“Non-intrusive” Testing Techniques for Communication Protocols

12^{th} TAROT Summer School, Paris, France

Jorge López
<jorge.lopez[at]telecom-sudparis.eu>
Télécom SudParis / Université Paris-Saclay

July 8^{th}, 2016

Outline

Acknowledgment...

The results presented in this talk were obtained along with different co-authors. Therefore, a special thank note is dedicated to those who directly contributed in this talk, they are:

- Ana Cavalli
- Natalia Kushik
- Stephane Maag
- Gerardo Morales
- Nina Yevtushenko

Спасибо! ¡Gracias! Merci !

I would also like to thank all the speakers of TAROT, including students. Your presentations were fantastic and will enrich this presentation.

Motivation

- The widespread adoption of applications using networks (i.e., communication protocols)

MOTIVATION

- The widespread adoption of applications using networks (i.e., communication protocols)
  - A second life (for some, their life)

MOTIVATION

- The widespread adoption of applications using networks (i.e., communication protocols)
  - A second life (for some, their life)
  - Critical Operations + Sensitive Information

WhatsApp

Motivation

- The widespread adoption of applications using networks (i.e., communication protocols)
  - A second life (for some, their life)
  - Critical Operations + Sensitive Information
  - Tell me the last application you used which does not interact over the network?

WhatsApp

Motivation

- The widespread adoption of applications using networks (i.e., communication protocols)
  - A second life (for some, their life)
  - Critical Operations + Sensitive Information
  - Tell me the last application you used which does not interact over the network?

WhatsApp

Important

Testing is crucial for such systems (or applications)!

"NON-INTRUSIVE" TESTING OF COMMUNICATION SYSTEMS

Testing of communication systems

"NON-INTRUSIVE" TESTING OF COMMUNICATION SYSTEMS

Testing of communication systems

- **Active testing** feels like an intuitive method for testing software and systems

"Non-Intrusive" Testing of Communication Systems

Testing of communication systems

- **Active testing** feels like an intuitive method for testing software and systems
  - Sometimes you can't interfere the system!

"NON-INTRUSIVE" TESTING OF COMMUNICATION SYSTEMS

Testing of communication systems

- **Active testing** feels like an intuitive method for testing software and systems
  - Sometimes you can't interfere the system!

Reasons not to interfere the system?

"NON-INTRUSIVE" TESTING OF COMMUNICATION SYSTEMS

Testing of communication systems

- **Active testing** feels like an intuitive method for testing software and systems
  - Sometimes you can't interfere the system!

Reasons not to interfere the system?

- The data on the system are susceptible towards the execution of tests

"NON-INTRUSIVE" TESTING OF COMMUNICATION SYSTEMS

Testing of communication systems

- **Active testing** feels like an intuitive method for testing software and systems
  - Sometimes you can't interfere the system!

Reasons not to interfere the system?

- The data on the system are susceptible towards the execution of tests
- Certain functionality is not available if "real" data are not processed

"NON-INTRUSIVE" TESTING OF COMMUNICATION SYSTEMS

Testing of communication systems

- **Active testing** feels like an intuitive method for testing software and systems
  - Sometimes you can't interfere the system!

Reasons not to interfere the system?

- The data on the system are susceptible towards the execution of tests
- Certain functionality is not available if "real" data are not processed
- Even if a system can be interrupted, we might *want* to test the "real" service / application / data

Methods for Non-Intrusive Testing

We focus on two of them. . .

Methods for Non-Intrusive Testing

We focus on two of them…

- Static (code) analysis

Methods for Non-Intrusive Testing

We focus on two of them...

- Static (code) analysis
  - A "white-box' approach

METHODS FOR NON-INTRUSIVE TESTING

We focus on two of them…

- Static (code) analysis
  - A “white-box’ approach
  - We assume we do not disrupt the implementation, we have an access to its source code

METHODS FOR NON-INTRUSIVE TESTING

We focus on two of them…

- Static (code) analysis
  - A "white-box' approach
  - We assume we do not disrupt the implementation, we have an access to its source code
  - Analyzing its code, we test the implementation

METHODS FOR NON-INTRUSIVE TESTING

We focus on two of them…

- Static (code) analysis
  - A “white-box’ approach
  - We assume we do not disrupt the implementation, we have an access to its source code
  - Analyzing its code, we test the implementation
- Passive network trace analysis

Methods for Non-Intrusive Testing

We focus on two of them...

- Static (code) analysis
  - A "white-box' approach
  - We assume we do not disrupt the implementation, we have an access to its source code
  - Analyzing its code, we test the implementation
- Passive network trace analysis
  - A "black-box' approach

METHODS FOR NON-INTRUSIVE TESTING

We focus on two of them…

- Static (code) analysis
  - A "white-box' approach
  - We assume we do not disrupt the implementation, we have an access to its source code
  - Analyzing its code, we test the implementation
- Passive network trace analysis
  - A "black-box' approach
  - We assume we do not disrupt the implementation, we have an access to the "messages" being exchanged by the implementation

METHODS FOR NON-INTRUSIVE TESTING

We focus on two of them…

- Static (code) analysis
  - A "white-box' approach
  - We assume we do not disrupt the implementation, we have an access to its source code
  - Analyzing its code, we test the implementation
- Passive network trace analysis
  - A "black-box' approach
  - We assume we do not disrupt the implementation, we have an access to the "messages" being exchanged by the implementation
  - The "source" where the messages are taken is called a Point of Observation (P.O.)

METHODS FOR NON-INTRUSIVE TESTING

We focus on two of them…

- Static (code) analysis
  - A “white-box’ approach
  - We assume we do not disrupt the implementation, we have an access to its source code
  - Analyzing its code, we test the implementation
- Passive network trace analysis
  - A “black-box’ approach
  - We assume we do not disrupt the implementation, we have an access to the “messages” being exchanged by the implementation
  - The “source” where the messages are taken is called a Point of Observation (P.O.)

Let’s take a look at each of them… “**on y va**”

Static Code Analysis 101

STATIC ANALYSIS

```
/*Test equal distribution of random number generation algorithm*/
#include <stdio.h>
#include <stdlib.h>
#define NUM 30
#define MEM_SIZE 512*1024 //512MB
int main()
{
        short i = 0 , j;
        long acc = 0;
        char *numbers = malloc(MEM_SIZE);
        if(!numbers)
        {
                printf(``Can't allocate memory\n'');
                exit(-1);
        }
        while (1)
        {
                numbers[i] = rand() % NUM ;  //random numbers from 0 - NUM
                acc = 0;
                for (j = 0; j < i; j++)
                        acc += numbers[j];
                printf(``New average: %ld\n'',  acc/++i); //should converge to NUM/2
        }
}
```

Do you see any problems with the code?

Static Analysis (cont.)

```
/*Test equal distribution of random number generation algorithm*/
#include <stdio.h>
#include <stdlib.h>
#define NUM 30
#define MEM_SIZE 512*1024 //512MB
int main()
{
        short i = 0 , j;
        long acc = 0;
        char *numbers = malloc(MEM_SIZE);
        if(!numbers)
        {
                printf(``Can't allocate memory\n'');
                exit(-1);
        }
        while (1)
        {
                numbers[i] = rand() % NUM ;  //random numbers from 0 - NUM
                acc = 0;
                for (j = 0; j < i; j++)
                        acc += numbers[j];
                printf(``New average: %ld\n'',  acc/++i); //should converge to NUM/2
        }
}
```

Hard to see, hard to detect (iteration # 32,768)

Static (Code) Analysis

We can look for *anything*…

Static (Code) Analysis

We can look for *anything*…

- Which is not *functional* related (the code compiles)

Static (Code) Analysis

We can look for *anything*…

- Which is not *functional* related (the code compiles)
- Usually we check properties

Static (Code) Analysis

We can look for *anything*...

- Which is not *functional* related (the code compiles)
- Usually we check properties
  - For instance, for the simple grammar $P \mapsto P\ P|(P)|()$, ensure that on the even parenthesis list elements, the depth level is not more than 2
    $(())()()((()))$ violates the property, but $((()))()$ does not (i.e., is valid)

STATIC (CODE) ANALYSIS

We can look for *anything*…

- Which is not *functional* related (the code compiles)
- Usually we check properties
  - For instance, for the simple grammar $P \mapsto P\ P|(P)|()$, ensure that on the even parenthesis list elements, the depth level is not more than 2
    $(())()()((()))$ violates the property, but $((((()))))()$ does not (i.e., is valid)
  - Ensure that starting from the N-th element of the list at least m levels are found
    For $N = 2, m = 2$, $()()()$ violates the property, and $()(())((()))$ does not

Static (Code) Analysis

We can look for *anything*…

- Which is not *functional* related (the code compiles)
- Usually we check properties
  - For instance, for the simple grammar $P \mapsto P\ P|(P)|()$, ensure that on the even parenthesis list elements, the depth level is not more than 2
    $(())()()((()))$ violates the property, but $(((())))()$ does not (i.e., is valid)
  - Ensure that starting from the N-th element of the list at least m levels are found
    For $N = 2, m = 2$, $()()()$ violates the property, and $()(())((()))$ does not
  - For a more complex grammar there is much more fun :)

Static (Code) Analysis

We can look for *anything*…

- Which is not *functional* related (the code compiles)
- Usually we check properties
  - For instance, for the simple grammar $P \mapsto P\ P|(P)|()$, ensure that on the even parenthesis list elements, the depth level is not more than 2
    $(())()()((()))$ violates the property, but $((()))()$ does not (i.e., is valid)
  - Ensure that starting from the N-th element of the list at least m levels are found
    For $N = 2, m = 2$, $()()()$ violates the property, and $()(())((()))$ does not
  - For a more complex grammar there is much more fun :)
- Let's look at more real-world examples…

Static Analysis for security properties

Source Code Security Issues

Static Analysis for security properties

Source Code Security Issues

- Many problems arise when the user input is not checked

Static Analysis for security properties

Source Code Security Issues

- Many problems arise when the user input is not checked
  - Cross-Site Scripting (XSS) attacks

Static Analysis for security properties

Source Code Security Issues

- Many problems arise when the user input is not checked
  - Cross-Site Scripting (XSS) attacks
  - SQL injection (SQLI) attacks

Static Analysis for security properties

Source Code Security Issues

- Many problems arise when the user input is not checked
  - Cross-Site Scripting (XSS) attacks
  - SQL injection (SQLI) attacks
  - Buffer overflow attacks

Static Analysis for security properties

Source Code Security Issues

- Many problems arise when the user input is not checked
  - Cross-Site Scripting (XSS) attacks
  - SQL injection (SQLI) attacks
  - Buffer overflow attacks

XSS

Static Analysis for security properties

Source Code Security Issues

- Many problems arise when the user input is not checked
  - Cross-Site Scripting (XSS) attacks
  - SQL injection (SQLI) attacks
  - Buffer overflow attacks

XSS

- Works by supplying data to users which can lead to insecure actions, executing unwanted javascript, or adding a sub-site filled with publicity or others

Static Analysis for security properties

Source Code Security Issues

- Many problems arise when the user input is not checked
  - Cross-Site Scripting (XSS) attacks
  - SQL injection (SQLI) attacks
  - Buffer overflow attacks

XSS

- Works by supplying data to users which can lead to insecure actions, executing unwanted javascript, or adding a sub-site filled with publicity or others
- Simple example: in a forum, users are allowed to insert comments. If the comment is displayed as-it-is, an attacker might successfully inject malicious code that will affect the forum users

Static Analysis for security properties II

SQLI

Static Analysis for security properties II

SQLI

- Essentially works by supplying data to the database for executing undesired actions, e.g., a user inputs a search criterion, and the database looks for the users matching this criterion:

  ```
  SELECT * from users where name='$CRIT';
  ```

  What if the criterion is:

  ```
  a'; DROP TABLE users
  ```

STATIC ANALYSIS FOR SECURITY PROPERTIES II

SQLI

- Essentially works by supplying data to the database for executing undesired actions, e.g., a user inputs a search criterion, and the database looks for the users matching this criterion:

  ```
  SELECT * from users where name='$CRIT';
  ```

  What if the criterion is:

  ```
  a'; DROP TABLE users
  ```

How to prevent this attack using static analysis?

STATIC ANALYSIS FOR SECURITY PROPERTIES II

SQLI

- Essentially works by supplying data to the database for executing undesired actions, e.g., a user inputs a search criterion, and the database looks for the users matching this criterion:

  `SELECT * from users where name='$CRIT';`

  What if the criterion is:

  `a'; DROP TABLE users`

How to prevent this attack using static analysis?

- A very simple approach is to guarantee that a **sanitization** function is called before the storing or displaying the input. Many languages provide such built-in functions, e.g., PHP provides the htmlspecialchars() function

Static Analysis for security properties III

Buffer Overflow

STATIC ANALYSIS FOR SECURITY PROPERTIES III

Buffer Overflow

- The canonical example:

```
#include <string.h>
#define BUFFSIZE 100
void load (char *userdata){
   char  buff[BUFFSIZE];
   strcpy(buff, userdata);  //not good
}

int main (int argc, char **argv){
   load(argv[1]);
   ...
}
```

STATIC ANALYSIS FOR SECURITY PROPERTIES III

Buffer Overflow

- The canonical example:

```
#include <string.h>
#define BUFFSIZE 100
void load (char *userdata){
   char  buff[BUFFSIZE];
   strcpy(buff, userdata);  //not good
}

int main (int argc, char **argv){
   load(argv[1]);
   ...
}
```

- A string which is longer than BUFFSIZE will be written into the memory space of the function load, potentially overwriting the return address

STATIC ANALYSIS FOR SECURITY PROPERTIES III

Buffer Overflow

- The canonical example:

```
#include <string.h>
#define BUFFSIZE 100
void load (char *userdata){
   char  buff[BUFFSIZE];
   strcpy(buff, userdata);  //not good
}

int main (int argc, char **argv){
   load(argv[1]);
   ...
}
```

- A string which is longer than BUFFSIZE will be written into the memory space of the function load, potentially overwriting the return address
- A string which contains code and the memory address of this code in the position of the return address will do the trick

Static Analysis for security properties IV

Buffer Overflow

Static Analysis for security properties IV

Buffer Overflow

- If a program with a potentially susceptible stack overflow runs with admin privileges, consequences can be devastating

Static Analysis for security properties IV

Buffer Overflow

- If a program with a potentially susceptible stack overflow runs with admin privileges, consequences can be devastating

Many other properties

Static Analysis for security properties IV

Buffer Overflow

- If a program with a potentially susceptible stack overflow runs with admin privileges, consequences can be devastating

Many other properties

- Each file should be closed only once

Static Analysis for security properties IV

Buffer Overflow

- If a program with a potentially susceptible stack overflow runs with admin privileges, consequences can be devastating

Many other properties

- Each file should be closed only once
- Each db connection that was open should be closed

STATIC ANALYSIS FOR SECURITY PROPERTIES IV

Buffer Overflow

- If a program with a potentially susceptible stack overflow runs with admin privileges, consequences can be devastating

Many other properties

- Each file should be closed only once
- Each db connection that was open should be closed
- Any other properties are welcome

Static Analysis for security properties IV

Buffer Overflow

- If a program with a potentially susceptible stack overflow runs with admin privileges, consequences can be devastating

Many other properties

- Each file should be closed only once
- Each db connection that was open should be closed
- Any other properties are welcome

How to perform static analysis (SA)?

Static Analysis Principles

Source Code is?

STATIC ANALYSIS PRINCIPLES

Source Code is?

- A text representation of “computer instructions” in a specific **programming language**

STATIC ANALYSIS PRINCIPLES

Source Code is?

- A text representation of “computer instructions” in a specific **programming language**
- Not text…Nor machine code

STATIC ANALYSIS PRINCIPLES

Source Code is?

- A text representation of "computer instructions" in a specific **programming language**
- Not text. . . Nor machine code
- A description of a system

Static Analysis Principles

Source Code is?

- A text representation of "computer instructions" in a specific **programming language**
- Not text...Nor machine code
- A description of a system

Static Analysis is?

STATIC ANALYSIS PRINCIPLES

Source Code is?

- A text representation of “computer instructions” in a specific **programming language**
- Not text…Nor machine code
- A description of a system

Static Analysis is?

- The analysis of source code is performed by a static analyzer without executing the program under test

Static Analysis Principles

Source Code is?

- A text representation of "computer instructions" in a specific **programming language**
- Not text. . . Nor machine code
- A description of a system

Static Analysis is?

- The analysis of source code is performed by a static analyzer without executing the program under test
- The code is assumed to be compilable, thus we do not look for lexical, syntactical, or *type* errors that a compiler finds

Static Analysis Principles (cont.)

How do we analyze the source code then?

Static Analysis Principles (cont.)

How do we analyze the source code then?

- Can we treat it as text?

Static Analysis Principles (cont.)

How do we analyze the source code then?

- Can we treat it as text?
  - This would be terribly^n hard

Static Analysis Principles (cont.)

How do we analyze the source code then?

- Can we treat it as text?
  - This would be terribly^n hard
- A data structure is needed to effectively manipulate it

Static Analysis Principles (cont.)

How do we analyze the source code then?

- Can we treat it as text?
  - This would be terribly^n hard
- A data structure is needed to effectively manipulate it
  - A famous structure to manipulate the source code is an **Abstract Syntax Tree (AST)**

STATIC ANALYSIS PRINCIPLES (CONT.)

How do we analyze the source code then?

- Can we treat it as text?
  - This would be terribly^n hard
- A data structure is needed to effectively manipulate it
  - A famous structure to manipulate the source code is an **Abstract Syntax Tree (AST)**
  - To build an AST, we need to parse the code

Static Analysis Principles (cont.)

How do we analyze the source code then?

- Can we treat it as text?
  - This would be terribly^n hard
- A data structure is needed to effectively manipulate it
  - A famous structure to manipulate the source code is an **Abstract Syntax Tree (AST)**
  - To build an AST, we need to parse the code

A parser…

Static Analysis Principles (cont.)

How do we analyze the source code then?

- Can we treat it as text?
  - This would be terribly^n hard
- A data structure is needed to effectively manipulate it
  - A famous structure to manipulate the source code is an **Abstract Syntax Tree (AST)**
  - To build an AST, we need to parse the code

A parser…

- Can be generated automatically by a parser generator (takes a context free grammar (CFG) as an input and produces a parser)

Static Analysis Principles (cont.)

How do we analyze the source code then?

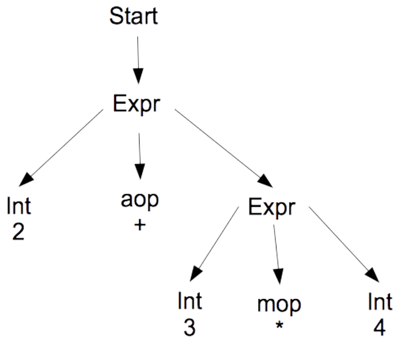
- Can we treat it as text?
  - This would be terribly^n hard
- A data structure is needed to effectively manipulate it
  - A famous structure to manipulate the source code is an **Abstract Syntax Tree (AST)**
  - To build an AST, we need to parse the code

A parser…

- Can be generated automatically by a parser generator (takes a context free grammar (CFG) as an input and produces a parser)
- Takes a CFG production (a sentence / source code) and produces a "parse tree"

An AST

AST for $2 + 3 * 4$



Dataflow Analysis

Mainly cares about data flow and data dependencies

Consider the following code:

Dataflow Analysis

Mainly cares about data flow and data dependencies

Consider the following code:

```
void func (int x){
   int y = 10;
   int z = 2 + y;

   if(x > 10){
      z=10;
      x = y + 1;
   }
   print(z);
}
```

Dataflow Analysis

Mainly cares about data flow and data dependencies

Consider the following code:

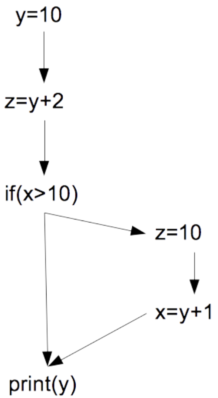
```
void func (int x){
   int y = 10;
   int z = 2 + y;

   if(x > 10){
      z=10;
      x = y + 1;
   }
   print(z);
}
```

How can Dataflow analysis help us?

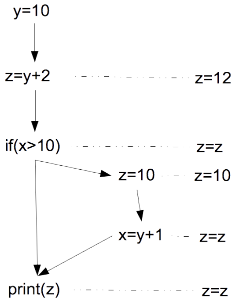
Dataflow Analysis — Forward Analysis

The data flow



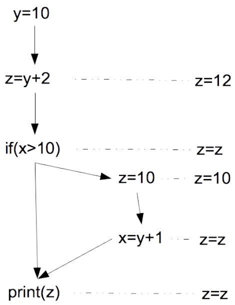
Dataflow Analysis — Forward Analysis

The data flow



Dataflow Analysis — Forward Analysis

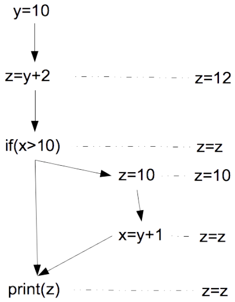
The data flow



Potential values

Dataflow Analysis — Forward Analysis

The data flow

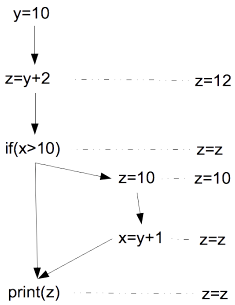


Potential values

- Negative array indices

Dataflow Analysis — Forward Analysis

The data flow

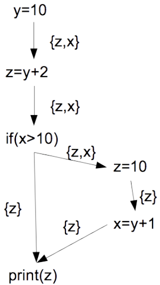


Potential values

- Negative array indices
- Closed DB connections

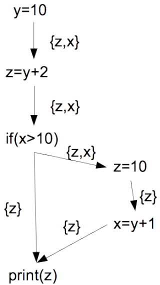
Dataflow Analysis — Backward Analysis

The data flow



Dataflow Analysis — Backward Analysis

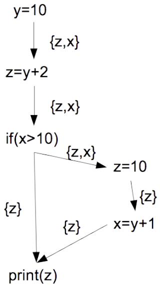
The data flow



Data dependencies

Dataflow Analysis — Backward Analysis

The data flow

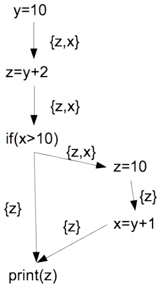


Data dependencies

- Useful for security testing

Dataflow Analysis — Backward Analysis

The data flow



Data dependencies

- Useful for security testing
- Useful for dead code elimination

Dataflow Analysis — Control Flow Graph

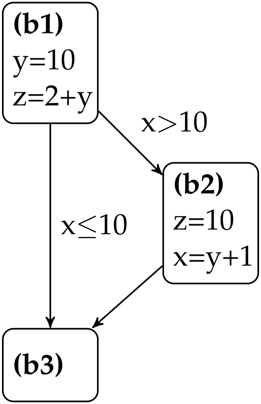
```
void func (int x){
    //b1
    int y = 10;
    int z = 2 + y;

    if(x > 10){
        //b2
        z=10;
        x = y + 1;
    }
    //b3
    print(z);
}
```

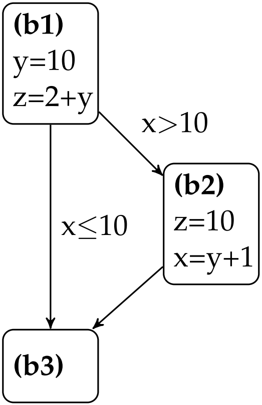
Dataflow Analysis — Control Flow Graph

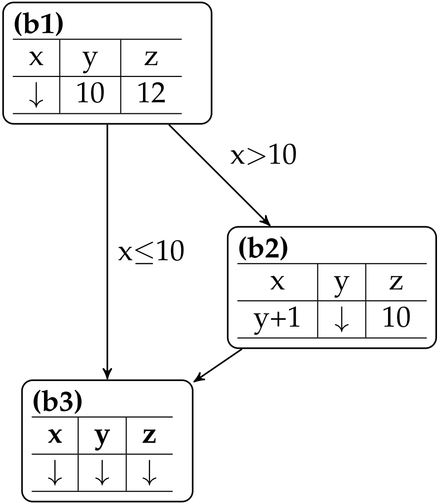
```
void func (int x){
    //b1
    int y = 10;
    int z = 2 + y;

    if(x > 10){
        //b2
        z=10;
        x = y + 1;
    }
    //b3
    print(z);
}
```

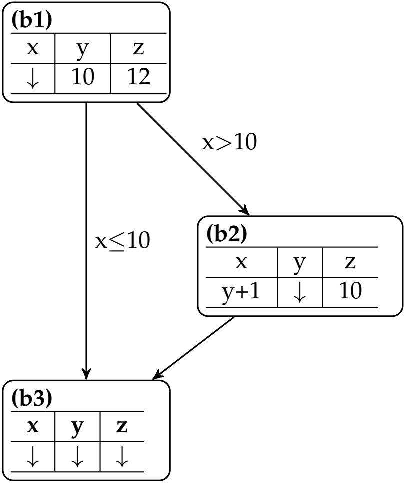


Dataflow notions



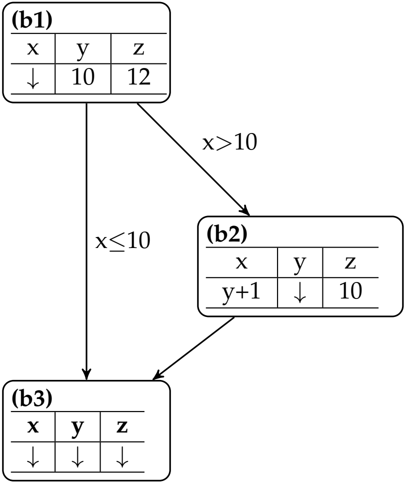


Dataflow notions (2)



Putting inputs and outputs…

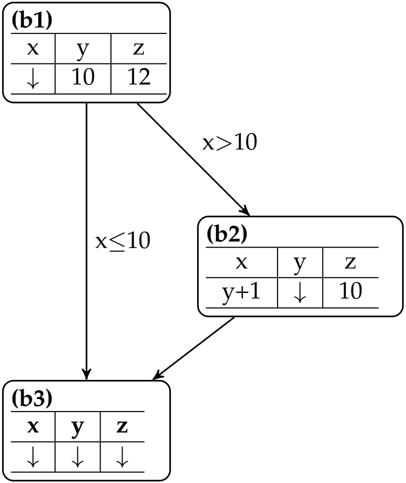
Dataflow notions (2)



Putting inputs and outputs…

- Assume ⊥ = not enough information

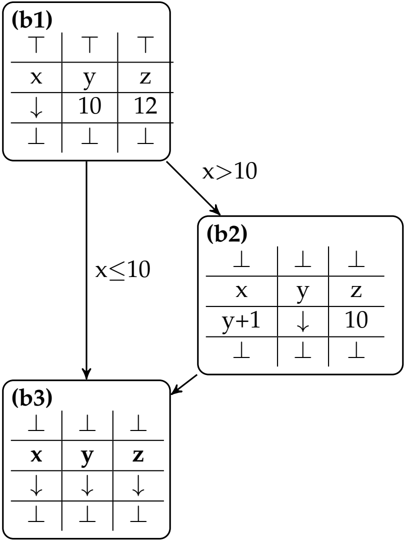
Dataflow notions (2)



Putting inputs and outputs...

- Assume ⊥ = not enough information
- Assume ⊤ = "too much" information (all possible values)

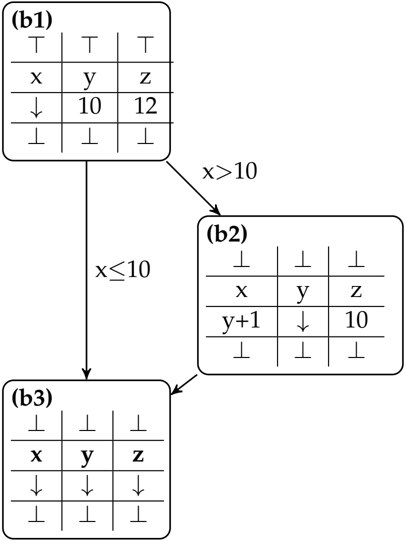
Dataflow notions (2)



Putting inputs and outputs...

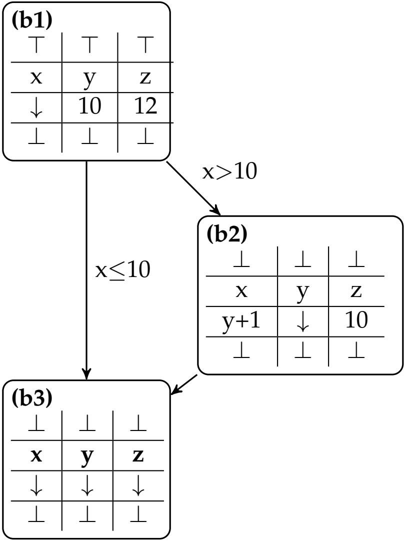
- Assume $\bot$ = not enough information
- Assume $\top$ = "too much" information (all possible values)

Dataflow Notions — The propagation game



Propagation…

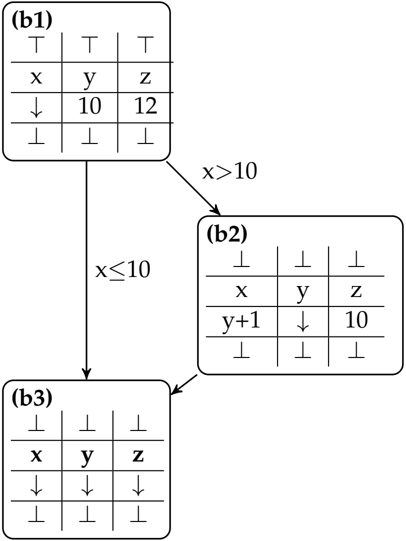
Dataflow Notions — The propagation game



Propagation…

- Propagate from inputs to outputs

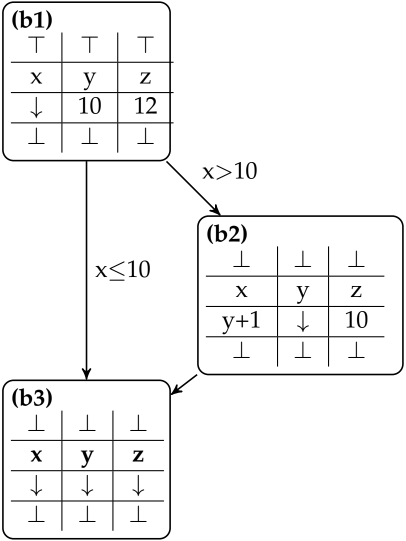
Dataflow Notions — The propagation game



Propagation…

- Propagate from inputs to outputs
- From one block to another

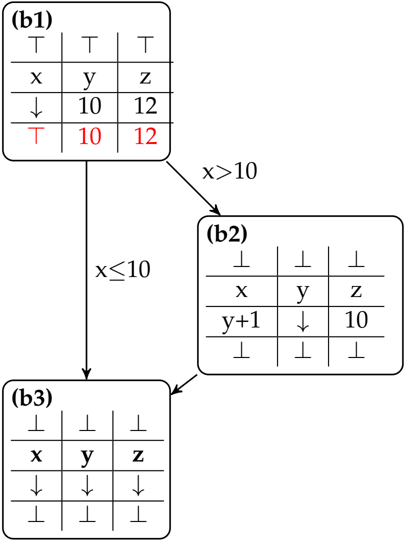
Dataflow Notions — The propagation game



Propagation…

- Propagate from inputs to outputs
- From one block to another
- *Join* the inputs

Dataflow Notions — The propagation game

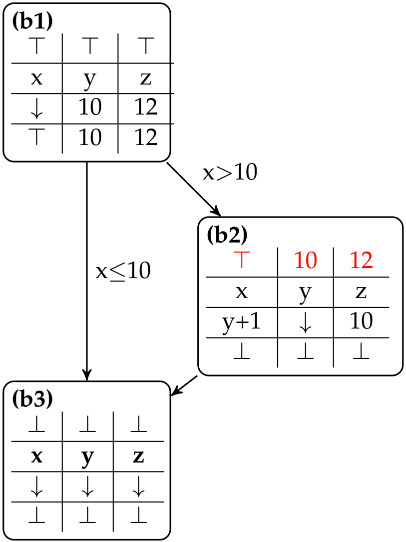


Propagation…

- Propagate from inputs to outputs
- From one block to another
- *Join* the inputs

(b1)

DATAFLOW NOTIONS — THE PROPAGATION GAME

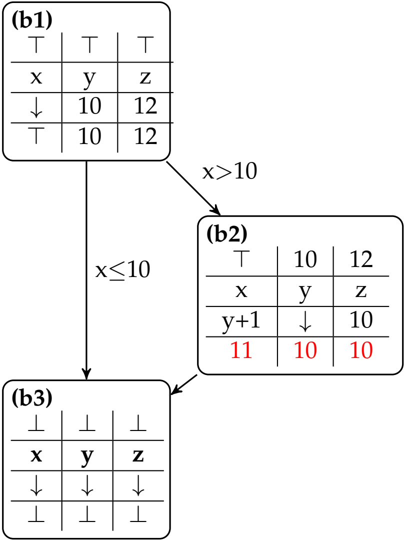


Propagation…

- Propagate from inputs to outputs
- From one block to another
- *Join* the inputs

(b2)

Dataflow Notions — The propagation game

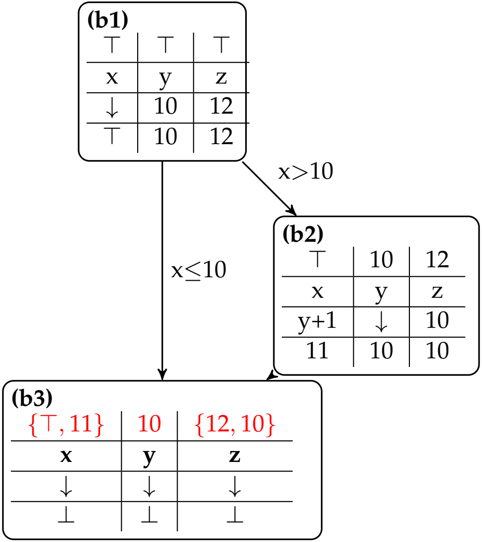


Propagation…

- Propagate from inputs to outputs
- From one block to another
- *Join* the inputs

(b2)

DATAFLOW NOTIONS — THE PROPAGATION GAME

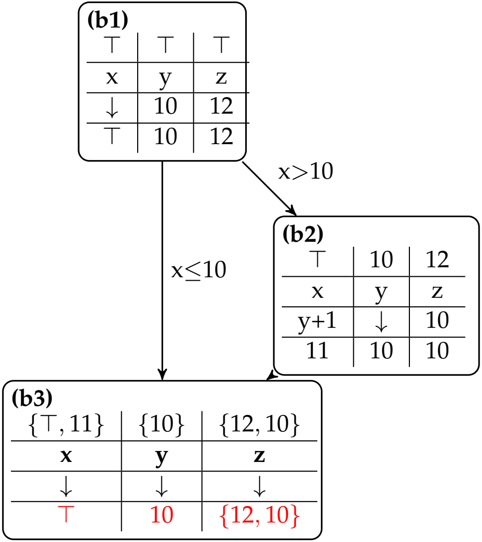


Propagation…

- Propagate from inputs to outputs
- From one block to another
- *Join* the inputs

(b3)

Dataflow Notions — The propagation game

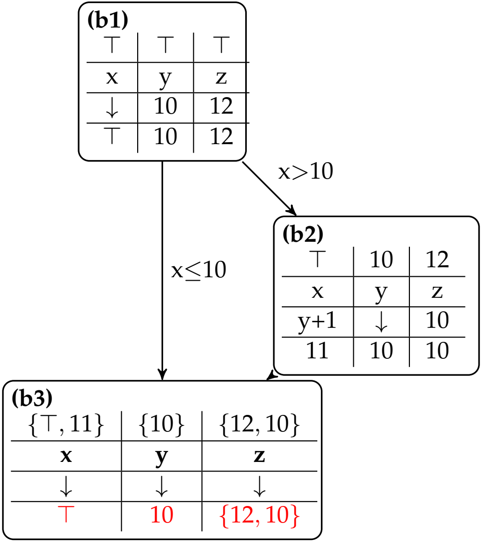


Propagation. . .

- Propagate from inputs to outputs
- From one block to another
- *Join* the inputs

(b3)

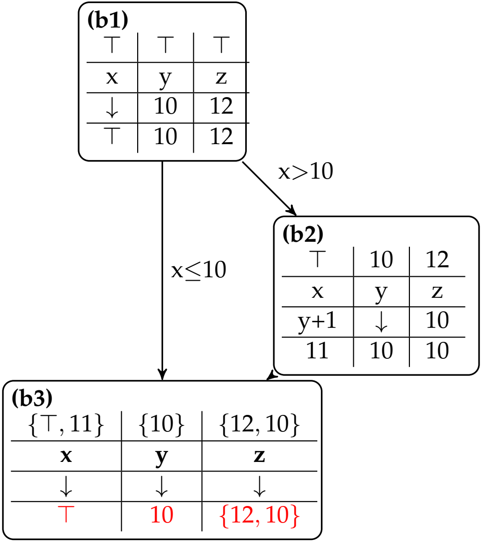
DATAFLOW NOTIONS — THE PROPAGATION GAME



Value Analysis

Yup... that's what we did

Dataflow Notions — The propagation game

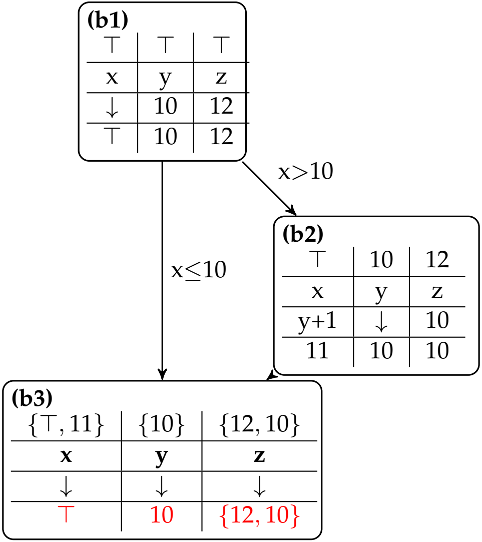


Value Analysis

Yup... that's what we did

- y=10 is truth for all execution paths
  Optimizations: constant propagation (less calculations), removal of y (smaller stack consumption)

Dataflow Notions — The propagation game



Value Analysis

Yup... that's what we did

- y=10 is truth for all execution paths
  Optimizations: constant propagation (less calculations), removal of y (smaller stack consumption)
- $z \in \{10, 12\}$
  Optimizations: smaller data type of z; perhaps this might be further used to verify that all functions return values between $[11 - 20]$?

Dataflow Analysis Basics

Lattice $\mathcal{L}$ (The analysis domain)

Dataflow Analysis Basics

Lattice $\mathcal{L}$ (The analysis domain)

- A partially ordered set $(\mathcal{L}, \leq)$

Dataflow Analysis Basics

Lattice $\mathcal{L}$ (The analysis domain)

- A partially ordered set $(\mathcal{L}, \leq)$
- $\forall x, y \in \mathcal{L}, \exists x \vee y$(sup) & $\exists x \wedge y$(inf)

DATAFLOW ANALYSIS BASICS

Lattice $\mathcal{L}$ (The analysis domain)

- A partially ordered set $(\mathcal{L}, \leq)$
- $\forall x, y \in \mathcal{L}, \exists x \vee y(\text{sup})$ & $\exists x \wedge y(\text{inf})$
- If $\forall \mathcal{S} \subseteq \mathcal{L}, \exists \vee \mathcal{S}(\textit{greatest element } \top)$ & $\wedge \mathcal{S}(\textit{least element } \bot)$ $\mathcal{L}$ is a complete lattice
  - $x_0 \leq x_1 \leq x_2 \ldots \implies \exists n : x_n = x_{n+1} = x_{n+2} = \ldots$
    Ensures ascending chain condition (no infinite progress)

DATAFLOW ANALYSIS BASICS

Lattice $\mathcal{L}$ (The analysis domain)

- A partially ordered set $(\mathcal{L}, \leq)$
- $\forall x, y \in \mathcal{L}, \exists x \vee y$(sup) & $\exists x \wedge y$(inf)
- If $\forall \mathcal{S} \subseteq \mathcal{L}, \exists \vee \mathcal{S}$(*greatest element* $\top$) & $\wedge \mathcal{S}$(*least element* $\perp$) $\mathcal{L}$ is a complete lattice
  - $x_0 \leq x_1 \leq x_2 \ldots \implies \exists n : x_n = x_{n+1} = x_{n+2} = \ldots$
    Ensures ascending chain condition (no infinite progress)

Inputs, outputs, and transfer functions, depend on the control flow block (CFB) b

DATAFLOW ANALYSIS BASICS

Lattice $\mathcal{L}$ (The analysis domain)

- A partially ordered set $(\mathcal{L}, \leq)$
- $\forall x, y \in \mathcal{L}, \exists x \vee y$(sup) & $\exists x \wedge y$(inf)
- If $\forall \mathcal{S} \subseteq \mathcal{L}, \exists \vee \mathcal{S}$(*greatest element* $\top$) & $\wedge \mathcal{S}$(*least element* $\bot$) $\mathcal{L}$ is a complete lattice
  - $x_0 \leq x_1 \leq x_2 \ldots \implies \exists n : x_n = x_{n+1} = x_{n+2} = \ldots$
    Ensures ascending chain condition (no infinite progress)

Inputs, outputs, and transfer functions, depend on the control flow block (CFB) b

- Transfer function $f_b : \mathcal{L} \to \mathcal{L}$

Dataflow Analysis Basics

Lattice $\mathcal{L}$ (The analysis domain)

- A partially ordered set $(\mathcal{L}, \leq)$
- $\forall x, y \in \mathcal{L}, \exists x \vee y(\text{sup})$ & $\exists x \wedge y(\text{inf})$
- If $\forall \mathcal{S} \subseteq \mathcal{L}, \exists \vee \mathcal{S}(\textit{greatest element } \top)$ & $\wedge \mathcal{S}(\textit{least element } \bot)$ $\mathcal{L}$ is a complete lattice
  - $x_0 \leq x_1 \leq x_2 ... \implies \exists n : x_n = x_{n+1} = x_{n+2} = ...$ Ensures ascending chain condition (no infinite progress)

Inputs, outputs, and transfer functions, depend on the control flow block (CFB) b

- Transfer function $f_b : \mathcal{L} \to \mathcal{L}$
- Output $out_b = f_b(in_n)$ is calculated with a transfer function

DATAFLOW ANALYSIS BASICS

Lattice $\mathcal{L}$ (The analysis domain)

- A partially ordered set $(\mathcal{L}, \leq)$
- $\forall x, y \in \mathcal{L}, \exists x \vee y$(sup) & $\exists x \wedge y$(inf)
- If $\forall \mathcal{S} \subseteq \mathcal{L}, \exists \vee \mathcal{S}(\textit{greatest element } \top)$ & $\wedge \mathcal{S}(\textit{least element } \bot)$ $\mathcal{L}$ is a complete lattice
  - $x_0 \leq x_1 \leq x_2 ... \implies \exists n : x_n = x_{n+1} = x_{n+2} = ...$
    Ensures ascending chain condition (no infinite progress)

Inputs, outputs, and transfer functions, depend on the control flow block (CFB) b

- Transfer function $f_b : \mathcal{L} \rightarrow \mathcal{L}$
- Output $out_b = f_b(in_n)$ is calculated with a transfer function
- in_b = denotes inputs of b, $in_b = \vee\{out_m | m \in pred(b)\}$ (usually $\vee = \cup$)

Dataflow Analysis Basics (cont.)

Requirements

Dataflow Analysis Basics (cont.)

Requirements

- The *join* operation $\vee$ must not lose information: $\vee(x, y) \supseteq x$ and $\vee(x, y) \supseteq y$

Dataflow Analysis Basics (cont.)

Requirements

- The *join* operation $\vee$ must not lose information: $\vee(x, y) \supseteq x$ and $\vee(x, y) \supseteq y$
- $\forall f \in F, x \subseteq y \implies f(x) \subseteq f(y)$, all transfer functions are monotones, i.e., they preserve the given order

Dataflow Analysis Basics (cont.)

Requirements

- The *join* operation $\vee$ must not lose information: $\vee(x, y) \supseteq x$ and $\vee(x, y) \supseteq y$
- $\forall f \in F, x \subseteq y \implies f(x) \subseteq f(y)$, all transfer functions are monotones, i.e., they preserve the given order

Necessary conditions for termination!

- Previous requirements + ascending chain condition

DATAFLOW ANALYSIS BASICS (CONT.)

Requirements

- The *join* operation $\vee$ must not lose information: $\vee(x, y) \supseteq x$ and $\vee(x, y) \supseteq y$
- $\forall f \in F, x \subseteq y \implies f(x) \subseteq f(y)$, all transfer functions are monotones, i.e., they preserve the given order

Necessary conditions for termination!

- Previous requirements + ascending chain condition

We won't see proofs

- However, this is certainly proven and easy to find…

Data Analysis Basics — Calculation Algorithm

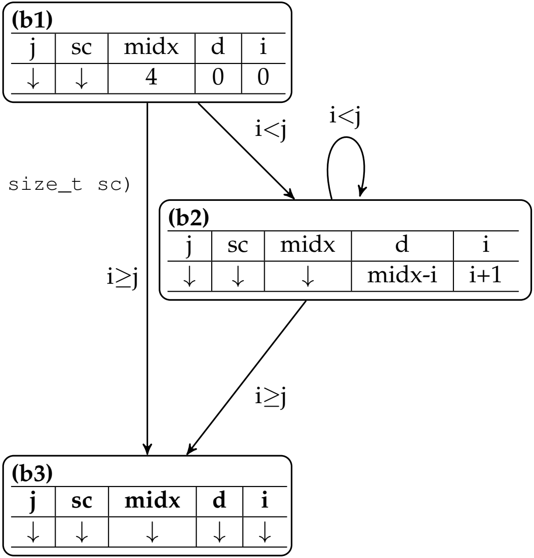
Data Analysis Basics — Calculation Algorithm

Maximal Fixed Point algorithm

for all $b \in B$ **do**
    $out_b \leftarrow f_b(\perp)$
end for
$in_{b0} \leftarrow \mathcal{I}$ //$\mathcal{I}$=initialization ($\top . \perp, \emptyset$ are usual)
$out_{b0} \leftarrow f_{n0}(I)$
worklist$\leftarrow B \setminus \{b0\}$
while worklist $\neq \emptyset$ **do**
    $b \leftarrow$ pop(worklist)
    $in_b \leftarrow \vee\{out_m | m \in pred(b)\}$
    $out_b \leftarrow f_b(in_b)$
    if out_b changed **then**
        worklist $\leftarrow$ worklist $\cup$ b
    end if
end while

Dataflow Analysis — Sign Analysis Example

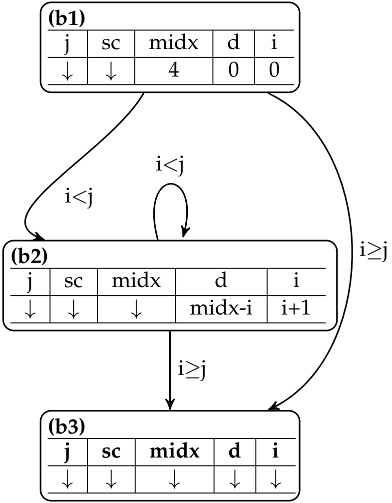
```
void bk (int j, char c[], size_t sc)
{
   size_t midx = 4;
   int d = 0,i =0;
   while(i< j)
   {
       d = midx - i;
      c[d] = i;
      i++;
   }
}
```



Dataflow Analysis — Sign Analysis Example (2)

Lattice $\mathcal{L}$ is defined over the set of all subsets of $\{-, 0, +\}$ and $\subseteq$

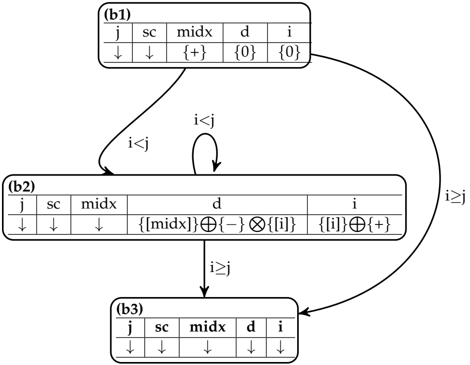
- $\vee = \cup$
  $\top \in \{+, -\}$
  $\bot =$ no information yet
- Operations over $\mathcal{L}$
  - Addition($\bigoplus$) and multiplication ($\bigotimes$) of lattice values:
    $\{-\} \bigoplus \{+\} =$
    $\{-, 0, +\} \wedge \{0\} \bigoplus \{+\} =$
    $\{+\} \wedge \{-\} \bigoplus \{-\} \wedge \ldots$
- $f_b \in F$ use operations + sign of constants



Dataflow Analysis — Sign Analysis Example (2)

Lattice $\mathcal{L}$ is defined over the set of all subsets of $\{-, 0, +\}$ and $\subseteq$

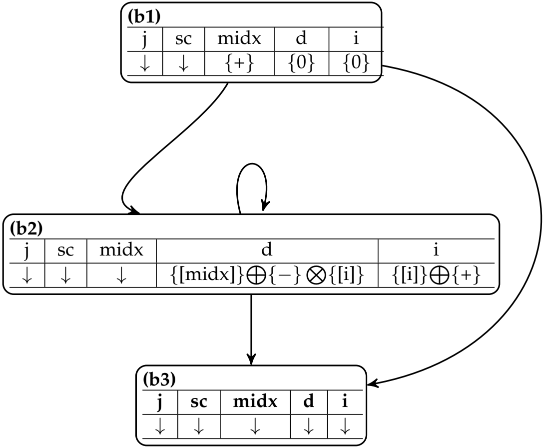
- $\vee = \cup$
  $\top \in \{+, -\}$
  $\bot =$ no information yet
- Operations over $\mathcal{L}$
  - Addition($\bigoplus$) and multiplication ($\bigotimes$) of lattice values:
    $\{-\} \bigoplus \{+\} = \{-, 0, +\} \wedge \{0\} \bigoplus \{+\} = \{+\} \wedge \{-\} \bigoplus \{-\} \wedge \ldots$
- $f_b \in F$ use operations + sign of constants



Dataflow Analysis — Sign Analysis Example (3)

for all $b \in B$ **do**
  $out_b \leftarrow f_b(\perp)$
end for
$in_{b0} \leftarrow \mathcal{I}$ //$\mathcal{I}$=initialization
($\top.\perp, \emptyset$ are usual)
$out_{b0} \leftarrow f_{n0}(I)$
worklist$\leftarrow B \setminus \{b0\}$
while worklist $\neq \emptyset$ **do**
  $b \leftarrow$ pop(worklist)

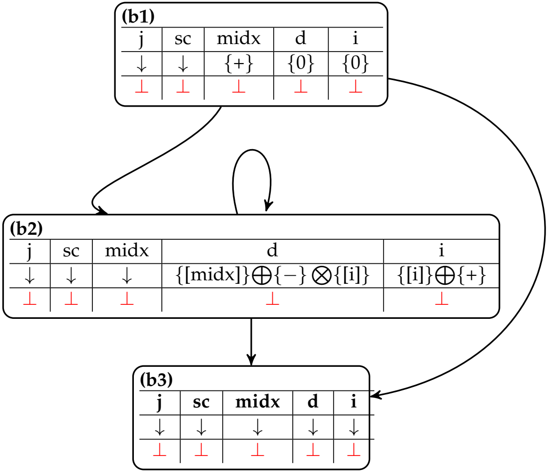
$in_b \leftarrow \vee\{out_m | m \in pred(b)\}$
  $out_b \leftarrow f_b(in_b)$
  if out_b changed **then**
    worklist $\leftarrow$ worklist
$\cup$ b
  end if
end while



Dataflow Analysis — Sign Analysis Example (3)

for all $b \in B$ **do**
  $out_b \leftarrow f_b(\bot)$
end for
$in_{b0} \leftarrow \mathcal{I}$ //$\mathcal{I}$=initialization
($\top.\bot, \emptyset$ are usual)
$out_{b0} \leftarrow f_{n0}(I)$
worklist$\leftarrow B \setminus \{b0\}$
while worklist $\neq \emptyset$ **do**
  $b \leftarrow$ pop(worklist)

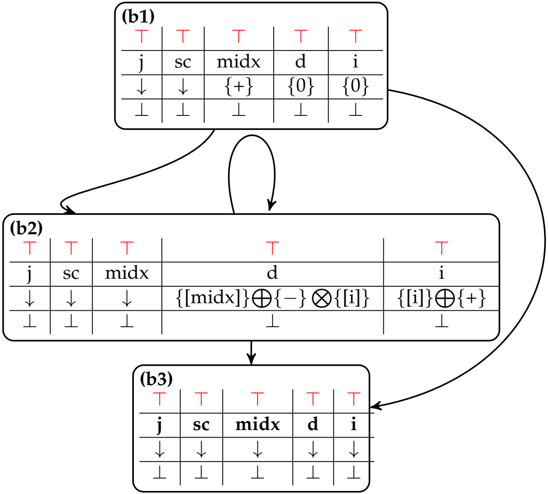
$in_b \leftarrow \vee\{out_m | m \in pred(b)\}$
  $out_b \leftarrow f_b(in_b)$
  if out_b changed **then**
    worklist $\leftarrow$ worklist
$\cup$ b
  end if
end while



Dataflow Analysis — Sign Analysis Example (3)

for all $b \in B$ **do**
  $out_b \leftarrow f_b(\bot)$
end for
$in_{b0} \leftarrow \mathcal{I}$ //$\mathcal{I}$=initialization ($\top.\bot, \emptyset$ are usual)
$out_{b0} \leftarrow f_{n0}(I)$
worklist$\leftarrow B \setminus \{b0\}$
while worklist $\neq \emptyset$ **do**
  $b \leftarrow$ pop(worklist)

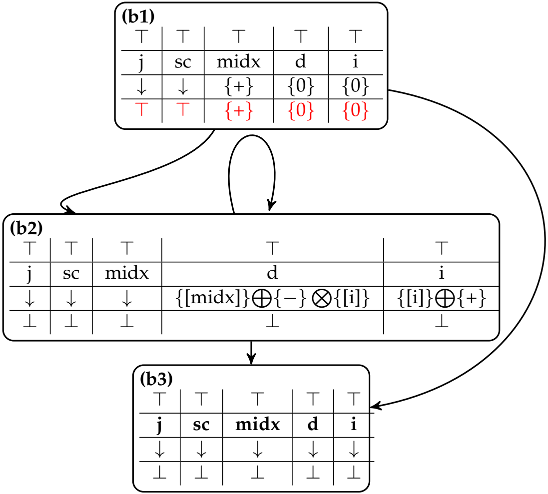
$in_b \leftarrow \vee\{out_m | m \in pred(b)\}$
  $out_b \leftarrow f_b(in_b)$
  if out_b changed **then**
    worklist $\leftarrow$ worklist $\cup$ b
  end if
end while



Dataflow Analysis — Sign Analysis Example (3)

for all $b \in B$ **do**
  $out_b \leftarrow f_b(\bot)$
end for
$in_{b0} \leftarrow \mathcal{I}$ //$\mathcal{I}$=initialization
($\top . \bot, \emptyset$ are usual)
$out_{b0} \leftarrow f_{n0}(I)$
worklist$\leftarrow B \setminus \{b0\}$
while worklist $\neq \emptyset$ **do**
  $b \leftarrow$ pop(worklist)

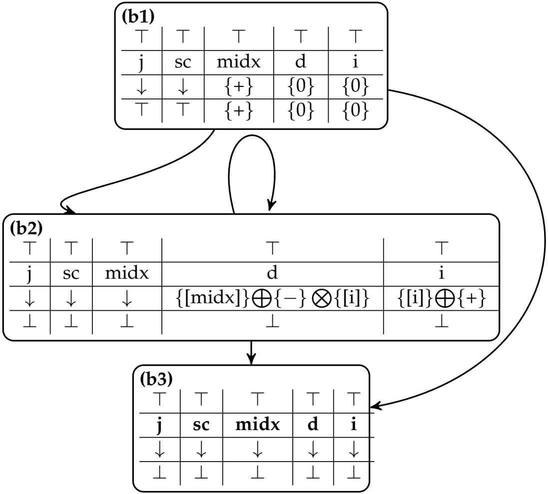
$in_b \leftarrow \vee\{out_m | m \in pred(b)\}$
  $out_b \leftarrow f_b(in_b)$
  if out_b changed **then**
    worklist $\leftarrow$ worklist
$\cup$ b
  end if
end while



Dataflow Analysis — Sign Analysis Example (3)

for all $b \in B$ **do**
  $out_b \leftarrow f_b(\bot)$
end for
$in_{b0} \leftarrow \mathcal{I}$ //$\mathcal{I}$=initialization
($\top.\bot, \emptyset$ are usual)
$out_{b0} \leftarrow f_{n0}(I)$
worklist$\leftarrow \{b2, b3\}$
while worklist $\neq \emptyset$ **do**
  $b \leftarrow$ pop(worklist)

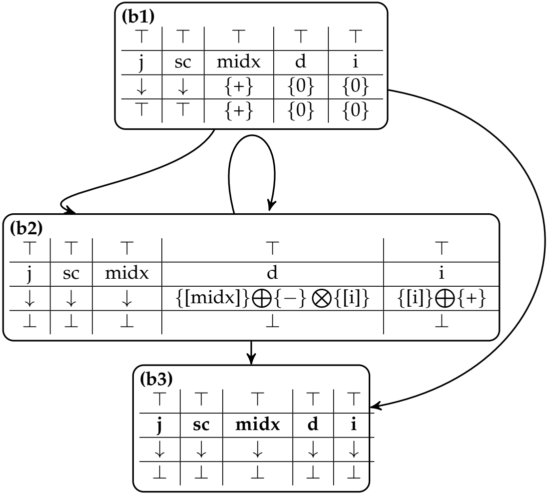
$in_b \leftarrow \vee\{out_m | m \in pred(b)\}$
  $out_b \leftarrow f_b(in_b)$
  if out_b changed **then**
    worklist $\leftarrow$ worklist
$\cup$ b
  end if
end while



DATAFLOW ANALYSIS — SIGN ANALYSIS EXAMPLE (3)

for all $b \in B$ **do**
  $out_b \leftarrow f_b(\bot)$
end for
$in_{b0} \leftarrow \mathcal{I}$ //$\mathcal{I}$=initialization
($\top.\bot, \emptyset$ are usual)
$out_{b0} \leftarrow f_{n0}(I)$
worklist$\leftarrow \{b2, b3\}$
while {b2,b3} $\neq \emptyset$ **do**
  $b \leftarrow$ pop(worklist)

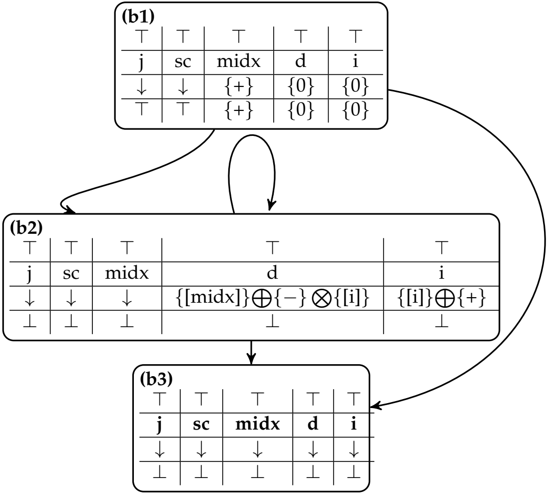
$in_b \leftarrow \vee\{out_m | m \in pred(b)\}$
  $out_b \leftarrow f_b(in_b)$
  if out_b changed **then**
    worklist $\leftarrow$ worklist
$\cup$ b
  end if
end while



Dataflow Analysis — Sign Analysis Example (3)

for all $b \in B$ **do**
  $out_b \leftarrow f_b(\bot)$
end for
$in_{b0} \leftarrow \mathcal{I}$ //$\mathcal{I}$=initialization
($\top.\bot, \emptyset$ are usual)
$out_{b0} \leftarrow f_{n0}(I)$
worklist$\leftarrow$ $\{b2, b3\}$
while {b2,b3} $\neq \emptyset$ **do**
  $b \leftarrow$ pop({b2,b3}
)//b=b2

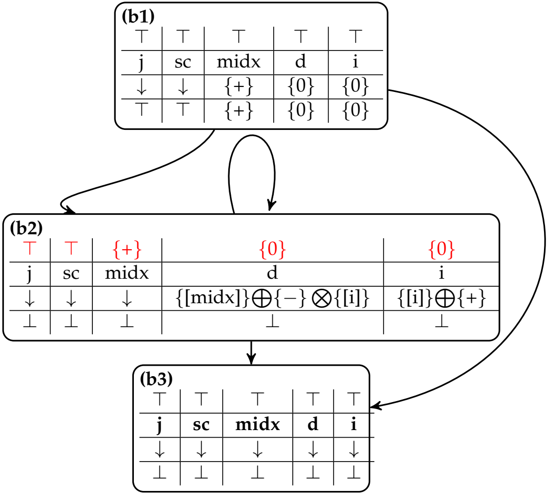
$in_b \leftarrow \vee\{out_m | m \in pred(b)\}$
  $out_b \leftarrow f_b(in_b)$
  if out_b changed **then**
    worklist $\leftarrow$ worklist
$\cup$ b
  end if
end while



Dataflow Analysis — Sign Analysis Example (3)

for all $b \in B$ **do**
  $out_b \leftarrow f_b(\perp)$
end for
$in_{b0} \leftarrow \mathcal{I}$ //$\mathcal{I}$=initialization
($\top.\perp, \emptyset$ are usual)
$out_{b0} \leftarrow f_{n0}(I)$
worklist$\leftarrow \{b2, b3\}$
while {b2,b3} $\neq \emptyset$ **do**
  $b \leftarrow$ pop({b3})

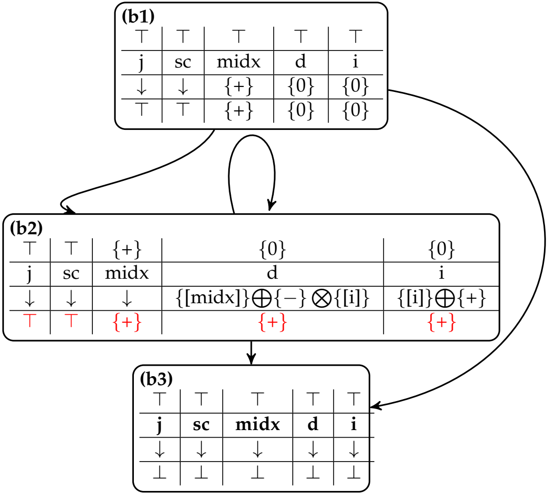
$in_{b2} \leftarrow \vee\{out_m | m \in pred(b2)\}$
  $out_b \leftarrow f_b(in_b)$
  if out_b changed **then**
    worklist $\leftarrow$ worklist
$\cup$ b
  end if
end while



Dataflow Analysis — Sign Analysis Example (3)

for all $b \in B$ **do**
  $out_b \leftarrow f_b(\bot)$
end for
$in_{b0} \leftarrow \mathcal{I}$ //$\mathcal{I}$=initialization
($\top.\bot, \emptyset$ are usual)
$out_{b0} \leftarrow f_{n0}(I)$
worklist$\leftarrow \{b2, b3\}$
while {b2,b3} $\neq \emptyset$ **do**
  $b \leftarrow$ pop({b3})

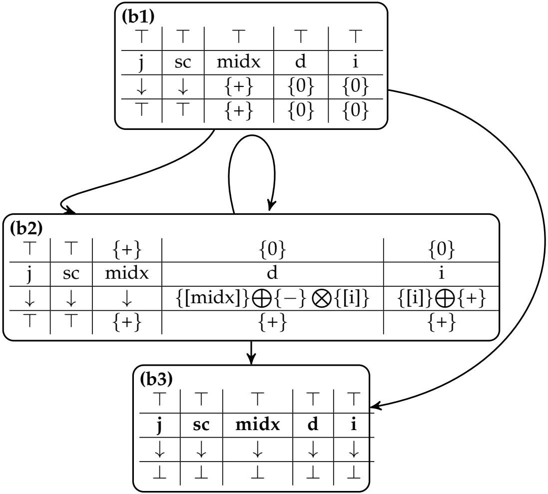
$in_{b2} \leftarrow \vee\{out_m | m \in pred(b2)\}$
  $out_{b2} \leftarrow f_{b2}(in_{b2})$
  if out_b changed **then**
    worklist $\leftarrow$ worklist
$\cup$ b
  end if
end while



Dataflow Analysis — Sign Analysis Example (3)

for all $b \in B$ **do**
  $out_b \leftarrow f_b(\bot)$
end for
$in_{b0} \leftarrow \mathcal{I}$ //$\mathcal{I}$=initialization
($\top.\bot, \emptyset$ are usual)
$out_{b0} \leftarrow f_{n0}(I)$
worklist$\leftarrow \{b2, b3\}$
while {b2,b3} $\neq \emptyset$ **do**
  $b \leftarrow$ pop({b3})

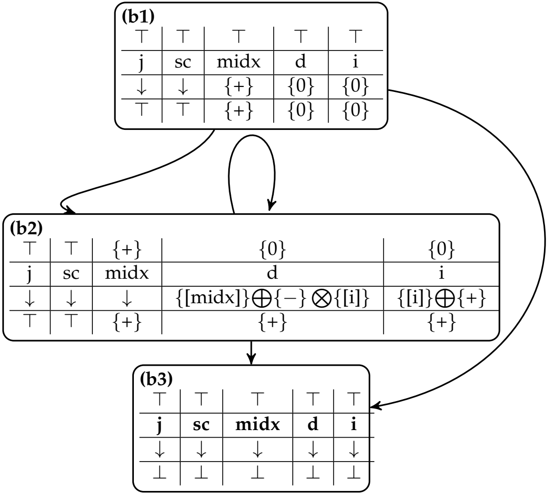
$in_{b2} \leftarrow \vee\{out_m | m \in pred(b2)\}$
  $out_{b2} \leftarrow f_{b2}(in_{b2})$
  if out_{b2} changed **then**
    worklist $\leftarrow$ worklist
$\cup$ b
  end if
end while



Dataflow Analysis — Sign Analysis Example (3)

for all $b \in B$ **do**
    $out_b \leftarrow f_b(\bot)$
end for
$in_{b0} \leftarrow \mathcal{I}$ //$\mathcal{I}$=initialization
($\top.\bot, \emptyset$ are usual)
$out_{b0} \leftarrow f_{n0}(I)$
worklist$\leftarrow \{b2, b3\}$
while {b2,b3} $\neq \emptyset$ **do**
    $b \leftarrow$ pop({b3})

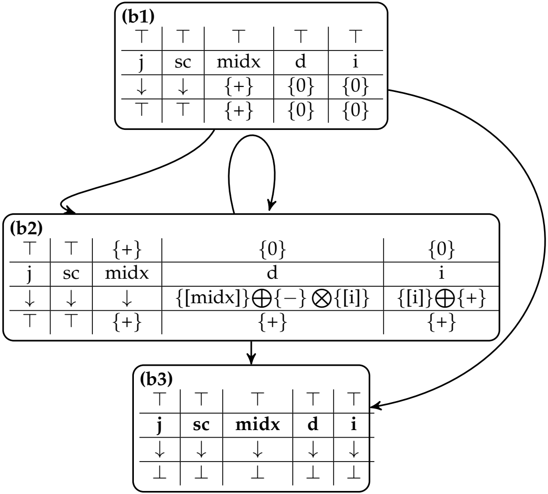
$in_{b2} \leftarrow \vee\{out_m | m \in pred(b2)\}$
    $out_{b2} \leftarrow f_{b2}(in_{b2})$
    if out_{b2} changed **then**
        {b3} ← {b3} ∪ {b2}
    end if
end while



DATAFLOW ANALYSIS — SIGN ANALYSIS EXAMPLE (3)

for all $b \in B$ **do**
  $out_b \leftarrow f_b(\bot)$
end for
$in_{b0} \leftarrow \mathcal{I}$ $//\mathcal{I}$=initialization
($\top.\bot, \emptyset$ are usual)
$out_{b0} \leftarrow f_{n0}(I)$
worklist$\leftarrow \{b2, b3\}$
while {b2,b3} $\neq \emptyset$ **do**
  $b \leftarrow$ pop(worklist)

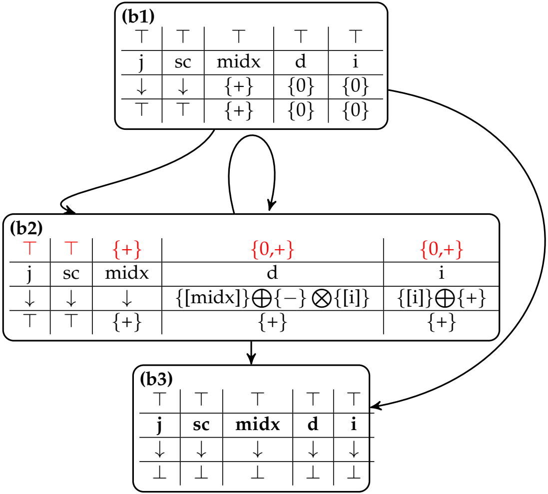
$in_b \leftarrow \vee\{out_m | m \in pred(b)\}$
  $out_b \leftarrow f_b(in_b)$
  if out_b changed **then**
    worklist $\leftarrow$ worklist
$\cup$ b
  end if
end while



DATAFLOW ANALYSIS — SIGN ANALYSIS EXAMPLE (3)

for all $b \in B$ **do**
    $out_b \leftarrow f_b(\bot)$
end for
$in_{b0} \leftarrow \mathcal{I}$ //$\mathcal{I}$=initialization
($\top.\bot, \emptyset$ are usual)
$out_{b0} \leftarrow f_{n0}(I)$
worklist$\leftarrow \{b2, b3\}$
while {b2,b3} $\neq \emptyset$ **do**
    $b \leftarrow$ pop({b3})

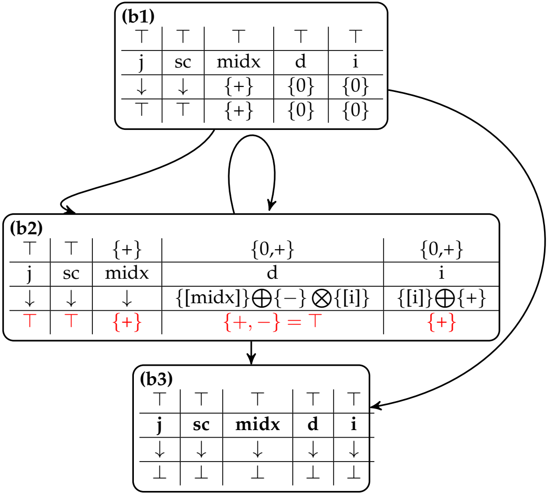
$in_{b2} \leftarrow \vee\{out_m | m \in pred(b2)\}$
    $out_b \leftarrow f_b(in_b)$
    if out_b changed **then**
        worklist $\leftarrow$ worklist
$\cup$ b
    end if
end while



Dataflow Analysis — Sign Analysis Example (3)

for all $b \in B$ **do**
  $out_b \leftarrow f_b(\bot)$
end for
$in_{b0} \leftarrow \mathcal{I}$ //$\mathcal{I}$=initialization
($\top.\bot, \emptyset$ are usual)
$out_{b0} \leftarrow f_{n0}(I)$
worklist$\leftarrow \{b2, b3\}$
while {b2,b3} $\neq \emptyset$ **do**
  $b \leftarrow$ pop({b3})

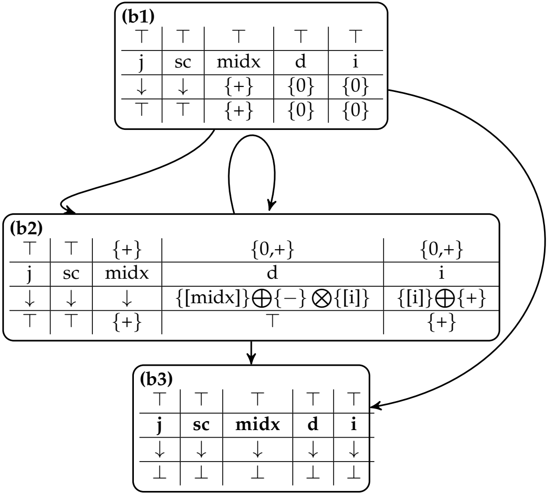
$in_{b2} \leftarrow \vee\{out_m | m \in pred(b2)\}$
  $out_{b2} \leftarrow f_{b2}(in_{b2})$
  if out_b changed **then**
    worklist ← worklist
∪ b
  end if
end while



DATAFLOW ANALYSIS — SIGN ANALYSIS EXAMPLE (3)

for all $b \in B$ **do**
    $out_b \leftarrow f_b(\bot)$
end for
$in_{b0} \leftarrow \mathcal{I}$ //$\mathcal{I}$=initialization
($\top.\bot, \emptyset$ are usual)
$out_{b0} \leftarrow f_{n0}(I)$
worklist$\leftarrow \{b2, b3\}$
while {b2,b3} $\neq \emptyset$ **do**
    $b \leftarrow$ pop({b3})

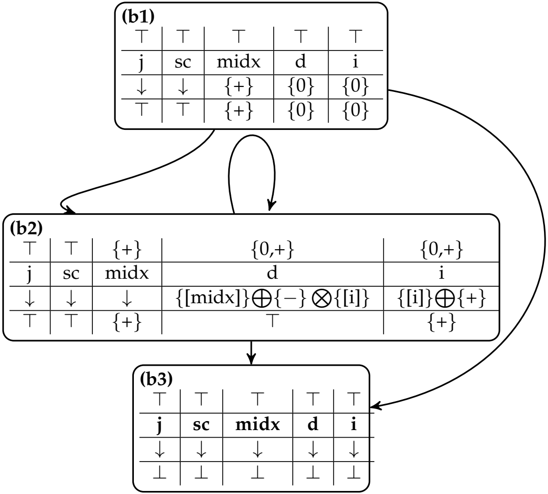
$in_{b2} \leftarrow \vee\{out_m | m \in pred(b2)\}$
    $out_{b2} \leftarrow f_{b2}(in_{b2})$
    if out_{b2} changed **then**
        {b3} ← {b3} ∪ {b2}
    end if
end while



Dataflow Analysis — Sign Analysis Example (3)

for all $b \in B$ **do**
    $out_b \leftarrow f_b(\bot)$
end for
$in_{b0} \leftarrow \mathcal{I}$ //$\mathcal{I}$=initialization
($\top.\bot, \emptyset$ are usual)
$out_{b0} \leftarrow f_{n0}(I)$
worklist$\leftarrow \{b2, b3\}$
while {b2,b3} $\neq \emptyset$ **do**
    $b \leftarrow$ pop(worklist)

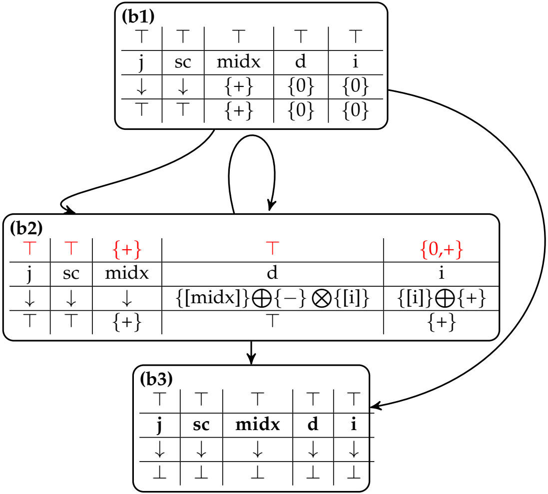
$in_b \leftarrow \vee\{out_m | m \in pred(b)\}$
    $out_b \leftarrow f_b(in_b)$
    if out_b changed **then**
        worklist $\leftarrow$ worklist
$\cup$ b
    end if
end while



Dataflow Analysis — Sign Analysis Example (3)

for all $b \in B$ **do**
    $out_b \leftarrow f_b(\bot)$
end for
$in_{b0} \leftarrow \mathcal{I}$ //$\mathcal{I}$=initialization
($\top.\bot, \emptyset$ are usual)
$out_{b0} \leftarrow f_{n0}(I)$
worklist$\leftarrow \{b2, b3\}$
while {b2,b3} $\neq \emptyset$ **do**
    $b \leftarrow$ pop({b3})

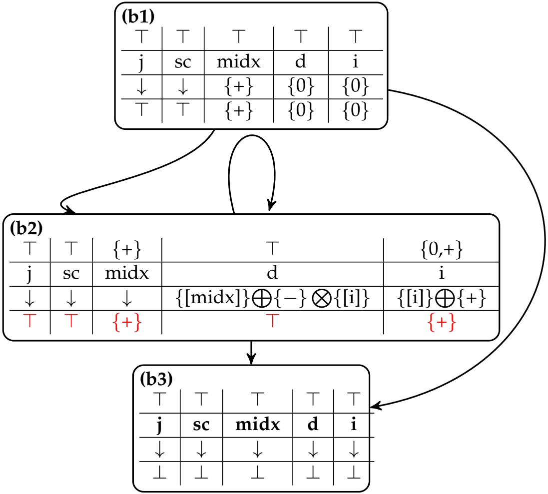
$in_{b2} \leftarrow \vee\{out_m | m \in pred(b2)\}$
    $out_b \leftarrow f_b(in_b)$
    if out_b changed **then**
        worklist $\leftarrow$ worklist
$\cup$ b
    end if
end while



DATAFLOW ANALYSIS — SIGN ANALYSIS EXAMPLE (3)

for all $b \in B$ **do**
  $out_b \leftarrow f_b(\bot)$
end for
$in_{b0} \leftarrow \mathcal{I}$ //$\mathcal{I}$=initialization
($\top.\bot, \emptyset$ are usual)
$out_{b0} \leftarrow f_{n0}(I)$
worklist$\leftarrow \{b2, b3\}$
while $\{$b2,b3$\} \neq \emptyset$ **do**
  $b \leftarrow$ pop($\{$b3$\}$)

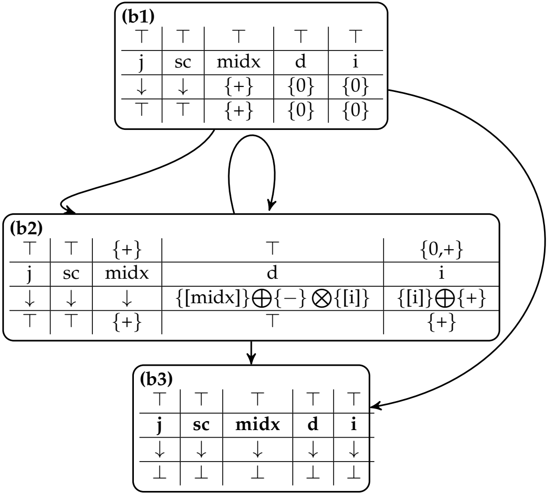
$in_{b2} \leftarrow \vee\{out_m | m \in pred(b2)\}$
  $out_{b2} \leftarrow f_{b2}(in_{b2})$
  if out_b changed **then**
    worklist $\leftarrow$ worklist
$\cup$ b
  end if
end while



Dataflow Analysis — Sign Analysis Example (3)

for all $b \in B$ **do**
  $out_b \leftarrow f_b(\bot)$
end for
$in_{b0} \leftarrow \mathcal{I}$ //$\mathcal{I}$=initialization
($\top . \bot, \emptyset$ are usual)
$out_{b0} \leftarrow f_{n0}(I)$
worklist$\leftarrow \{b2, b3\}$
while $\{b3\} \neq \emptyset$ **do**
  $b \leftarrow$ pop(worklist)

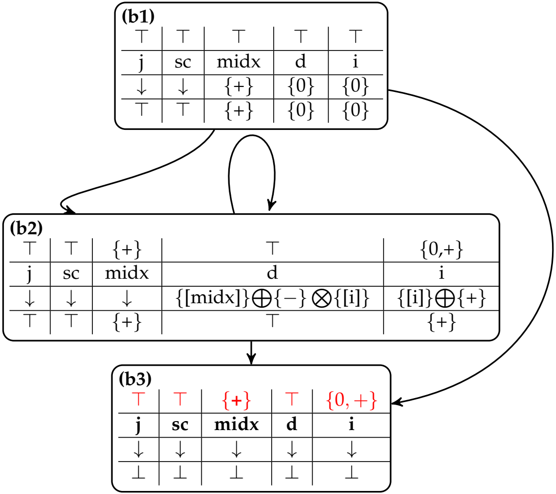
$in_b \leftarrow \vee\{out_m | m \in pred(b)\}$
  $out_b \leftarrow f_b(in_b)$
  if out_b changed **then**
    worklist $\leftarrow$ worklist
$\cup$ b
  end if
end while



Dataflow Analysis — Sign Analysis Example (3)

for all $b \in B$ **do**
  $out_b \leftarrow f_b(\bot)$
end for
$in_{b0} \leftarrow \mathcal{I}$ //$\mathcal{I}$=initialization
($\top . \bot, \emptyset$ are usual)
$out_{b0} \leftarrow f_{n0}(I)$
worklist$\leftarrow \{b2, b3\}$
while $\{\text{b3}\} \neq \emptyset$ **do**
  $b \leftarrow$ pop({})

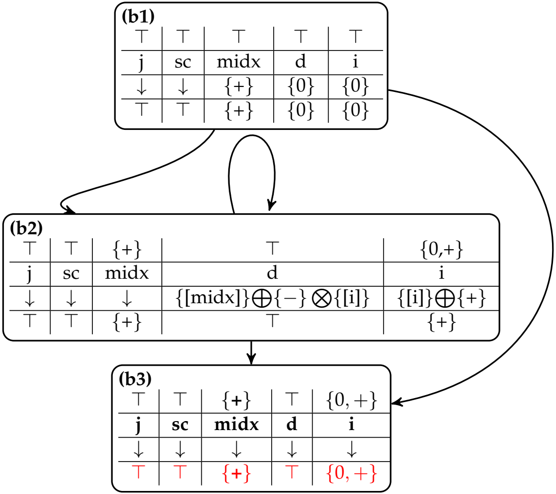
$in_{b3} \leftarrow \vee\{out_m | m \in pred(b3)\}$
  $out_b \leftarrow f_b(in_b)$
  if out_b changed **then**
    worklist $\leftarrow$ worklist
$\cup$ b
  end if
end while



Dataflow Analysis — Sign Analysis Example (3)

for all $b \in B$ **do**
  $out_b \leftarrow f_b(\bot)$
end for
$in_{b0} \leftarrow \mathcal{I}$ //$\mathcal{I}$=initialization
($\top.\bot, \emptyset$ are usual)
$out_{b0} \leftarrow f_{n0}(I)$
worklist$\leftarrow \{b2, b3\}$
while {b2,b3} $\neq \emptyset$ **do**
  $b \leftarrow$ pop({b3})

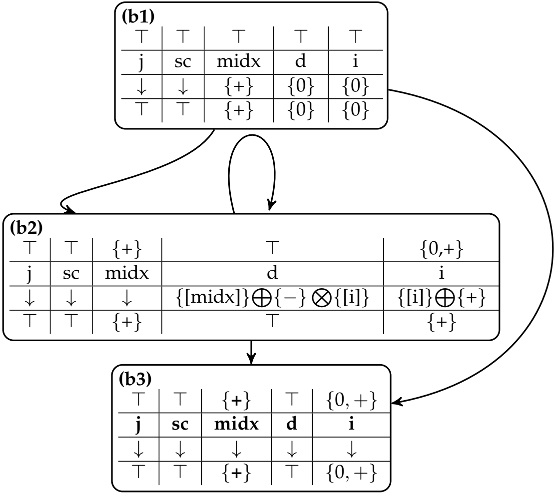
$in_{b3} \leftarrow \vee\{out_m | m \in pred(b3)\}$
  $out_{b3} \leftarrow f_{b3}(in_{b3})$
  if out_b changed **then**
    worklist $\leftarrow$ worklist
$\cup$ b
  end if
end while



Dataflow Analysis — Sign Analysis Example (3)

for all $b \in B$ **do**
  $out_b \leftarrow f_b(\bot)$
end for
$in_{b0} \leftarrow \mathcal{I}$ //$\mathcal{I}$=initialization
($\top . \bot, \emptyset$ are usual)
$out_{b0} \leftarrow f_{n0}(I)$
worklist$\leftarrow \{b2, b3\}$
while {b2,b3} $\neq \emptyset$ **do**
  $b \leftarrow$ pop({b3})

$in_{b3} \leftarrow \vee\{out_m | m \in pred(b3)\}$
  $out_{b3} \leftarrow f_{b3}(in_{b3})$
  if out_b changed **then**
    {} $\leftarrow$ worklist $\cup$ {b3}
  end if
end while



(b1)

⊤	⊤	⊤	⊤	⊤
j	sc	midx	d	i
↓	↓	{+}	{0}	{0}
⊤	⊤	{+}	{0}	{0}

(b2)

⊤	⊤	{+}	⊤	{0,+}
j	sc	midx	d	i
↓	↓	↓	{[midx]}⨁{−}⨂{[i]}	{[i]}⨁{+}
⊤	⊤	{+}	⊤	{+}

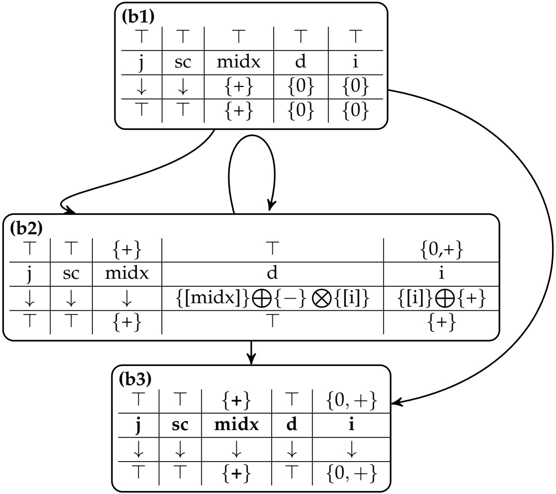
(b3)

⊤	⊤	{+}	⊤	{0, +}
j	**sc**	**midx**	**d**	**i**
↓	↓	↓	↓	↓
⊤	⊤	{+}	⊤	{0, +}

DATAFLOW ANALYSIS — SIGN ANALYSIS EXAMPLE (3)

for all $b \in B$ **do**
  $out_b \leftarrow f_b(\bot)$
end for
$in_{b0} \leftarrow \mathcal{I}$ //$\mathcal{I}$=initialization
($\top.\bot, \emptyset$ are usual)
$out_{b0} \leftarrow f_{n0}(I)$
worklist$\leftarrow \{b2, b3\}$
while $\{b3\} \neq \emptyset$ **do**
  $b \leftarrow$ pop(worklist)

$in_b \leftarrow \vee\{out_m | m \in pred(b)\}$
  $out_b \leftarrow f_b(in_b)$
  if out_b changed **then**
    worklist $\leftarrow$ worklist
$\cup$ b
  end if
end while



DATAFLOW ANALYSIS — SIGN ANALYSIS EXAMPLE (3)

for all $b \in B$ **do**

  $out_b \leftarrow f_b(\bot)$

end for

$in_{b0} \leftarrow \mathcal{I}$ //$\mathcal{I}$=initialization

($\top.\bot, \emptyset$ are usual)

$out_{b0} \leftarrow f_{n0}(I)$

worklist$\leftarrow \{b2, b3\}$

while $\{\text{b3}\} \neq \emptyset$ **do**

  $b \leftarrow$ pop({})

$in_{b3} \leftarrow \vee\{out_m | m \in pred(b3)\}$

  $out_b \leftarrow f_b(in_b)$

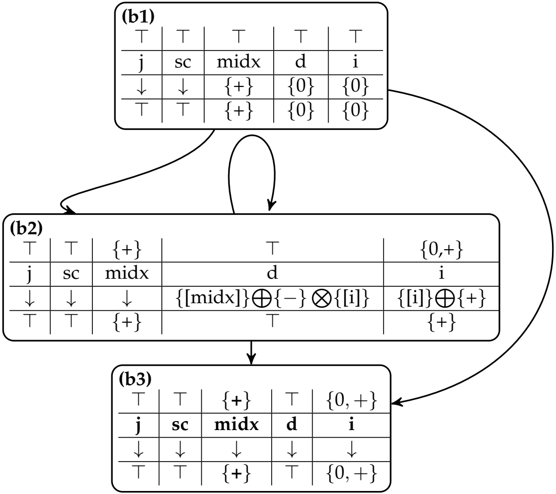
  if out_b changed **then**

    worklist $\leftarrow$ worklist

$\cup$ b

  end if

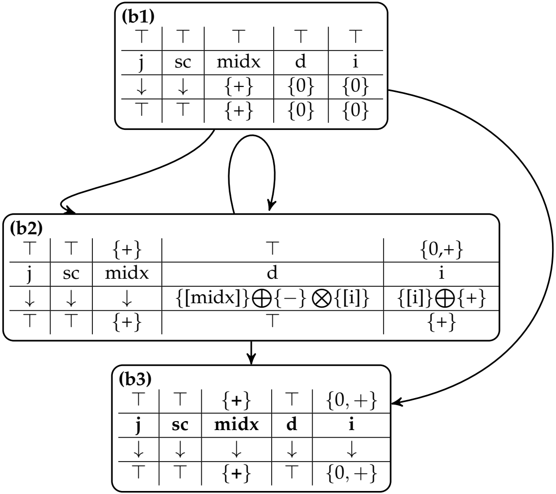
end while



Dataflow Analysis — Sign Analysis Example (3)

for all $b \in B$ **do**
  $out_b \leftarrow f_b(\bot)$
end for
$in_{b0} \leftarrow \mathcal{I}$ //$\mathcal{I}$=initialization
($\top.\bot, \emptyset$ are usual)
$out_{b0} \leftarrow f_{n0}(I)$
worklist$\leftarrow \{b2, b3\}$
while {b2,b3} $\neq \emptyset$ **do**
  $b \leftarrow$ pop({b3})

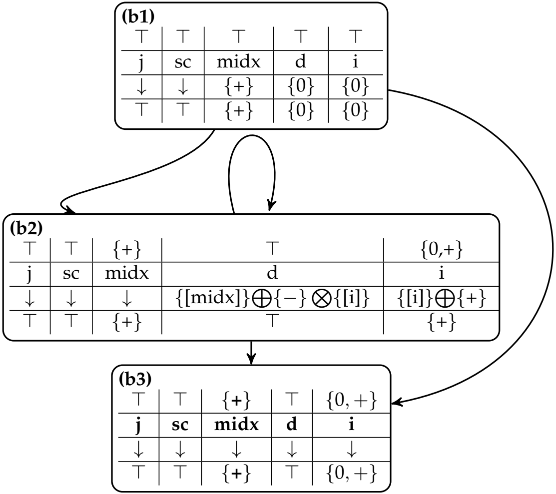
$in_{b3} \leftarrow \vee\{out_m | m \in pred(b3)\}$
  $out_{b3} \leftarrow f_{b3}(in_{b3})$
  if out_b changed **then**
    worklist $\leftarrow$ worklist
$\cup$ b
  end if
end while



Dataflow Analysis — Sign Analysis Example (3)

for all $b \in B$ **do**
  $out_b \leftarrow f_b(\bot)$
end for
$in_{b0} \leftarrow \mathcal{I}$ //$\mathcal{I}$=initialization
($\top.\bot, \emptyset$ are usual)
$out_{b0} \leftarrow f_{n0}(I)$
worklist$\leftarrow \{b2, b3\}$
while $\{\} \neq \emptyset$ **do**
  $b \leftarrow$ pop(worklist)

$in_b \leftarrow \vee\{out_m | m \in pred(b)\}$
  $out_b \leftarrow f_b(in_b)$
  if out_b changed **then**
    worklist $\leftarrow$ worklist
$\cup$ b
  end if
end while



DATAFLOW ANALYSIS — SIGN ANALYSIS EXAMPLE (3.1)

for all $b \in B$ **do**
  $out_b \leftarrow f_b(\bot)$
end for
$in_{b0} \leftarrow \mathcal{I}$ //$\mathcal{I}$=initialization
($\top.\bot, \emptyset$ are usual)
$out_{b0} \leftarrow f_{n0}(I)$
worklist$\leftarrow \{b2, b3\}$
while $\{\} \neq \emptyset$ **do**
  $b \leftarrow$ pop(worklist)

$in_b \leftarrow \vee\{out_m | m \in pred(b)\}$
  $out_b \leftarrow f_b(in_b)$
  if out_b changed **then**
    worklist $\leftarrow$ worklist
$\cup$ b
  end if
end while

Fixpoint Reached!

Dataflow Analysis — Sign Analysis Example (3.1)

for all $b \in B$ **do**
 $out_b \leftarrow f_b(\bot)$
end for
$in_{b0} \leftarrow \mathcal{I}$ //$\mathcal{I}$=initialization
($\top.\bot, \emptyset$ are usual)
$out_{b0} \leftarrow f_{n0}(I)$
worklist$\leftarrow \{b2, b3\}$
while $\{\} \neq \emptyset$ **do**
 $b \leftarrow$ pop(worklist)

$in_b \leftarrow \vee\{out_m | m \in pred(b)\}$
 $out_b \leftarrow f_b(in_b)$
 if out_b changed **then**
  worklist $\leftarrow$ worklist
$\cup$ b
 end if
end while

Fixpoint Reached!

Possible values:

j	sc	midx	d	i
$\top$	$\top$	$\{+\}$	$\top$	$\{0, +\}$

(specially in b2)

DATAFLOW ANALYSIS — SIGN ANALYSIS EXAMPLE (3.1)

for all $b \in B$ **do**
  $out_b \leftarrow f_b(\bot)$
end for
$in_{b0} \leftarrow \mathcal{I}$ //$\mathcal{I}$=initialization
($\top.\bot, \emptyset$ are usual)
$out_{b0} \leftarrow f_{n0}(I)$
worklist$\leftarrow \{b2, b3\}$
while $\{\} \neq \emptyset$ **do**
  $b \leftarrow$ pop(worklist)

$in_b \leftarrow \vee\{out_m | m \in pred(b)\}$
  $out_b \leftarrow f_b(in_b)$
  if out_b changed **then**
    worklist $\leftarrow$ worklist
$\cup$ b
  end if
end while

Fixpoint Reached!

Possible values:

j	sc	midx	d	i
$\top$	$\top$	$\{+\}$	$\top$	$\{0, +\}$

(specially in b2)

```
void bk (int j, char c[], size_t sc)
{
    size_t midx = 4;
    int d = 0,i =0;
    while(i< j)
    {
        d = midx - i;
        c[d] = i;
        i++;
    }
}
```

Static Analysis – Final Remarks

Static Analysis – Final Remarks

- It can be used for many things including statement reachability

Static Analysis – Final Remarks

- It can be used for many things including statement reachability
- It can be used for test generation, for determining the inputs that will cover the code

Static Analysis – Final Remarks

- It can be used for many things including statement reachability
- It can be used for test generation, for determining the inputs that will cover the code
- False-positives can be acceptable

Static Analysis – Final Remarks

- It can be used for many things including statement reachability
- It can be used for test generation, for determining the inputs that will cover the code
- False-positives can be acceptable
- There exist well-known plug-in (or feature) based tools, e.g., Frama-C

Static Analysis – Final Remarks

- It can be used for many things including statement reachability
- It can be used for test generation, for determining the inputs that will cover the code
- False-positives can be acceptable
- There exist well-known plug-in (or feature) based tools, e.g., Frama-C
- There exist other formal approaches, e.g., abstract interpretation

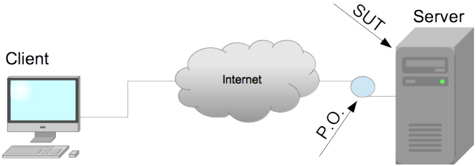
Passive Testing using Network Traces

Architecture(example)

An image, 10^3 words…

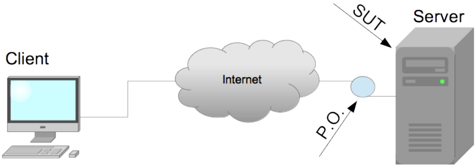
ARCHITECTURE(EXAMPLE)

An image, 10^3 words…



Architecture(example)

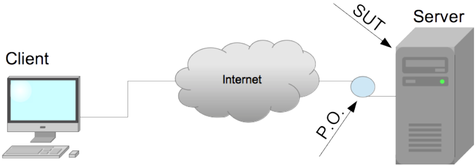
An image, 10^3 words...



- Direction: From server to client

ARCHITECTURE(EXAMPLE)

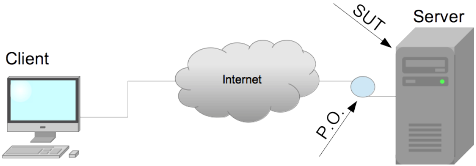
An image, 10^3 words...



- Direction: From server to client
- P.O. = **one** network interface of the server (usually a P.O. is associated with a network host, it can vary...)

ARCHITECTURE(EXAMPLE)

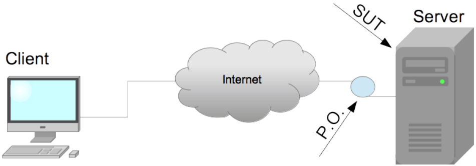
An image, 10^3 words...



- Direction: From server to client
- P.O. = **one** network interface of the server (usually a P.O. is associated with a network host, it can vary...)
- No information regarding on-line or off-line trace collection (perhaps on-line is more interesting)

ARCHITECTURE(EXAMPLE)

An image, 10^3 words...



- Direction: From server to client
- P.O. = **one** network interface of the server (usually a P.O. is associated with a network host, it can vary...)
- No information regarding on-line or off-line trace collection (perhaps on-line is more interesting)

How do we test this?

Deep Packet Inspection (DPI) 101

What is DPI?

Deep Packet Inspection (DPI) 101

What is DPI?

- The process of examining the data of a network packet when searching for *specific* parameter values in it:

DEEP PACKET INSPECTION (DPI) 101

What is DPI?

- The process of examining the data of a network packet when searching for *specific* parameter values in it:
  - Protocol non-compliance, e.g., TCP SYN-FIN (or xmas tree packet)

Deep Packet Inspection (DPI) 101

What is DPI?

- The process of examining the data of a network packet when searching for *specific* parameter values in it:
  - Protocol non-compliance, e.g., TCP SYN-FIN (or xmas tree packet)
  - Viruses, buffer overflows, etc.

Deep Packet Inspection (DPI) 101

What is DPI?

- The process of examining the data of a network packet when searching for *specific* parameter values in it:
  - Protocol non-compliance, e.g., TCP SYN-FIN (or xmas tree packet)
  - Viruses, buffer overflows, etc.
    - These can be identified by a **signature**
    - A signature is a known binary sequence inside a packet that identifies the attack

Deep Packet Inspection (DPI) 101

What is DPI?

- The process of examining the data of a network packet when searching for *specific* parameter values in it:
  - Protocol non-compliance, e.g., TCP SYN-FIN (or xmas tree packet)
  - Viruses, buffer overflows, etc.
    - These can be identified by a **signature**
    - A signature is a known binary sequence inside a packet that identifies the attack
  - Specific application layer data, for instance:

    ```
    a'; DROP TABLE users
    ```

Deep Packet Inspection (DPI) 101

What is DPI?

- The process of examining the data of a network packet when searching for *specific* parameter values in it:
  - Protocol non-compliance, e.g., TCP SYN-FIN (or xmas tree packet)
  - Viruses, buffer overflows, etc.
    - These can be identified by a **signature**
    - A signature is a known binary sequence inside a packet that identifies the attack
  - Specific application layer data, for instance:

    `a'; DROP TABLE users`

What to do, once certain value is found?

Deep Packet Inspection (DPI) 101

What is DPI?

- The process of examining the data of a network packet when searching for *specific* parameter values in it:
  - Protocol non-compliance, e.g., TCP SYN-FIN (or xmas tree packet)
  - Viruses, buffer overflows, etc.
    - These can be identified by a **signature**
    - A signature is a known binary sequence inside a packet that identifies the attack
  - Specific application layer data, for instance:

    `a'; DROP TABLE users`

What to do, once certain value is found?

- Report the finding (usually the search targets for prohibited elements)

Deep Packet Inspection (DPI) 102

Deep Packet Inspection (DPI) 102

- Mostly in firewalls, Intrusion Detection Systems (IDS), and Intrusion Prevention Systems (IPS)

Deep Packet Inspection (DPI) 102

- Mostly in firewalls, Intrusion Detection Systems (IDS), and Intrusion Prevention Systems (IPS)
- Off-line approaches are not very popular

Deep Packet Inspection (DPI) 102

- Mostly in firewalls, Intrusion Detection Systems (IDS), and Intrusion Prevention Systems (IPS)
- Off-line approaches are not very popular
- Off-line approaches are sometimes considered as a form of computer forensics (examining an already killed computer)

Deep Packet Inspection (DPI) 102

- Mostly in firewalls, Intrusion Detection Systems (IDS), and Intrusion Prevention Systems (IPS)
- Off-line approaches are not very popular
- Off-line approaches are sometimes considered as a form of computer forensics (examining an already killed computer)

How to describe which values to search?

Deep Packet Inspection (DPI) 102

- Mostly in firewalls, Intrusion Detection Systems (IDS), and Intrusion Prevention Systems (IPS)
- Off-line approaches are not very popular
- Off-line approaches are sometimes considered as a form of computer forensics (examining an already killed computer)

How to describe which values to search?

- There exist various approaches (Cisco, Snort, etc.), nonetheless, they tend to have common points…

Describing values in DPI

Based on “rules”

Describing values in DPI

Based on “rules”

- For common protocols (IP, TCP, UDP, HTTP, etc.) variables are provided, for example:

DESCRIBING VALUES IN DPI

Based on "rules"

- For common protocols (IP, TCP, UDP, HTTP, etc.) variables are provided, for example:

```
alert tcp any any -> any 21 (msg:"FTP ROOT"; content:"USER root"; nocase;)
```

Describing values in DPI

Based on “rules”

- For common protocols (IP, TCP, UDP, HTTP, etc.) variables are provided, for example:

```
alert tcp any any -> any 21 (msg:"FTP ROOT"; content:"USER root"; nocase;)
```

- A signature can be described as a set of strings (potentially binary) of a regular language

Describing values in DPI

Based on “rules”

- For common protocols (IP, TCP, UDP, HTTP, etc.) variables are provided, for example:

```
alert tcp any any -> any 21 (msg:"FTP ROOT"; content:"USER root"; nocase;)
```

- A signature can be described as a set of strings (potentially binary) of a regular language
  - It can be described by a regular expression

Describing values in DPI

Based on “rules”

- For common protocols (IP, TCP, UDP, HTTP, etc.) variables are provided, for example:

```
alert tcp any any -> any 21 (msg:"FTP ROOT"; content:"USER root"; nocase;)
```

- A signature can be described as a set of strings (potentially binary) of a regular language
  - It can be described by a regular expression

```
alert tcp any any -> any 80 (content:"/foo.php?id=";
 pcre:"/foo.php?id=[0-9]{1,10}/iU";)
```

Describing values in DPI

Based on "rules"

- For common protocols (IP, TCP, UDP, HTTP, etc.) variables are provided, for example:

```
alert tcp any any -> any 21 (msg:"FTP ROOT"; content:"USER root"; nocase;)
```

- A signature can be described as a set of strings (potentially binary) of a regular language
  - It can be described by a regular expression

```
alert tcp any any -> any 80 (content:"/foo.php?id=";
 pcre:"/foo.php?id=[0-9]{1,10}/iU";)
```

(All the previous rules were written using the syntax of snort)

Stateless and Stateful DPI concepts

Stateless DPI

Stateless and Stateful DPI concepts

Stateless DPI

- Each rule is applied to each network packet and no state is saved

Stateless and Stateful DPI concepts

Stateless DPI

- Each rule is applied to each network packet and no state is saved
- It can be good if we are tying to search for a virus transmitted over SMTP, for instance

Stateless and Stateful DPI concepts

Stateless DPI

- Each rule is applied to each network packet and no state is saved
- It can be good if we are tying to search for a virus transmitted over SMTP, for instance

Stateful DPI

Stateless and Stateful DPI concepts

Stateless DPI

- Each rule is applied to each network packet and no state is saved
- It can be good if we are tying to search for a virus transmitted over SMTP, for instance

Stateful DPI

- Certain information regarding the state of the connection gets stored

Stateless and Stateful DPI concepts

Stateless DPI

- Each rule is applied to each network packet and no state is saved
- It can be good if we are tying to search for a virus transmitted over SMTP, for instance

Stateful DPI

- Certain information regarding the state of the connection gets stored
- For example, the FTP data channel get associated to the FTP control channel

Beyond DPI

What if we want more?.. What is more?

Beyond DPI

What if we want more?.. What is more?

- The Very Simple Network Protocol (VSNP)
  - Every client request has an integer ID
  - Each client request is followed by a VSNP server response
  - The VSNP server response should be even if the request ID is odd, and vice-versa

Beyond DPI

What if we want more?.. What is more?

- The Very Simple Network Protocol (VSNP)
  - Every client request has an integer ID
  - Each client request is followed by a VSNP server response
  - The VSNP server response should be even if the request ID is odd, and vice-versa
- Assume we want to check the odd/even, even/odd constraints

Beyond DPI

What if we want more?.. What is more?

- The Very Simple Network Protocol (VSNP)
  - Every client request has an integer ID
  - Each client request is followed by a VSNP server response
  - The VSNP server response should be even if the request ID is odd, and vice-versa
- Assume we want to check the odd/even, even/odd constraints
- Or we want to check the behavior of non-typical protocol implementations or not predefined set of rules

Beyond DPI

What if we want more?.. What is more?

- The Very Simple Network Protocol (VSNP)
  - Every client request has an integer ID
  - Each client request is followed by a VSNP server response
  - The VSNP server response should be even if the request ID is odd, and vice-versa
- Assume we want to check the odd/even, even/odd constraints
- Or we want to check the behavior of non-typical protocol implementations or not predefined set of rules
- Or to choose what to save and how to correlate it with future packets

Beyond DPI

What if we want more?.. What is more?

- The Very Simple Network Protocol (VSNP)
  - Every client request has an integer ID
  - Each client request is followed by a VSNP server response
  - The VSNP server response should be even if the request ID is odd, and vice-versa
- Assume we want to check the odd/even, even/odd constraints
- Or we want to check the behavior of non-typical protocol implementations or not predefined set of rules
- Or to choose what to save and how to correlate it with future packets

Passive Testing using Network Traces

Passive Testing using Network Traces 101

Passive Testing using Network Traces 101

We want to guarantee that:

Passive Testing using Network Traces 101

We want to guarantee that:

- Certain functional and non-functional requirements hold over the network traces, also known as **properties** (or rules, or invariants, more on this later)

Passive Testing using Network Traces 101

We want to guarantee that:

- Certain functional and non-functional requirements hold over the network traces, also known as **properties** (or rules, or invariants, more on this later)
- We are able to analyze properties that go beyond single packet analysis or simple associations

Passive Testing using Network Traces 101

We want to guarantee that:

- Certain functional and non-functional requirements hold over the network traces, also known as **properties** (or rules, or invariants, more on this later)
- We are able to analyze properties that go beyond single packet analysis or simple associations

Consider the VSNP protocol and its even/odd, odd/even property

PASSIVE TESTING USING NETWORK TRACES 101

We want to guarantee that:

- Certain functional and non-functional requirements hold over the network traces, also known as **properties** (or rules, or invariants, more on this later)
- We are able to analyze properties that go beyond single packet analysis or simple associations

Consider the VSNP protocol and its even/odd, odd/even property

- Let's take a look at a potential network trace to list some properties

UNDERSTANDING CORRELATED NETWORK INTERACTIONS

Consider the following trace

Understanding correlated network interactions

Consider the following trace

ID:2	**ID:3**	**ID:4**	**ID:2**	**ID:4**	**ID:21**	**ID:21**
N:	N:	N:	N: 77	N: 89	N:	N: 101

Understanding correlated network interactions

Consider the following trace

ID:2	ID:3	ID:4	ID:2	ID:4	ID:21	ID:21
N:	N:	N:	N: 77	N: 89	N:	N: 101

Questions

Understanding correlated network interactions

Consider the following trace

ID:2	**ID:3**	**ID:4**	**ID:2**	**ID:4**	**ID:21**	**ID:21**
N:	N:	N:	N: 77	N: 89	N:	N: 101

Questions

- How can two requests / responses be together?

UNDERSTANDING CORRELATED NETWORK INTERACTIONS

Consider the following trace

ID:2	**ID:3**	**ID:4**	**ID:2**	**ID:4**	**ID:21**	**ID:21**
N:	N:	N:	N: 77	N: 89	N:	N: 101

Questions

- How can two requests / responses be together?
  - As packets go through, the P.O. allows to observe n sequential client(s) requests before a response arrives to the P.O.

Understanding correlated network interactions

Consider the following trace

ID:2	**ID:3**	**ID:4**	**ID:2**	**ID:4**	**ID:21**	**ID:21**
N:	N:	N:	N: 77	N: 89	N:	N: 101

Questions

- How can two requests / responses be together?
  - As packets go through, the P.O. allows to observe *n* sequential client(s) requests before a response arrives to the P.O.
- What do we do with a non-replied request?

Understanding correlated network interactions

Consider the following trace

ID:2	**ID:3**	**ID:4**	**ID:2**	**ID:4**	**ID:21**	**ID:21**
N:	N:	N:	N: 77	N: 89	N:	N: 101

Questions

- How can two requests / responses be together?
  - As packets go through, the P.O. allows to observe n sequential client(s) requests before a response arrives to the P.O.
- What do we do with a non-replied request?
  - It depends if the analysis is performed on-line or off-line

The On-line vs. Off-line Monitoring Verdicts

Incoming (or read) packets

Tester Storing queue / Memory

Actions:

THE ON-LINE VS. OFF-LINE MONITORING VERDICTS

Incoming (or read) packets

ID:2

N:

Tester Storing queue / Memory

Actions:

Read REQ with ID = 2

The On-line vs. Off-line Monitoring Verdicts

Incoming (or read) packets

Tester Storing queue / Memory

ID:2

N:

Actions:

Store packet in the requests to be replied queue

The On-line vs. Off-line Monitoring Verdicts

Incoming (or read) packets

ID:3

N:

Tester Storing queue / Memory

ID:2

N:

Actions:

Read REQ with ID = 3

The On-line vs. Off-line Monitoring Verdicts

Incoming (or read) packets

Tester Storing queue / Memory

ID:2	**ID:3**
N:	N:

Actions:

Store packet in the requests to be replied queue

The On-line vs. Off-line Monitoring Verdicts

Incoming (or read) packets

ID:4
N:

Tester Storing queue / Memory

ID:2	**ID:3**
N:	N:

Actions:
Read REQ with ID = 4

The On-line vs. Off-line Monitoring Verdicts

Incoming (or read) packets

Tester Storing queue / Memory

ID:2	**ID:3**	**ID:4**
N:	N:	N:

Actions:

Store packet in the requests to be replied queue

The On-line vs. Off-line Monitoring Verdicts

Incoming (or read) packets

ID:2
N: 77

Tester Storing queue / Memory

ID:2	**ID:3**	**ID:4**
N:	N:	N:

Actions:

Read RES with ID = 2

The On-line vs. Off-line Monitoring Verdicts

Incoming (or read) packets

ID:2
N: 77

Tester Storing queue / Memory

ID:2	**ID:3**	**ID:4**
N:	N:	N:

Actions:
Check to which stored packet it *corresponds*

The On-line vs. Off-line Monitoring Verdicts

Incoming (or read) packets

ID:2
N: 77

Tester Storing queue / Memory

ID:2	**ID:3**	**ID:4**
N:	N:	N:

Actions:
Verify the property

The On-line vs. Off-line Monitoring Verdicts

Incoming (or read) packets

Tester Storing queue / Memory

ID:2	**ID:3**	**ID:4**
N:	N:	N:

Actions:

Report PASS (+)

The On-line vs. Off-line Monitoring Verdicts

Incoming (or read) packets

Tester Storing queue / Memory

ID:3	**ID:4**
N:	N:

Actions:
Remove corresponding stored packet from the stored requests queue

THE ON-LINE VS. OFF-LINE MONITORING VERDICTS

Incoming (or read) packets

ID:4
N: 89

Tester Storing queue / Memory

ID:3	**ID:4**
N:	N:

Actions:
Read RES with ID = 4

THE ON-LINE VS. OFF-LINE MONITORING VERDICTS

Incoming (or read) packets

ID:4
N: 89

Tester Storing queue / Memory

ID:3	**ID:4**
N:	N:

Actions:
Check to which stored packet it *corresponds*

The On-line vs. Off-line Monitoring Verdicts

Incoming (or read) packets

ID:4
N: 89

Tester Storing queue / Memory

ID:3	**ID:4**
N:	N:

Actions:
Verify the property

THE ON-LINE VS. OFF-LINE MONITORING VERDICTS

Incoming (or read) packets

Tester Storing queue / Memory

ID:3	**ID:4**
N:	N:

Actions:
Report PASS (+)

The On-line vs. Off-line Monitoring Verdicts

Incoming (or read) packets

Tester Storing queue / Memory

ID:3

N:

Actions:

Remove corresponding stored packet from the stored requests queue

The On-line vs. Off-line Monitoring Verdicts

Incoming (or read) packets

ID:21

N:

Tester Storing queue / Memory

ID:3

N:

Actions:

Read REQ with ID = 21

The On-line vs. Off-line Monitoring Verdicts

Incoming (or read) packets

Tester Storing queue / Memory

ID:3	**ID:21**
N:	N:

Actions:

Store packet in the requests to be replied queue

The On-line vs. Off-line Monitoring Verdicts

Incoming (or read) packets

ID:21
N: **101**

Tester Storing queue / Memory

ID:3	**ID:21**
N:	N:

Actions:
Read RES with ID = 21

The On-line vs. Off-line Monitoring Verdicts

Incoming (or read) packets

ID:21
N: **101**

Tester Storing queue / Memory

ID:3	**ID:21**
N:	N:

Actions:
Check to which stored packet it *corresponds*

The On-line vs. Off-line Monitoring Verdicts

Incoming (or read) packets

ID:21
N: **101**

Tester Storing queue / Memory

ID:3	**ID:21**
N:	N:

Actions:
Verify the property

THE ON-LINE VS. OFF-LINE MONITORING VERDICTS

Incoming (or read) packets

Tester Storing queue / Memory

ID:3	ID:21
N:	N:

Actions:

Report FAIL (-)

The On-line vs. Off-line Monitoring Verdicts

Incoming (or read) packets

Tester Storing queue / Memory

ID:3

N:

Actions:

Remove corresponding stored packet from the stored requests queue

The On-line vs. Off-line Monitoring Verdicts

Incoming (or read) packets

Tester Storing queue / Memory

ID:3

N:

Actions:

What's next?

THE ON-LINE VS. OFF-LINE MONITORING VERDICTS

Incoming (or read) packets

Tester Storing queue / Memory

ID:3

N:

Actions:

Wait. . . wait until another packet comes (live capture)

THE ON-LINE VS. OFF-LINE MONITORING VERDICTS

Incoming (or read) packets

Tester Storing queue / Memory

ID:3

N:

Actions:

Until when do we wait? What do we do with this left-alone packet

THE ON-LINE VS. OFF-LINE MONITORING VERDICTS

Incoming (or read) packets

Tester Storing queue / Memory

ID:3

N:

Actions:

Until a determined **timeout**

THE ON-LINE VS. OFF-LINE MONITORING VERDICTS

Incoming (or read) packets

Tester Storing queue / Memory

Actions:
After the determined timeout, report TIME_FAIL (!) and remove packets whose time stored > timeout

THE ON-LINE VS. OFF-LINE MONITORING VERDICTS

Incoming (or read) packets

Tester Storing queue / Memory

ID:3

N:

Actions:

Now assume it was off-line

THE ON-LINE VS. OFF-LINE MONITORING VERDICTS

Incoming (or read) packets

EOT

(end of trace)

Tester Storing queue / Memory

ID:3

N:

Actions:

Read EOT

THE ON-LINE VS. OFF-LINE MONITORING VERDICTS

Incoming (or read) packets

Tester Storing queue / Memory

ID:3

N:

Actions:

For each packet left on memory report **?**

THE ON-LINE VS. OFF-LINE MONITORING VERDICTS

Incoming (or read) packets

Tester Storing queue / Memory

Actions:
Report INCONCLUSIVE (?)! (What if the trace was cut before the packet arrived?) and delete the left packets

Correlated network interactions (cont.)

Some conclusions / questions

Correlated network interactions (cont.)

Some conclusions / questions

- Given the nature of properties, matching packets cannot be expressed by a regular language

Correlated network interactions (cont.)

Some conclusions / questions

- Given the nature of properties, matching packets cannot be expressed by a regular language
  - How do we express the properties?

CORRELATED NETWORK INTERACTIONS (CONT.)

Some conclusions / questions

- Given the nature of properties, matching packets cannot be expressed by a regular language
  - How do we express the properties?
- Given the network interactions, each packet can represent a connection in any state (bad, very bad…)

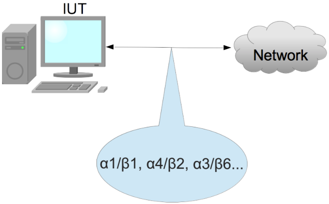
Correlated network interactions (cont.)

Some conclusions / questions

- Given the nature of properties, matching packets cannot be expressed by a regular language
  - How do we express the properties?
- Given the network interactions, each packet can represent a connection in any state (bad, very bad...)
  - How to avoid resource consumption?

Understanding correlated network interactions (cont. 2)

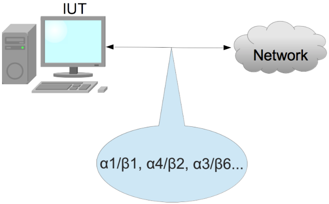
Interaction



Invariants or properties

UNDERSTANDING CORRELATED NETWORK INTERACTIONS (CONT. 2)

Interaction

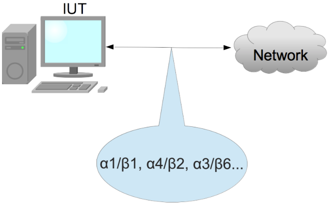


Invariants or properties

- *Test purposes* hold over all the observed network traces

Understanding correlated network interactions (cont. 2)

Interaction

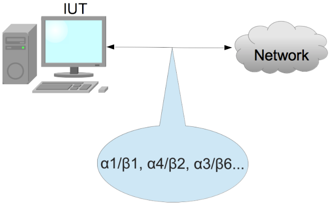


Invariants or properties

- *Test purposes* hold over all the observed network traces
- E.g., $\beta 6$ is not allowed to occur before the occurrence of $\alpha 4$

Understanding correlated network interactions (cont. 2)

Interaction



Invariants or properties

- *Test purposes* hold over all the observed network traces
- E.g., $\beta 6$ is not allowed to occur before the occurrence of $\alpha 4$
- There have been proposed **many** languages to express invariants

Expressing properties

Many languages have been proposed…

Expressing properties

Many languages have been proposed...

- Linear Temporal Logic (LTL)

Expressing properties

Many languages have been proposed…

- Linear Temporal Logic (LTL)
  - Describes an ω-regular language (over infinite words)

EXPRESSING PROPERTIES

Many languages have been proposed…

- Linear Temporal Logic (LTL)
  - Describes an ω-regular language (over infinite words)
  - Fits pretty well if assuming an interaction has a single state (the P.O. is not situated on a server, for instance)

Expressing properties

Many languages have been proposed…

- Linear Temporal Logic (LTL)
  - Describes an ω-regular language (over infinite words)
  - Fits pretty well if assuming an interaction has a single state (the P.O. is not situated on a server, for instance)
  - Is not ideal if many states are assumed to be possible for different connections (many sequential requests)

Expressing properties

Many languages have been proposed...

- Linear Temporal Logic (LTL)
  - Describes an ω-regular language (over infinite words)
  - Fits pretty well if assuming an interaction has a single state (the P.O. is not situated on a server, for instance)
  - Is not ideal if many states are assumed to be possible for different connections (many sequential requests)
- Languages based on Context Free Grammars (CFG)

Expressing properties

Many languages have been proposed…

- Linear Temporal Logic (LTL)
  - Describes an ω-regular language (over infinite words)
  - Fits pretty well if assuming an interaction has a single state (the P.O. is not situated on a server, for instance)
  - Is not ideal if many states are assumed to be possible for different connections (many sequential requests)
- Languages based on Context Free Grammars (CFG)
  - Languages are adjusted to the task

EXPRESSING PROPERTIES

Many languages have been proposed…

- Linear Temporal Logic (LTL)
  - Describes an ω-regular language (over infinite words)
  - Fits pretty well if assuming an interaction has a single state (the P.O. is not situated on a server, for instance)
  - Is not ideal if many states are assumed to be possible for different connections (many sequential requests)
- Languages based on Context Free Grammars (CFG)
  - Languages are adjusted to the task
  - Examples: XML-based, or others…

Expressing properties

Many languages have been proposed...

- Linear Temporal Logic (LTL)
  - Describes an ω-regular language (over infinite words)
  - Fits pretty well if assuming an interaction has a single state (the P.O. is not situated on a server, for instance)
  - Is not ideal if many states are assumed to be possible for different connections (many sequential requests)
- Languages based on Context Free Grammars (CFG)
  - Languages are adjusted to the task
  - Examples: XML-based, or others...
  - Concepts behind the languages are more important...

Passive Testing with Network Traces
Concepts

Passive Testing with Network Traces
Concepts

- A property expresses a sequence of premises and consequences of non-chronologically arranged, but related network packets

Passive Testing with Network Traces
Concepts

- A property expresses a sequence of premises and consequences of non-chronologically arranged, but related network packets
  - Example: in English, if(A) then B (before or after), or if{if (A) then B(before or after)} then C (before or after), etc...

Passive Testing with Network Traces
Concepts

- A property expresses a sequence of premises and consequences of non-chronologically arranged, but related network packets
  - Example: in English, if(A) then B (before or after), or if{if (A) then B(before or after)} then C (before or after), etc...
- A property must be able to characterize different packets

Passive Testing with Network Traces
Concepts

- A property expresses a sequence of premises and consequences of non-chronologically arranged, but related network packets
  - Example: in English, if(A) then B (before or after), or if{if (A) then B(before or after)} then C (before or after), etc...
- A property must be able to characterize different packets
  - That is, to describe a template of each packet
    Example: a packet that uses TCP in the port 1010 and the first value is an ID, etc.

Passive Testing with Network Traces
Concepts

- A property expresses a sequence of premises and consequences of non-chronologically arranged, but related network packets
  - Example: in English, if(A) then B (before or after), or if{if (A) then B(before or after)} then C (before or after), etc...
- A property must be able to characterize different packets
  - That is, to describe a template of each packet
    Example: a packet that uses TCP in the port 1010 and the first value is an ID, etc.
- A property must be able to create relationships between the different network packets

Passive Testing with Network Traces
Concepts

- A property expresses a sequence of premises and consequences of non-chronologically arranged, but related network packets
  - Example: in English, if(A) then B (before or after), or if{if (A) then B(before or after)} then C (before or after), etc...
- A property must be able to characterize different packets
  - That is, to describe a template of each packet
    Example: a packet that uses TCP in the port 1010 and the first value is an ID, etc.
- A property must be able to create relationships between the different network packets
  - That is, to describe how packet A relates to packet B (request port is equal to response port, etc.)

Passive Testing with Network Traces
Concepts (cont.)

How can we proceed to characterize packets and relationships between them?

Passive Testing with Network Traces Concepts (cont.)

How can we proceed to characterize packets and relationships between them?

- Comparisons…

Passive Testing with Network Traces Concepts (cont.)

How can we proceed to characterize packets and relationships between them?

- Comparisons…
  - Request TCP source port = response destination port?

PASSIVE TESTING WITH NETWORK TRACES
CONCEPTS (CONT.)

How can we proceed to characterize packets and relationships between them?

- Comparisons…
  - Request TCP source port = response destination port?
  - Individual comparisons are performed against constants or previous (chronological) packet values

Passive Testing with Network Traces
Concepts (cont.)

How can we proceed to characterize packets and relationships between them?

- Comparisons…
  - Request TCP source port = response destination port?
  - Individual comparisons are performed against constants or previous (chronological) packet values
  - These comparisons make relationships between packets

Passive Testing with Network Traces Concepts (cont.)

How can we proceed to characterize packets and relationships between them?

- Comparisons…
  - Request TCP source port = response destination port?
  - Individual comparisons are performed against constants or previous (chronological) packet values
  - These comparisons make relationships between packets
  - A set of individual comparisons characterizes a packet

Passive Testing with Network Traces
Concepts (cont.)

How can we proceed to characterize packets and relationships between them?

- Comparisons...
  - Request TCP source port = response destination port?
  - Individual comparisons are performed against constants or previous (chronological) packet values
  - These comparisons make relationships between packets
  - A set of individual comparisons characterizes a packet

To make individual comparisons we need granular data access

Passive Testing with Network Traces
Concepts (cont.)

How can we proceed to characterize packets and relationships between them?

- Comparisons…
  - Request TCP source port = response destination port?
  - Individual comparisons are performed against constants or previous (chronological) packet values
  - These comparisons make relationships between packets
  - A set of individual comparisons characterizes a packet

To make individual comparisons we need granular data access

- The SYN flag of the TCP header of the *i*-th the packet

Passive Testing with Network Traces
Concepts (cont.)

How can we proceed to characterize packets and relationships between them?

- Comparisons…
  - Request TCP source port = response destination port?
  - Individual comparisons are performed against constants or previous (chronological) packet values
  - These comparisons make relationships between packets
  - A set of individual comparisons characterizes a packet

To make individual comparisons we need granular data access

- The SYN flag of the TCP header of the *i*-th the packet
- Hierarchical as you can see…

Passive Testing with Network Traces Concepts (cont. cont.)

Granular data access is needed

Passive Testing with Network Traces Concepts (cont. cont.)

Granular data access is needed

- Because byte offset access is not feasible

Passive Testing with Network Traces Concepts (cont. cont.)

Granular data access is needed

- Because byte offset access is not feasible
  - Example: the value of the IHL (Internet header length) affects the offset mapping

Passive Testing with Network Traces Concepts (cont. cont.)

Granular data access is needed

- Because byte offset access is not feasible
  - Example: the value of the IHL (Internet header length) affects the offset mapping
  - The communication protocol data point to other locations (semantics of the protocol)

Passive Testing with Network Traces Concepts (cont. cont.)

Granular data access is needed

- Because byte offset access is not feasible
  - Example: the value of the IHL (Internet header length) affects the offset mapping
  - The communication protocol data point to other locations (semantics of the protocol)
- Otherwise, how to refer to the TCP SYN flag inside the TCP header of the packet?

Passive Testing with Network Traces Concepts (cont. cont.)

Granular data access is needed

- Because byte offset access is not feasible
  - Example: the value of the IHL (Internet header length) affects the offset mapping
  - The communication protocol data point to other locations (semantics of the protocol)
- Otherwise, how to refer to the TCP SYN flag inside the TCP header of the packet?

Granular data access can be achieved with hierarchical key-value structure of the packet

Passive Testing with Network Traces Concepts (cont. cont.)

Granular data access is needed

- Because byte offset access is not feasible
  - Example: the value of the IHL (Internet header length) affects the offset mapping
  - The communication protocol data point to other locations (semantics of the protocol)
- Otherwise, how to refer to the TCP SYN flag inside the TCP header of the packet?

Granular data access can be achieved with hierarchical key-value structure of the packet

- A mapping function between the raw data bytes and the structure is needed

PASSIVE TESTING WITH NETWORK TRACES CONCEPTS (CONT. CONT. CONT.)

P packet

```
...

(TCP Header)
eb5d01bbd3e75a55cfa6
e7c0801810001cd50000

...
```

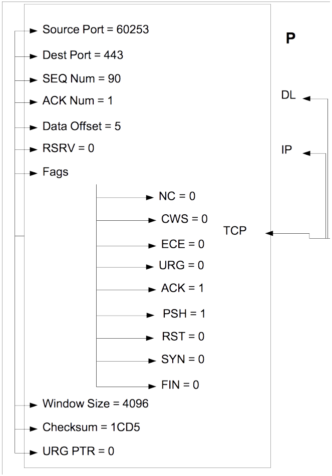
PASSIVE TESTING WITH NETWORK TRACES CONCEPTS (CONT. CONT. CONT.)

P packet

...

```
(TCP Header)
eb5d01bbd3e75a55cfa6
e7c0801810001cd50000
```

...



Passive Testing with Network Traces
Concepts (cont. cont. cont.)

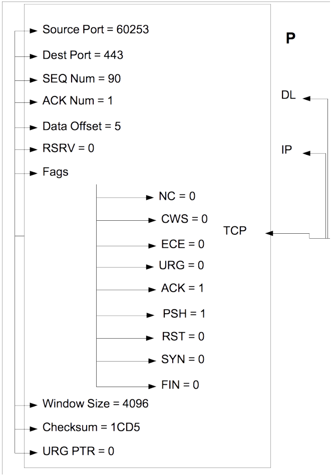
P packet

...

```
(TCP Header)
eb5d01bbd3e75a55cfa6
e7c0801810001cd50000
```

...

Accessing the ACK flag of the TCP header



Passive Testing with Network Traces
Concepts (cont. cont. cont.)

P packet

...

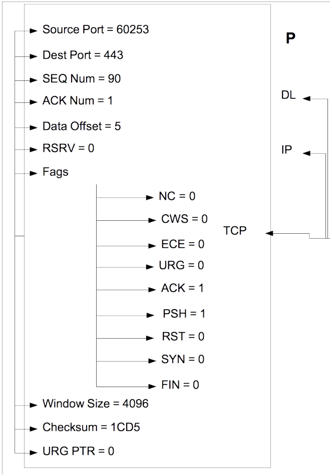
```
(TCP Header)
eb5d01bbd3e75a55cfa6
e7c0801810001cd50000
```

...

Accessing the ACK flag of the TCP header

- Given packet P, the value is 1 for

`P->TCP->flags->ACK`



Expressing invariants

Without a “formal” language

Expressing invariants

Without a "formal" language

- For each response with an even number a "corresponding" request with an odd ID should have been received

Expressing invariants

Without a “formal” language

- For each response with an even number a “corresponding” request with an odd ID should have been received
- Example:

```
if RES
(
   RES->TCP->srcP = 1010 &
   RES->VSNP->Num % 2 = 0 &
   RES->IP->srcIP = REQ->IP->dstIP &
   REQ->VSNP->ID = RES->VSNP->ID &
   REQ->VSNP->ID %2 != 0
) then REQ<RES
(
   REQ->VSNP->Num = NULL
)
```

WHY IS THIS NOT COMMERCIAL / WIDESPREAD?

WHY IS THIS NOT COMMERCIAL / WIDESPREAD?

Resource intensive!!! The industry lost interest…

Why is this not commercial / widespread?

Resource intensive!!! The industry lost interest…

What if I could identify a subset of properties to check at the current execution point?..

Why is this not commercial / widespread?

Resource intensive!!! The industry lost interest…

What if I could identify a subset of properties to check at the current execution point?..

How?

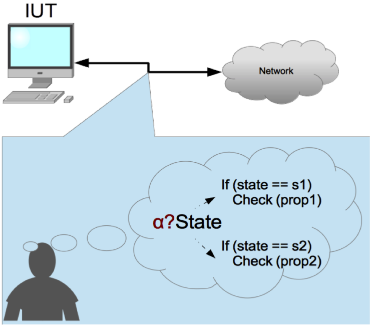
Why is this not commercial / widespread?

Resource intensive!!! The industry lost interest…

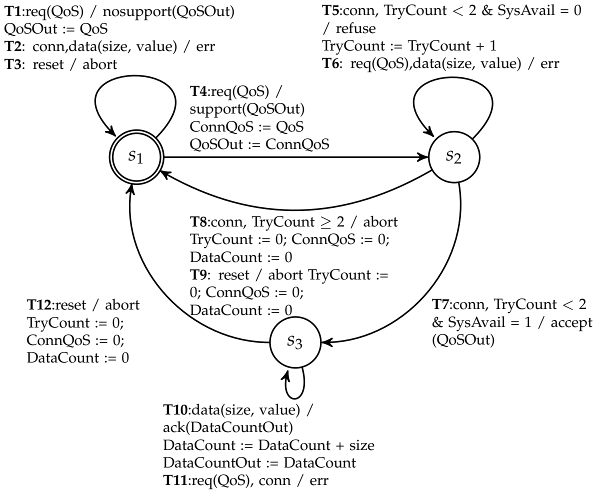
What if I could identify a subset of properties to check at the current execution point?..

How?

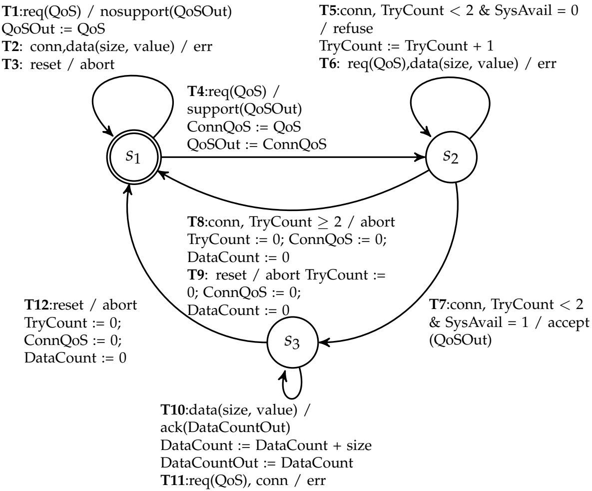
An idea: model the protocol as an FSM to identify its states?



The Simple Connection Protocol (SCP)

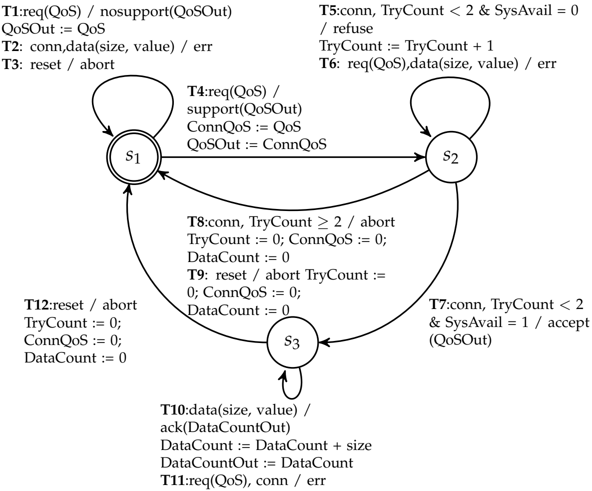


The Simple Connection Protocol (SCP)



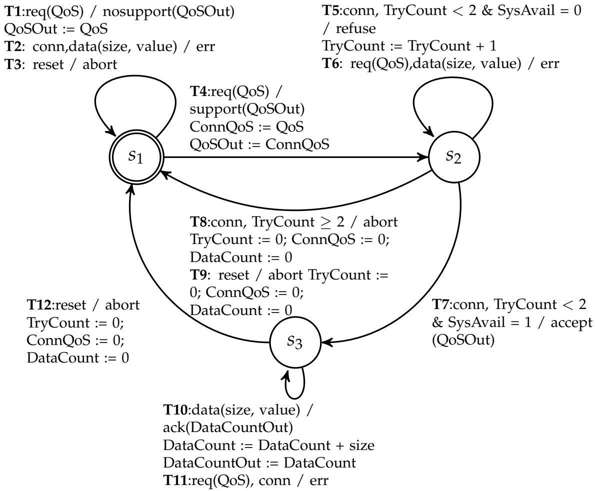
Used to transmit data from one entity (called the upper layer) to the other (called the lower layer)

The Simple Connection Protocol (SCP)



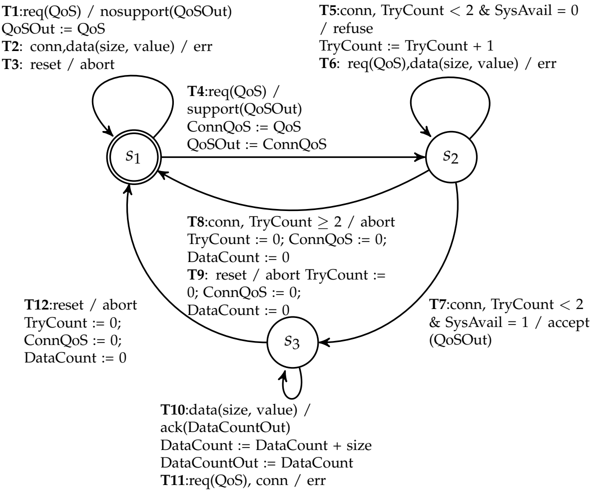
It starts with a QoS negotiation phase, the sending entity proposes the level, the receiving decides

THE SIMPLE CONNECTION PROTOCOL (SCP)



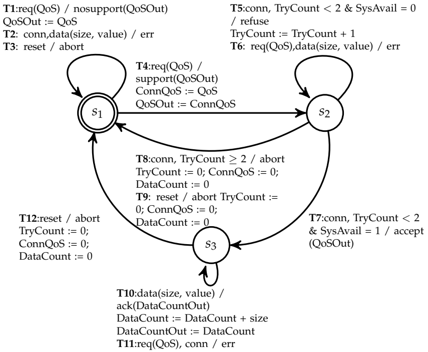
Then, the transmitting entity attempts connecting to the receiving one (up to 3 times)

The Simple Connection Protocol (SCP)



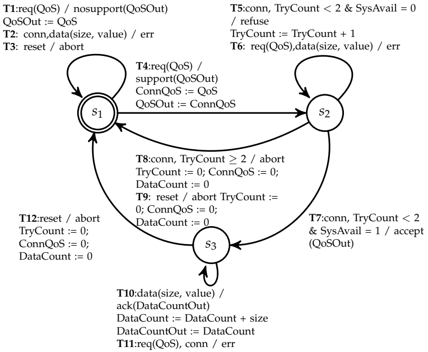
After a successful connection, the transmitting entity sends data, the receiving entity acknowledges

CHECKING SCP PROPERTIES...



Assume we want to check that:

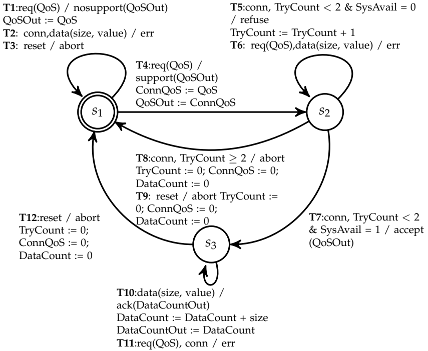
Checking SCP properties...



Assume we want to check that:

- An acknowledgment is always sent after transmitting data

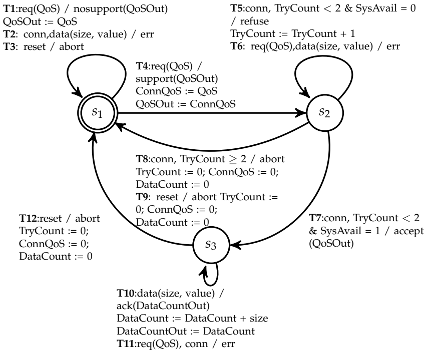
CHECKING SCP PROPERTIES...



Assume we want to check that:

- An acknowledgment is always sent after transmitting data
  Why to check for data packets when the connection has not been even established?

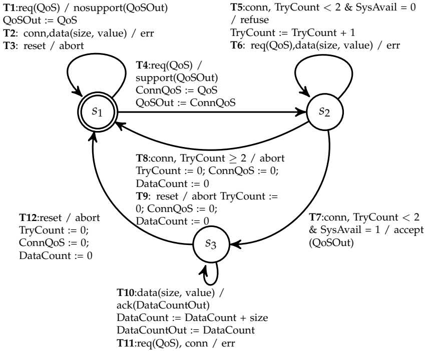
CHECKING SCP PROPERTIES...



Assume we want to check that:

- An acknowledgment is always sent after transmitting data
- After a successful connection a reset is sent

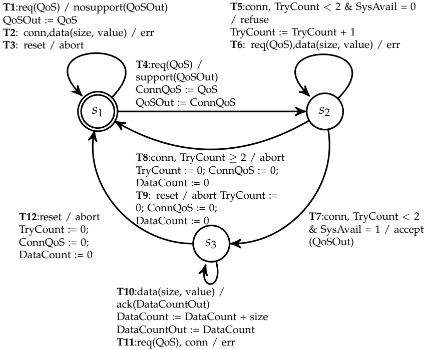
CHECKING SCP PROPERTIES...



Assume we want to check that:

- An acknowledgment is always sent after transmitting data
- After a successful connection a reset is sent
  Why to check for connection requests/replies if no QoS level has been agreed on?

CHECKING SCP PROPERTIES...



Assume we want to check that:

- An acknowledgment is always sent after transmitting data
- After a successful connection a reset is sent
- Only **relevant** properties to the current execution state!

Finite State Machine (FSM)

Do you remember?..

An FSM is a 5-tuple
$M = \langle S, I, O, h_s, S' \rangle$

Finite State Machine (FSM)

Do you remember?..

An FSM is a 5-tuple
$M = \langle S, I, O, h_s, S^{'} \rangle$

- S is a finite nonempty set of states with a non-empty subset $S^{'}$ of initial states

Finite State Machine (FSM)

Do you remember?..

An FSM is a 5-tuple
$M = \langle S, I, O, h_s, S^{'} \rangle$

- S is a finite nonempty set of states with a non-empty subset $S^{'}$ of initial states
- I and O are finite input and output alphabets

Finite State Machine (FSM)

Do you remember?..

An FSM is a 5-tuple
$M = \langle S, I, O, h_s, S^{'} \rangle$

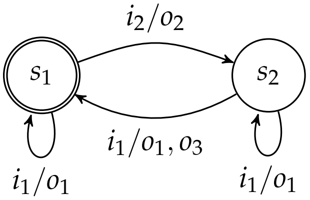
- S is a finite nonempty set of states with a non-empty subset $S^{'}$ of initial states
- I and O are finite input and output alphabets
- $h_s \subseteq S \times I \times O \times S$ is a behavior relation

Finite State Machine (FSM)

Do you remember?..

An FSM is a 5-tuple
$M = \langle S, I, O, h_s, S' \rangle$

- S is a finite nonempty set of states with a non-empty subset S' of initial states
- I and O are finite input and output alphabets
- $h_s \subseteq S \times I \times O \times S$ is a behavior relation



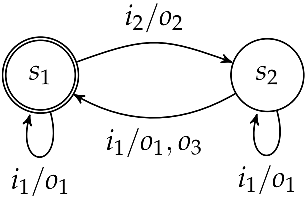
$S' = \{s_1\}$

Finite State Machine (FSM)

Do you remember?..

An FSM is a 5-tuple
$M = \langle S, I, O, h_s, S^{'} \rangle$

- S is a finite nonempty set of states with a non-empty subset $S^{'}$ of initial states
- I and O are finite input and output alphabets
- $h_s \subseteq S \times I \times O \times S$ is a behavior relation



$S^{'} = \{s_1\}$

Extended FSM (EFSM)

EFSM augments an FSM with

Extended FSM (EFSM)

EFSM augments an FSM with

- Context variables

Extended FSM (EFSM)

EFSM augments an FSM with

- Context variables
- Input and output parameters

Extended FSM (EFSM)

EFSM augments an FSM with

- Context variables
- Input and output parameters
- Predicates

Extended FSM (EFSM)

EFSM augments an FSM with

- Context variables
- Input and output parameters
- Predicates
- Update functions

Extended FSM (EFSM)

EFSM augments an FSM with

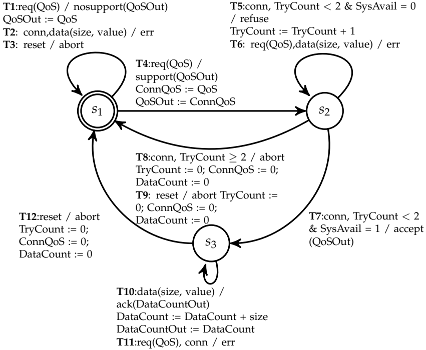
- Context variables
- Input and output parameters
- Predicates
- Update functions

A transition is executed if the corresponding predicate is true

EXTENDED FSM (EFSM)

EFSM augments an FSM with

- Context variables
- Input and output parameters
- Predicates
- Update functions



A transition is executed if the corresponding predicate is true

i and o can have parameters
Context variables are updated when a transition is executed
Predicates allow to execute the transition if it is true

Properties of an FSM $M = \langle S, I, O, h_s, S' \rangle$

An FSM can be

Properties of an FSM $M = \langle S, I, O, h_s, S' \rangle$

An FSM can be

- *deterministic* if for each pair $(s, i) \in S \times I$ there exists at most one pair $(o, s') \in O \times S$ such that $(s, i, o, s') \in h_s$, otherwise M is *nondeterministic*

Properties of an FSM $M = \langle S, I, O, h_s, S' \rangle$

An FSM can be

- *deterministic* if for each pair $(s, i) \in S \times I$ there exists at most one pair $(o, s') \in O \times S$ such that $(s, i, o, s') \in h_s$, otherwise M is *nondeterministic*
- *complete* if for each $(s, i) \in S \times I$ there exists $(o, s') \in O \times S$ such that $(s, i, o, s') \in h_s$, otherwise M is *partial*

PROPERTIES OF AN FSM $M = \langle S, I, O, h_s, S' \rangle$

An FSM can be

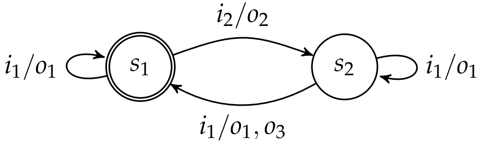
- *deterministic* if for each pair $(s, i) \in S \times I$ there exists at most one pair $(o, s') \in O \times S$ such that $(s, i, o, s') \in h_s$, otherwise M is *nondeterministic*
- *complete* if for each $(s, i) \in S \times I$ there exists $(o, s') \in O \times S$ such that $(s, i, o, s') \in h_s$, otherwise M is *partial*
- *observable* if for each triple $(s, i, o) \in S \times I \times O$ there exist at most one state $s' \in S$ such that $(s, i, o, s') \in h_s$, otherwise M is *nonobservable*

Properties of an FSM $M = \langle S, I, O, h_s, S' \rangle$

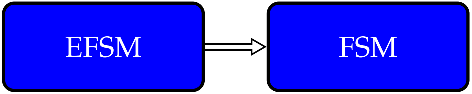
An FSM can be

- *deterministic* if for each pair $(s, i) \in S \times I$ there exists at most one pair $(o, s') \in O \times S$ such that $(s, i, o, s') \in h_s$, otherwise M is *nondeterministic*
- *complete* if for each $(s, i) \in S \times I$ there exists $(o, s') \in O \times S$ such that $(s, i, o, s') \in h_s$, otherwise M is *partial*
- *observable* if for each triple $(s, i, o) \in S \times I \times O$ there exist at most one state $s' \in S$ such that $(s, i, o, s') \in h_s$, otherwise M is *nonobservable*

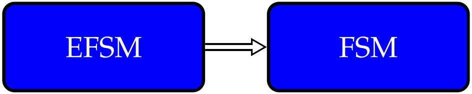
This is a partial, nondeterministic and nonobservable FSM



From an EFSM to an FSM

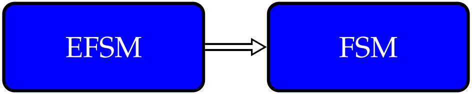


From an EFSM to an FSM



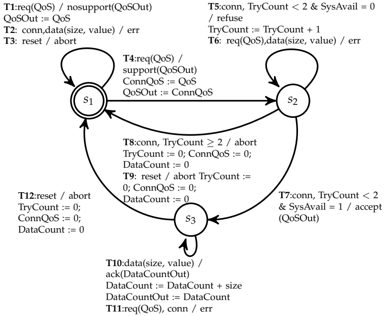
- Deleting context variables and predicates that significantly depend on them (Context-free Slice)

From an EFSM to an FSM

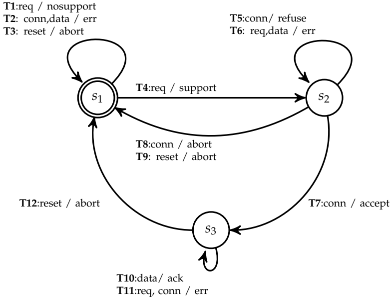


- Deleting context variables and predicates that significantly depend on them (Context-free Slice)

EFSM



FSM



Homing Sequence (HS)

Homing Sequence (HS)

- The sequence α allows to conclude about the final state s_i' trough the observation of β_i (the output reaction)

Homing Sequence (HS)

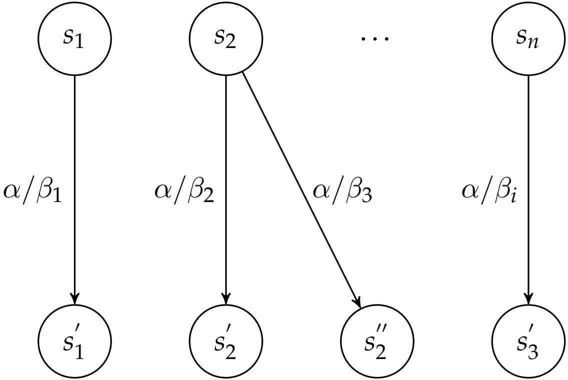
- The sequence α allows to conclude about the final state s_i' trough the observation of β_i (the output reaction)
- After applying α at any state $s_i \in S$ the final state s_i' becomes known, depending on the observed β_i

Homing Sequence (HS)

- The sequence α allows to conclude about the final state s_i' trough the observation of β_i (the output reaction)
- After applying α at any state $s_i \in S$ the final state s_i' becomes known, depending on the observed β_i
- **α is a homing sequence**

Homing Sequence (HS)

- The sequence α allows to conclude about the final state $s_i^{'}$ trough the observation of β_i (the output reaction)
- After applying α at any state $s_i \in S$ the final state $s_i^{'}$ becomes known, depending on the observed β_i
- **α is a homing sequence**

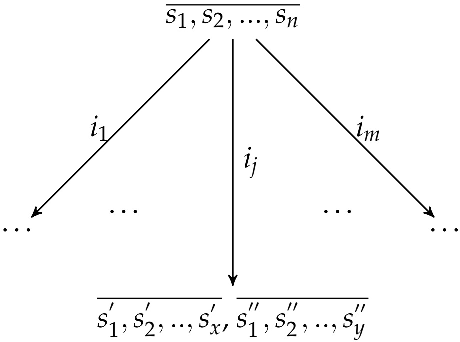


If $\beta_i = \beta_j \implies s_i^{'} = s_j^{'}$ (For observable FSMs)

Deriving ALL (non-redundant) HS of length l

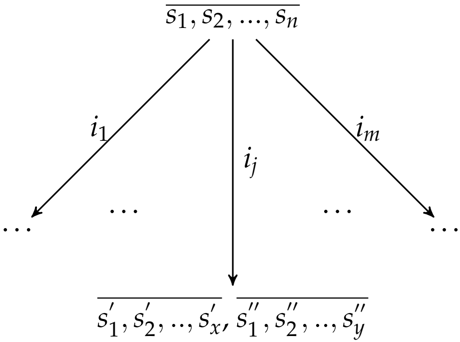
Deriving ALL (non-redundant) HS of length l

- Derive a **Truncated Successor Tree (TST)** $\exists o \in ((s_1, i_j, o, s_1^{'}) \in h_s \& (s_2, i_j, o, s_2^{'}) \in h_s \& (s_3, i_j, o, s_3^{'}) \in h_s \ldots)$



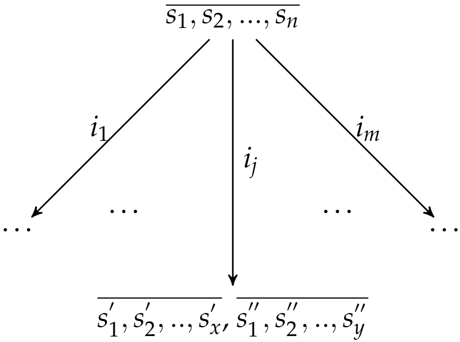
Deriving ALL (non-redundant) HS of length l

- Derive a **Truncated Successor Tree (TST)** $\exists o \in ((s_1, i_j, o, s_1^{'}) \in h_s \& (s_2, i_j, o, s_2^{'}) \in h_s \& (s_3, i_j, o, s_3^{'}) \in h_s \ldots)$
- **Truncating rules**
  - **Rule 1** The node P has only singletons



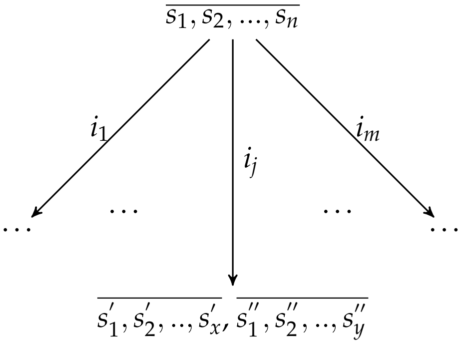
DERIVING ALL (NON-REDUNDANT) HS OF LENGTH l

- Derive a **Truncated Successor Tree (TST)** $\exists o \in ((s_1, i_j, o, s_1') \in h_s \& (s_2, i_j, o, s_2') \in h_s \& (s_3, i_j, o, s_3') \in h_s \ldots)$
- **Truncating rules**
  - **Rule 1** The node P has only singletons
  - **Rule 2** The depth of the node P is greater than l



DERIVING ALL (NON-REDUNDANT) HS OF LENGTH l

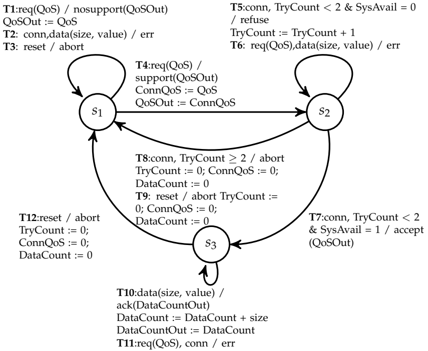
- Derive a **Truncated Successor Tree (TST)** $\exists o \in ((s_1, i_j, o, s_1^{'}) \in h_s \& (s_2, i_j, o, s_2^{'}) \in h_s \& (s_3, i_j, o, s_3^{'}) \in h_s \ldots)$
- **Truncating rules**
  - **Rule 1** The node P has only singletons
  - **Rule 2** The depth of the node P is greater than l



α is a homing sequence iff it labels the path truncated by **Rule 1**

Applying state identification to SCP

Applying state identification to SCP

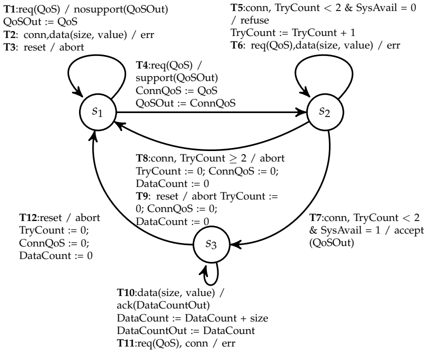


The resulting set of HSs of length $l = 2$

$\{(req.reset), (req.conn), (req.data), (reset), (data.rec), (data.conn),$
$data.reset), (conn.req), (conn.data), (conn.reset)\}$

APPLYING STATE IDENTIFICATION TO SCP

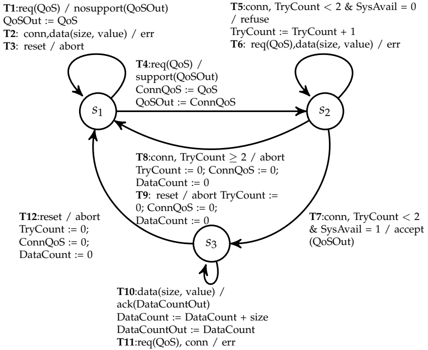
Let's assume $\alpha = (req.conn)$



APPLYING STATE IDENTIFICATION TO SCP

Let's assume $\alpha = (req.conn)$

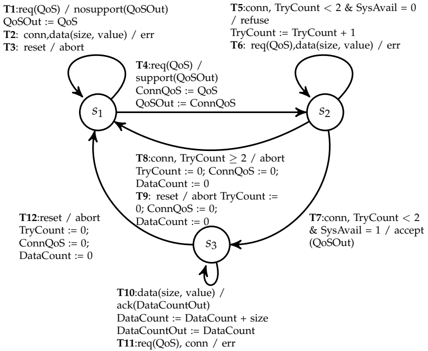
- If $\beta \in \{(nosupport.err), (err.abort)\}$, then current state is s_1, check no properties



Applying state identification to SCP

Let's assume $\alpha = (req.conn)$

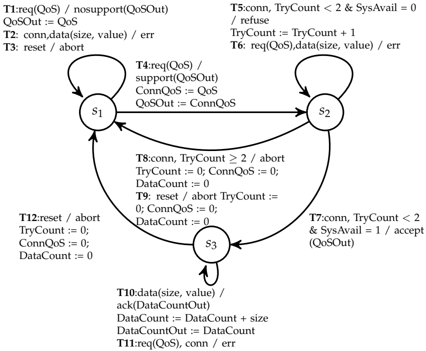
- If $\beta \in \{(nosupport.err), (err.abort)\}$, then current state is s_1, check no properties
- If $\beta \in \{(support.refuse), (err.refuse)\}$, then current state is s_2, check for $req/conn$



Applying state identification to SCP

Let's assume $\alpha = (req.conn)$

- If $\beta \in \{(nosupport.err), (err.abort)\}$, then current state is s_1, check no properties
- If $\beta \in \{(support.refuse), (err.refuse)\}$, then current state is s_2, check for $req/conn$
- Else current state is s_3 check for $data/ack$ + $reset/abort$ (small heuristic)



State Identification for Network Trace Analysis — Final Remarks

What if the TST up to the length l does not contain a HS?..

State Identification for Network Trace Analysis — Final Remarks

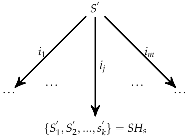
What if the TST up to the length l does not contain a HS?..

- Continue with your TST with $l = l + 1$

State Identification for Network Trace Analysis — Final Remarks

What if the TST up to the length l does not contain a HS?..

- Continue with your TST with $l = l + 1$

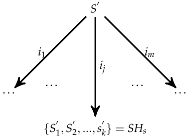


- Or perhaps not... If $\forall S'_j \in SH_s, |S'| >> |S'_i|$
  Only check properties for the identified subset!

State Identification for Network Trace Analysis — Final Remarks

What if the TST up to the length l does not contain a HS?..

- Continue with your TST with $l = l + 1$



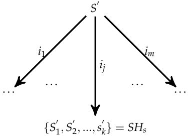
- Or perhaps not... If $\forall S'_j \in SH_s, |S'| >> |S'_i|$
  Only check properties for the identified subset!

What sequences to choose?

State Identification for Network Trace Analysis — Final Remarks

What if the TST up to the length l does not contain a HS?..

- Continue with your TST with $l = l + 1$



- Or perhaps not... If $\forall S'_j \in SH_s, |S'| >> |S'_i|$
  Only check properties for the identified subset!

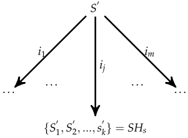
What sequences to choose?

- Sequences that are frequent to observe!

State Identification for Network Trace Analysis — Final Remarks

What if the TST up to the length l does not contain a HS?..

- Continue with your TST with $l = l + 1$



- Or perhaps not... If $\forall S'_j \in SH_s, |S'| >> |S'_i|$
  Only check properties for the identified subset!

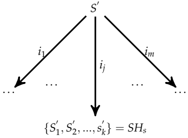
What sequences to choose?

- Sequences that are frequent to observe!
- Sequences that follow the protocol flow, e.g., not $(conn.req)$

State Identification for Network Trace Analysis — Final Remarks

What if the TST up to the length l does not contain a HS?..

- Continue with your TST with $l = l + 1$



- Or perhaps not... If $\forall S'_j \in SH_s, |S'| >> |S'_i|$
  Only check properties for the identified subset!

What sequences to choose?

- Sequences that are frequent to observe!
- Sequences that follow the protocol flow, e.g., not $(conn.req)$
- Can be chosen by experimental evaluation

PASSIVE TESTING USING NETWORK TRACES – FINAL REMARKS

Passive Testing using Network Traces – Final Remarks

- Very useful when: no access to the code is possible, real data analysis is desirable, the system cannot be influenced by test cases

Passive Testing using Network Traces – Final Remarks

- Very useful when: no access to the code is possible, real data analysis is desirable, the system cannot be influenced by test cases
- It can be computationally expensive, but techniques to reduce the complexity are being actively studied

Passive Testing using Network Traces – Final Remarks

- Very useful when: no access to the code is possible, real data analysis is desirable, the system cannot be influenced by test cases
- It can be computationally expensive, but techniques to reduce the complexity are being actively studied
- The approach was presented with network traces, but it can be applied to passive testing of other software/hardware (embedded) systems

Future work / Conclusions

Research Opportunities / Future Work

Static Code Analysis

Research Opportunities / Future Work

Static Code Analysis

- Reducing the complexity when analyzing

Research Opportunities / Future Work

Static Code Analysis

- Reducing the complexity when analyzing
  - New methods / Heuristics

Research Opportunities / Future Work

Static Code Analysis

- Reducing the complexity when analyzing
  - New methods / Heuristics
- New analysis approaches using different algebraic structures

Research Opportunities / Future Work

Static Code Analysis

- Reducing the complexity when analyzing
  - New methods / Heuristics
- New analysis approaches using different algebraic structures

Passive Testing using Network Traces

Research Opportunities / Future Work

Static Code Analysis

- Reducing the complexity when analyzing
  - New methods / Heuristics
- New analysis approaches using different algebraic structures

Passive Testing using Network Traces

- Reducing the complexity when checking properties[2] (!)

RESEARCH OPPORTUNITIES / FUTURE WORK

Static Code Analysis

- Reducing the complexity when analyzing
  - New methods / Heuristics
- New analysis approaches using different algebraic structures

Passive Testing using Network Traces

- Reducing the complexity when checking properties[2] (!)
  - Through simplifying the checking algorithms

RESEARCH OPPORTUNITIES / FUTURE WORK

Static Code Analysis

- Reducing the complexity when analyzing
  - New methods / Heuristics
- New analysis approaches using different algebraic structures

Passive Testing using Network Traces

- Reducing the complexity when checking properties[2] (!)
  - Through simplifying the checking algorithms
  - Through minimizing the number checks :)

Research Opportunities / Future Work

Static Code Analysis

- Reducing the complexity when analyzing
  - New methods / Heuristics
- New analysis approaches using different algebraic structures

Passive Testing using Network Traces

- Reducing the complexity when checking properties[2] (!)
  - Through simplifying the checking algorithms
  - Through minimizing the number checks :)
- Considering:
  Synchronizing distributed traces…
  Encrypted protocol testing…

Research Opportunities / Future Work

Static Code Analysis

- Reducing the complexity when analyzing
  - New methods / Heuristics
- New analysis approaches using different algebraic structures

Passive Testing using Network Traces

- Reducing the complexity when checking properties[2] (!)
  - Through simplifying the checking algorithms
  - Through minimizing the number checks :)
- Considering:
  Synchronizing distributed traces…
  Encrypted protocol testing…
- Developing efficient tools…

End

Thank you for your attention!

Q&A / SOME CLARIFICATIONS

Ariadna Barinova from ITMO asked about the applicability of static code analysis to scripting languages

In the example, we saw how static analysis can be used for sign analysis and detected a negative array index. For scripting languages this is not important as array indexes can be anything they are not the offset of a memory address. However, sign analysis can reveal errors in the code, e.g., a function that should guarantee a positive return value does not, etc. Furthermore, many other analyses can be useful, like the live variables analysis to detect potential waste of resources, etc. I hope the explanation was clear during the presentation, or at least it is now :)

Q&A / Some clarifications (2)

Natalia Kushik from Télécom SudParis pointed out the fact that the extended finite state machine under experiment to obtain a homing sequence had an initial state

In fact, as she correctly pointed out, if the initial state is known, there is no need for the entire experiment. My intention was to show a communication protocol description; this protocol description has an initial state in fact. However, it is theoretically incorrect depict an initial state when performing the state identification experiment. These slides were corrected and updated taking this into consideration.

Thank you, Natalia :)