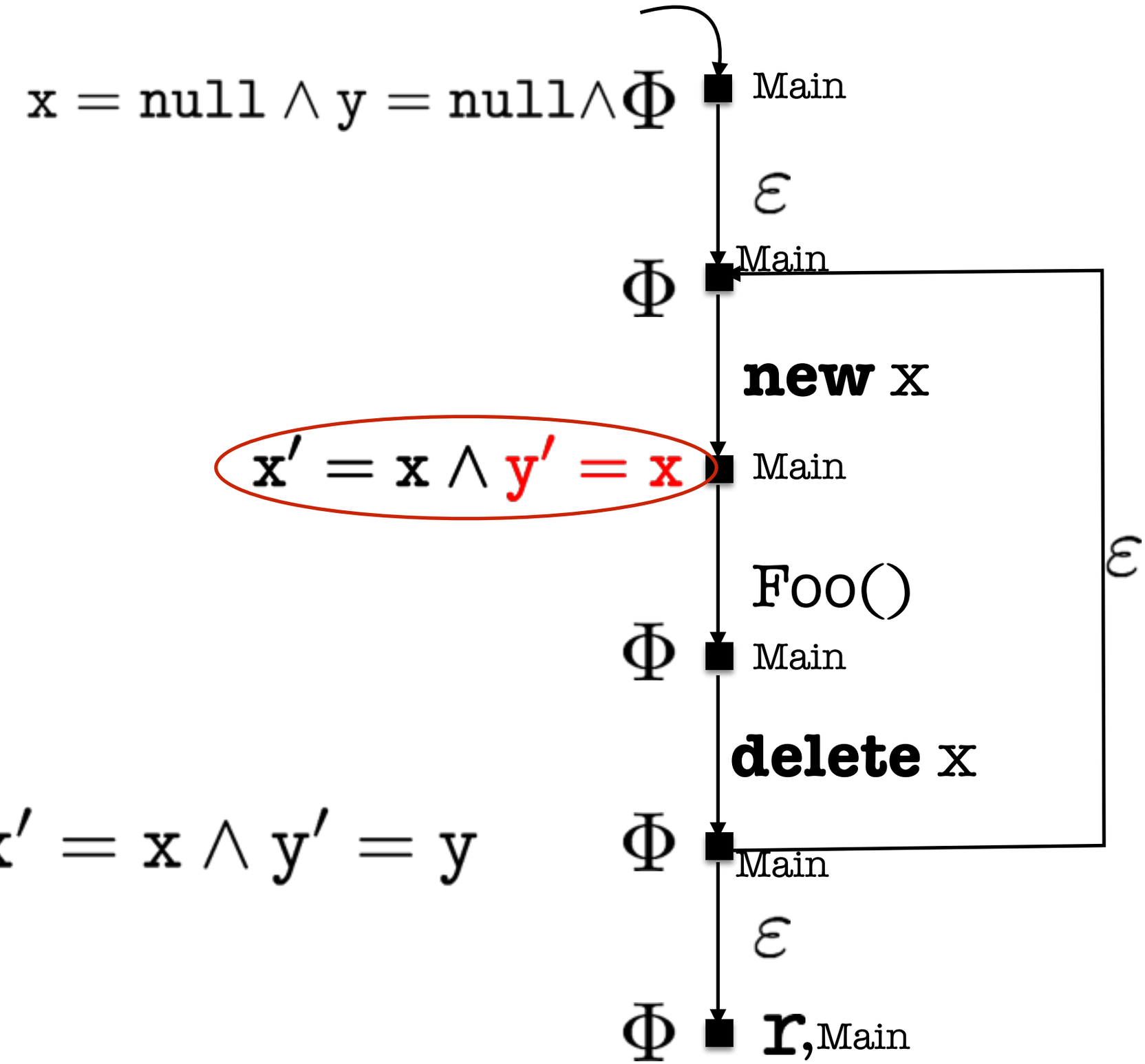


Flow Graphs

```
decl x = null;
decl y = null;

Main() {
   while(*) {
      new x;
      y = x;
      Foo();
      delete x;
   }
}
```

```
Foo() {
   .....
}
```

Φ is $x' = x \wedge y' = y$

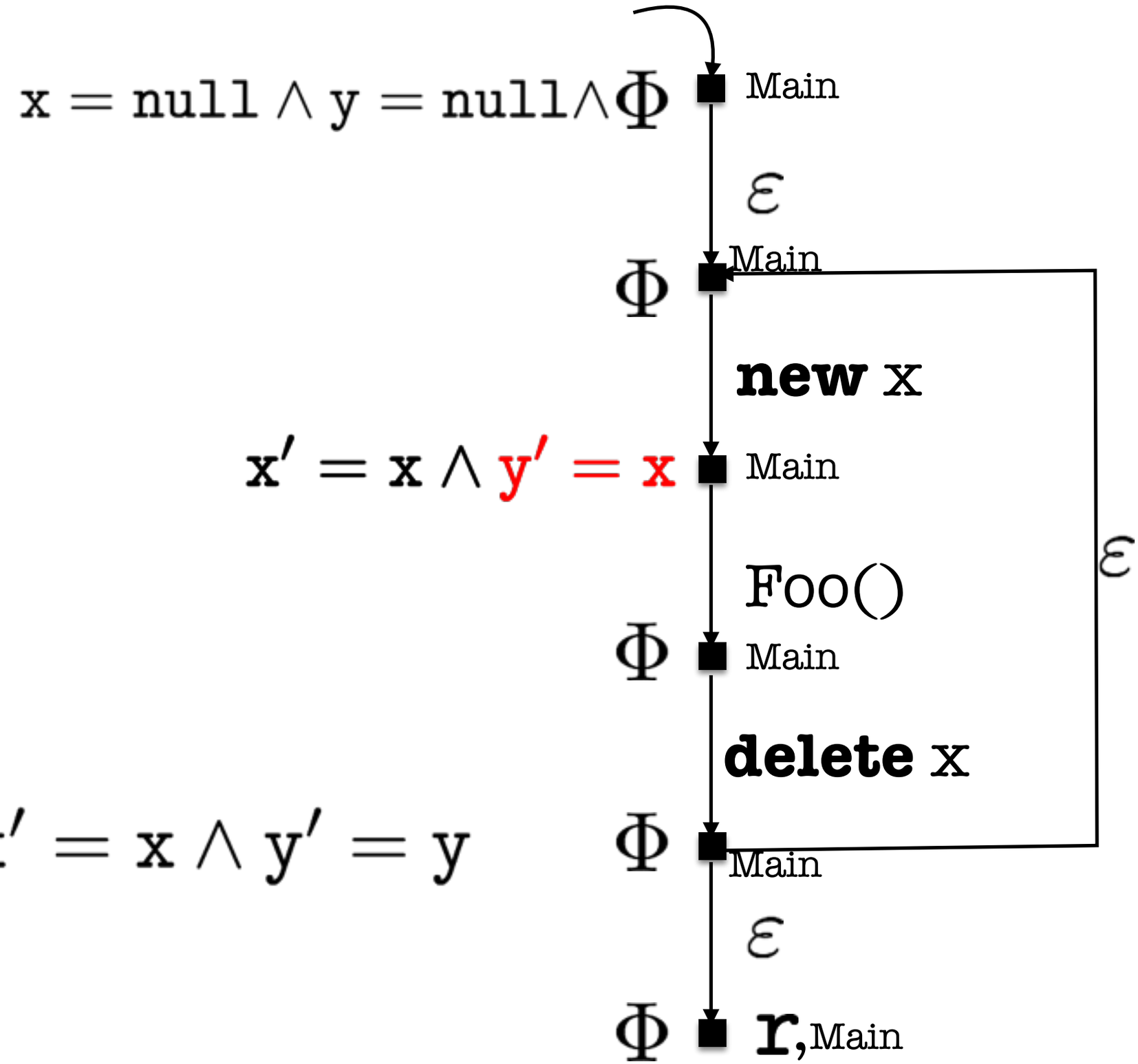


Flow Graphs

```
decl x = null;
decl y = null;

Main() {
   while(*) {
     new x;
     y = x;
     Foo();
     delete x;
   }
}
```

```
Foo() {
   .....
}
```

Φ is $\mathtt{x}' = \mathtt{x} \wedge \mathtt{y}' = \mathtt{y}$



Overview

User Tasks

1. Specification: Local specification (variable) & Global property
2. Define observed instructions

Task I

1. Model check the code of the variable components against their local specifications

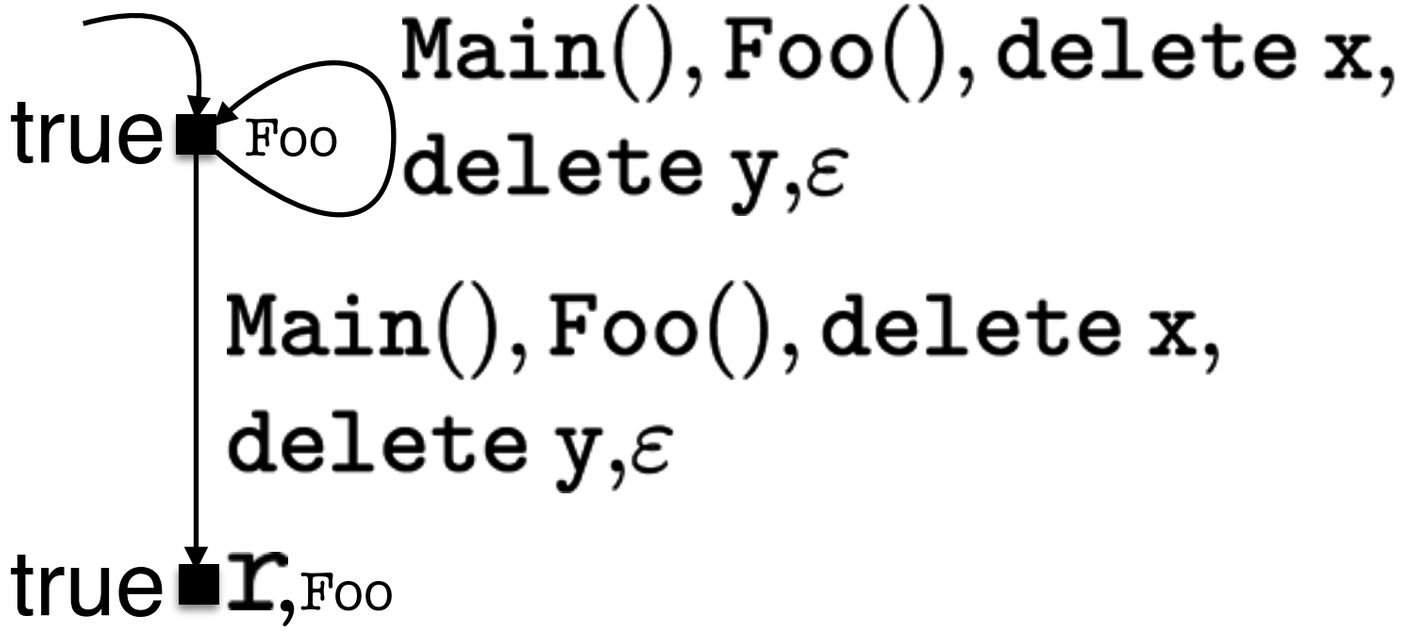
Task II

1. Model extraction from stable code
2. Maximal model construction from local specification
3. Compose models and induce the behavior of the system
4. Model check the behavior

Maximal Flow Graphs

```
decl x = null;
decl y = null;

Main() {
   while(*) {
     new x;
     y = x;
     Foo();
     delete x;
   }
}
```

```
Foo() {
   .....
}
```



Local Structural Specification of Foo

"no **new** statement"

Overview

User Tasks

1. Specification: Local specification (variable) & Global property
2. Define observed instructions

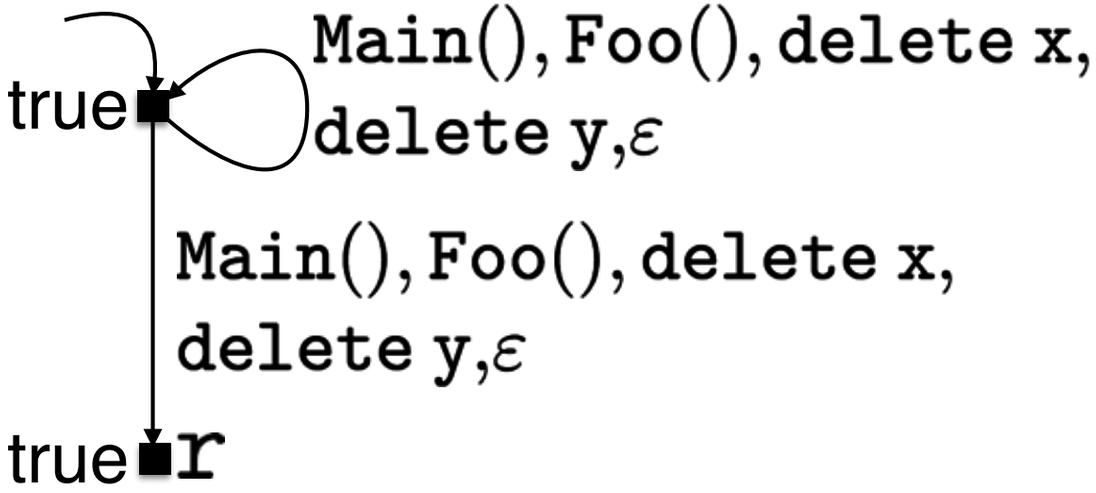
Task I

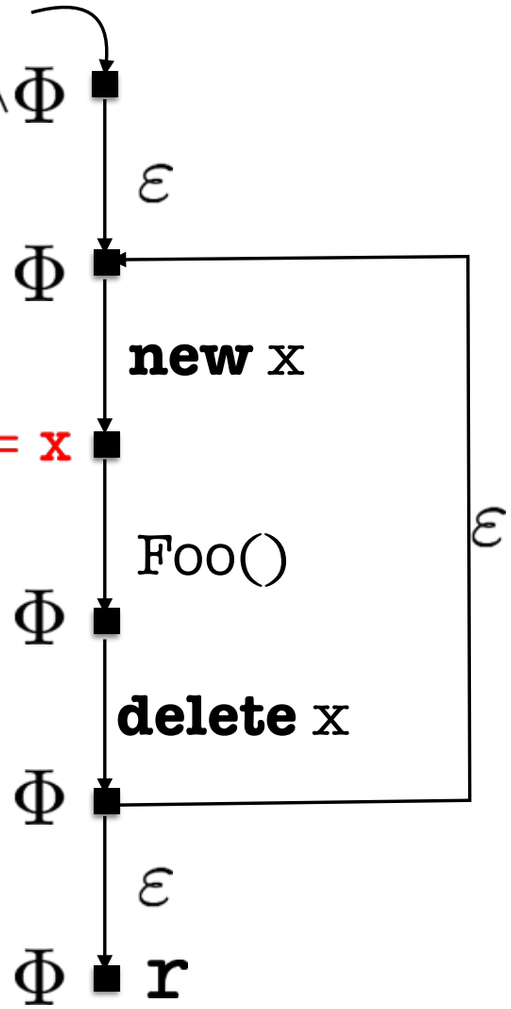
1. Model check the code of the variable components against their local specifications

Task II

1. Model extraction from stable code
2. Maximal model construction from local specification
3. Compose models and induce the behavior of the system
4. Model check the behavior

Composition of Flow Graphs





Φ is $\mathtt{x}' = \mathtt{x} \wedge \mathtt{y}' = \mathtt{y}$

Overview

User Tasks

1. Specification: Local specification (variable) & Global property
2. Define observed instructions

Task I

1. Model check the code of the variable components against their local specifications

Task II

1. Model extraction from stable code
2. Maximal model construction from local specification
3. Compose models and induce the behavior of the system
4. Model check the behavior

Overview

User Tasks

1. Specification: Local specification (variable) & Global property
2. Define observed instructions

Task I

1. Model check the code of the variable components against their local specifications (**Quick and Easy**)

Task II

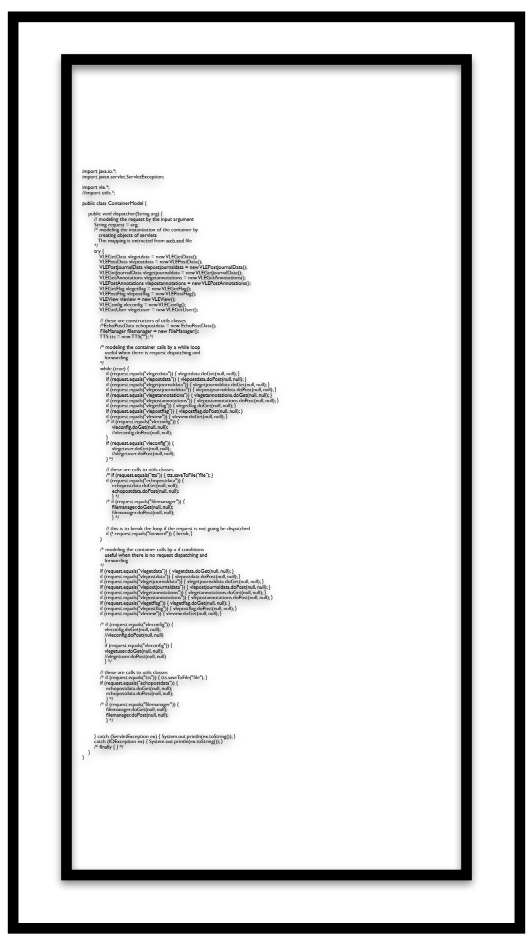
1. Model extraction from stable code
2. Maximal model construction from local specification
3. Compose models and induce the behavior of the system
4. Model check the behavior

A Case Study

Global Property

"only a single database connection should be created for each request and it should be properly closed"

J2EE Application

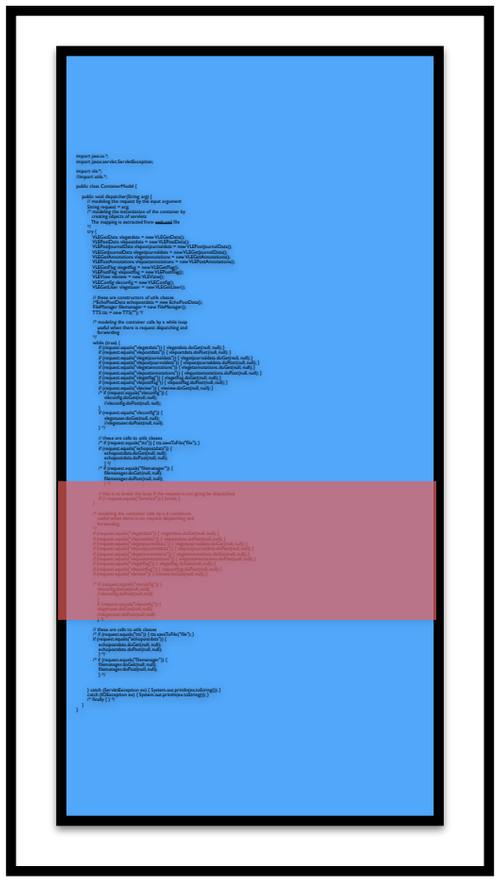


A Case Study

Global Property

"only a single database connection should be created for each request and it should be properly closed"

J2EE Application

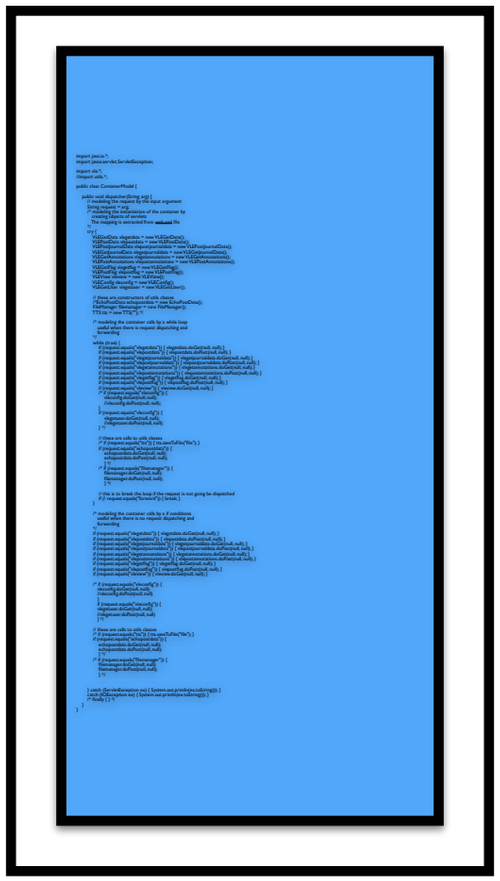


A Case Study

Global Property

"only a single database connection should be created for each request and it should be properly closed"

J2EE Application



```
import java.io.*;
import javax.servlet.ServletException;

import vlib.*;
//import vlib.*;

public class ContainerModel {

    public void dispatcher(String arg) {
        // modeling the request by vlib input argument
        String request = arg;
        // the following emulates of the container by
            creating objects of servlets
            The mapping is extracted from web.xml file
        try {
            VLEGenData vlegendata = new VLEGenData();
            VLEPostData vlepostdata = new VLEPostData();
            VLEPostJournalData vlepostjournaldata = new VLEPostJournalData();
            VLEPostJournalItemData vlepostjournalitemdata = new VLEPostJournalItemData();
            VLEGetAnnotations vlegetannotations = new VLEGetAnnotations();
            VLEPostAnnotations vlepostannotations = new VLEPostAnnotations();
            VLEGetReport vlegetreport = new VLEGetReport();
            VLEPostFlag vlepostflag = new VLEPostFlag();
            VLEView vlereview = new VLEView();
            VLECertify vlecertify = new VLECertify();
            VLEConfig vleconfig = new VLEConfig();

            // these are constructors of vdk classes
            FilePostData echopostdata = new FilePostData();
            FileManager filemanager = new FileManager();
            TTS tts = new TTS("","");

            /* modeling the container calls by a while loop
               useful when there is request dispatching and
               forwarding
            */
            while (true) {
              if (request.equals("vlegetdata")) { vlegetdata.doGet(null, null);
                if (request.equals("vlepostdata")) { vlepostdata.doPost(null, null);}
                if (request.equals("vlepostjournaldata")) { vlepostjournaldata.doPost(null, null);}
                if (request.equals("vlepostjournalitemdata")) { vlepostjournalitemdata.doPost(null, null);}
                if (request.equals("vlegetannotations")) { vlegetannotations.doGet(null, null);}
                if (request.equals("vlepostannotations")) { vlepostannotations.doPost(null, null);}
                if (request.equals("vlepostflag")) { vlepostflag.doPost(null, null);}
                if (request.equals("vlereview")) { vlereview.doGet(null, null);}
                if (request.equals("vlecertify")) {
                    vlecertify.doGet(null, null);
                    //vlecertify.doPost(null, null);
                }
                if (request.equals("vlecertify")) {
                    vlegetuser.doGet(null, null);
                    //vlegetuser.doPost(null, null);
                }

                // these are calls to vdk classes
                /* if (request.equals("tts")) { tts.nowToFile("file");}
                if (request.equals("echopostdata")) {
                    echopostdata.doGet(null, null);
                    echopostdata.doPost(null, null);
                }
                /* if (request.equals("filemanager")) {
                    filemanager.doGet(null, null);
                    filemanager.doPost(null, null);
                }*/

                // this is to break the loop if the request is not going be dispatched
                // if ( request.equals("forward")) { break; }
            }

            /* modeling the container calls by a if conditions
               useful when there is no request dispatching and
               forwarding
            */
            if (request.equals("vlegetdata")) { vlegetdata.doGet(null, null);}
            if (request.equals("vlepostdata")) { vlepostdata.doPost(null, null);}
            if (request.equals("vlepostjournaldata")) { vlepostjournaldata.doGet(null, null);}
            if (request.equals("vlepostjournalitemdata")) { vlepostjournalitemdata.doPost(null, null);}
            if (request.equals("vlegetannotations")) { vlegetannotations.doGet(null, null);}
            if (request.equals("vlepostannotations")) { vlepostannotations.doPost(null, null);}
            if (request.equals("vlegetflag")) { vlegetflag.doGet(null, null);}
            if (request.equals("vlepostflag")) { vlepostflag.doPost(null, null);}
            if (request.equals("vlereview")) { vlereview.doGet(null, null);}

            /* if (request.equals("vlecertify")) {
                vlecertify.doGet(null, null);
                //vlecertify.doPost(null, null);
                }
                if (request.equals("vlecertify")) {
                vlegetuser.doGet(null, null);
                //vlegetuser.doPost(null, null)
                } */

            // these are calls to vdk classes
            /* if (request.equals("tts")) { tts.nowToFile("file");}
            if (request.equals("echopostdata")) {
                echopostdata.doGet(null, null);
                echopostdata.doPost(null, null);
                }
            /* if (request.equals("filemanager")) {
                filemanager.doGet(null, null);
                filemanager.doPost(null, null);
                } */

        } catch (ServletException ex) { System.out.println(ex.toString()); }
          catch (IOException ex) { System.out.println(ex.toString()); }
          /* finally { } */
    }
}
```

A Case Study

Global Property

"only a single database connection should be created for each request and it should be properly closed"

Application	Lines of Code	Variable Part	Task I	Maximal Model Cons.	Global Model Check	Total Time
J2EE application	1087	297	0.5 sec	4.1 sec	2.1 sec	8.1 sec

Summary

Summary

- Generic framework for modular verification

Summary

- Generic framework for modular verification
- Modularity allows dealing with variability

Summary

- Generic framework for modular verification
- Modularity allows dealing with variability
- Modular model checking for pushdown infinite state sys.

Summary

- Generic framework for modular verification
- Modularity allows dealing with variability
- Modular model checking for pushdown infinite state sys.
  - Structural maximal models

Summary

- Generic framework for modular verification
- Modularity allows dealing with variability
- Modular model checking for pushdown infinite state sys.
  - Structural maximal models
  - Representing unobserved instructions through Hoare style assertions

Summary

- Generic framework for modular verification
- Modularity allows dealing with variability
- Modular model checking for pushdown infinite state sys.
  - Structural maximal models
  - Representing unobserved instructions through Hoare style assertions
- Instantiation of the framework

Summary

- Generic framework for modular verification
- Modularity allows dealing with variability
- Modular model checking for pushdown infinite state sys.
  - Structural maximal models
  - Representing unobserved instructions through Hoare style assertions
- Instantiation of the framework
  - Pointer programs

Summary

- Generic framework for modular verification
- Modularity allows dealing with variability
- Modular model checking for pushdown infinite state sys.
  - Structural maximal models
  - Representing unobserved instructions through Hoare style assertions
- Instantiation of the framework
  - Pointer programs
    - ✴ Tool support and a case study

Summary

- Generic framework for modular verification
- Modularity allows dealing with variability
- Modular model checking for pushdown infinite state sys.
  - Structural maximal models
  - Representing unobserved instructions through Hoare style assertions
- Instantiation of the framework
  - Pointer programs
    - ✷ Tool support and a case study

Future Work

- More instantiations of the framework, e.g., integers
- More tool support, e.g., for Boolean programs

Contributions

Development of the framework:

1. A framework for compositional verification with full data abstraction (existed before)

2. A generic framework for verification of procedural programs in the presence of variability
   - Three instantiations of the framework

Tool Support:

1. A set of stand alone tools, CVPP toolset (existed before)

2. A fully automated tool for procedure-modular verification of programs with full data abstraction
   ProMoVer

Contributions cont.

Verification of product families:

1. A hierarchical variable model for capturing commonality and variability of products

2. A extension of our verification technique for the efficient verification of product families represented by hierarchical variability models