CS3215: Software Engineering Project

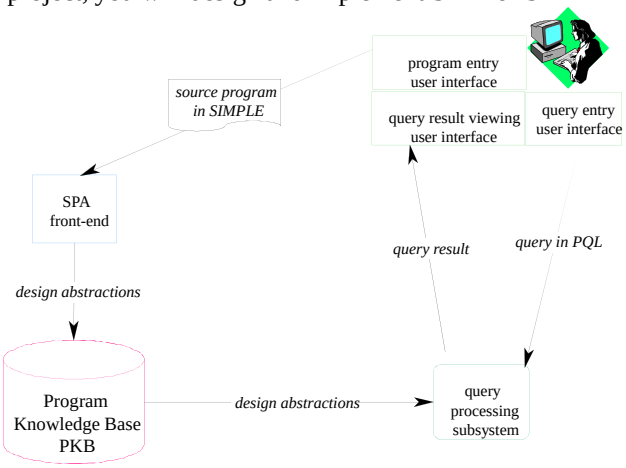
CS3215, LN set #2: SPA requirements

Motivation:

- Some companies spend up to 80% of the total computer budget on software maintenance
- During maintenance, programmers spend 50% trying to understand a program
- Examples of questions programmers ask about programs:
  - Where is variable x modified?
  - Where are all the statements that contain sub-expression x*y+z?
  - Which program statements affect value of x at statement #120?
  - If I change value of x in statement #20, which other statements will be affected?

SPA - Static Program Analysis tool

- SPA helps programmers understand a program by answering program queries
- in the project, you will design and implement SPA for SIMPLE



An outline of this lecture:

1. Source language SIMPLE
2. Questions programmers ask about programs
3. What program information will be stored in the PKB?
4. Writing program queries in PQL (Program Query Language)

Source language: SIMPLE

```
procedure First {
   x = 2;
   z = 3;
   call Second; }
```

```
procedure Second {
1.  x = 0;
2.  i = 5;
3.  while i  {
4.    x = x + 2*y;
5.   call Third;
6.    i = i - 1; }
7.  if x then {
8.    x = x+1; }
     else {
9.    z = 1; }
10. z = z + x + i;
11. y = z + 2;
12.  x = x * y + z; }
```

```
procedure Third {
   z = 5;
   v = z;  }
```

SIMPLE language rules

- A program consists of one or more procedures,
- Program execution starts by calling the first procedure,
- Procedures: no parameters, no nesting, no recursion,
- Variables: unique names, global scope, integer type, no declarations - can be introduced in a program as needed,
- Program statements:
  - procedure call: call p
  - assignment: x = 2 ; x = a + 2 * b ;
  - while loop: while i { statement list }
  - if-then-else: if i then { statement list } else { statement list}
  - a control variable: FALSE if 0, otherwise - TRUE

SPA for SIMPLE

- You will design and implement an SPA for SIMPLE
- SPA front-end extracts design information from programs and stores it in a Program Knowledge Base (PKB)
- Programmer formulates queries in a Program Query Language (PQL)
- Query Processor answers queries
  - Query Pre-processor and Query Evaluator
- Query Result Projector displays query results

Relationships Calls and Calls*

- for any procedures p and q:

Calls (p, q) - procedure p directly calls q

e.g., Calls (First, Second), Calls (Second, Third)

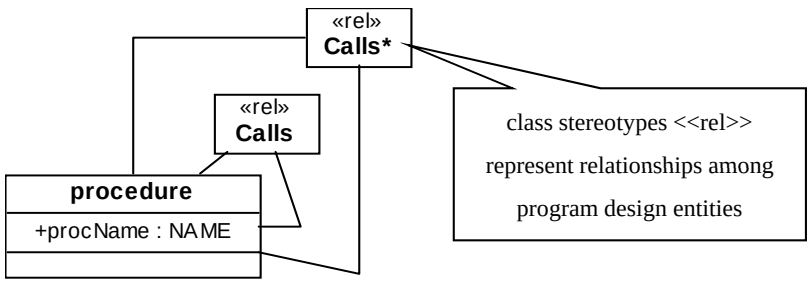
Calls* (p,q) - procedure p calls directly or indirectly q

e.g., Calls* (First, Second), Calls* (First, Third)

Call and Calls* in UML

Calls (procedure, procedure)

Calls* (procedure, procedure)



Attributes of program design entities

program.progName

procedure.procName

variable.varName

stmt.stmt#

Relationships *Modifies* and *Uses*

For stmt ‘s’ and variable ‘v’: Modifies (s, v) holds if it is possible that statement ‘s’ modifies the value of variable ‘v’ during execution

For procedure ‘p’ and variable ‘v’: Modifies (p, v) holds if it is possible that procedure ‘p’ modifies the value of variable ‘v’ when ‘p’ is called

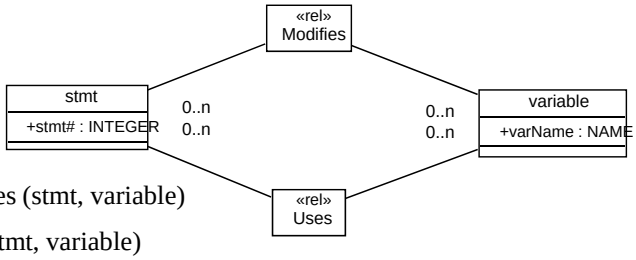
that is v appears on the left hand side in some assignment statement in p or is modified in some other procedure called (directly or indirectly) from p

For stmt ‘s’ and variable ‘v’: Uses (s, v) holds if it is possible that statement ‘s’ uses the value of variable ‘v’ during execution

For procedure ‘p’ and variable ‘v’: Uses (p, v) holds if it is possible that procedure ‘p’ uses the value of variable ‘v’ when ‘p’ is called

that is, v appears in the right hand side of an assignment statement or in the condition in p or is used in other procedure called (directly or indirectly) from p

Modifies and *Uses* in UML

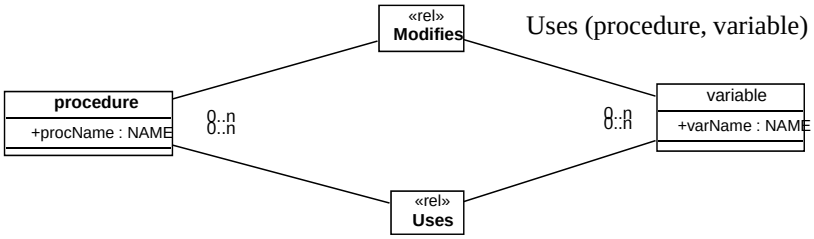


Modifies (stmt, variable)

Uses (stmt, variable)

Modifies (procedure, variable)

Uses (procedure, variable)



Examples

```
procedure Second {
1.  x = 0;
2.  i = 5;
3.  while i  {
4.    x = x + 2*y;
5.   call Third;
6.    i = i - 1; }
7.  if x then {
8.    x = x+1; }
    else {
9.    z = 1; }
10. z = z + x + i;
11. y = z + 2;
12.  x = x * y + z; }
```

which relationships hold?

Modifies (1, “x”)

Modifies (6, “i”)

Modifies (7, “x”)

Modifies (7, “z”)

which variables are modified in statement #3?

Examples

```
procedure Second {
1.  x = 0;
2.  i = 5;
3.  while i  {
4.    x = x + 2*y;
5.   call Third;
6.    i = i - 1; }
7.  if x then {
8.    x = x+1; }
    else {
9.    z = 1; }
10. z = z + x + i;
11. y = z + 2;
12.  x = x * y + z; }
```

which relationships hold?

Uses (4, “x”)

Uses (4, “y’)

Uses (5, “z”)

Uses (“Second”, “i”)

Uses (“Second”,“x”)

Uses (“Second”,“y”)

Uses (“Second”,“z”)

Examples

```
procedure Second {
1.  x = 0;
2.  i = 5;
3.  while i  {
4.    x = x + 2*y;
5.   call Third;
6.    i = i - 1; }
7.  if x then {
8.    x = x+1; }
    else {
9.    z = 1; }
10. z = z + x + i;
11. y = z + 2;
12.  x = x * y + z; }
```

which relationships hold?

Modifies (“Second“, “x“)

Modifies (“Second“, “i“)

Modifies (“Second“, “z“)

Modifies (“Second“, “y“)

Modifies (“Second“, “v“)

Examples of program queries

Q1. Which procedures a program consists of?

Q2. Which procedures call a given procedure p?

directly? indirectly?

Q3. Which variables have their values modified in procedure p?

Q4. Which variables are used in procedure p?

Q5. Which procedures modify a given variable v?

Q6. Which procedures modify variable v and at the same time call procedure p?

Examples of program queries in PQL

Q: Select all procedures in the program

procedure p;

Select p

-- *answer:* procedures First, Second and Third

Select p.procName

-- *answer:* procedure names: First, Second, Third

Q: Which procedures directly call procedure Third?

procedure p, q;

Select p **such that** Calls (p, q) **with** q.procName = “Third”

Select p **such that** Calls (p, “Third”)

-- *answer:* procedure Second

Examples of program queries in PQL, cont.

Q: Which procedures call procedure "Third" directly or indirectly?

Select p **such that** Calls* (p, "Third")

-- *answer:* procedures First and Second

Q: Which procedures are called from "First"?

Select p **such that** Calls ("First", p)

-- *answer*: procedure Second

Q: Which variables have their values modified in procedure "First"?

variable v; procedure p;

Select v **such that** Modifies (p, v) **with** p.procName = "First"

Select v **such that** Modifies ("First", v)

-- *answer:* variables x, z, i, y, v

Examples of program queries in PQL, cont.

Q. Which variables are used in procedure "Second"?

variable v;

Select v **such that** Uses ("Second", v)

-- *answer*: variable x, y, i, z

Q. Which procedures modify variable "x"?

procedure p;

Select p **such that** Modifies (p, "x")

-- *answer*: procedures First and Second

Examples of program queries in PQL, cont.

Q. Which procedures modify variable "x" and at the same time call procedure "Third"?

procedure p;

Select p **such that** Modifies (p, "x") **and** Calls (p,"Third")

-- *answer*: procedure Second

Comments on PQL conventions

- All program design entities, relationships and attributes referred to in a query must be defined in a program design model

procedure p; variable v;

-- declaration of synonyms to be used in the query: p represents program entity "procedure" and v – "variable"

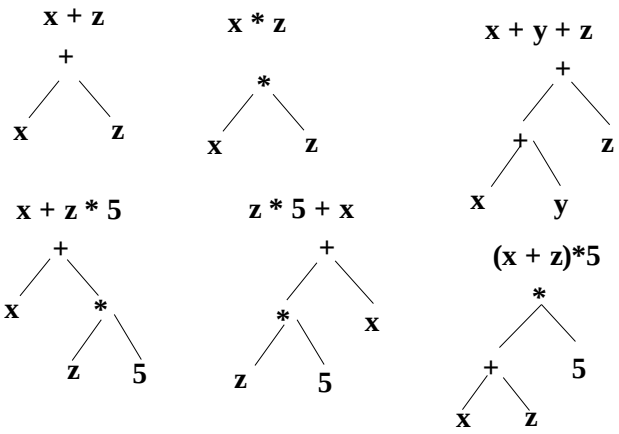
Select p – specifies query result procedures in this case

such that clause constrains the results in terms of relationship membership

with clause constrains the results in terms of attribute values

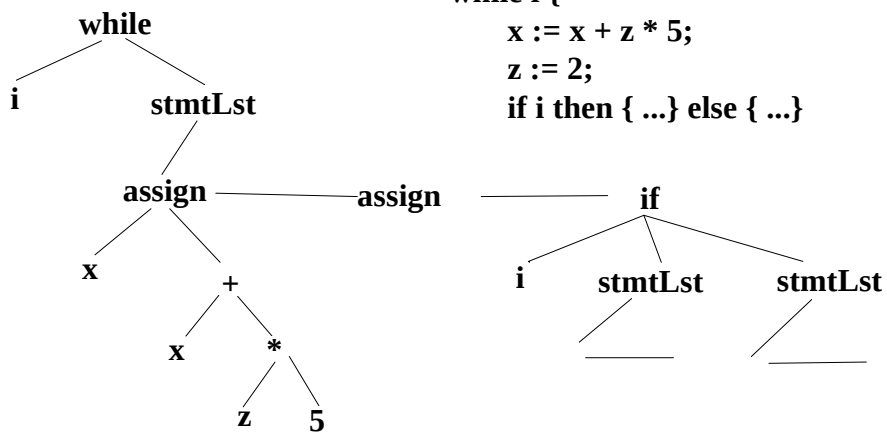
Abstract Syntax Tree (AST)

- AST depicts syntactic structure of a program
- examples of expressions as ASTs:



Example: while loop as an AST

```
while i {
    x := x + z * 5;
    z := 2;
    if i then { ...} else { ...}
```



Abstract syntax of SIMPLE

Meta symbols:
a* - repetition 0 or more times of a
a+ - repetition 1 or more times of a
'|' *means or*

Lexical tokens:
LETTER : A-Z | a-z -- capital or small letter
DIGIT : 0-9
NAME, VAR : LETTER (LETTER | DIGIT | '#')*
procedure, variable and attribute names are strings of letters, digits and '#', starting with a letter
INTEGER : DIGIT+ -- constants are sequences of digits

Abstract syntax of SIMPLE

program : procedure+
procedure : stmtLst
stmtLst : stmt+
stmt : assign | call | while | if
assign : variable expr
expr : plus | minus| times | ref
plus : expr expr
minus : expr expr
times : expr expr
ref : variable | constant
while: variable stmtLst
if : variable stmtLst stmtLst

Relationships Follows and Parent

Follows (s1, s2) – statement s2 appears ***immediately after*** s1 at the same nesting level

Follows* (s1, s2) – statement s2 appears ***after*** s1 at the same nesting level

Parent (s1, s2) – s2 is directly nested in s1

Parent* (s1, s2) – s2 is nested in s1

Examples from procedure Second:

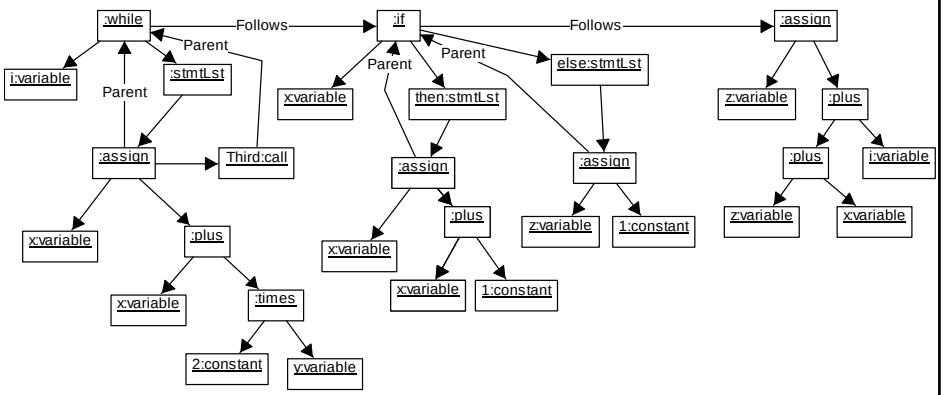
Follows (1,2), Follows (2,3), Follows (4,5)

Follows (3,7), Follows (7,10)

Follows *(1,2), Follows * (1,3), Follows * (1,7), Follows* (1,10)

Parent (3,4), Parent (3,5), Parent(3,6), Parent (7,8), Parent (7,9)

Partial AST for procedure Second



Searching for code patterns in PQL, cont.

Q. Find all the while loops
while w;
Select w
-- *answer*: statement #3
Q. Find while loops with "i" as a control variable
while w;
Select w **pattern** w ("i",_)
-- *answer*: statement #3
Q. Find while loops with assignment to variable "x"
assign a; while w; variable v;
Select a **such that** Parent* (w, a) **pattern** a ("x", _)
-- *answer*: statement #4

Programmers often trace control and data flow paths in a program

- if I change value of 'i' at statement #2 - which statements can be affected?

```
procedure Second {
1.  x = 0;
2.  i = 5;
3.  while i  {
4.     x = x + 2*y;
5.    call Third;
6.     i = i - 1; }
7.  if x then {
8.     x = x+1; }
     else {
9.     z = 1; }
10. z = z + x + i;
11. y = z + 2;
12.  x = x * y + z; }
```

Control flow relationships Next and Next*

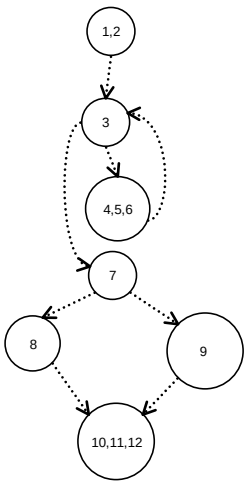
Relationship Next() defines control flow:

Next (n1, n2) – program line n2 can be executed ***immediately after*** n1 in some execution sequence

Next* (n1, n2) - program line n2 can be executed ***after*** n1 in some execution sequence

A control flow graph (CFG) for procedure Second

```
procedure Second {
1.  x = 0;
2.  i = 5;
3.  while i  {
4.     x = x + 2*y;
5.    call Third;
6.     i = i - 1; }
7.  if x then {
8.     x = x+1; }
     else {
9.     z = 1; }
10. z = z + x + i;
11. y = z + 2;
12.  x = x * y + z; }
```



Examples

```
procedure Second {
1.  x = 0;
2.  i = 5;
3.  while i  {
4.     x = x + 2*y;
5.    call Third;
6.     i = i - 1; }
7.  if x then {
8.     x = x+1; }
    else {
9.     z = 1; }
10. z = z + x + i;
11. y = z + 2;
12.  x = x * y + z; }
```

Next (1,2)

Next (2,3)

Next (3,4)

Next (3,7)

Next (6,3)

Next (7,8)

Next (7,9)

Next (8,10)

Next (9,10)

Next* (1,2)

Next* (1,3)

Next* (3,4)

Next* (4,3)

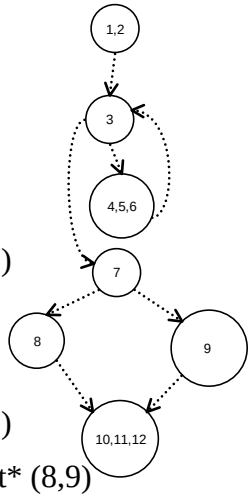
Next* (1,12)

Next* (2,5)

Next* (5,9)

Next* (5,12)

Next* (8,9)



Data flow relationships Affects and Affects*

Affects (a1, a2) holds if the value of 'v' as computed at assignment a1 can be actually used in a2

- there must be a computational path in the CFG from a1 to a2 on which v is not modified

Affects* (a1, a2) holds if a1 has either direct or indirect impact on a2

Affects(a1, x) Affects (x, y) Affects (y, z) Affects (z, a2)

Examples

```
procedure Second {
1.  x = 0;
2.  i = 5;
3.  while i  {
4.     x = x + 2*y;
5.    call Third;
6.     i = i - 1; }
7.  if x then {
8.     x = x+1; }
    else {
9.     z = 1; }
10. z = z + x + i;
11. y = z + 2;
12.  x = x * y + z; }
```

Affects (1, 4)

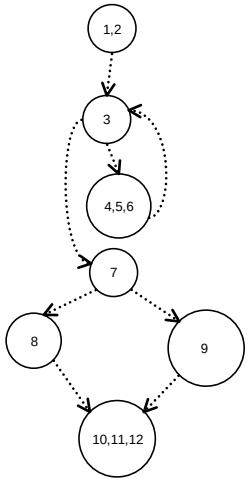
Affects (2, 6)

Affects (4, 8)

Affects(4, 10)

Affects(9, 10)

Affects(9,11)



Affects

1. x=a;
2. call p;
3. v=x;

Affects (1,3) ??

if procedure p does NOT modify “x” - Affects (1,3) HOLDS

otherwise - Affects (1,3) does NOT HOLD

Affects

```
procedure p {
1.      x = 1;
2.      y = 2;
3.      z = y;
4.      call q;
5.      z = x + y + z;          }
procedure q {
6.      x = 5;
7.      if z then {
8.              t = x + 1; }
    else {
9.              y = z + x; } }
```

Affects (3,5) ??
Affects (1,5) ??
Affects (2,5) ??

Affects*

1. x=a;
2. v=x;
3. z=v;

Affects*(1,3) ??

Examples from procedure Second:

Affects*(1, 4), Affects*(1, 11), Affects*(1, 12)

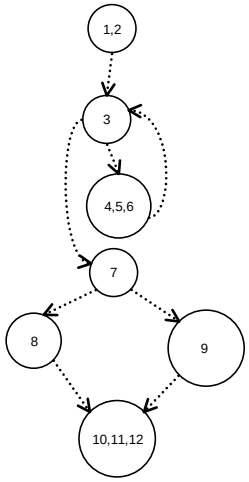
Examples

```
procedure Second {
1.  x = 0;
2.  i = 5;
3.  while i  {
4.     x = x + 2*y;
5.    call Third;
6.     i = i - 1; }
7.  if x then {
8.     x = x+1; }
    else {
9.     z = 1; }
10. z = z + x + i;
11. y = z + 2;
12.  x = x * y + z; }
```

Affects*(1, 4)

Affects*(1, 11)

Affects*(1, 12)



Now we can ask program queries such as:

Is there a control path from statement #2 to statement #9?

Is there a control path from statement #2 to statement #9 that passes through statement #8?

Find all program lines that can be executed between line #5 and line #12.

Which assignments affect the value computed at assignment #12 (directly or indirectly)?

Writing program queries in PQL

Q. Is there a control path from statement #2 to statement #9?

Select BOOLEAN **such that** Next* (2, 9)

-- *answer*: TRUE

Q. Is there a control path from statement #2 to statement #9 that passes through statement #8?

Select BOOLEAN **such that** Next* (2, 8) **and** Next* (8, 9)

-- *answer*: FALSE

Writing program queries in PQL, cont.

Q. Which program lines can be executed between line #5 and line #12?

prog_line n;

Select n **such that** Next* (5, n) **and** Next* (n, 12)

-- *answer*: lines #6, #3, #4, #5, …, #7, #8, #9, #10, #11

Q. Which assignments directly affect value computed at assignment #12?

assign a;

Select a **such that** Affects (a, 12)

-- *answer*: assignments #1, #4, #8, #10, #11

Q. Which assignments directly or indirectly affect value computed at assignment #10?

assign a;

Select a **such that** Affects (a, 10)

-- *answer*: statements #9, #1, #4, #8, #2, #6

Writing program queries in PQL, cont.

Q. Find assignments to variable "x" located in a loop, that can be reached (in terms of control flow) from statement #1

assign a; while w;

Select a **pattern** a("x", _) **such that** Parent* (w, a)
and Next* (1, a)

-- *answer*: statement #4

Select a **such that** Modifies (a, "x") **and** Parent* (w, a)
and Next* (1, a)

-- *answer*: statement #4

Notice that the two queries above yield the same result for SIMPLE programs. Notice also that this might not be the case in other languages. Why?

Final comment on program queries

Changing the order of conditions in a query does not change the query result:

assign a; while w;

Select a **such that** Modifies (a, "x") **and** Parent* (w, a) **and** Next* (1, a)

Select a **such that** Parent* (w, a) **and** Modifies (a, "x") **and** Next* (1, a)

Select a **such that** Next* (1, a) **and** Parent* (w, a) **and** Modifies (a, "x")

Select a **pattern** a ("x", _) **such that** Parent* (w, a) **and** Next* (1, a)

Select a **such that** Parent* (w, a) **and** Next* (1, a) **pattern** a ("x", _)

Select a **such that** Next* (1, a) **and** Parent* (w, a) **pattern** a ("x", _)

-- *answer*: statement #4

BUT: Changing the order of conditions may affect query evaluation time

--- The end of set #2 ---