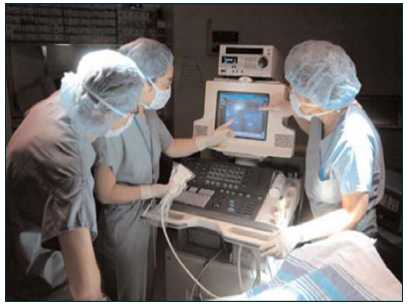
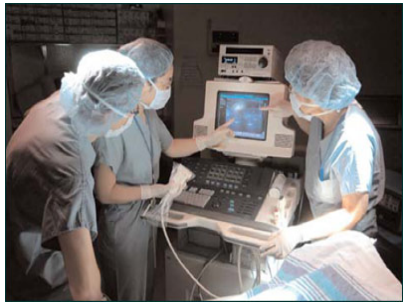
Improving Software Quality using Automatic Invariant Discovery and Program Repair

ThanhVu (Vu) Nguyen

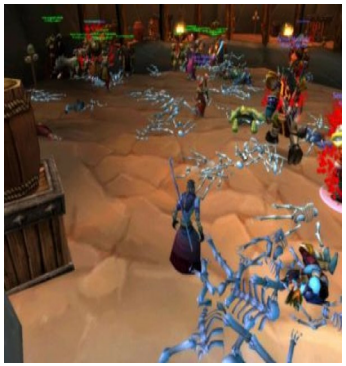



A380
A380 AIRBUS





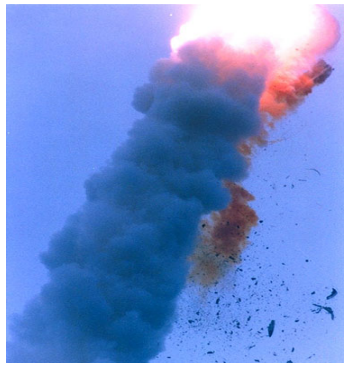

Software Bugs



World of Warcraft bug

Therac-25 machines

X-rays overdose



Ariane-5 rocket self-destructs

North America blackout

– Mozilla Developer

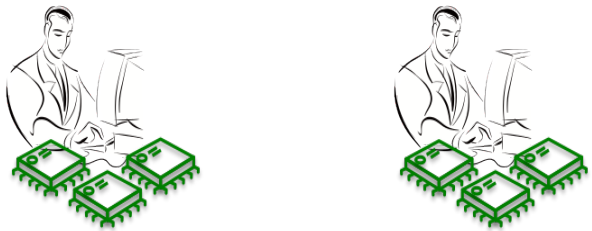
"Everyday, almost 300 bugs appear [..] far too many for only the Mozilla programmers to handle."

Software bugs annually cost 0.6% of the U.S GDP and $312 billion to the global economy

Average time to fix a security-critical error: 28 days

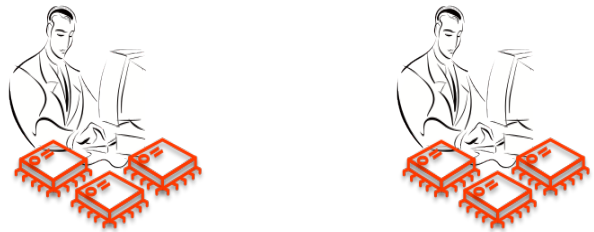
Automated program analysis techniques and tools can decrease debugging time by an average of 26% and $41 billion annually

Program Verification



Check if a program meets a given specification

Program Repair



Fix a buggy program to satisfy a given specification

Invariant Generation

```
def intdiv(x, y):
  q = 0
  r = x
  while r ≥ y:
    a = 1
    b = y
    while [??] r ≥ 2b:
      a = 2a
      b = 2b
    r = r - b
    q = q + a
    [??]
    return q
```

- Discover invariant properties at certain program locations
- Answer the question *"what does this program do ?"*

Invariant Generation

```
def intdiv(x, y):
  q = 0
  r = x
  while r ≥ y:
    a = 1
    b = y
    while [??] r ≥ 2b:
      a = 2a
      b = 2b
    r = r - b
    q = q + a
    [??]
    return q
```

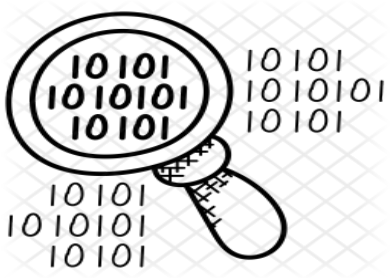
- Discover invariant properties at certain program locations
- Answer the question *"what does this program do ?"*

Automatic Program Repair

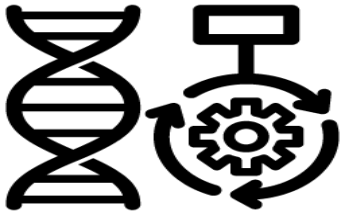
```
def intdiv(x, y):
  q = 0
  r = x
  while r ≥ y:   (≥ → ≠)
    a = 1
    b = y   (y → 3*y)
    while r ≥ 2b:
      a = 2a
      b = 2b
    r = r - b
    q = q + a   (+a → −2*a)
  return q
```

- Localize errors and modify code to fix bugs
- A form of *program synthesis*

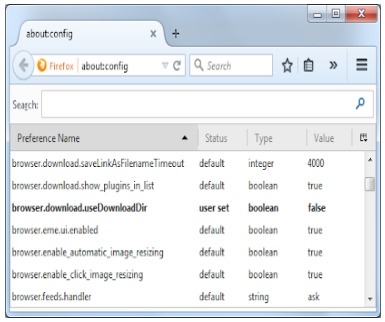
Core Research Areas



Invariant Discovery

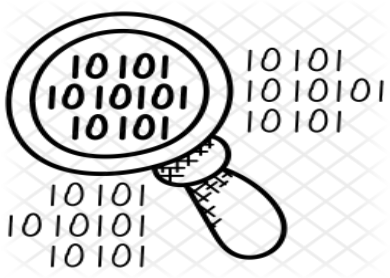


Program Repair

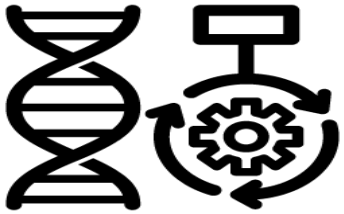


Configurable System

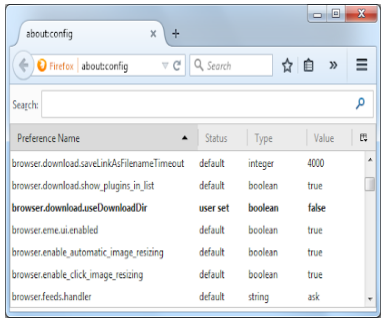
Core Research Areas



Invariant Discovery



Program Repair



Configurable System

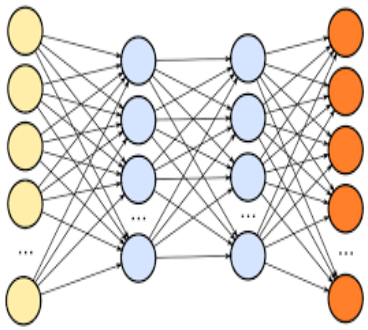
New Research Directions

makefile

```
RM := rm -rf
TARGET := hello
OBJS := hello.o
SRCS := hello.c

all: $(TARGET)

$(TARGET): $(OBJS) $(SRCS)
        @echo 'Bulding ' $(TARGET)
        @gcc -o $(TARGET) $(OBJS)
        @echo 'Built Successfully'

%.o: %.c
        @echo 'building $@ from $<'
        @gcc -o $@ -c $<

clean:
        $(RM) $(OBJS) $(TARGET)
```

Unix Build Systems

IoT systems

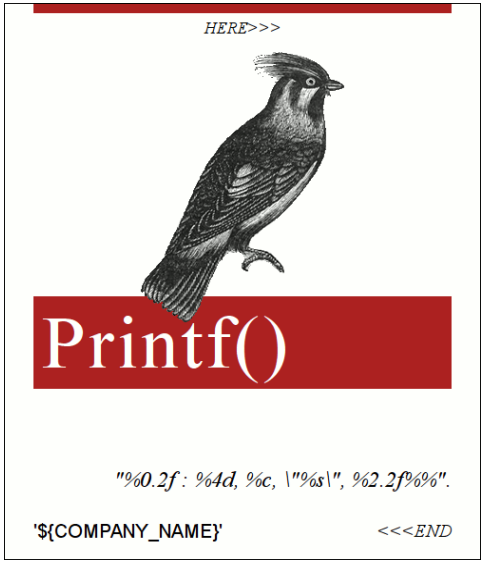


AI-generated Software

Outline

- SE/PL Research
  - Invariant Generation
  - Automatic Program Repair
  - Highly-Configurable Systems
- Current/New Research Works

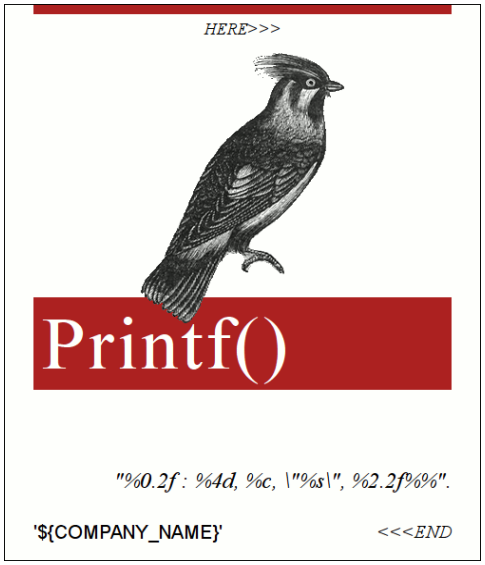
How We Analyze Programs




```
File  Edit  Options  Buffers  Tools  Help

def intdiv(x, y):

    q = 0
    r = x
    while r >= y:
        a = 1
        b = y

        while r >= 2*b:
            a = 2 * a
            b = 2 * b
        r = r - b
        q = q + a

    print "x %d, y %d, q %d, r %d" %(x,y,q,r)
    return q,r

-U:---  intdiv.py      All (18,0)     (Python)--5:
```

```
x   0,  y   1,  q   0,  r  0
x   1,  y   1,  q   1,  r  0
x   1,  y   5,  q   0,  r  1
x   1,  y  10,  q   0,  r  1
x   3,  y   1,  q   3,  r  0
x   3,  y   4,  q   0,  r  3
x   3,  y   7,  q   0,  r  3
x   8,  y   1,  q   8,  r  0
x   8,  y   2,  q   4,  r  0
x   8,  y   9,  q   0,  r  8
x   8,  y  10,  q   0,  r  8
x  15,  y   1,  q  15,  r  0
x  15,  y   5,  q   3,  r  0
x  15,  y   7,  q   2,  r  1
x  20,  y   2,  q  10,  r  0
x  20,  y   7,  q   2,  r  6
x  20,  y  10,  q   2,  r  0
x 100,  y   1,  q 100,  r  0
x 100,  y   5,  q  20,  r  0
x 100,  y  10,  q  10,  r  0

-U:---  intdiv.traces   All (21,0)
```

How We Analyze Programs




```
File  Edit  Options  Buffers  Tools  Help

def intdiv(x, y):

    q = 0
    r = x
    while r >= y:
        a = 1
        b = y

        while r >= 2*b:
            a = 2 * a
            b = 2 * b
        r = r - b
        q = q + a

    print "x %d, y %d, q %d, r %d" %(x,y,q,r)
    return q,r

-U:---  intdiv.py      All (18,0)      (Python)--5:
```

```
-U:---  intdiv.traces   All (21,0)
x   0,  y   1,  q   0,  r  0
x   1,  y   1,  q   1,  r  0
x   1,  y   5,  q   0,  r  1
x   1,  y  10,  q   0,  r  1
x   3,  y   1,  q   3,  r  0
x   3,  y   4,  q   0,  r  3
x   3,  y   7,  q   0,  r  3
x   8,  y   1,  q   8,  r  0
x   8,  y   2,  q   4,  r  0
x   8,  y   9,  q   0,  r  8
x   8,  y  10,  q   0,  r  8
x  15,  y   1,  q  15,  r  0
x  15,  y   5,  q   3,  r  0
x  15,  y   7,  q   2,  r  1
x  20,  y   2,  q  10,  r  0
x  20,  y   7,  q   2,  r  6
x  20,  y  10,  q   2,  r  0
x 100,  y   1,  q 100,  r  0
x 100,  y   5,  q  20,  r  0
x 100,  y  10,  q  10,  r  0
```

```
File  Edit  Options  Buffers  Tools  Python  Help

def intdiv(x, y):
    assert y != 0

    # .. compute result ..

    assert r >= 0
    assert x >= q

    return q,r

--:---  intdiv.py      All (11,0)
```

UDACITY – Software Testing course

"GCC: 9000 assertions,
LLVM: 13,000 assertions [..]
1 assertion per 110 loc"

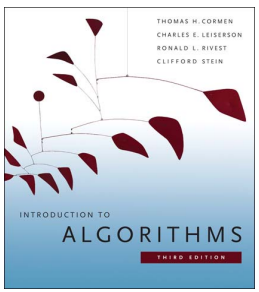
"program invariants are asserted properties, such as relations among variables, at certain locations in a program"


```
assert(x == 2*y);
assert(0 <= idx < |arr|);
```

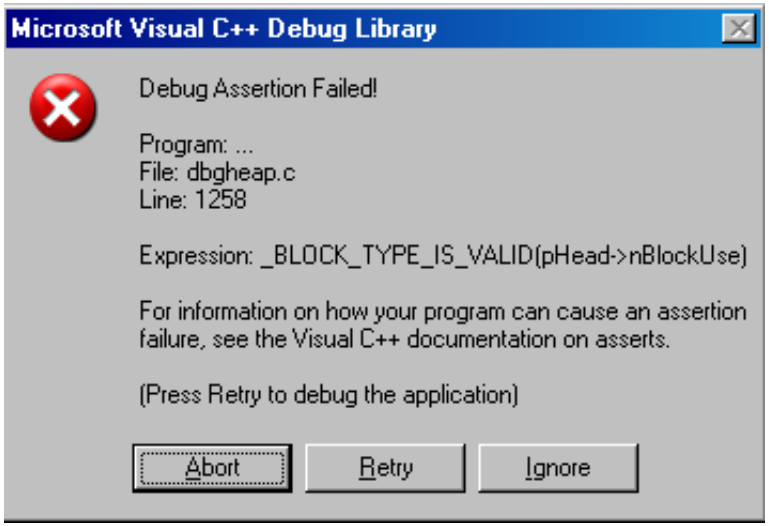
"program invariants are asserted properties, such as relations among variables, at certain locations in a program"


```
assert(x == 2*y);
assert(0 <= idx < |arr|);
```


```
int getDateOfMonth(int m){
   /*pre: 1 <= m <= 12*/
   ...
   /*post: 0 <= result <= 31*/
}
```



"a loop invariant is a condition that is true on entry into a loop and is guaranteed to remain true on every iteration of the loop [..]"



Uses

- Understand and verify programs
- Formal proofs
- Debug (locate errors)
- Documentations

Approaches to Finding Invariants

```
int intdiv(int x, int y){
  int q=0; int r=x;
  while(r ≥ y){
    int a=1; int b=y;
    while[L](r ≥ 2*b){
      a = 2*a; b = 2*b;
    }
    r=r-b; q=q+a;
  }
  return q;
}
```

Static Analysis

- Analyze source code directly
- Pros: guaranteed results
- Cons: computationally intensive, infer simple invariants

Approaches to Finding Invariants

```
int intdiv(int x, int y){
  int q=0; int r=x;
  while(r ≥ y){
    int a=1; int b=y;
    while[L](r ≥ 2*b){
      a = 2*a; b = 2*b;
    }
    r=r-b; q=q+a;
  }
  return q;
}
```

x	y	q	r
0	1	0	0
1	1	1	0
3	4	0	3
8	1	8	0
15	5	3	0
20	2	10	0
100	1	100	0
	⋮	⋮	

Static Analysis

- Analyze source code directly
- Pros: guaranteed results
- Cons: computationally intensive, infer simple invariants

Dynamic Analysis

- Analyze program traces
- Pros: fast, source code not required
- Cons: results depend on traces, might not hold for all runs

Example

```
int intdiv(int x, int y){
  assert(x>0 && y>0);
  int q=0; int r=x;
  while(r ≥ y){
    int a=1;
    int b=y;
    while[L](r ≥ 2*b){
      a = 2*a;
      b = 2*b;
    }
    r=r-b;
    q=q+a;
  }
  return q;
}
```

Example

```
int intdiv(int x, int y){
  assert(x>0 && y>0);
  int q=0; int r=x;
  while(r ≥ y){
    int a=1;
    int b=y;
    while[L](r ≥ 2*b){
      a = 2*a;
      b = 2*b;
    }
    r=r-b;
    q=q+a;
  }
  return q;
}
```

x	y	a	b	q	r
15	2	1	2	0	15
15	2	2	4	0	15
15	2	1	2	4	7
4	1	1	1	0	4
4	1	2	2	0	4

Invariants at L: $b = ya,\ x = qy + r,\ r \geq 2ya$

DIG discovers polynomial relations of the forms

Equalities	$c_0 + c_1 x_1 + c_2 x_n + c_3 x_1 x_2 + \cdots + c_m x_1^{d_1} \ldots x_n^{d_n} = 0$
Inequalities	$c_0 + c_1 x_1 + c_2 x_n + c_3 x_1 x_2 + \cdots + c_m x_1^{d_1} \ldots x_n^{d_n} \geq 0, \quad c_i \in \mathbb{R}$

Examples

cubic	$z - 6n = 6,\ \frac{1}{12} z^2 - y - \frac{1}{2} z = -1$
extended gcd	$\gcd(a, b) = ia + jb$
sqrt	$x + \varepsilon \geq y^2 \geq x - \varepsilon$

DIG discovers polynomial relations of the forms

Equalities $c_0 + c_1 x_1 + c_2 x_n + c_3 x_1 x_2 + \cdots + c_m x_1^{d_1} \ldots x_n^{d_n} = 0$

Inequalities $c_0 + c_1 x_1 + c_2 x_n + c_3 x_1 x_2 + \cdots + c_m x_1^{d_1} \ldots x_n^{d_n} \geq 0, \quad c_i \in \mathbb{R}$

Examples

cubic	$z - 6n = 6,\ \frac{1}{12} z^2 - y - \frac{1}{2} z = -1$
extended gcd	$\gcd(a, b) = ia + jb$
sqrt	$x + \varepsilon \geq y^2 \geq x - \varepsilon$

Method

- Equalities: solve equations
- Inequalities: construct polyhedra

Example

```
int intdiv(int x, int y){
  assert(x>0 && y>0);
  int q=0; int r=x;
  while(r ≥ y){
    int a=1;
    int b=y;
    while[L](r ≥ 2*b){
      a = 2*a;
      b = 2*b;
    }
    r=r-b;
    q=q+a;
  }
  return q;
}
```

x	y	a	b	q	r
15	2	1	2	0	15
15	2	2	4	0	15
15	2	1	2	4	7
4	1	1	1	0	4
4	1	2	2	0	4

Invariants at L: $b = ya,\ x = qy + r,\ r \geq 2ya$

Finding Nonlinear Equations using Linear Equation Solving

x	y	a	b	q	r
15	2	1	2	0	15
15	2	2	4	0	15
15	2	1	2	4	7
4	1	1	1	0	4
4	1	2	2	0	4

Finding Nonlinear Equations using Linear Equation Solving

- Terms and degrees

$$V = \{r, y, a\};\ \deg = 2$$

$$\downarrow$$

$$T = \{1, r, y, a, ry, ra, ya, r^2, y^2, a^2\}$$

x	y	a	b	q	r
15	2	1	2	0	15
15	2	2	4	0	15
15	2	1	2	4	7
4	1	1	1	0	4
4	1	2	2	0	4

Finding Nonlinear Equations using Linear Equation Solving

- Terms and degrees

$$V = \{r, y, a\};\ \deg = 2$$

$$\downarrow$$

$$T = \{1, r, y, a, ry, ra, ya, r^2, y^2, a^2\}$$

$$T = \{\ldots, \log(r), a^y, \sin(y), \ldots\}$$

x	y	a	b	q	r
15	2	1	2	0	15
15	2	2	4	0	15
15	2	1	2	4	7
4	1	1	1	0	4
4	1	2	2	0	4

Finding Nonlinear Equations using Linear Equation Solving

- Terms and degrees

$$V = \{r, y, a\};\ \deg = 2$$

$$\downarrow$$

$$T = \{1, r, y, a, ry, ra, ya, r^2, y^2, a^2\}$$

x	y	a	b	q	r
15	2	1	2	0	15
15	2	2	4	0	15
15	2	1	2	4	7
4	1	1	1	0	4
4	1	2	2	0	4

- Nonlinear equation template

$$c_1 + c_2 r + c_3 y + c_4 a + c_5 ry + c_6 ra + c_7 ya + c_8 r^2 + c_9 y^2 + c_{10} a^2 = 0$$

Finding Nonlinear Equations using Linear Equation Solving

- Terms and degrees

$$V = \{r, y, a\};\ \deg = 2$$

$$\downarrow$$

$$T = \{1, r, y, a, ry, ra, ya, r^2, y^2, a^2\}$$

x	y	a	b	q	r
15	**2**	**1**	2	0	**15**
15	2	2	4	0	15
15	2	1	2	4	7
4	1	1	1	0	4
4	1	2	2	0	4

- Nonlinear equation template

$$c_1 + c_2 r + c_3 y + c_4 a + c_5 ry + c_6 ra + c_7 ya + c_8 r^2 + c_9 y^2 + c_{10} a^2 = 0$$

- System of linear equations

$$\begin{aligned} \text{trace 1} &: \{r = 15, y = 2, a = 1\} \\ \text{eq 1} &: c_1 + 15c_2 + 2c_3 + c_4 + 30c_5 + 15c_6 + 2c_7 + 225c_8 + 4c_9 + c_{10} = 0 \\ &\vdots \end{aligned}$$

Finding Nonlinear Equations using Linear Equation Solving

- Terms and degrees

$$V = \{r, y, a\};\ \deg = 2$$
$$\downarrow$$
$$T = \{1, r, y, a, ry, ra, ya, r^2, y^2, a^2\}$$

x	y	a	b	q	r
15	2	1	2	0	15
15	2	2	4	0	15
15	2	1	2	4	7
4	1	1	1	0	4
4	1	2	2	0	4

- Nonlinear equation template

$$c_1 + c_2 r + c_3 y + c_4 a + c_5 ry + c_6 ra + c_7 ya + c_8 r^2 + c_9 y^2 + c_{10} a^2 = 0$$

- System of linear equations

$$\begin{aligned} \text{trace 1} &: \{r = 15, y = 2, a = 1\} \\ \text{eq 1} &: c_1 + 15c_2 + 2c_3 + c_4 + 30c_5 + 15c_6 + 2c_7 + 225c_8 + 4c_9 + c_{10} = 0 \\ &\vdots \end{aligned}$$

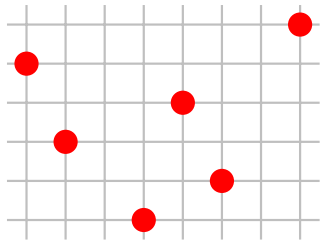
- Solve for coefficients c_i

$$V = \{x, y, a, b, q, r\};\ \deg = 2 \quad \longrightarrow \quad b = ya, x = qy + r$$

Geometric Invariant Inference

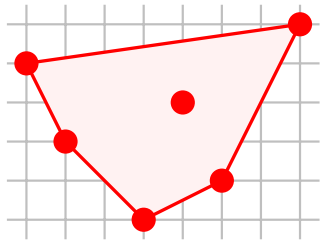
x	y
-2	1
-1	-1
1	-3
2	0
3	-2
5	2

program traces



trace points

in 2D



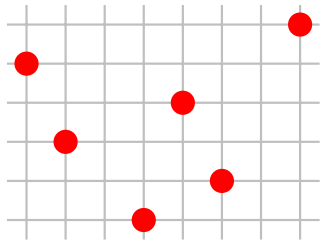
polygon

$c_1 x + c_2 y \geq c$

Geometric Invariant Inference

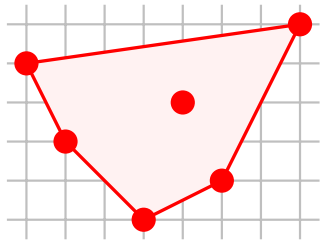
x	y
-2	1
-1	-1
1	-3
2	0
3	-2
5	2

program traces



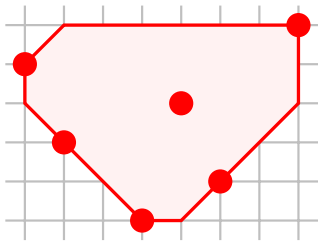
trace points

in 2D



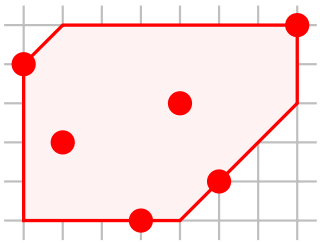
polygon

$c_1x + c_2y \geq c$



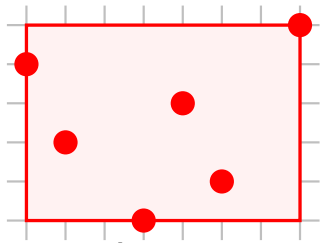
octagon

$\pm x \pm y \geq c$



zone

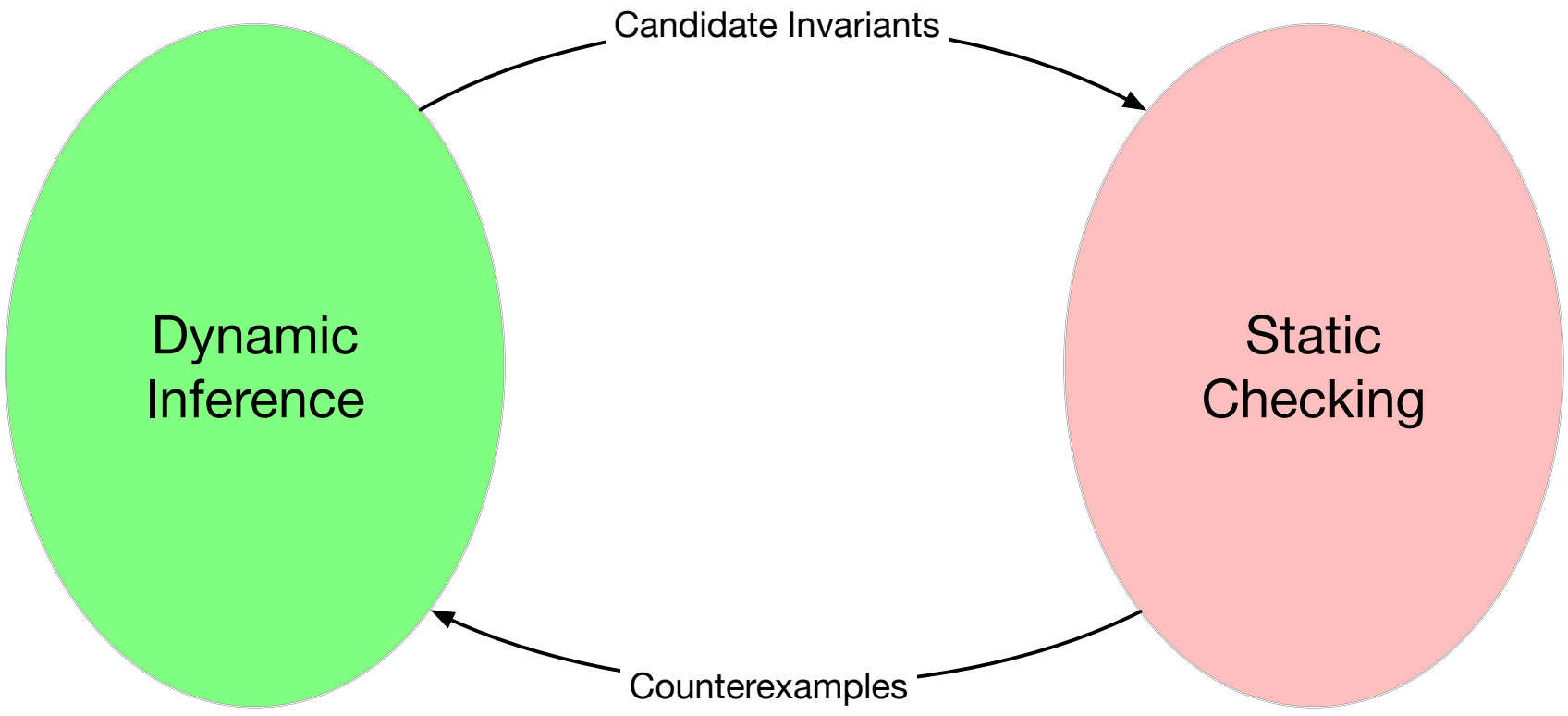
$x - y \geq c$



box

$\pm x, y \geq c$

Iterative Invariant Generation



- Dynamic Analysis: learn *candidate* invariants from execution traces
- Static Analysis: use *theorem proving* and *constraint solving* to check invariants against program code and return *counterexample inputs*

Complexity Analysis

```
void triple(int M, int N, int P){
  assert (0 <= M);
  assert (0 <= N);
  assert (0 <= P);
  int i = 0, j = 0, k = 0;
  int t = 0;
  while(i < N){
    j = 0; t++;
    while(j < M){
      j++; k = i; t++;
      while (k < P){
        k++; t++;
      }
      i = k;
    }
    i++;
  }
  [L]
}
```

Complexity of this program?

- Use t to count loop iterations

Complexity Analysis

```
void triple(int M, int N, int P){
  assert (0 <= M);
  assert (0 <= N);
  assert (0 <= P);
  int i = 0, j = 0, k = 0;
  int t = 0;
  while(i < N){
    j = 0; t++;
    while(j < M){
      j++; k = i; t++;
      while (k < P){
        k++; t++;
      }
      i = k;
    }
    i++;
  }
  [L]
}
```

Complexity of this program?

- Use t to count loop iterations
- At first glance: $t = O(MNP)$

Complexity Analysis

```
void triple(int M, int N, int P){
  assert (0 <= M);
  assert (0 <= N);
  assert (0 <= P);
  int i = 0, j = 0, k = 0;
  int t = 0;
  while(i < N){
    j = 0; t++;
    while(j < M){
      j++; k = i; t++;
      while (k < P){
        k++; t++;
      }
      i = k;
    }
    i++;
  }
  [L]
}
```

Complexity of this program?

- Use t to count loop iterations
- At first glance: $t = O(MNP)$
- DIg found an interesting (and unexpected) nonlinear invariant at L:

$$P^2Mt + PM^2t - PMNt - M^2Nt - PMt^2 + MNt^2 + PMt - PNt - 2MNt + Pt^2 + Mt^2 + Nt^2 - t^3 - Nt + t^2 = 0$$

Complexity Analysis

```
void triple(int M, int N, int P){
  assert (0 <= M);
  assert (0 <= N);
  assert (0 <= P);
  int i = 0, j = 0, k = 0;
  int t = 0;
  while(i < N){
    j = 0; t++;
    while(j < M){
      j++; k = i; t++;
      while (k < P){
        k++; t++;
      }
      i = k;
    }
    i++;
  }
  [L]
}
```

Complexity of this program?

- Use t to count loop iterations
- At first glance: $t = O(MNP)$
- DIg found an interesting (and unexpected) nonlinear invariant at L:

$$P^2Mt + PM^2t - PMNt - M^2Nt - PMt^2 + MNt^2 + PMt - PNt - 2MNt + Pt^2 + Mt^2 + Nt^2 - t^3 - Nt + t^2 = 0$$

- Solve for t yields the most precise, unpublished bound:

$$\begin{array}{lll} t = 0 & \text{when} & N = 0, \\ t = P + M + 1 & \text{when} & N \leq P, \\ t = N - M(P - N) & \text{when} & N > P \end{array}$$

Applications

Security

- *complexity* and *side-channel attacks* (**FSE'17, ASE'17, SEAD Workshop '20**)
- *AES* analysis (**ICSE'12, TOSEM'13**)

Applications

Security

- *complexity* and *side-channel attacks* (**FSE'17, ASE'17, SEAD Workshop '20**)
- *AES* analysis (**ICSE'12, TOSEM'13**)

Others

- *termination/liveness* (**OOPSLA'20**)
- *heap/pointer* (**PLDI'19**)
- *program (API) synthesis* (**OOPSLA'19**)
- *disjunctive/geometric invs* (**ICSE'14, J. Automated Reasoning'13**)

Applications

Security

- *complexity* and *side-channel attacks* (**FSE'17, ASE'17, SEAD Workshop '20**)
- *AES* analysis (**ICSE'12, TOSEM'13**)

Others

- *termination/liveness* (**OOPSLA'20**)
- *heap/pointer* (**PLDI'19**)
- *program (API) synthesis* (**OOPSLA'19**)
- *disjunctive/geometric invs* (**ICSE'14, J. Automated Reasoning'13**)

Highly-Configurable Systems: iGen (**FSE '16**), GenTree (**ICSE '21**)

Outline

- SE/PL Research
  - Invariant Generation
  - Automatic Program Repair
  - Highly-Configurable Systems
- Current/New Research Works

Zune Bug

Zune Bug

Wed morning, Dec 31, 2008: Microsoft Zune music players mysteriously froze

Zune Bug

Wed morning, Dec 31, 2008: Microsoft Zune music players mysteriously froze

– Matt Akers (Microsoft Zune spokesman)

"By [Thursday] you should allow the battery to fully run out of power before the unit can restart successfully, then simply ensure that your device is recharged, then turn it back on"

Zune Bug

```
int zunebug(int days) {
  int year = 1980;
  while (days > 365) {
    if (isLeapYear(year)){
      if (days > 366) {
        days -= 366;
        year += 1;
      }
    }
    else {
      days -= 365;
      year += 1;
    }
  }
  return year;
}
```

Zune Bug

```
int zunebug(int days) {
  int year = 1980;
  while (days > 365) {
    if (isLeapYear(year)){
      if (days > 366) {
        days -= 366;
        year += 1;
      }
    }
    else {
      days -= 365;
      year += 1;
    }
  }
  return year;
}
```

```
int zunebug_repair(int days) {
  int year = 1980;
  while (days > 365) {
    if (isLeapYear(year)){
      if (days > 366) {
        // days -= 366; // repair deletes
        year += 1;
      }
      days -= 366;     // repair inserts
    } else {
      days -= 365;
      year += 1;
    }
  }
  return year;
}
```

GenProg: Program Repair using Genetic Algorithm

1. Isolate faults
2. Mutate program statements and reuse existing code
3. Check repair candidates

GenProg: Program Repair using Genetic Algorithm

1. Isolate faults
2. Mutate program statements and reuse existing code
3. Check repair candidates

Results

- demonstrated on bugs in real-world software (repair 16 programs over 1.25 MLocs, 2 mins avg)
- 10-year Most Influential Paper award (ICSE '19) and 10-year Most Impact Paper award (GECCO '19)

APR Techniques

- *Evolutionary computing* (**ICSE'09, GECCO'09, ICST'09, CACM'10, TSE'13**)
- *Theory and Formal Analysis*: *equivalence* between program repair and reachability, apply input generation techniques to repair programs (**TACAS'17**)

APR Techniques

- *Evolutionary computing* (**ICSE'09, GECCO'09, ICST'09, CACM'10, TSE'13**)
- *Theory and Formal Analysis*: *equivalence* between program repair and reachability, apply input generation techniques to repair programs (**TACAS'17**)

Non-traditional Repairs

- Corrupted *data structures* (**Google Summer of Code'18, FSE JPF Workshop'18**)
- *Fault localization* in declarative models (**ICSE '21**)
- Repair *declarative* programs (**ICSE '21**)

Outline

- SE/PL Research
  - Invariant Generation
  - Automatic Program Repair
  - Highly-Configurable Systems
- Current/New Research Works

Analyzing Configurable Software

Modern software are highly-configurable

- Allowing for customization and flexiblity

Analyzing Configurable Software

Modern software are highly-configurable

- Allowing for customization and flexiblity
- But can have misconfigurations (rank 6^{th} on 2020 OWASP list of most critical security risks)


```
# a. ~/.htaccess
<Limit
  PUT DELLETE TRACE
  ..
</Limit>

# b. /etc/apache/httpd.conf
RewriteCond TRACE
..
```

```
# c. load mod_rewrite
a2enmod mod_rewrite

# d. /etc/apache/httpd.conf
LoadModule rewrite_module
  "mod_rewrite.so"
...
```

- Program with *7 options*: $s, t, u, v, x, y, z \in \{0 \ldots 4\}$

- Program with *7 options*: $s, t, u, v, x, y, z \in \{0 \dots 4\}$
- Interactions discovery

interaction	behavior
$x \wedge y$	$B0$
$x \wedge y \wedge (z \in \{0, 3, 4\})$	$B1$
$s \vee t$	$B2$
$(\neg s \wedge \neg t) \vee (\neg u \vee \neg v)$	$B3$
$\vdots$	

- Program with *7 options*: $s, t, u, v, x, y, z \in \{0 \ldots 4\}$
- Interactions discovery

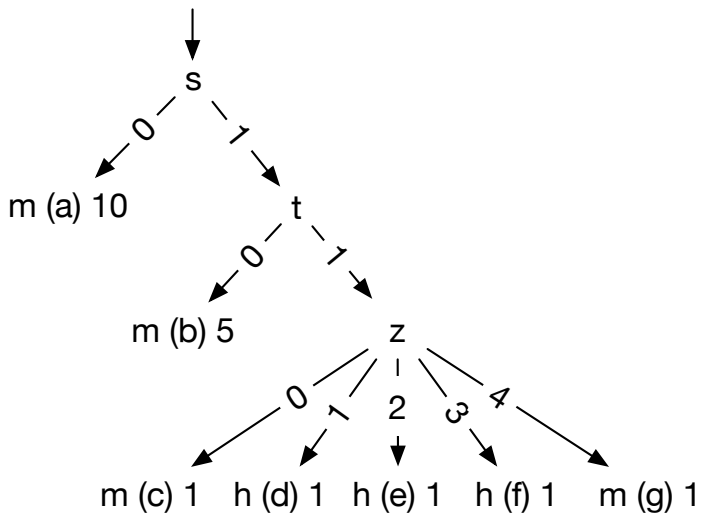
interaction	behavior
$x \wedge y$	$B0$
$x \wedge y \wedge (z \in \{0, 3, 4\})$	$B1$
$s \vee t$	$B2$
$(\neg s \wedge \neg t) \vee (\neg u \vee \neg v)$	$B3$
$\vdots$	

- Use *dynamic analysis*

config	s	t	u	v	x	y	z	behavior
c_1	1	0	1	1	1	1	4	$B0, B1$
c_2	0	0	1	1	1	1	0	$B0, B3$
c_3	0	1	1	1	1	0	3	$B2$
	$\vdots$							

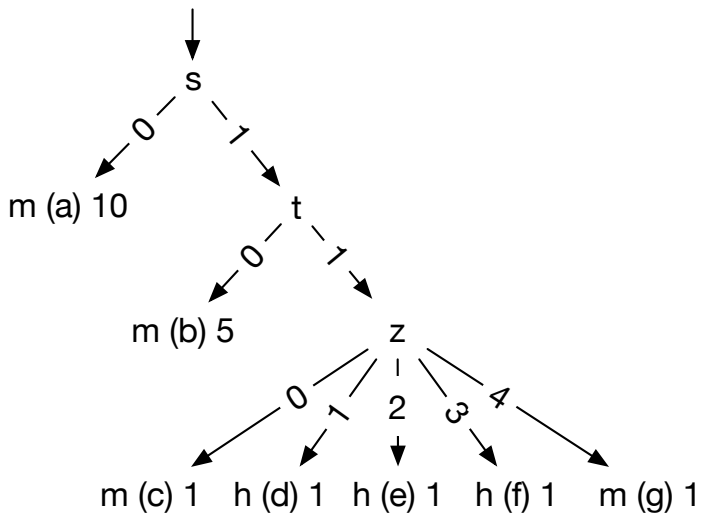
GenTree: Dynamic Interaction Discovery (FSE'16, ICSE'21)

- Use decision trees to represent interactions



GenTree: Dynamic Interaction Discovery (FSE'16, ICSE'21)

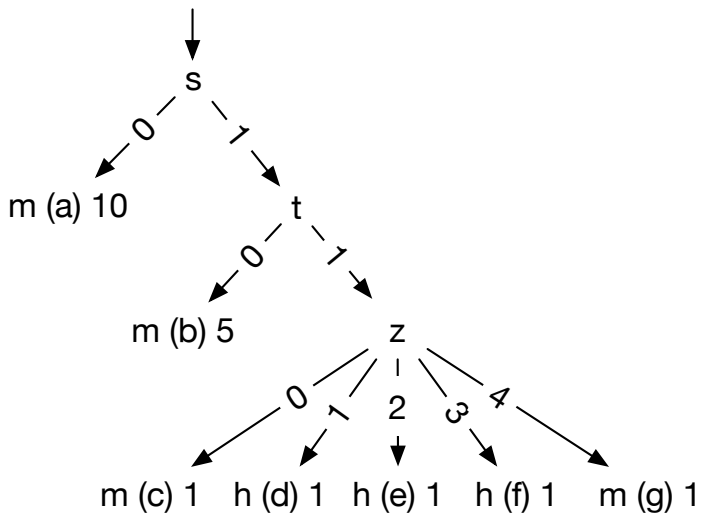
- Use decision trees to represent interactions



- $C5_i$: new classification algorithm

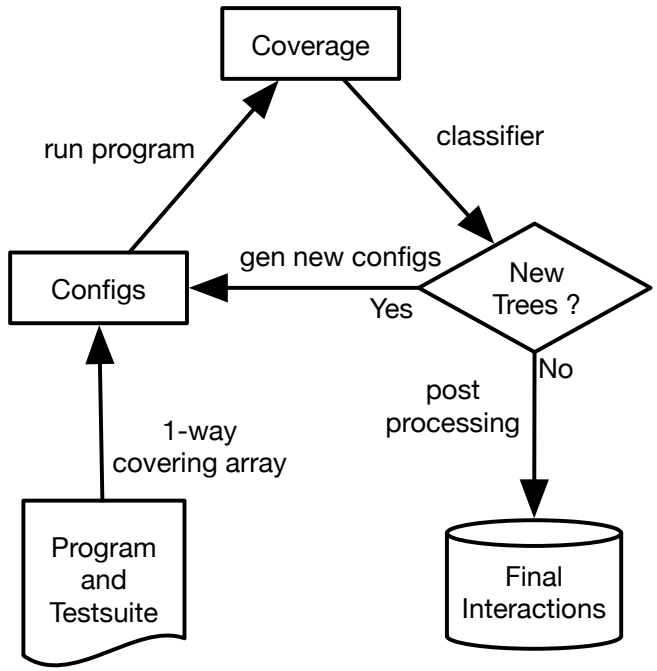
GenTree: Dynamic Interaction Discovery (FSE'16, ICSE'21)

- Use decision trees to represent interactions



- C5$_i$: new classification algorithm

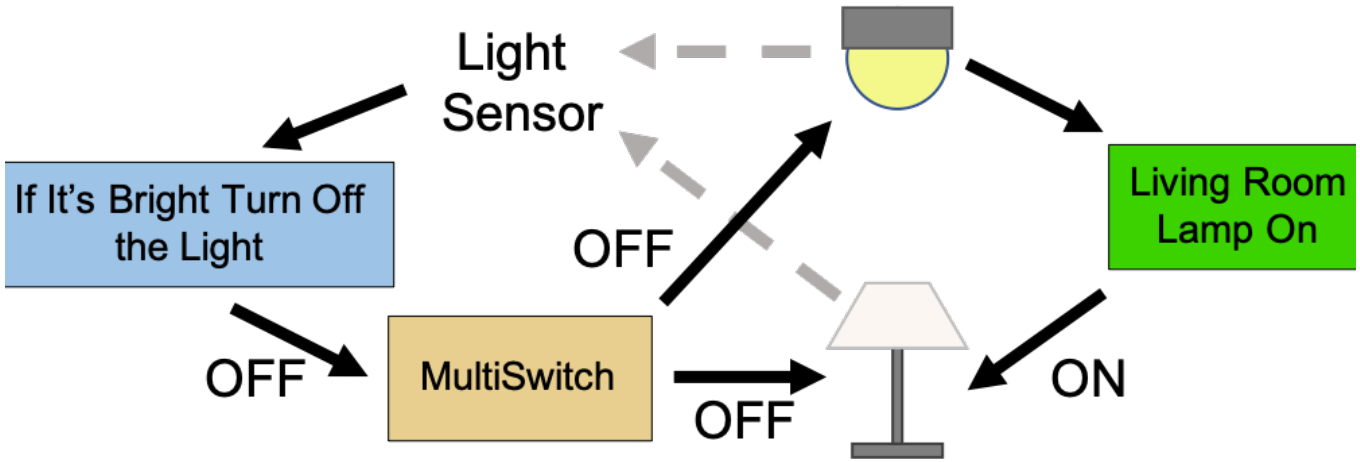
Iterative approach



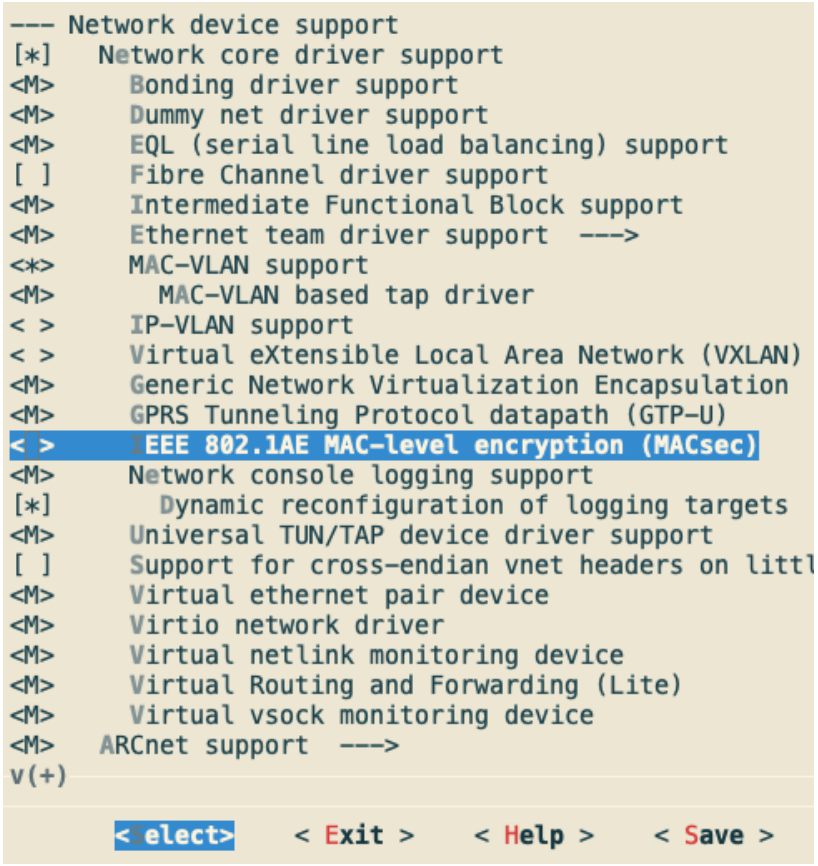
Outline

- SE/PL Research
  - Invariant Generation
  - Automatic Program Repair
  - Highly-Configurable Systems
- Current/New Research Works

IoT Interaction Analysis and Repair (**UNL Faculty grant'21**)

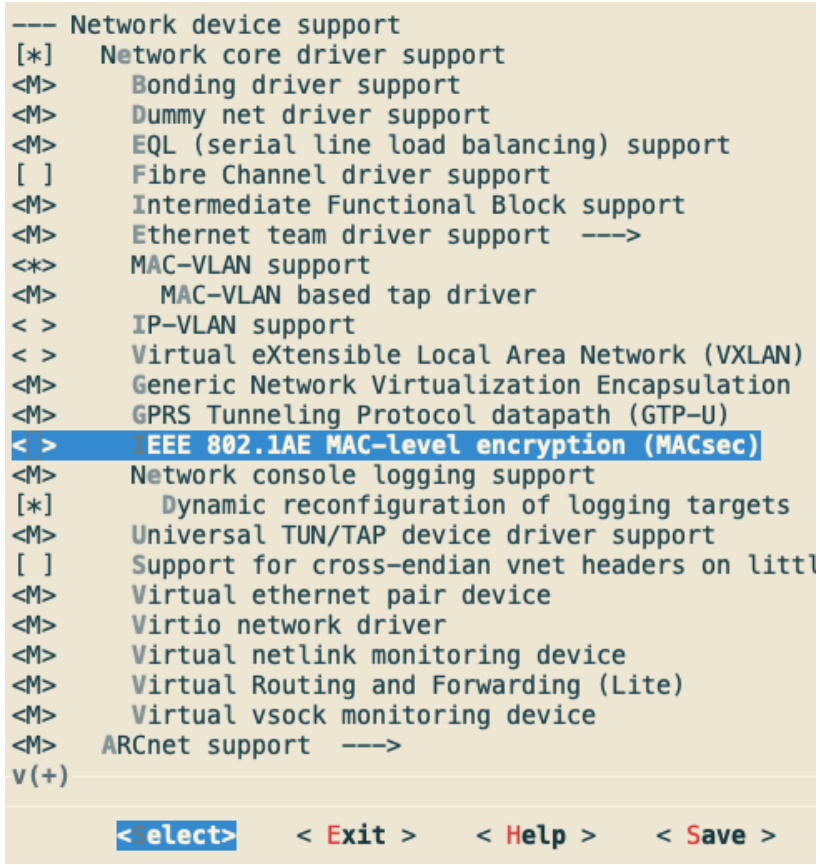


Build Systems (SPLC Workshop'20, ICSME NIER'20, NSF CRII'20)



Linux/Unix Build Systems

Build Systems (SPLC Workshop'20, ICSME NIER'20, NSF CRII'20)

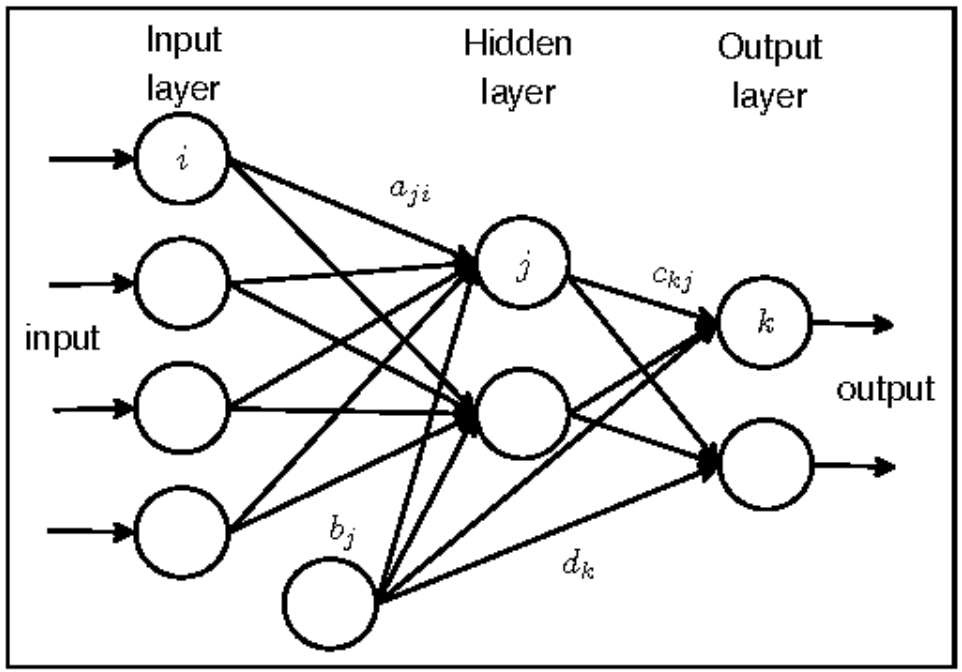


Linux/Unix Build Systems

```
ifeq ($(abs_otree),$(CURDIR))
MAKEFLAGS += --no-print-dir
else
need-sub-make := 1
endif

abs_stree := $(rpath $(dir $
     (this-makefile)))
ifneq ($(words $(subst :,
     ,$(abs_stree))), 1)
$(error src dir cannot contain
     spaces or colons)
endif
ifneq ($(abs_stree),$(abs_otree))
MAKEFLAGS +=
     --include-dir=$(abs_stree)
```

Make/CMake

Deep Neural Networks

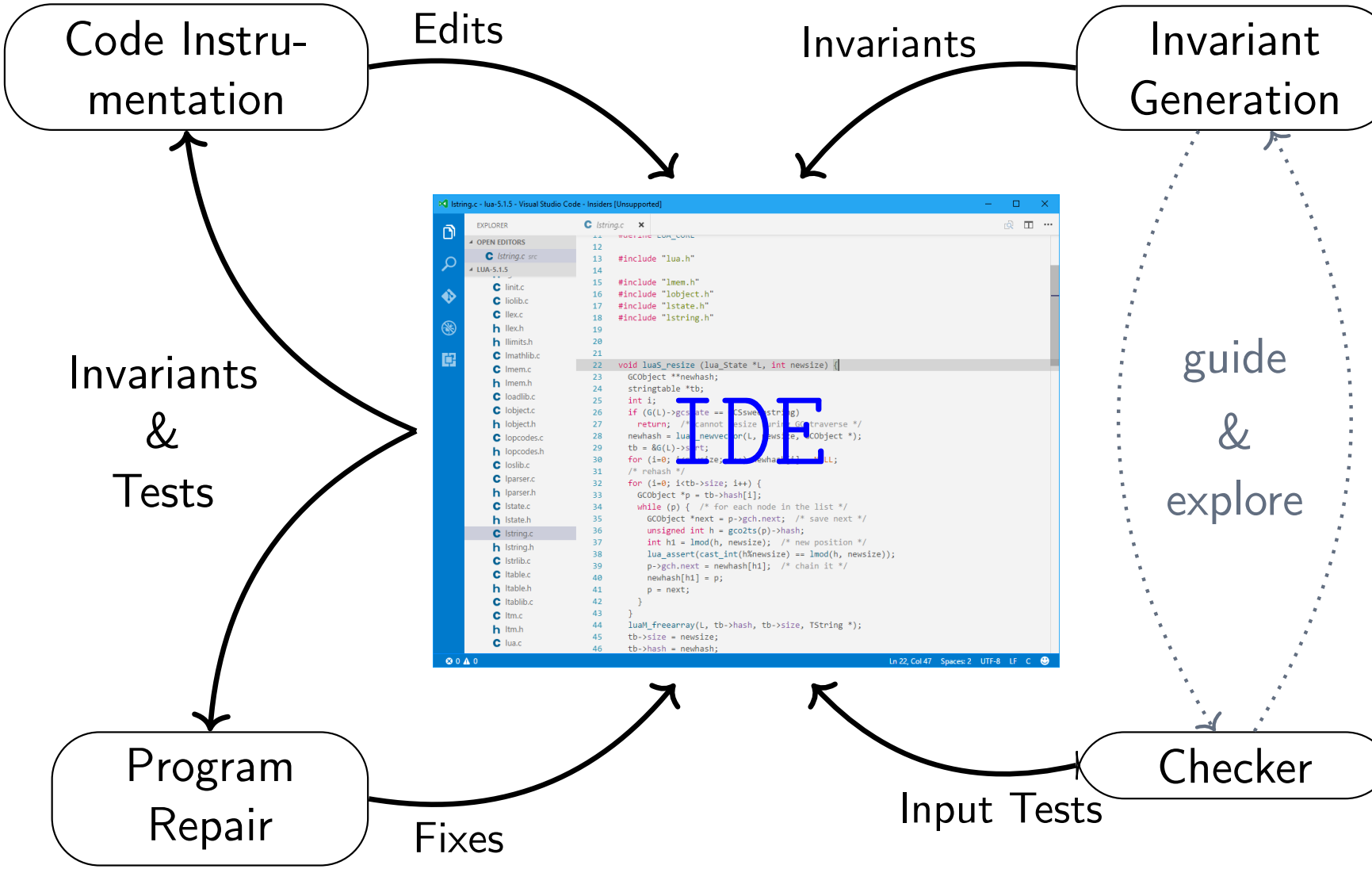


$a[1,\ldots,h][1,\ldots,n]$ ▷ Input weights
$c[1,\ldots,m][1,\ldots,h]$ ▷ Output weights
$b[1,\ldots,h]$ ▷ Hidden nodes' bias
$d[1,\ldots,m]$ ▷ Output nodes' bias

FUNCTION $\nu(\underline{x})$
  for $j \leftarrow 1$ **to** h **do**
    $r_j \leftarrow 0$;
    for $i \leftarrow 1$ **to** n **do**
      $r_j \leftarrow r_j + a_{ji} \cdot x_i + b_j$
  for $k \leftarrow 1$ **to** m **do**
    $s_k \leftarrow 0$;
    for $j \leftarrow 1$ **to** h **do**
      $s_k \leftarrow s_k + c_{kj} \cdot \sigma_h(r_j) + d_k$
    $y_k \leftarrow \sigma_o(s_k)$
  return $\underline{y}$

- Invariants (activation patterns) discovery
- Symbolic testing

IDE Integration (https://grammatech.gitlab.io/Mnemosyne/docs/muses/)



Thank you!