Pattern Matching Security Properties of Code using Dependence Graphs

John Wilander
and Pia Fåk

Computer Science
Linköpings universitet

Comparison of Static Tools

Five static analysis tools trying to find security bugs

Tool	True Positives	False Positives	True Negatives	False Negatives
Flawfinder	96%	71%	29%	4%
ITS4	91%	52%	48%	9%
RATS	83%	67%	33%	17%
Splint	30%	19%	81%	70%
BOON	27%	31%	69%	73%

New Tools On The Block

- Cqual, Metal/xgcc, MOPS, IPSSA, Mjonir and Eau Claire
- Program models: CFG, AST, PDA, SSA ...
- Security flaws: buffer overflows, integer flaws, race conditions ...

Drawbacks

- Check for one or two flaws each
- Different models and analyses
- Textual modeling of flaws
- Textual feedback to user
- Lack of ranking

Textual Models

```
// Start state. Matches any copy_from_user
// call and puts parameter x in tainted state.
start: { copy_from_user(x, y, len) }
        ==> x.tainted
;
// Catch operations illegal on unsafe values.
x.tainted, x.need_ub, x.need_lb:
   { v[x] } ==>{ err(”Dangerous index!”); }
...
;
```

Textual Feedback

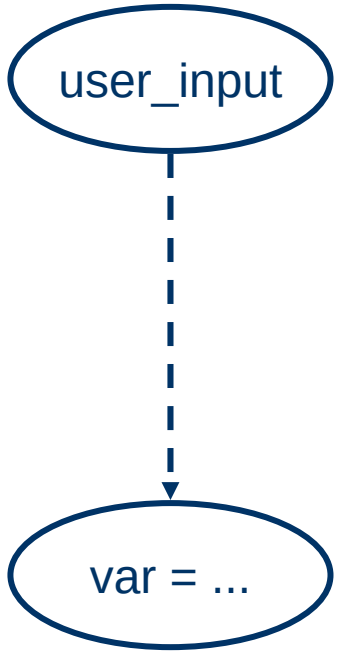
```
Johns_program.c:7:3: Possible out-of-bounds store:

  strcpy(buffer, input_string)

  Unable to resolve constraint:
  requires maxRead(input_string @ test.c:7:18) <= 7

  needed to satisfy precondition:
  requires maxSet(buffer @ test.c:7:10) >=
           maxRead(input_string @ test.c:7:18)

  derived from strcpy precondition:
  requires maxSet(<parameter 1>) >=
           maxRead(<parameter 2>)
```

Research Goal

- More generic formalism
- Ranking of potential flaws
- Visual communcation

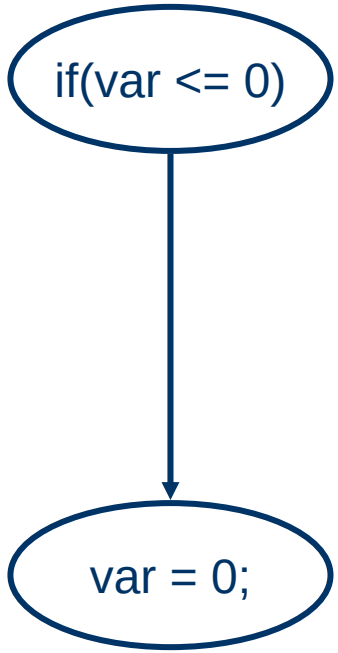
$\Rightarrow$ Our proposal is *dependence graphs*

Data Dependence




```
int func(int user_input){
  int var;
  ...
  var = user_input * 5;
  ...
}
```

Control Dependence

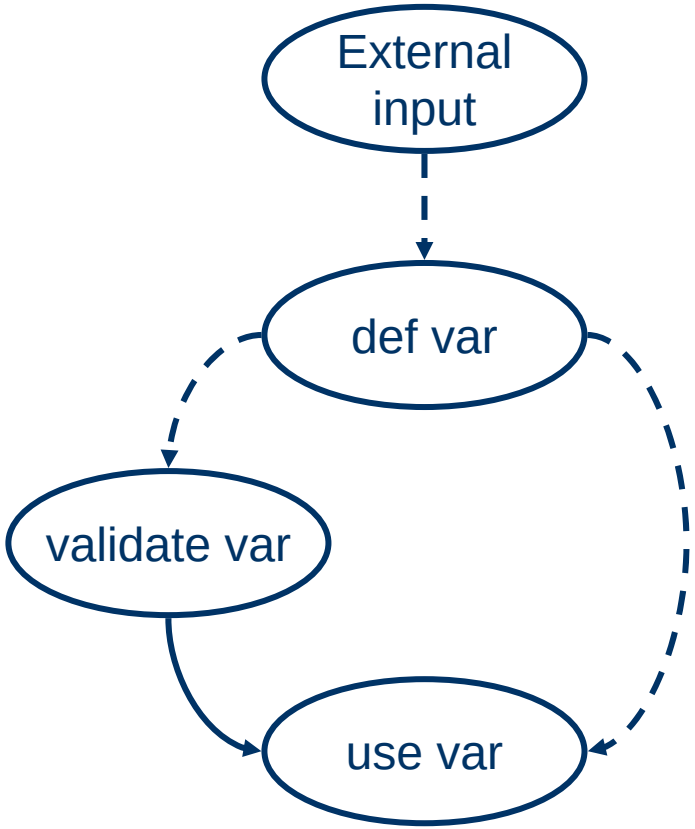



```
int func(int user_input){
  int var;
  ...
  if(var <= 0)
    var = 0;
  ...
}
```

Input Validation

```
int func(int user_input){
  int var;
  ...
  var = user_input * 5;
  ...
  if(var >= 0 && var < MAX)
    for(i = 0; i < var; i++)
      array[i] = 0;
  ...
}
```

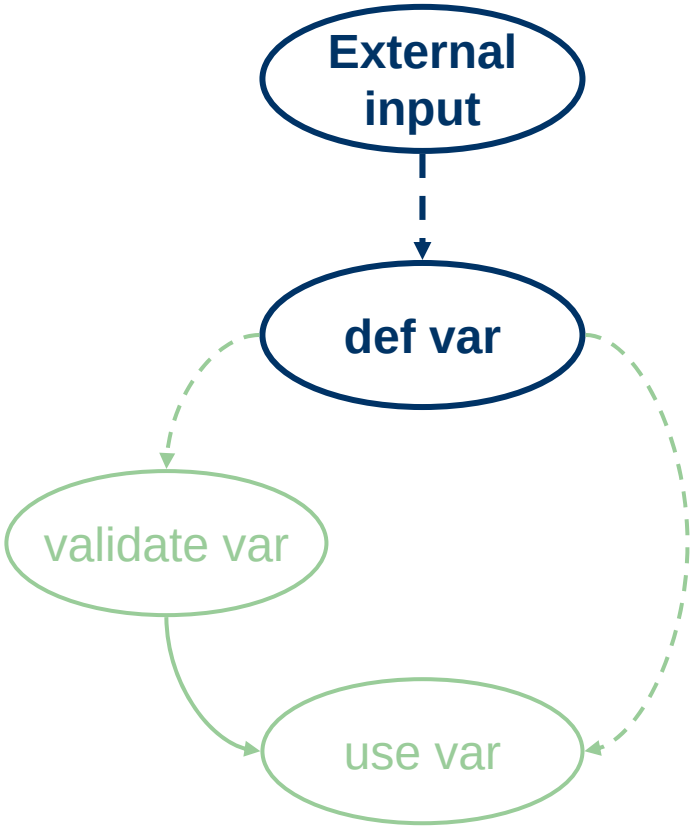
Input Validation




```
int func(int user_input){
  int var;
  ...
  var = user_input * 5;
  ...
  if(var >= 0 && var < MAX)
    for(i = 0; i < var; i++)
      array[i] = 0;
  ...
}
```

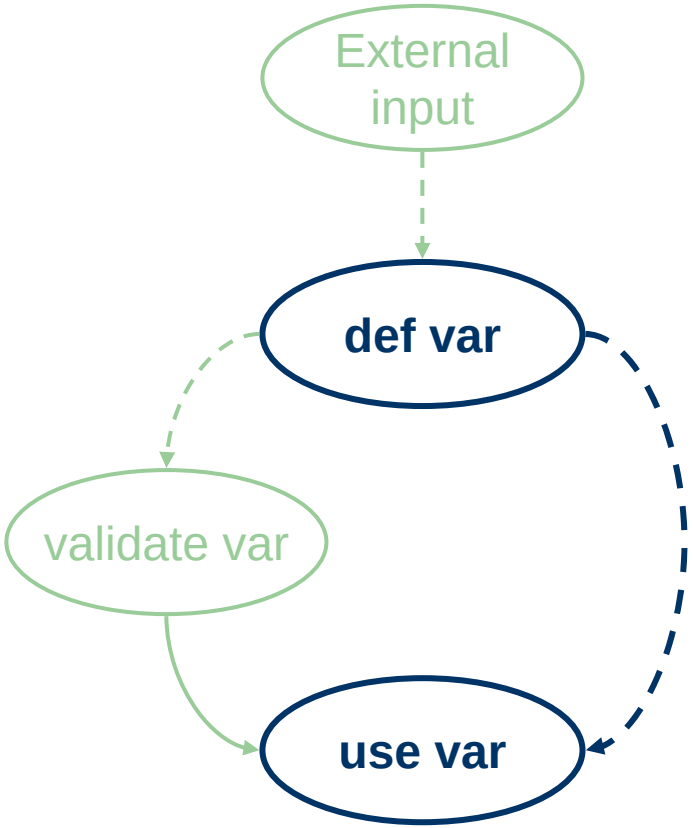
Model of good programming practice

Input Validation



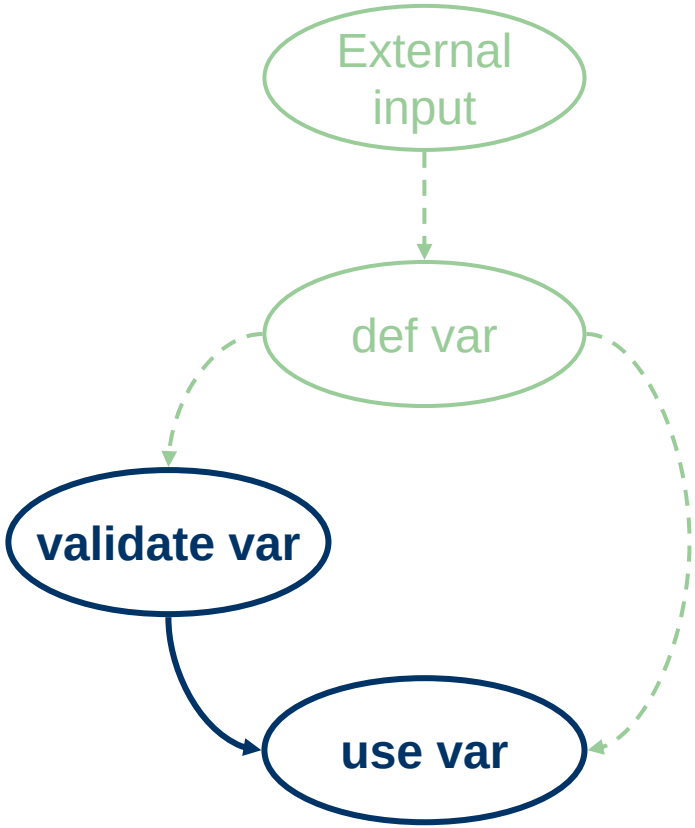

```
int func(int user_input){
  int var;
  ...
  var = user_input * 5;
  ...
  if(var >= 0 && var < MAX)
    for(i = 0; i < var; i++)
      array[i] = 0;
  ...
}
```

Input Validation



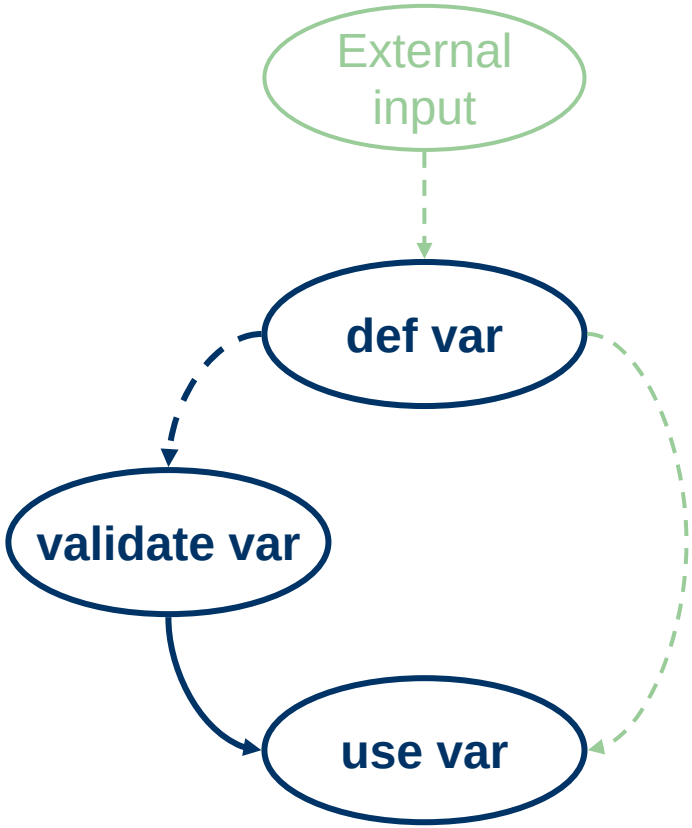

```
int func(int user_input){
  int var;
  ...
  var = user_input * 5;
  ...
  if(var >= 0 && var < MAX)
    for(i = 0; i < var; i++)
      array[i] = 0;
  ...
}
```

Input Validation



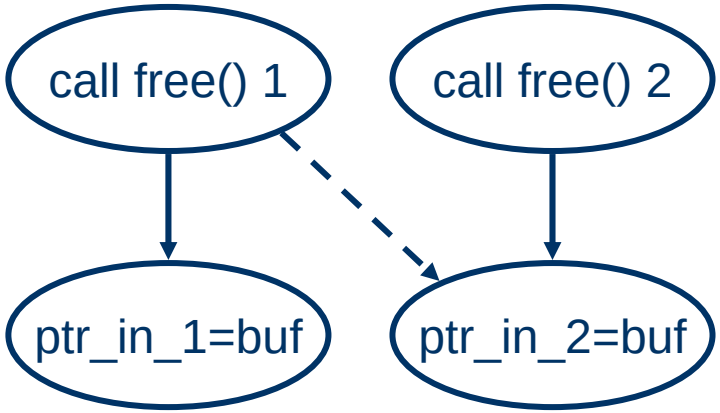

```
int func(int user_input){
  int var;
  ...
  var = user_input * 5;
  ...
  if(var >= 0 && var < MAX)
    for(i = 0; i < var; i++)
      array[i] = 0;
  ...
}
```

Input Validation




```
int func(int user_input){
  int var;
  ...
  var = user_input * 5;
  ...
  if(var >= 0 && var < MAX)
    for(i = 0; i < var; i++)
      array[i] = 0;
  ...
}
```

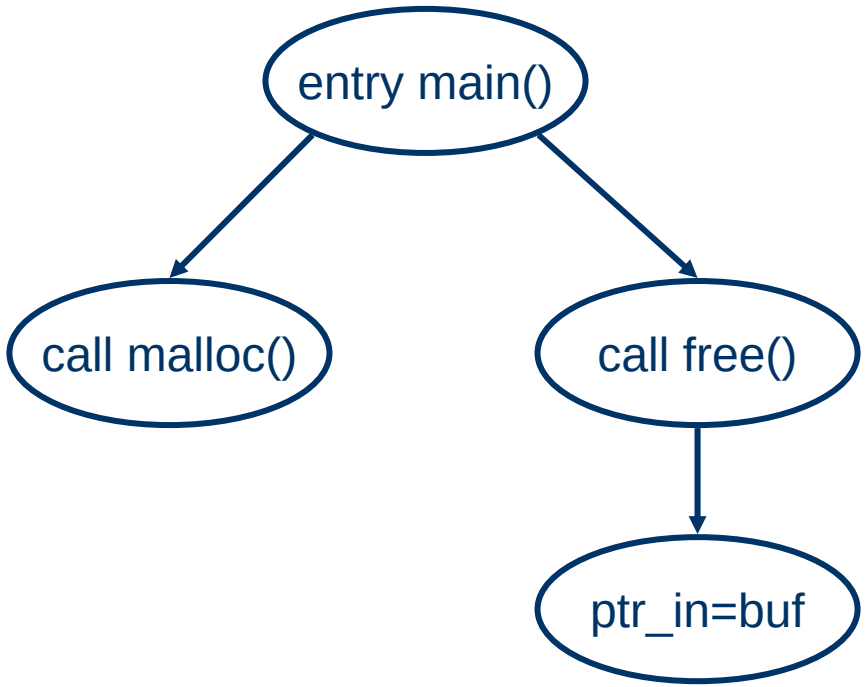
Double free



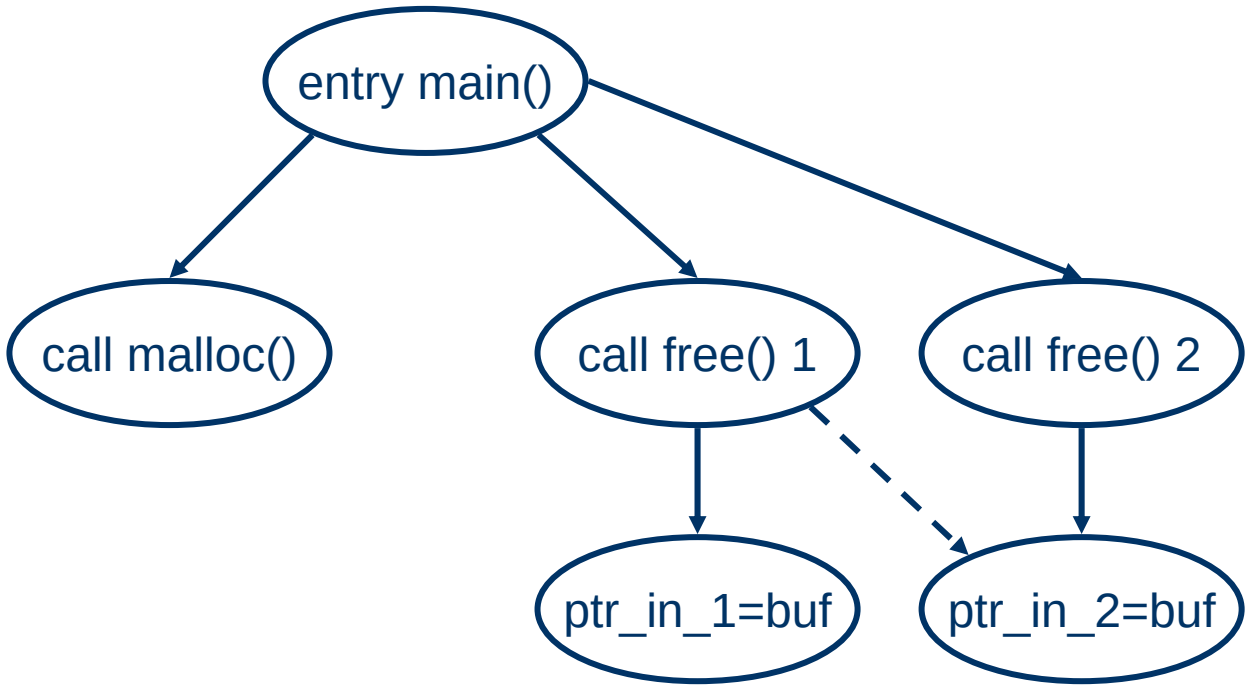

```
char *buf=(char *)malloc(MAX);
...
free(buf);
...
free(buf);
```

Model of bad programming practice

Double free



Double free


entry main()
call malloc()
call free() 1
call free() 2
ptr_in_1=buf
ptr_in_2=buf

Current State

- CodeSurfer $\rightarrow$ XML $\rightarrow$ C++
- 15h for analysis of 20k lines of code
- Pattern matches integer input validation

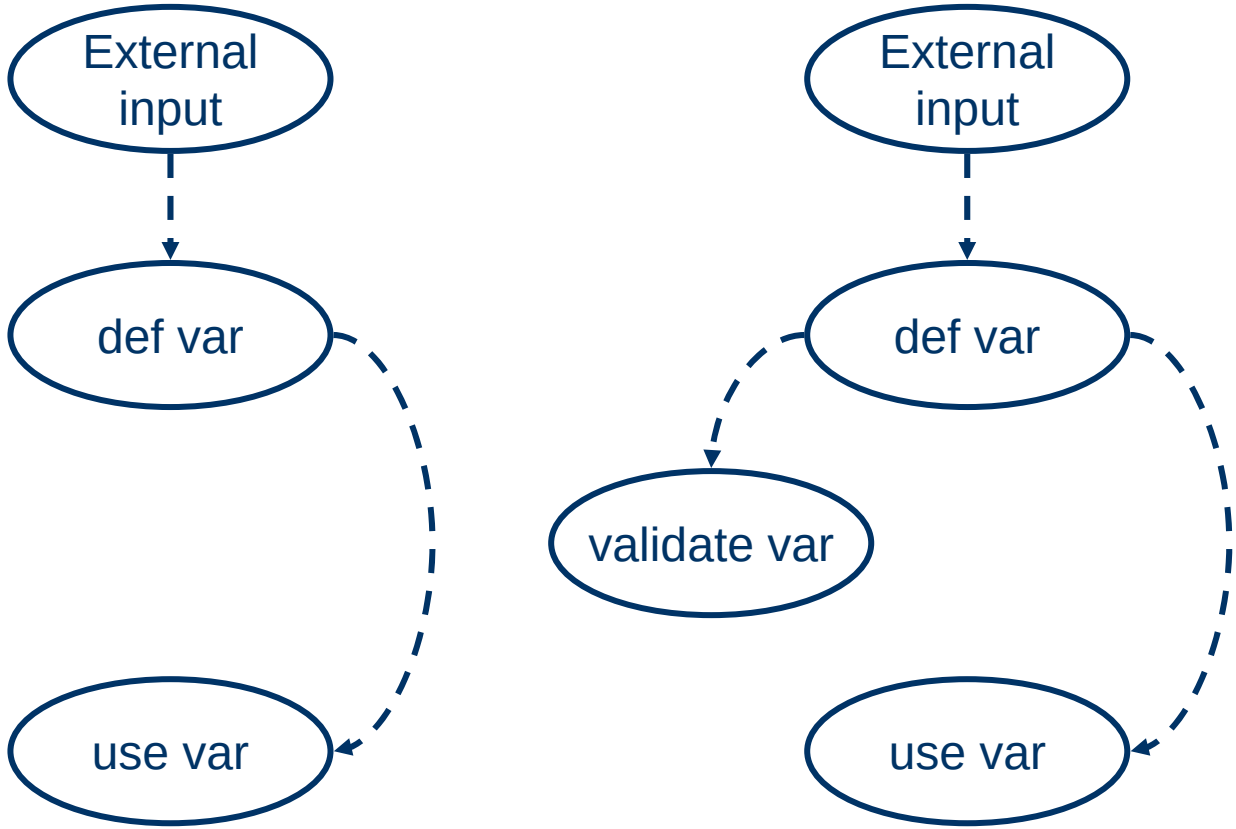
More Generic?

- We believe so
- Models for buffer overflow, integer flaws and double free

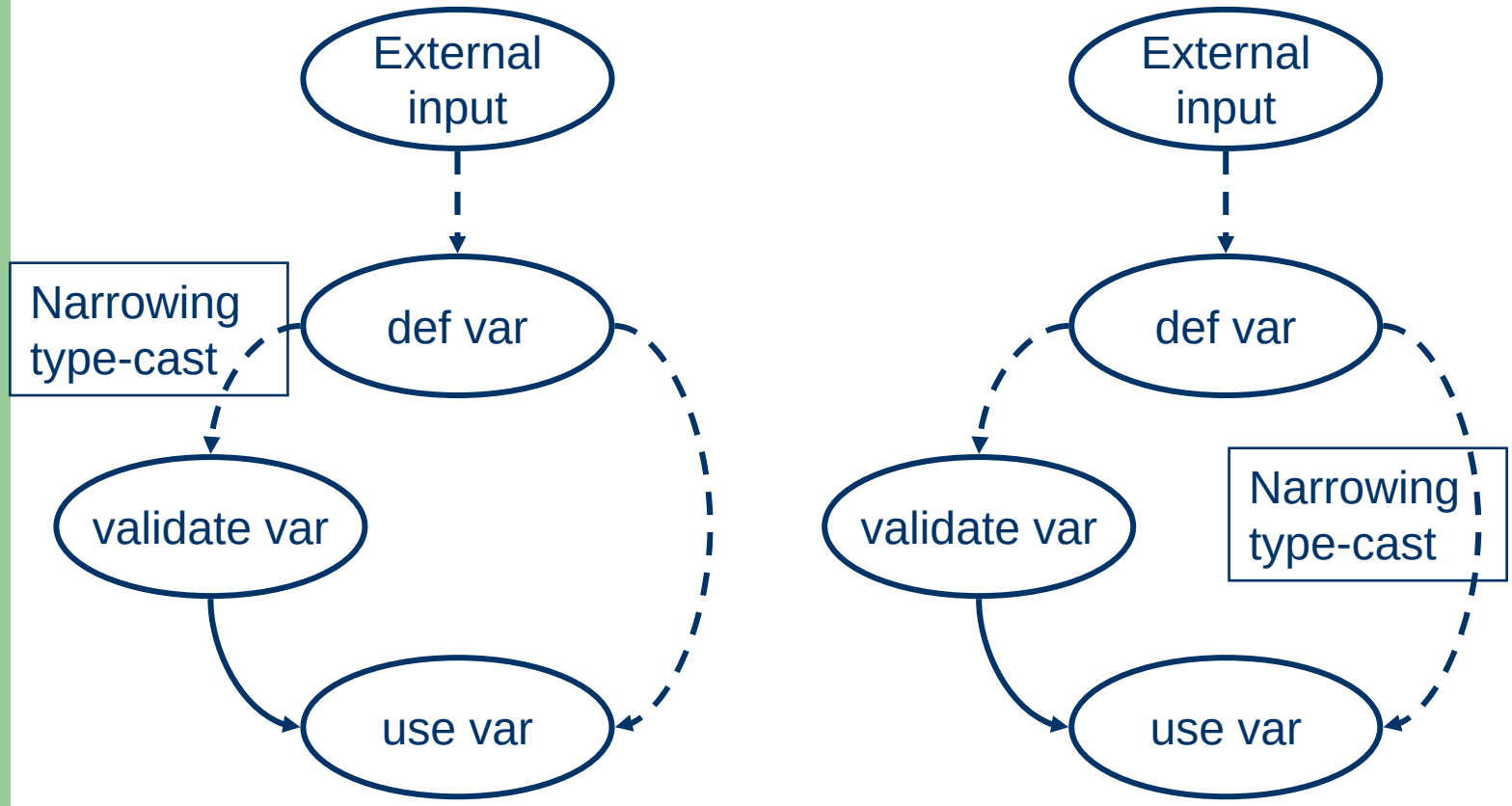
Ranking?

- More inaccuracy $\Rightarrow$ lower rank
- Exploit *dual* of models

Ranking?


External input
def var
use var
External input
def var
validate var
use var

Ranking?


External input
Narrowing type-cast
def var
validate var
use var
External input
def var
validate var
Narrowing type-cast
use var

Visual?

- Graph models of flaw types
- Graph models of reported flaws

Future Work

- Complexity and scalability
- Accuracy
- Generality
- Usability
- Heuristic ranking
- Model Updates