

Soundness and completeness considered harmful

ESCJ20

K.Rustan M. Leino
DIGITAL SRC

Joint work with Cormac Flanagan,
Mark Lillibridge, Greg Nelson, James B. Saxe,
and Raymie Stata.

SRI CS seminar, 4 May 1998

Vision



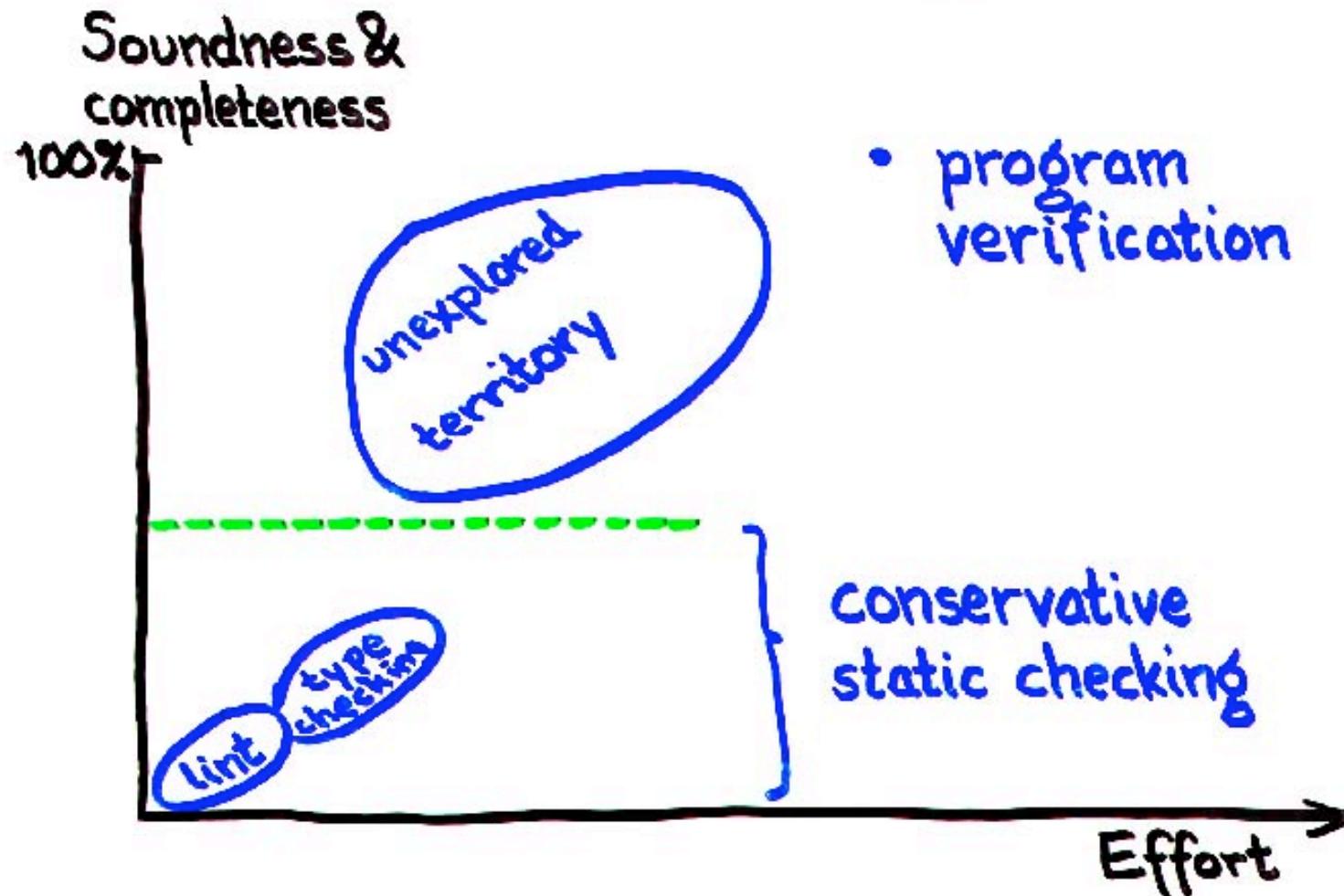
Sound: catch all errors

Complete: no spurious warnings

Benefit vs. cost

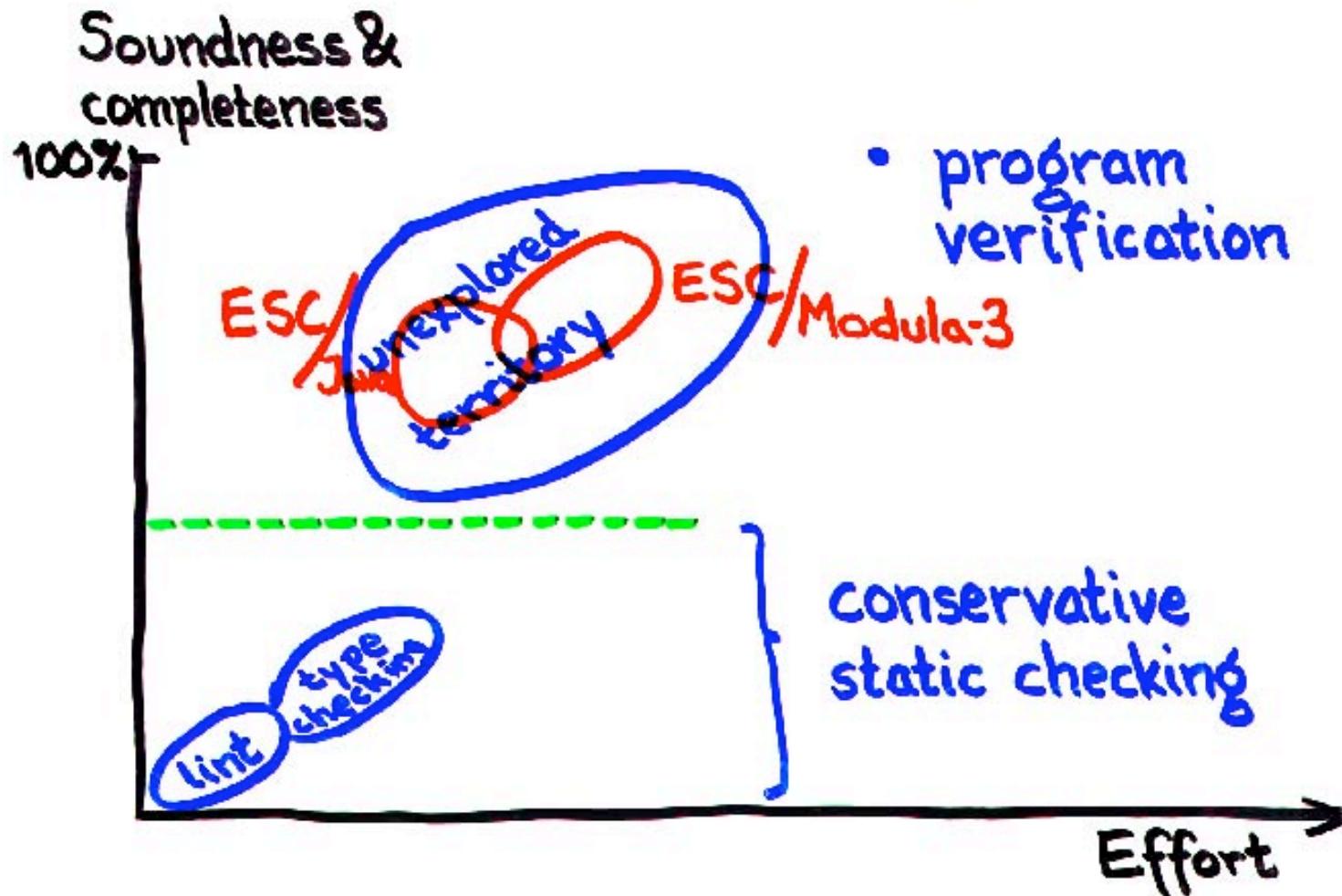
- + Find errors early
 - Annotating the program
 - Running the tool
 - Analyzing the output

Static Checking



Note: Illustration not to scale

Static Checking

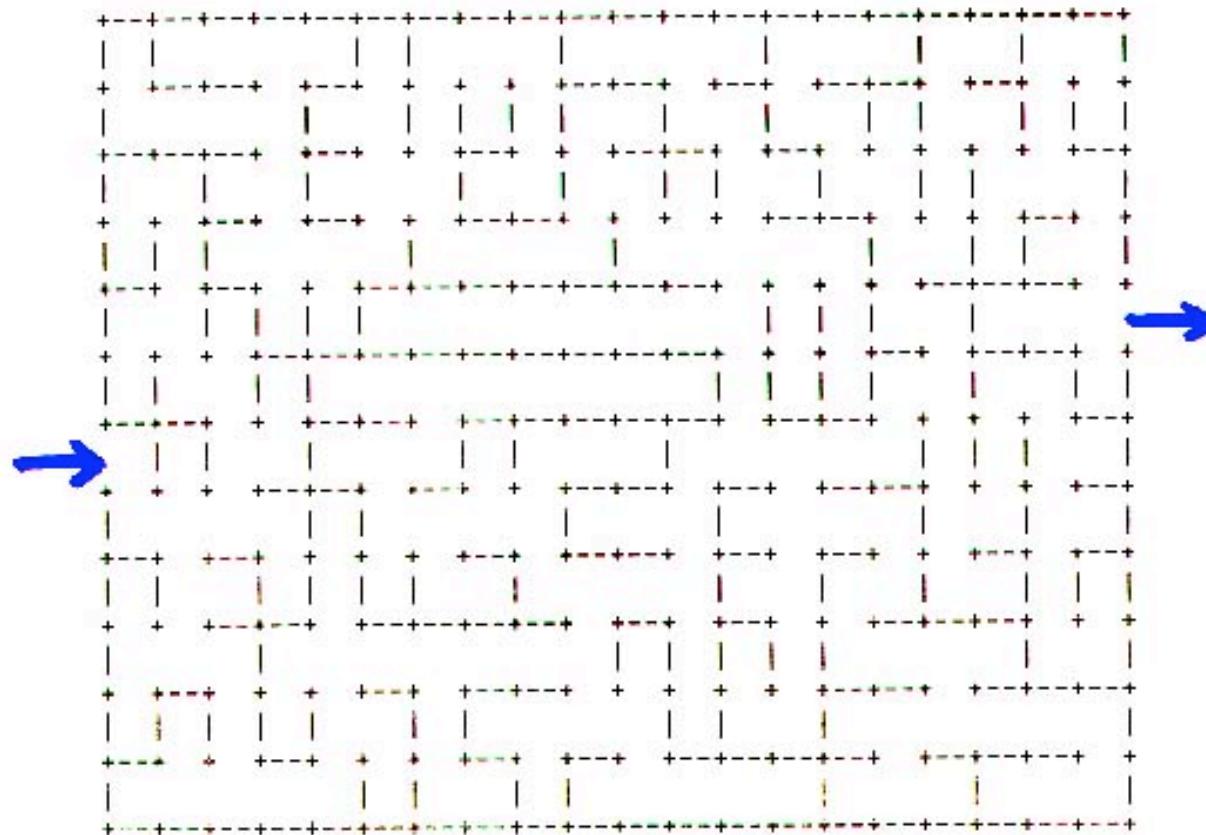


Note: Illustration not to scale

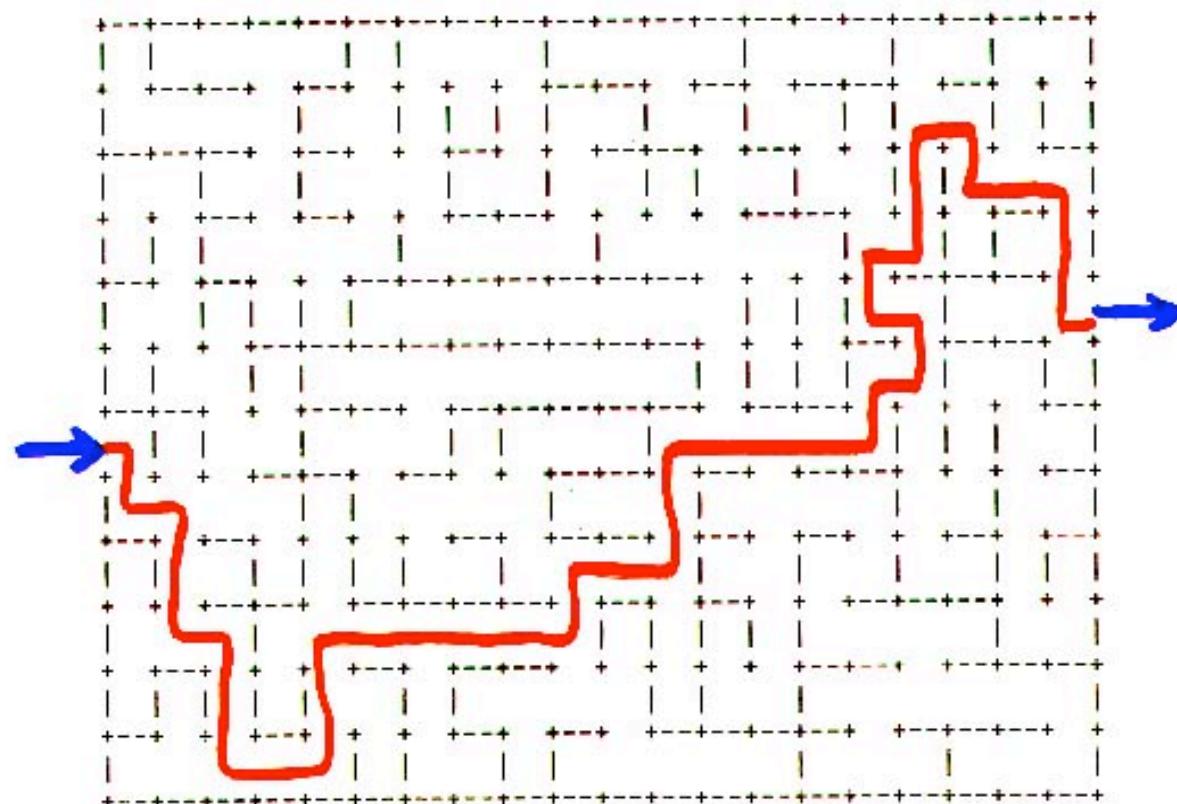
How ESC works



Generating mazes



Generating mazes



Creating a maze

```
uf := new UnionFind;  
uf.Init(...);  
while ( uf.NumberOfClasses() != 1 ) {  
    pick a new door d that joins rooms p,q;  
    if ( uf.Find(p) != uf.Find(q) ) {  
        open door d;  
        uf.Union(p,q);  
    } }  
6.
```

```
class UnionFind {  
    abstract var valid: bool;  
    abstract var size: nat;  
    abstract var state: any;  
  
proc Init( uf: UnionFind; n: nat );  
    requires uf ≠ null;  
    modifies uf.valid, uf.size, uf.state;  
    ensures uf.valid ∧ uf.size = n;  
};
```

:

proc Find(uf: UnionFind; p: nat): nat ;
requires uf ≠ null ∧ uf.valid ∧ p < uf.size ;
modifies uf.state ;
ensures result < uf.size ;

proc Union(uf: UnionFind; p, q: nat) :
requires uf ≠ null ∧ uf.valid ∧ p < uf.size ∧ q < uf.size ;
modifies uf.state ;

proc NumberOfClasses(uf: UnionFind): nat ;
requires uf ≠ null ∧ uf.valid ;
modifies uf.state ;
ensures result ≤ uf.size ;

```
reveal class UnionFind {  
    var r : nat[] ;  
    var numClasses : nat ;  
  
    rep valid ≡  
        r ≠ null  
        ∧ ( $\forall i : \text{nat} :: i < r.\text{length} \Rightarrow r[i] < r.\text{length}$ )  
        ∧ numClasses ≤ r.length ;  
  
    rep size ≡ r.length ;  
  
depends state on numClasses, r[*] ;  
    :  
}
```

```
proc Union( uf: UnionFind; p,q:nat ) {  
    var rp := uf.Find(p);  
    var rq := uf.Find(q);  
    if ( rp ≠ rq ) {  
        if ( ... )  
            uf.r[rp] := rq ;  
        else  
            uf.r[rq] := rp ;  
    } } uf.numClasses-- ;
```

```
proc Union( uf: UnionFind; p,q:nat ) {  
    var rp := uf.Find(p);  
    var rq := uf.Find(q);  
    if ( rp ≠ rq ) {  
        if ( ... )  
            uf.r[rp] := rq ;  
        else  
            uf.r[rq] := rp ;  
        assume 0 < uf.numClasses;  
        uf.numClasses -- ;  
    } }  
}
```

On the design of ESC/Java

- Simplify annotation language
- Improve robustness and performance
- Enhance error reporting

Validity vs. object invariants

abstract var valid: bool;

:

requires uf.valid;

:

rep valid \equiv r \neq null \wedge ...

\wedge numClasses \leq r.length;

* * *

inv r \neq null;

:

inv numClasses \leq r.length;

Problem with object invariants

```
proc P( a:A; b:B; c:C ) {  
    :  
    x.Q( y, z ) :  
    :  
}
```

Specifying modifications

abstract var state: any;

:

modifies uf.state;

:

depends state on numClasses, ... ;

* * *

modifies uf.numClasses, ... ;

ESC|Java modifies checking

Callers: Use the specified modifies clause, except for what is not visible to caller

Implementation:

- Level 0: anything can be modified
- Level 1: fields of parameters subject to specified modifies clause

Conclusions

- Verification technology can find more errors than conservative methods
- Sound + complete \neq useful
- Automate and find the right errors

www.research.digital.com/SRC/esc/Esc.html