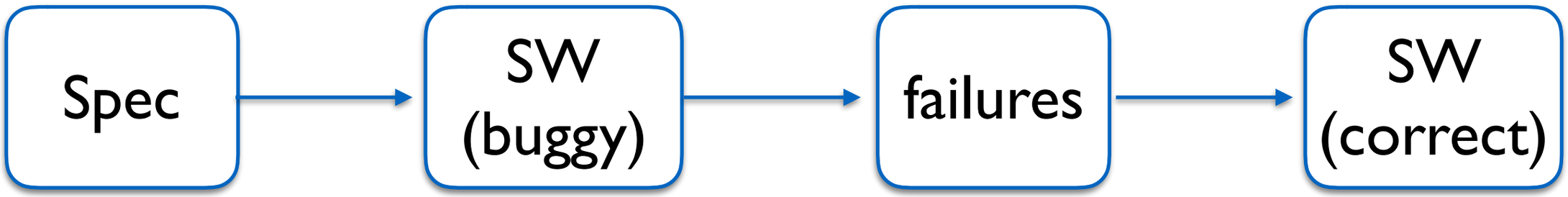
소프트웨어 오류 자동 수정 기법

오학주
소프트웨어 분석 연구실
고려대학교 정보대학 컴퓨터학과

Jun 8, 2018@고려대-삼성전자 교류회

연구 분야

- Automated programming / testing / debugging



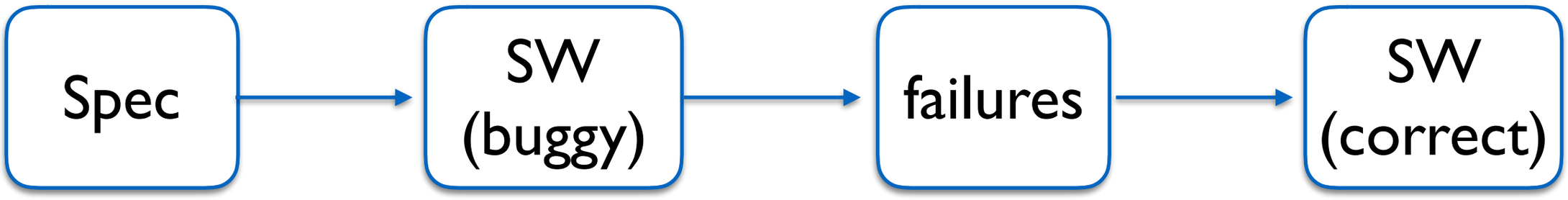
프로그래밍
(manual)

테스팅
(manual)

디버깅
(manual)

연구 분야

- Automated programming / testing / debugging



프로그래밍
(manual)

프로그램 합성
(Program Synthesis)

테스팅
(manual)

프로그램 분석
(Program Analysis)

디버깅
(manual)

프로그램 수정
(Program Repair)

Automatic Debugging (Automatic Program Repair)

```
1 p = malloc(); // o1
2
3 if(...){
4     q = malloc(); // o2
5     *q = *p;
6 } else {
7     q = p;
8 }
9 // Use q
10 free(p);
11
12 // o2 leaks here
13 return;
```

buggy program

오류 자동 수정기

```
1 p = malloc(); // o1
2
3 if(...){
4     q = malloc(); // o2
5     *q = *p;
6     free(p);
7 } else {
8     q = p;
9 }
10 // Use q
11 free(q);
12
13 return;
```

correct program

MemFix: 메모리 관리 오류 자동 수정기

- 메모리 관리 오류: C/C++에서 빈번하게 발생

Memory Leak

```
p = malloc(1);
...
return;
```

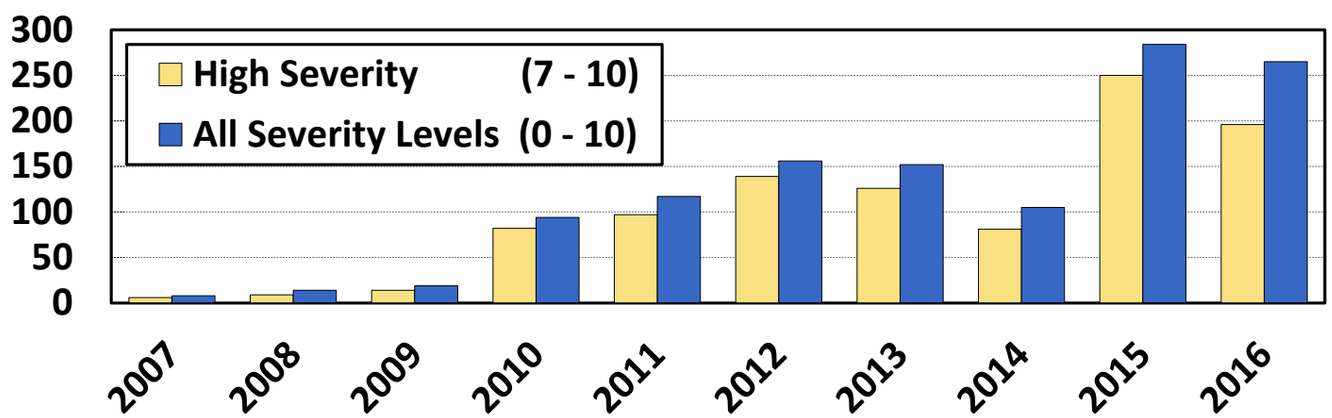
Use-After-Free

```
p = malloc(1);
...
free(p);
...
use(p);
```

Double-Free

```
p = malloc(1);
...
free(p);
...
free(p);
```

- 심각한 소프트웨어 보안취약점의 주요 원인



(Yan et al. ICSE 2018)

- 사전 탐지 및 정확한 수정이 매우 어려움

Example (Linux Kernel)

```
in = malloc(1);
out = malloc(1);
... // use in, out
free(out);
free(in);

in = malloc(2);
if (in == NULL) {

  goto err;
}

out = malloc(2);
if (out == NULL) {
  free(in);

  goto err;
}
... // use in, out
err:
  free(in);
  free(out);
  return;
```

Example (Linux Kernel)

```
in = malloc(1);
out = malloc(1);
... // use in, out
free(out);
free(in);

in = malloc(2);
if (in == NULL) {
  goto err;
}

out = malloc(2);
if (out == NULL) {
  free(in);

  goto err;
}
... // use in, out
err:
  free(in);
  free(out);
  return;
```

double-free

Example (Linux Kernel)

```
in = malloc(1);
out = malloc(1);
... // use in, out
free(out);
free(in);

in = malloc(2);
if (in == NULL) {

  goto err;
}

out = malloc(2);
if (out == NULL) {
  free(in);

  goto err;
}
... // use in, out
err:
  free(in);
  free(out);
  return;
```

Example (Linux Kernel)

USB: fix double frees in error code paths of ipaq driver

the error code paths can be enter with buffers to freed buffers.
Serial core would do a kfree() on memory already freed.

Signed-off-by: Oliver Neukum <oneukum@suse.de>
Signed-off-by: Greg Kroah-Hartman <gregkh@suse.de>

master v4.15-rc1 ... v2.6.24-rc1

Oliver Neukum committed with **gregkh** on 18 Sep 2007 1 par

```
in = malloc(1);
out = malloc(1);
... // use in, out
free(out);
free(in);

in = malloc(2);
if (in == NULL) {
  out = NULL;
  goto err;
}

out = malloc(2);
if (out == NULL) {
  free(in);
  in = NULL;
  goto err;
}
... // use in, out
err:
  free(in);
  free(out);
  return;
```

Example (Linux Kernel)

USB: fix double kfree in ipaq in error case

in the error case the ipaq driver leaves a dangling pointer to already freed memory that will be freed again.

Signed-off-by: Oliver Neukum <oneukum@suse.de>
Signed-off-by: Greg Kroah-Hartman <gregkh@suse.de>

master v4.15-rc1 ... v2.6.27-rc1

Oliver Neukum committed with **gregkh** on 30 Jun 2008 1 parent 35

```
in = malloc(1);
out = malloc(1);
... // use in, out
// removed
free(in);

in = malloc(2);
if (in == NULL) {
  out = NULL;
  goto err;
}
free(out);
out = malloc(2);
if (out == NULL) {
  free(in);
  in = NULL;
  goto err;
}
... // use in, out
err:
  free(in);
  free(out);
  return;
```

Example (Linux Kernel)

USB: fix double kfree in ipaq in error case

in the error case the ipaq driver leaves a dangling pointer to already freed memory that will be freed again.

Signed-off-by: Oliver Neukum <oneukum@suse.de>
Signed-off-by: Greg Kroah-Hartman <gregkh@suse.de>

master v4.15-rc1 ... v2.6.27-rc1

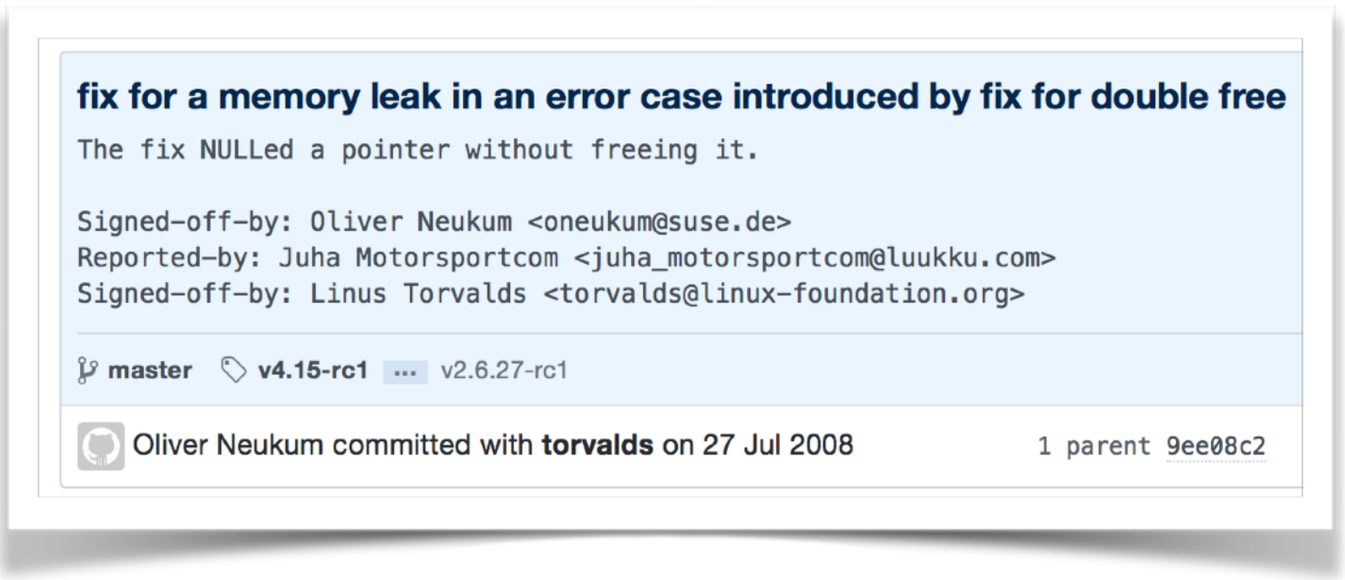
Oliver Neukum committed with **gregkh** on 30 Jun 2008 1 parent 35

memory leak

```
in = malloc(1);
out = malloc(1);
... // use in, out
// removed
free(in);

in = malloc(2);
if (in == NULL) {
  out = NULL;
  goto err;
}
free(out);
out = malloc(2);
if (out == NULL) {
  free(in);
  in = NULL;
  goto err;
}
... // use in, out
err:
  free(in);
  free(out);
  return;
```

Example (Linux Kernel)




```
in = malloc(1);
out = malloc(1);
... // use in, out
free(out);
free(in);
out = NULL;
in = malloc(2);
if (in == NULL) {
  out = NULL;
  goto err;
}
// removed
out = malloc(2);
if (out == NULL) {
  free(in);
  in = NULL;
  goto err;
}
... // use in, out
err:
  free(in);
  free(out);
  return;
```

```
in = malloc(1);
out = malloc(1);
... // use in, out
free(out);
free(in);

in = malloc(2);
if (in == NULL) {

  goto err;
}

out = malloc(2);
if (out == NULL) {
  free(in);

  goto err;
}
... // use in, out
err:
  free(in);
  free(out);
  return;
```

MemFix

```
in = malloc(1);
out = malloc(1);
... // use in, out
// removed
free(in);

in = malloc(2);
if (in == NULL) {

  goto err;
}
free(out);
out = malloc(2);
if (out == NULL) {
  // removed

  goto err;
}
... // use in, out
err:
  free(in);
  free(out);
  return;
```

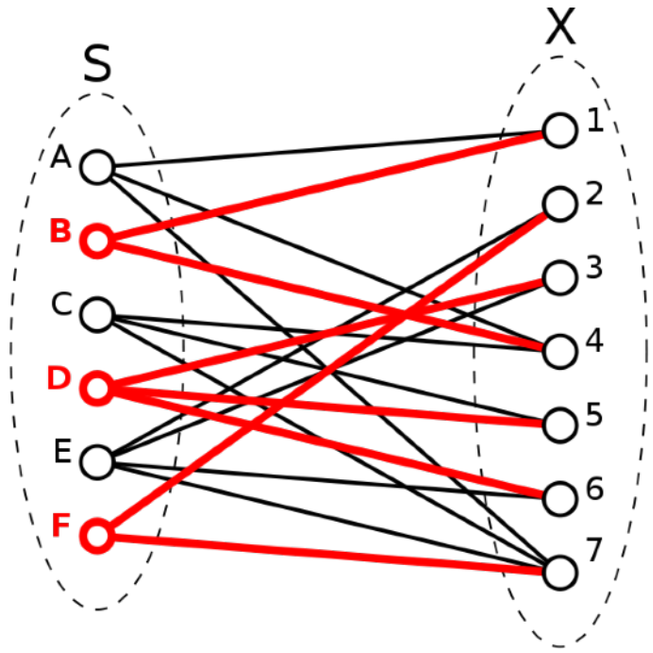
MemFix 알고리즘

approx. by static analysis

```
in = malloc(2);
if (in == NULL) {
  goto err;
}
free(out);
out = malloc(2);
if (out == NULL) {
  // removed

  goto err;
}
```

Fixing memory errors
(undecidable)



Exact cover problem
(NP-complete)

SAT encoding

$$\varphi_1 = \bigwedge_{j=1}^{m} \bigvee_{i=1}^{n} T_{ij} \wedge S_i$$

$$\varphi_2 = \bigwedge_{j=1}^{m} \bigwedge_{i_1=1}^{n} \bigwedge_{i_2=1}^{n} \left((i_1 \neq i_2)\right.$$

Boolean satisfiability
(NP-complete)

MemFix 알고리즘

- Soundness and safety proved formally
  - Soundness: the patch gurantees to fix the error
  - Safety: no new errors are introduced

Automatic Feedback Generation for Programming Assignments

- In my programming language course,
  - students hardly receive personalized feedback, and
  - instructor's solutions are not very helpful.

모범 답안

```
let rec map f (l,var) =
    match l with
    | [] -> []
    | hd::tl -> (f (hd,var))::(map f (tl,var))
...
| Sum lst -> Sum (map diff (lst,var))
...
```

오답 코드

```
...
| Sum plus ->
 (match plus with
   []     -> Const 0
   | [hd]   -> diff( hd, var)
   | hd::tl -> Sum [diff(hd, var); diff(Times tl, var)]
 ) ...
```

Sum

학생 제출 답안

```
type aexp =
   |CONST of int
   | VAR of string
   | POWER of string * int
   | TIMES of aexp list
   | SUM of aexp list

type env = (string * int * int) list

let diff : aexp * string -> aexp
= fun (aexp, x) ->

   let rec deployEnv : env -> int -> aexp list
   = fun env flag ->
   match env with
   | hd::tl ->
   (
    match hd with
    |(x, c, p) ->
       if (flag = 0 && c = 0) then deployEnv tl flag
       else if (x = "const" && flag = 1 && c = 1) then deployEnv tl flag
       else if (p = 0) then (CONST c)::(deployEnv tl flag)
       else if (c = 1 && p = 1) then (VAR x)::(deployEnv tl flag)
       else if (p = 1) then TIMES[CONST c; VAR x]::(deployEnv tl flag)
       else if (c = 1) then POWER(x, p)::(deployEnv tl flag)
       else TIMES [CONST c; POWER(x, p)]::(deployEnv tl flag)
)
| [] -> []
in

let rec updateEnv : (string * int * int) -> env -> int -> env
= fun elem env flag ->
match env with
| (hd::tl) ->
   (
   match hd with
   | (x, c, p) ->
      (
      match elem with
      |(x2, c2, p2) ->
         if (flag = 0) then
            if (x = x2 && p = p2) then (x, (c + c2), p)::tl
            else hd::(updateEnv elem tl flag)
         else
            if (x = x2) then (x, (c*c2), (p + p2))::tl
            else hd::(updateEnv elem tl flag)
      )
   )
| [] -> elem::[]
in

let rec doDiff : aexp * string -> aexp
= fun (aexp, x) ->
match aexp with
| CONST _ -> CONST 0
| VAR v ->
   if (x = v) then CONST 1
   else CONST 0
| POWER (v, p) ->
   if (p = 0) then CONST 0
   else if (x = v) then TIMES ((CONST p)::POWER (v, p-1)::[])
   else CONST 0
| TIMES lst ->
   (
      match lst with
         (
            match (hd, diff_hd, tl, diff_tl) with
            | (CONST p, CONST s, [CONST r], CONST q) -> CONST (p*q + r*s)
            | (CONST p, _, _, CONST q) ->
               if (diff_hd = CONST 0 || tl = [CONST 0]) then CONST (p*q)
               else SUM [CONST(p*q); TIMES(diff_hd::tl)]
            | (_, CONST s, [CONST r], _) ->
               if (hd = CONST 0 || diff_tl = CONST 0) then CONST (r*s)
               else  SUM [TIMES [hd; diff_tl]; CONST(r*s)]
            | _ ->
               if (hd = CONST 0 || diff_tl = CONST 0) then TIMES(diff_hd::tl)
               else if (tl = [CONST 0] || diff_hd = CONST 0) then TIMES [hd; diff_tl]
               else SUM [TIMES [hd; diff_tl]; TIMES (diff_hd::tl)]
         )
      | [] -> CONST 0
   )
| SUM lst -> SUM(List.map (fun aexp -> doDiff(aexp, x)) lst)
in

let rec simplify : aexp -> env -> int -> aexp list
= fun aexp env flag ->
match aexp with
| SUM lst ->
   (
      match lst with
      | (CONST c)::tl -> simplify (SUM tl) (updateEnv ("const", c, 0) env 0) 0
      | (VAR x)::tl -> simplify (SUM tl) (updateEnv (x, 1, 1) env 0) 0
      | (POWER (x, p))::tl -> simplify (SUM tl) (updateEnv (x, 1, p) env 0) 0
      | (SUM lst)::tl -> simplify (SUM (List.append lst tl)) env 0
      | (TIMES lst)::tl ->
         (
            let l = simplify (TIMES lst) [] 1 in
            match l with
            | h::t ->
               if (t = []) then List.append l (simplify (SUM tl) env 0)
               else List.append (TIMES l::[]) (simplify (SUM tl) env 0)
            | [] -> []
         )
      | [] -> deployEnv env 0
   )
| TIMES lst ->
   (
      match lst with
      | (CONST c)::tl -> simplify (TIMES tl) (updateEnv ("const", c, 0) env 1) 1
      | (VAR x)::tl -> simplify (TIMES tl) (updateEnv (x, 1, 1) env 1) 1
      | (POWER (x, p))::tl -> simplify (TIMES tl) (updateEnv (x, 1, p) env 1) 1
      | (SUM lst)::tl ->
         (
            let l = simplify (SUM lst) [] 0 in
            match l with
            | h::t ->
               if (t = []) then List.append l (simplify (TIMES tl) env 1)
                else List.append (SUM l::[]) (simplify (TIMES tl) env 1)
             | [] -> []  (* Feedback : Replace [] by ((Sum lst) :: tl) *)
         )
      | (TIMES lst)::tl -> simplify (TIMES (List.append lst tl)) env 1
      | [] -> deployEnv env 1
   )
in

let result = doDiff (aexp, x) in
match result with
| SUM _ -> SUM (simplify result [] 0)
| TIMES _ -> TIMES (simplify result [] 1)
| _ -> result
```

모범답안

```
let rec diff : aexp * string -> aexp
= fun (e, x) ->
   match e with
   | Const n -> Const 0
   | Var a -> if (a <> x) then Const 0 else Const 1
   | Power (a, n) -> if (a <> x) then Const 0 else Times [Const n; Power (a, n-1)]
   | Times l ->
      begin
      match l with
      | [] -> Const 0
      | hd::tl -> Sum [Times ((diff (hd, x))::tl); Times [hd; diff (Times tl, x)]]
      end
   | Sum l -> Sum (List.map (fun e -> diff (e,x)) l)
```

학생 제출 답안

```
type aexp =
   |CONST of int
   | VAR of string
   | POWER of string * int
   | TIMES of aexp list
   | SUM of aexp list

type env = (string * int * int) list

let diff : aexp * string -> aexp
= fun (aexp, x) ->

   let rec deployEnv : env -> int -> aexp list
   = fun env flag ->
   match env with
   | hd::tl ->
   (
    match hd with
    |(x, c, p) ->
       if (flag = 0 && c = 0) then deployEnv tl flag
       else if (x = "const" && flag = 1 && c = 1) then deployEnv tl flag
       else if (p = 0) then (CONST c)::(deployEnv tl flag)
       else if (c = 1 && p = 1) then (VAR x)::(deployEnv tl flag)
       else if (p = 1) then TIMES[CONST c; VAR x]::(deployEnv tl flag)
       else if (c = 1) then POWER(x, p)::(deployEnv tl flag)
       else TIMES [CONST c; POWER(x, p)]::(deployEnv tl flag)
)
| [] -> []
in

let rec updateEnv : (string * int * int) -> env -> int -> env
= fun elem env flag ->
match env with
| (hd::tl) ->
   (
   match hd with
   | (x, c, p) ->
      (
      match elem with
      |(x2, c2, p2) ->
         if (flag = 0) then
            if (x = x2 && p = p2) then (x, (c + c2), p)::tl
            else hd::(updateEnv elem tl flag)
         else
            if (x = x2) then (x, (c*c2), (p + p2))::tl
            else hd::(updateEnv elem tl flag)
      )
   )
| [] -> elem::[]
in

let rec doDiff : aexp * string -> aexp
= fun (aexp, x) ->
match aexp with
| CONST _ -> CONST 0
| VAR v ->
   if (x = v) then CONST 1
   else CONST 0
| POWER (v, p) ->
   if (p = 0) then CONST 0
   else if (x = v) then TIMES ((CONST p)::POWER (v, p-1)::[])
   else CONST 0
| TIMES lst ->
   (
      match lst with
      (
         match (hd, diff_hd, tl, diff_tl) with
         | (CONST p, CONST s, [CONST r], CONST q) -> CONST (p*q + r*s)
         | (CONST p, _, _, CONST q) ->
            if (diff_hd = CONST 0 || tl = [CONST 0]) then CONST (p*q)
            else SUM [CONST(p*q); TIMES(diff_hd::tl)]
         | (_, CONST s, [CONST r], _) ->
            if (hd = CONST 0 || diff_tl = CONST 0) then CONST (r*s)
            else  SUM [TIMES [hd; diff_tl]; CONST(r*s)]
         | _ ->
            if (hd = CONST 0 || diff_tl = CONST 0) then TIMES(diff_hd::tl)
            else if (tl = [CONST 0] || diff_hd = CONST 0) then TIMES [hd; diff_tl]
            else SUM [TIMES [hd; diff_tl]; TIMES (diff_hd::tl)]
      )
      | [] -> CONST 0
   )
| SUM lst -> SUM(List.map (fun aexp -> doDiff(aexp, x)) lst)
in

let rec simplify : aexp -> env -> int -> aexp list
= fun aexp env flag ->
match aexp with
| SUM lst ->
   (
      match lst with
      | (CONST c)::tl -> simplify (SUM tl) (updateEnv ("const", c, 0) env 0) 0
      | (VAR x)::tl -> simplify (SUM tl) (updateEnv (x, 1, 1) env 0) 0
      | (POWER (x, p))::tl -> simplify (SUM tl) (updateEnv (x, 1, p) env 0) 0
      | (SUM lst)::tl -> simplify (SUM (List.append lst tl)) env 0
      | (TIMES lst)::tl ->
         (
            let l = simplify (TIMES lst) [] 1 in
            match l with
            | h::t ->
               if (t = []) then List.append l (simplify (SUM tl) env 0)
               else List.append (TIMES l::[]) (simplify (SUM tl) env 0)
            | [] -> []
         )
      | [] -> deployEnv env 0
   )
| TIMES lst ->
   (
      match lst with
      | (CONST c)::tl -> simplify (TIMES tl) (updateEnv ("const", c, 0) env 1) 1
      | (VAR x)::tl -> simplify (TIMES tl) (updateEnv (x, 1, 1) env 1) 1
      | (POWER (x, p))::tl -> simplify (TIMES tl) (updateEnv (x, 1, p) env 1) 1
      | (SUM lst)::tl ->
         (
            let l = simplify (SUM lst) [] 0 in
            match l with
            | h::t ->
               if (t = []) then List.append l (simplify (TIMES tl) env 1)
               else List.append (SUM l::[]) (simplify (TIMES tl) env 1)
            | [] -> [] (* Feedback : Replace [] by ((Sum lst) :: tl) *)
         )
      | (TIMES lst)::tl -> simplify (TIMES (List.append lst tl)) env 1
      | [] -> deployEnv env 1
   )
in

let result = doDiff (aexp, x) in
match result with
| SUM _ -> SUM (simplify result [] 0)
| TIMES _ -> TIMES (simplify result [] 1)
| _ -> result
```

모범답안

```
let rec diff : aexp * string -> aexp
= fun (e, x) ->
   match e with
   | Const n -> Const 0
   | Var a -> if (a <> x) then Const 0 else Const 1
   | Power (a, n) -> if (a <> x) then Const 0 else Times [Const n; Power (a, n-1)]
   | Times l ->
      begin
      match l with
      | [] -> Const 0
      | hd::tl -> Sum [Times ((diff (hd, x))::tl); Times [hd; diff (Times tl, x)]]
      end
   | Sum l -> Sum (List.map (fun e -> diff (e,x)) l)
```

((Sum lst)::tl)

Thank you!

- Research areas: programming languages, software engineering, software security
  - program analysis and testing
  - program synthesis and repair
- Publication: top-venues in PL, SE, Security, and AI:
  - PLDI('12,'14), OOPSLA('15,'17,'17), TOPLAS('14,'16,'17), ICSE('17,'18), FSE'18, S&P'17, IJCAI('17,'18), etc

http://prl.korea.ac.kr