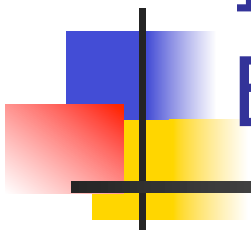
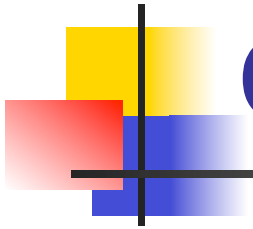


Inferring Specifications to Detect Errors in Code

Mana Taghdiri

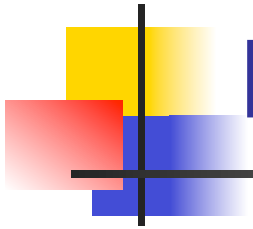
Presented by: Robert Seater

MIT Computer Science & AI Lab



Outline

- Problem
- Overview and motivation
- Details
- Experimental results
- Related work
- Conclusions



Problem Statement

Software model checkers ignore code modularization

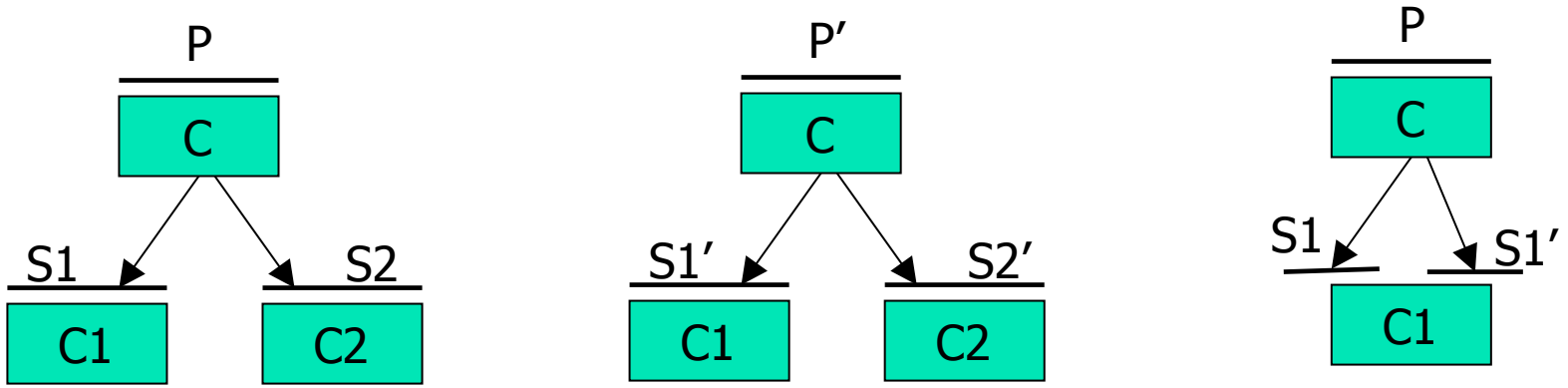
- Procedures are treated as control constructs
- Procedure boundaries not exploited in the analysis
- This is odd !

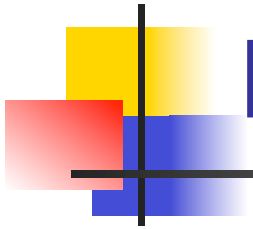
Problem Statement

In contrast,

Traditional methods are based on code structure

- Examples: ESC/Java , Jalloy
- User has to provide procedure specs





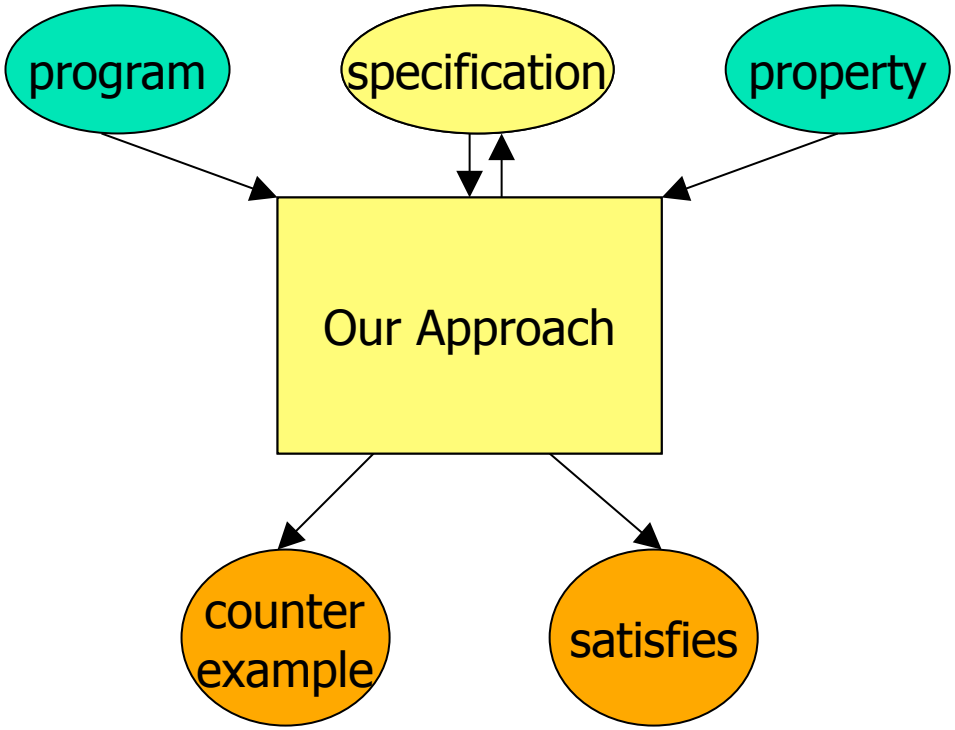
Example

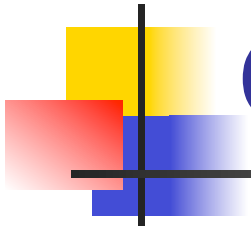
if (b) **then** x is acyclic

```
procedure f(bool b, List x, List y) {
    g(b, x);
    h(y);
}

procedure g(bool b, List x) {
    if (b) then mutate x;
    else mutate' x;
}
```

User's View


program
specification
property
Our Approach
counter example
satisfies



Our Approach

- A procedure-based automatic analysis
  - Procedures => modularity => help scale
- To detect bugs
  - Reported bugs are not spurious
- Procedure specs are inferred automatically
  - Property-dependent
  - Call site-dependent
- A counterexample-guided refinement technique

Example

if (b) **then** x is acyclic

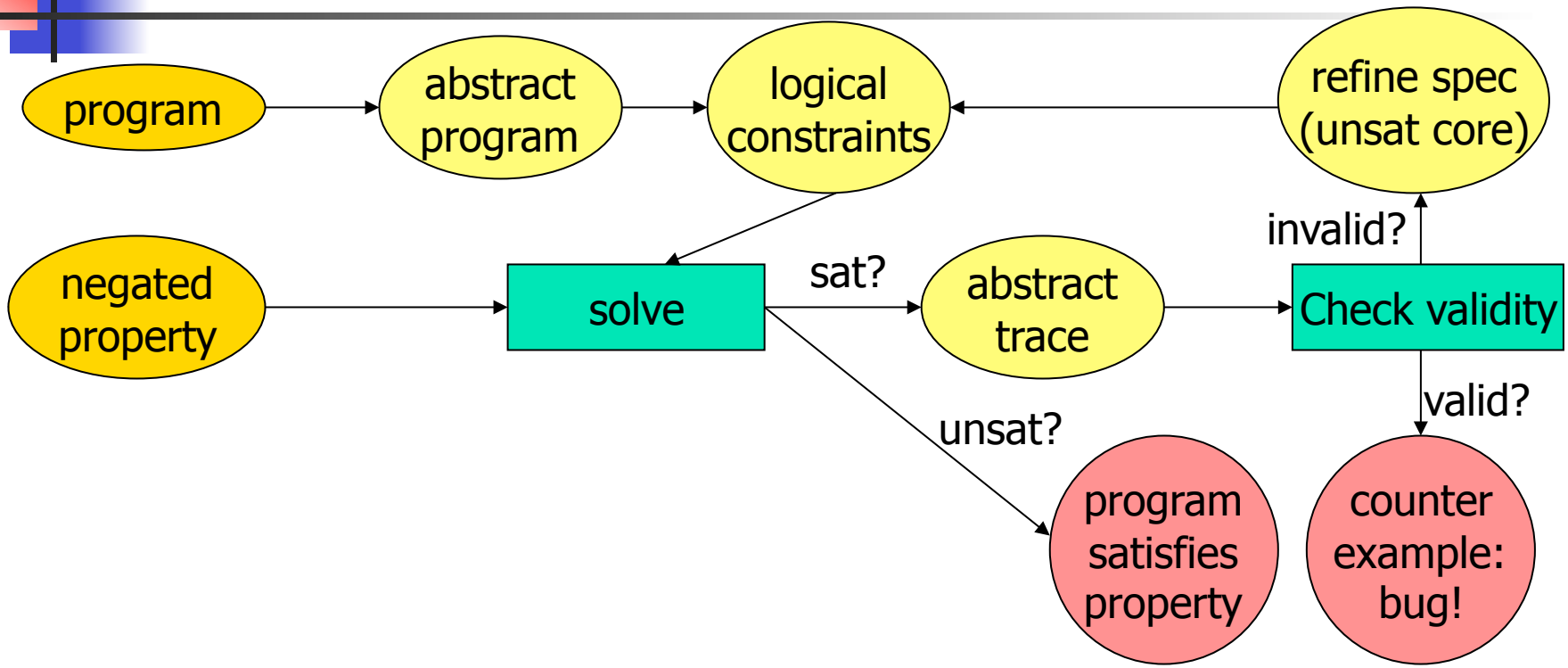
```
procedure f(bool b, List x, List y) {
    g(b, x);
    h(y);
}

procedure g(bool b, List x) {
    if (b) then mutate x;
    else mutate' x;
}
```

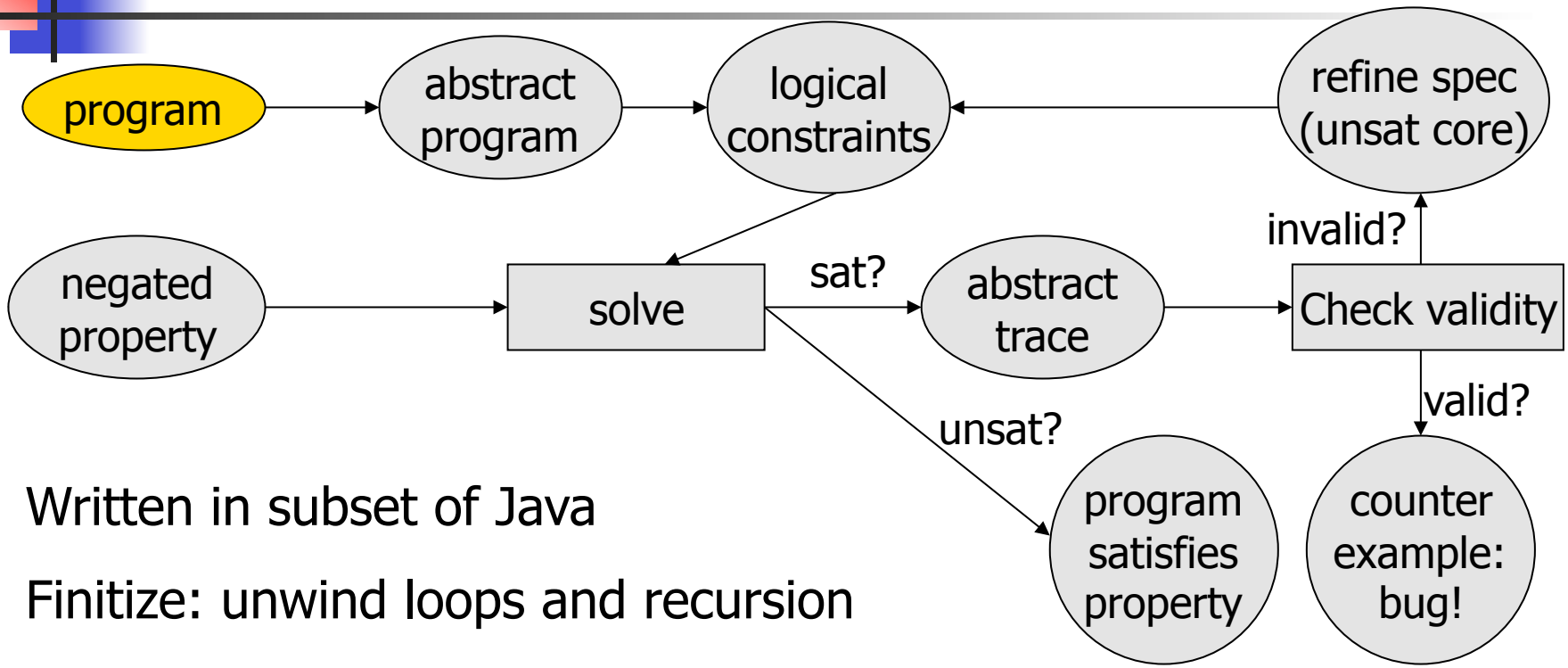
b=true
x=empty
g(b,x)=mutated x
y=…
h(y)=…

mutate(empty) is acyclic

Algorithmic Overview



Algorithmic Overview

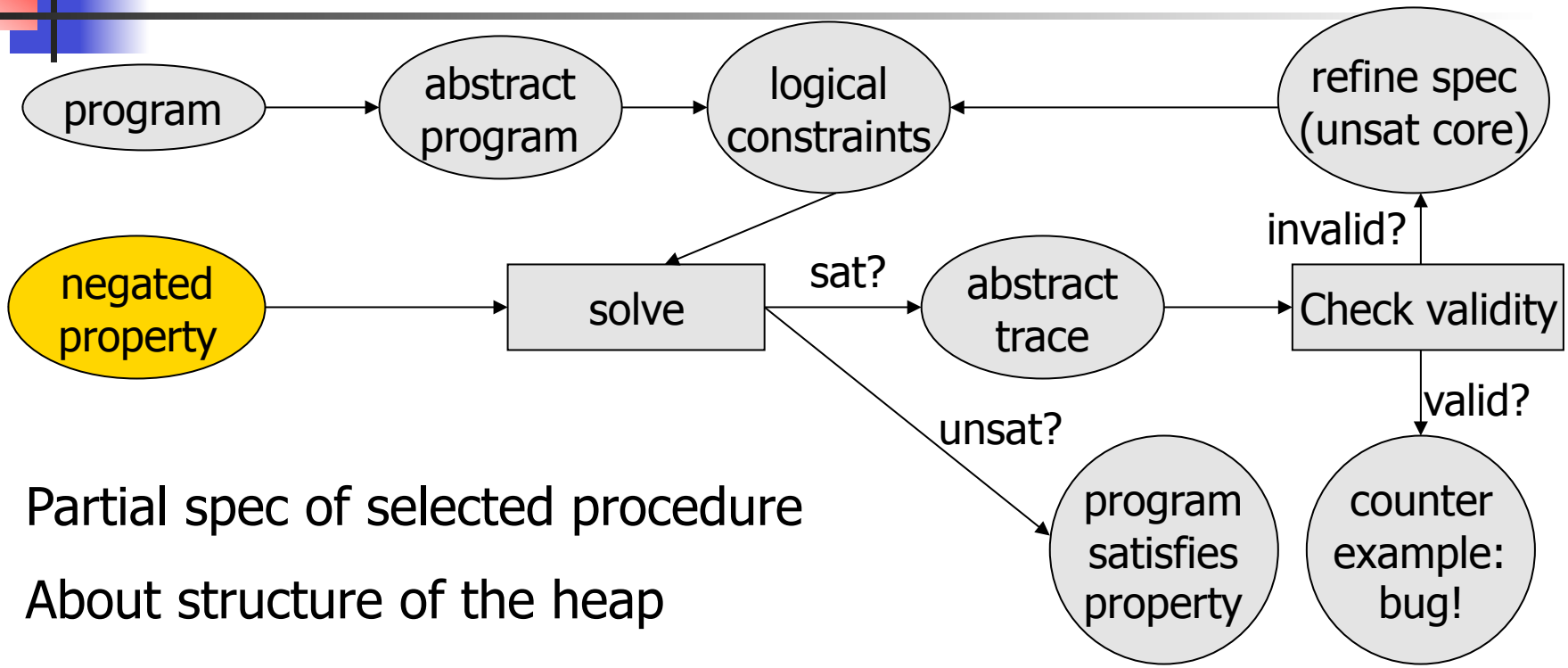


Written in subset of Java

Finitize: unwind loops and recursion

based on a user-provided number

Algorithmic Overview



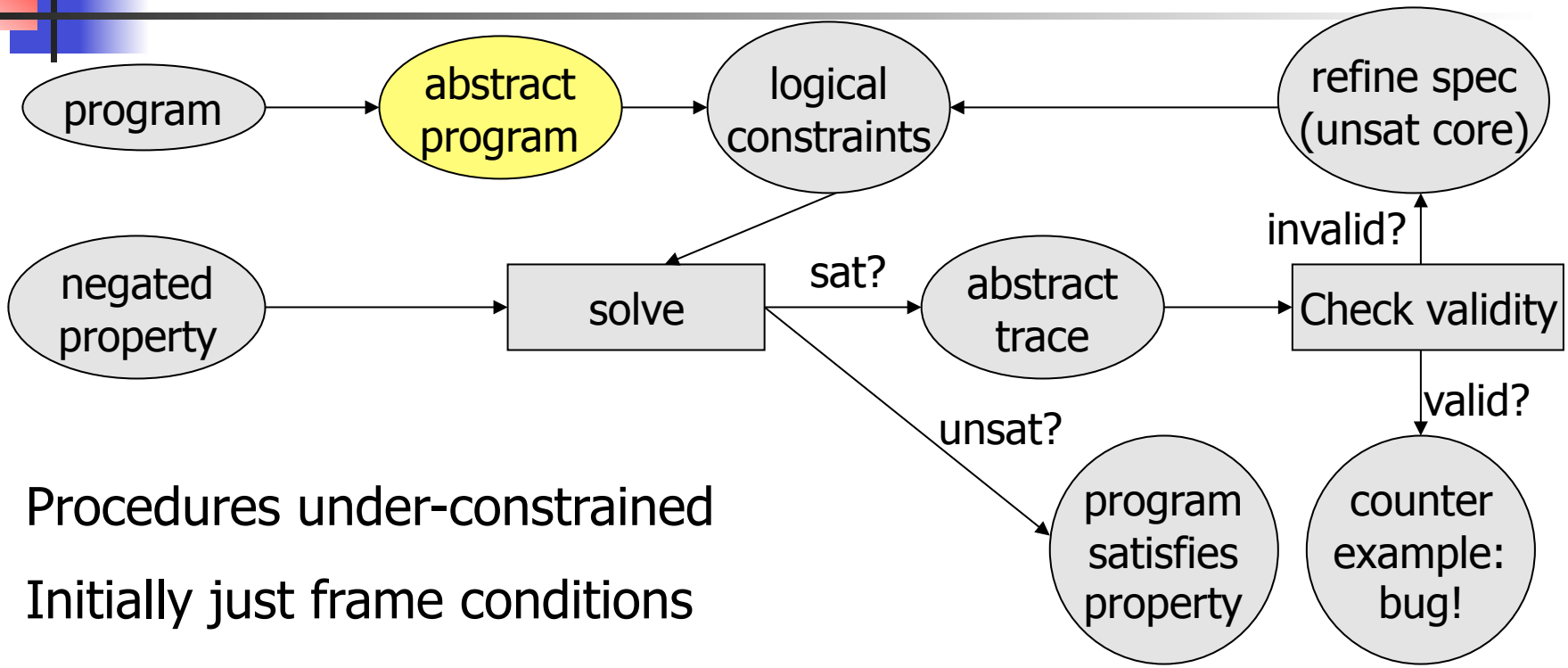
Partial spec of selected procedure

About structure of the heap

Written in logical constraints

Solution=counterexample

Algorithmic Overview

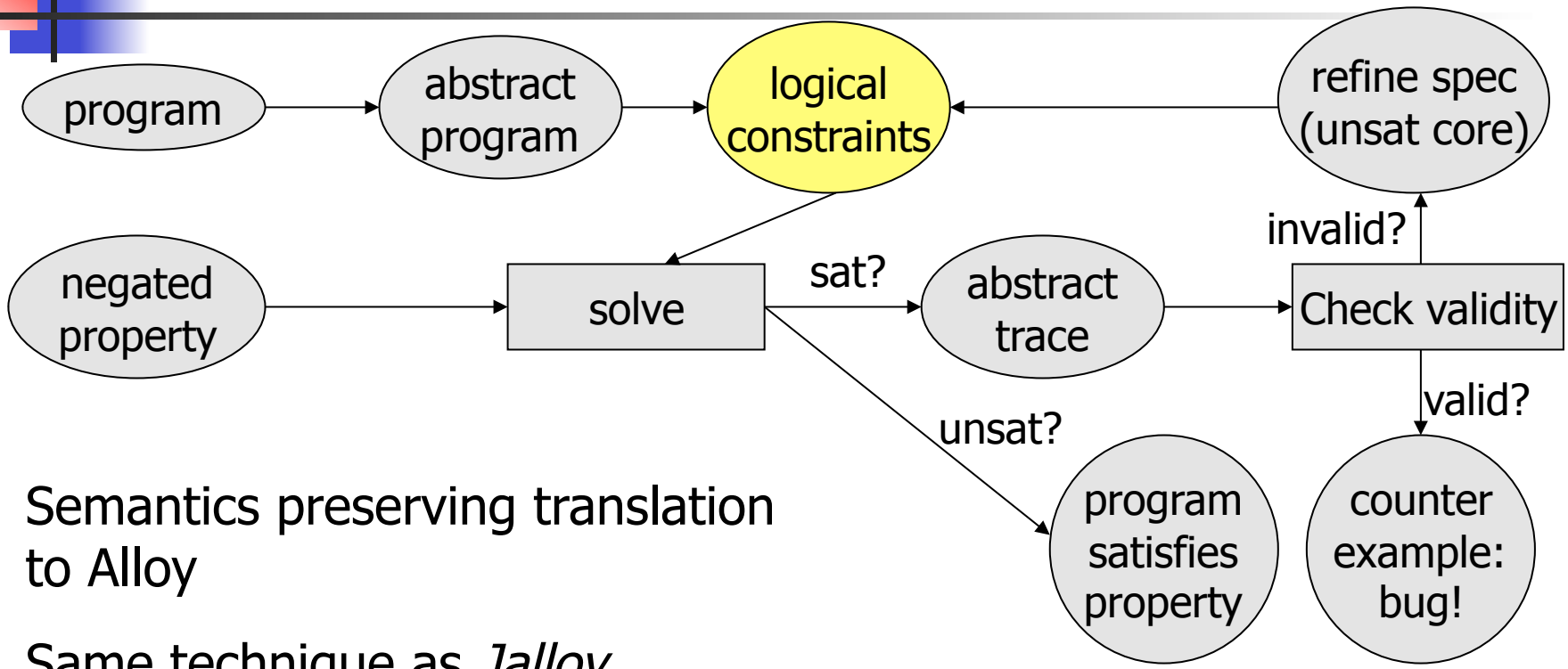


Procedures under-constrained

Initially just frame conditions

(generated bottom up)

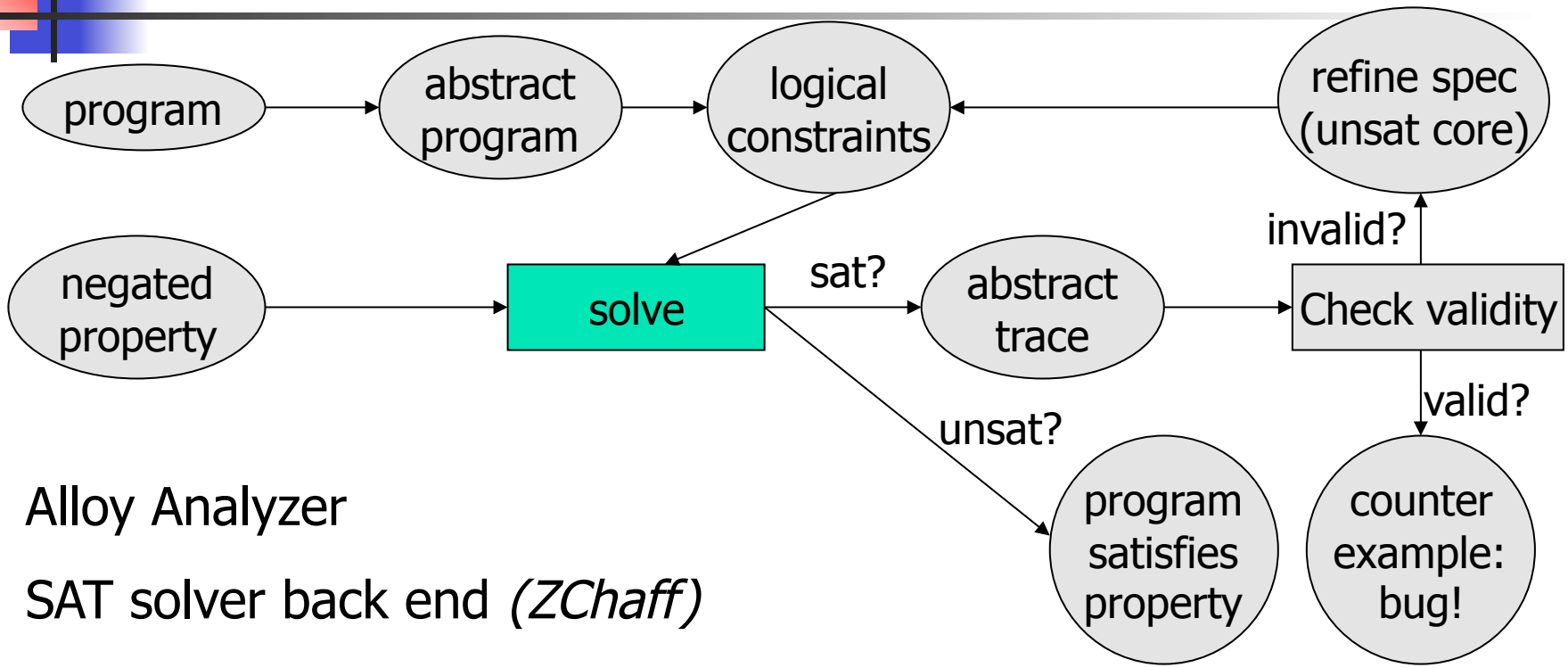
Algorithmic Overview



Semantics preserving translation to Alloy

Same technique as *Jalloy*

Algorithmic Overview

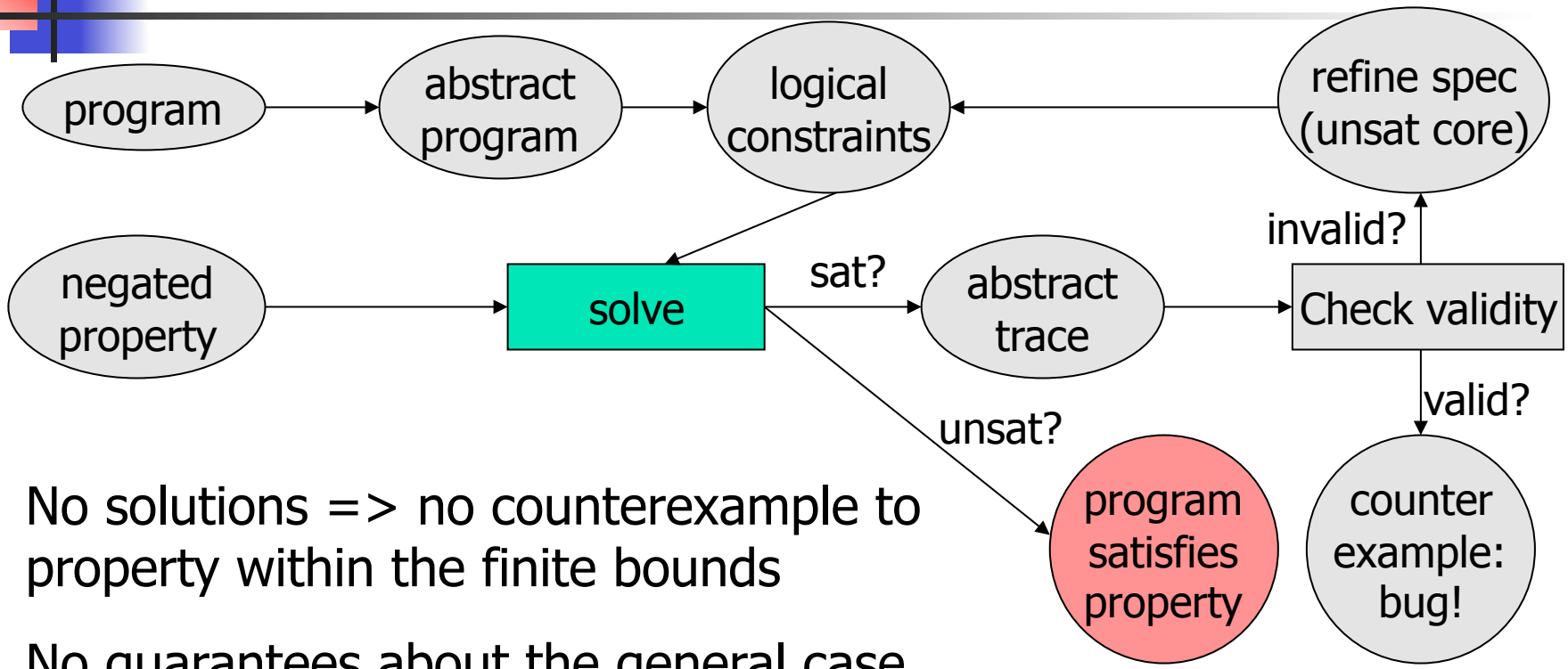


Alloy Analyzer

SAT solver back end *(ZChaff)*

Solutions satisfy current spec but violate property

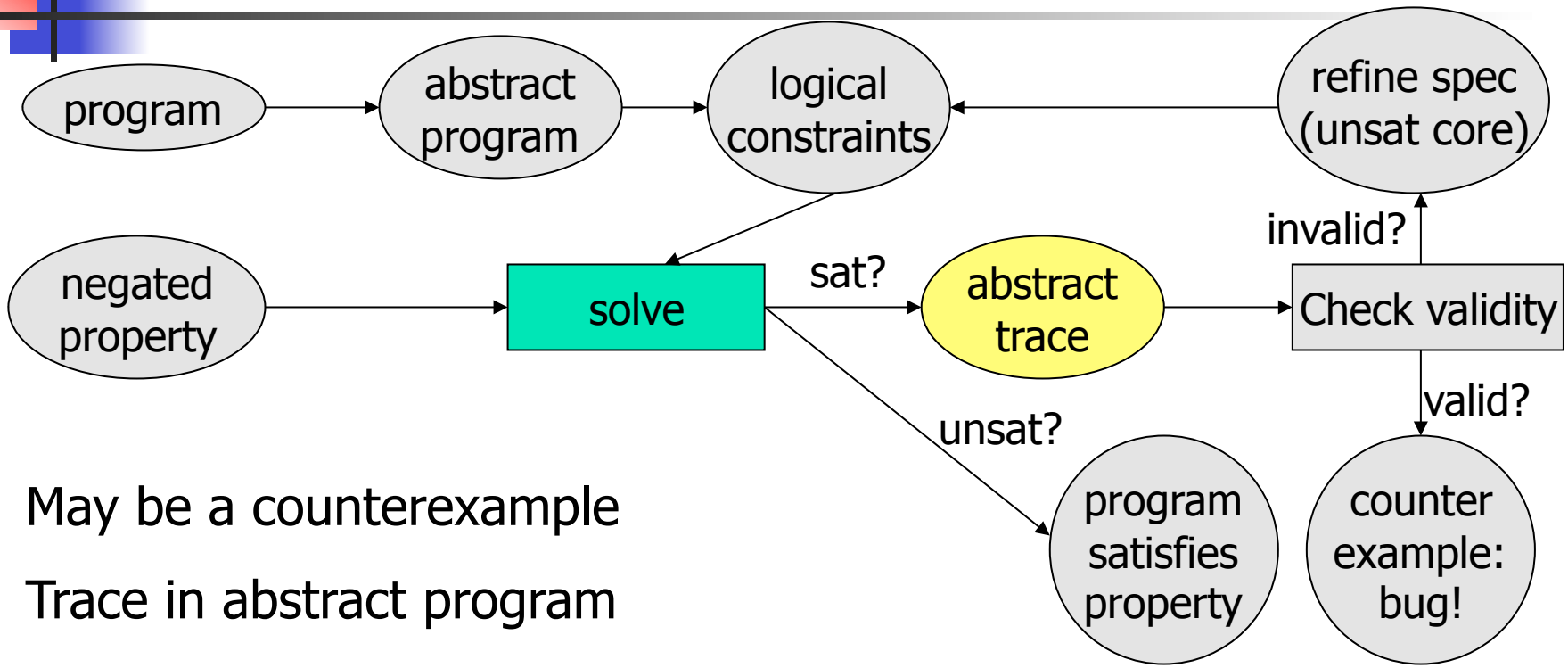
Algorithmic Overview



No solutions => no counterexample to property within the finite bounds

No guarantees about the general case

Algorithmic Overview

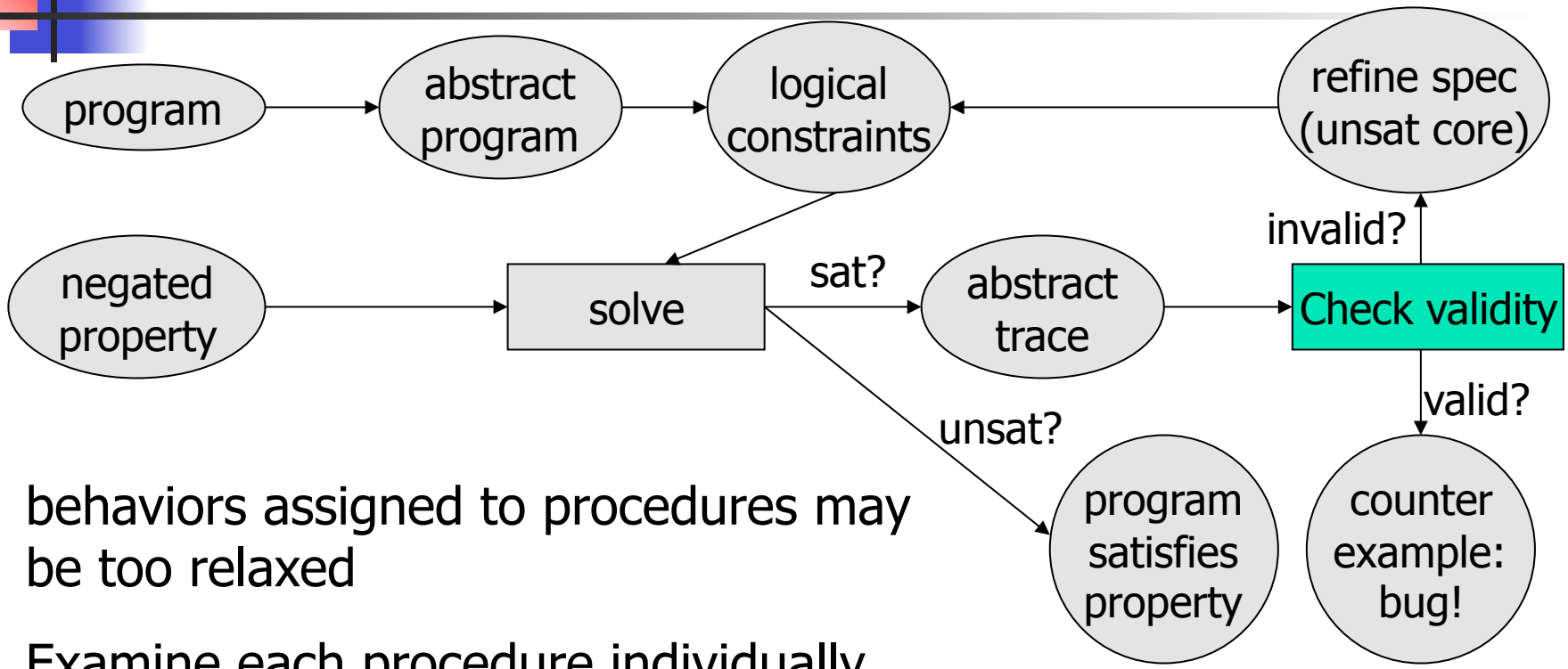


May be a counterexample

Trace in abstract program

Assigns behavior to each procedure: before/after pairs of program states for the procedure call site

Algorithmic Overview

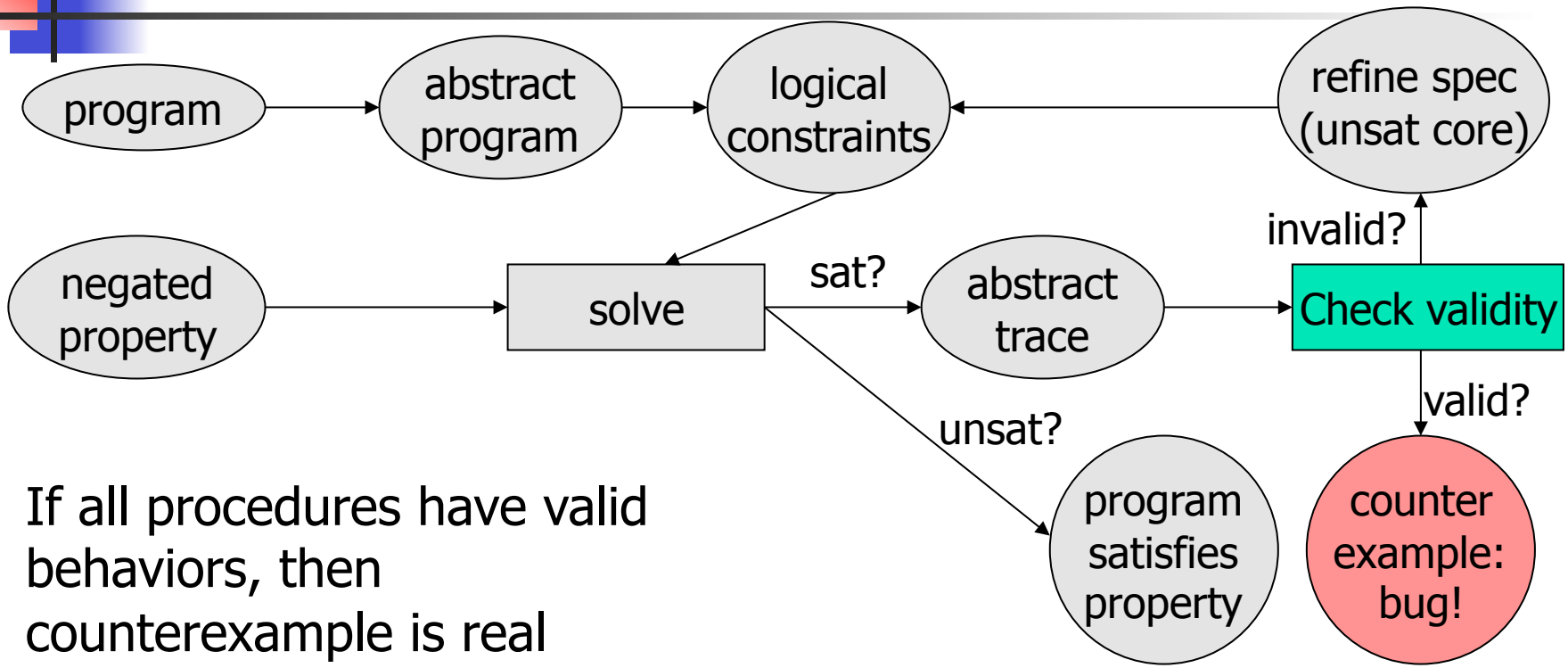


behaviors assigned to procedures may be too relaxed

Examine each procedure individually

Check behavior against full procedure, not just current (partial) spec

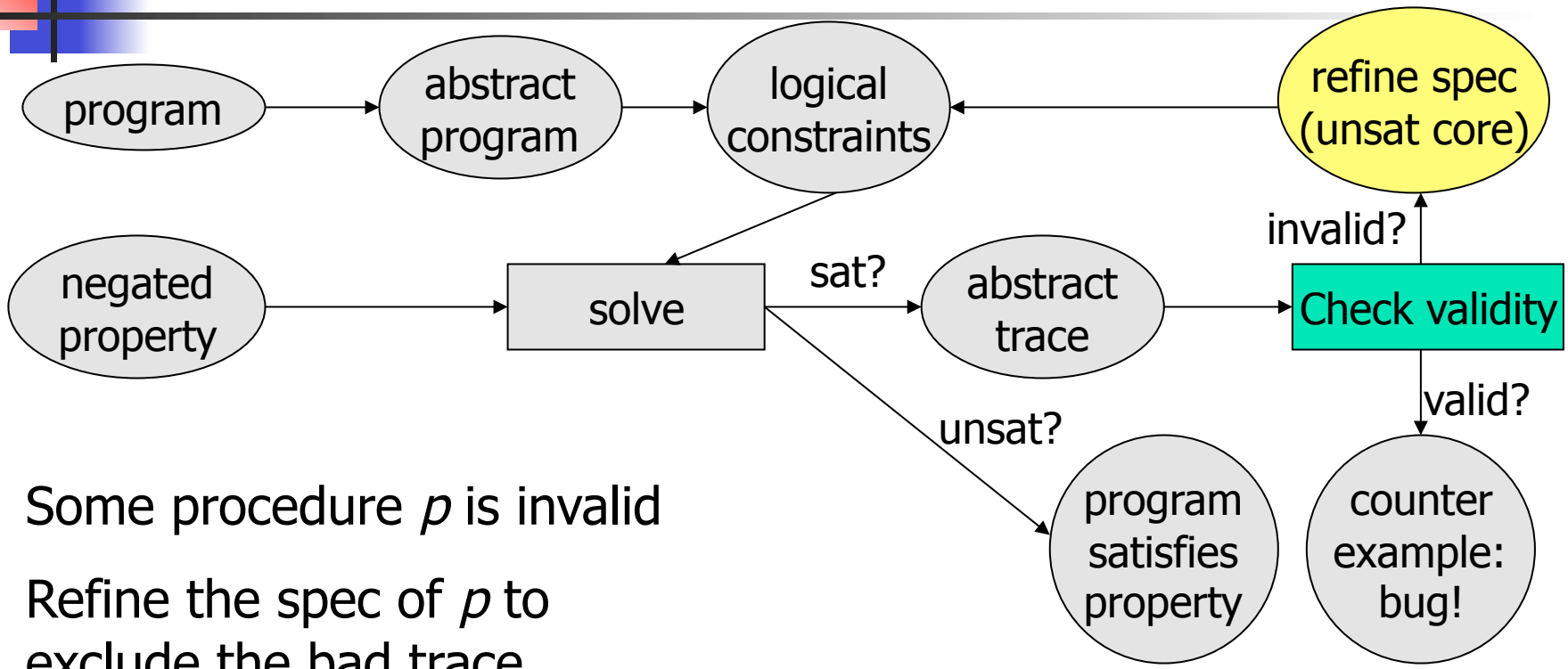
Algorithmic Overview



If all procedures have valid behaviors, then counterexample is real

Bug in original code!

Algorithmic Overview

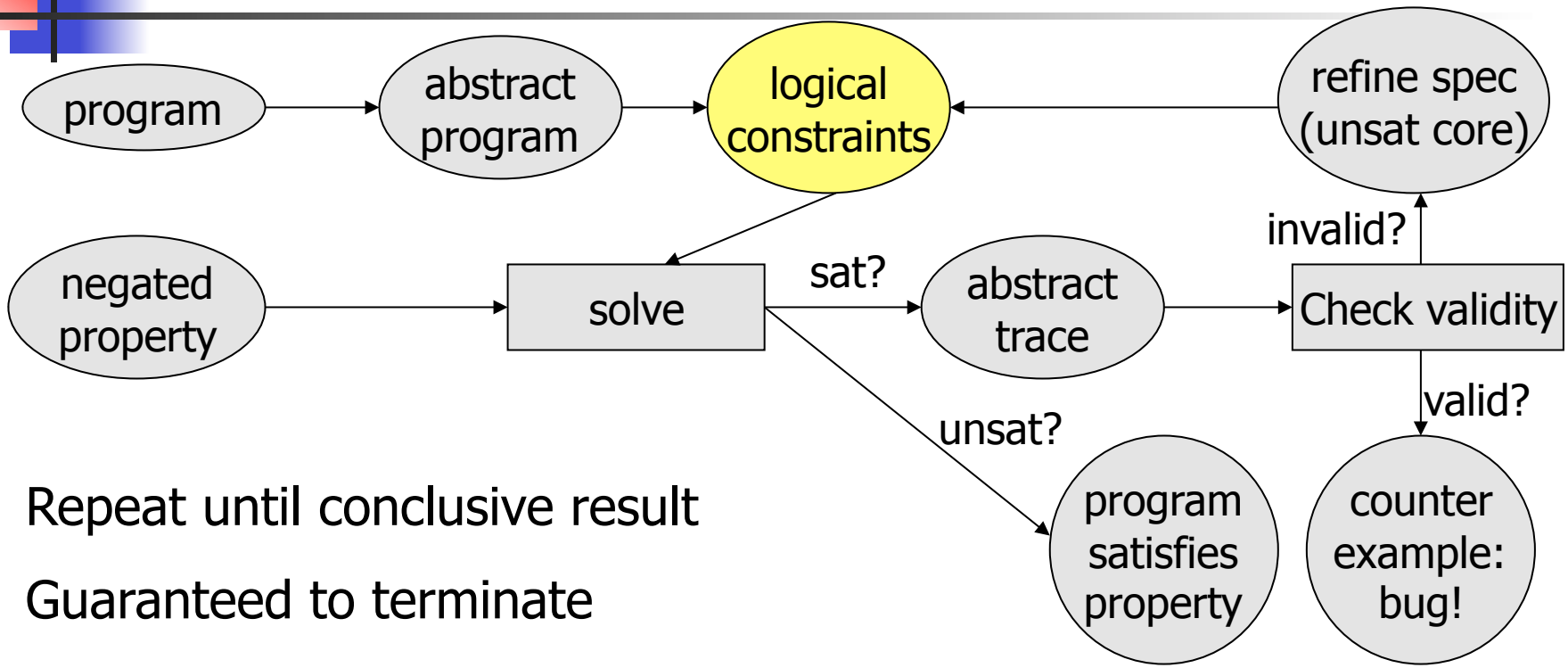


Some procedure *p* is invalid

Refine the spec of *p* to exclude the bad trace

Use unsat core to choose which constraints to add

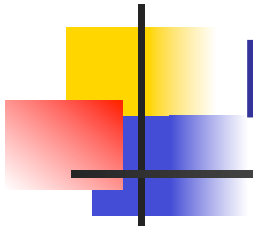
Algorithmic Overview



Repeat until conclusive result

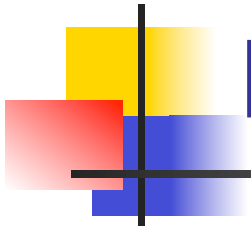
Guaranteed to terminate

At worst, each spec is just the entire procedure



Experiments

- Preliminary results
- Graph manipulation code
  - Structural properties
- All properties hold in the code
- shows benefits of procedure abstraction
  - Comparison with Jalloy
  - Same translation technique
  - Same SAT solver
  - Inlines all procedure calls



Experimental Results

List.removeAll :

- The result list is a subset of the original list
- No refinement iterations needed!

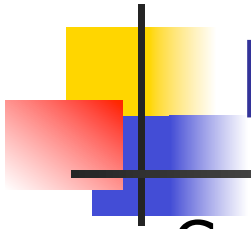
Loop Iter	Heap Size	Jalloy			Our Method			Improvement
		#Var	#Clause	Time	#Var	#Clause	Time	
4	4	8216	18126	15	4928	10260	9	1.67
5	5	14555	34704	162	8611	19002	98	1.65
6	4	13554	30555	40	6702	14013	12	3.33
6	5	18137	43760	234	9857	21776	83	2.82

Experimental Results

Graph.remove :

- If given node set is empty, graph edges aren't changed
- No refinement iterations needed!

Loop Iter	Heap Size	Jalloy			Our Method			Improvement
		#Var	#Clause	Time	#Var	#Clause	Time	
3	3	27112	56241	61	3284	6589	5	12.2
4	4	66566	151323	164	6187	13507	8	20.5
4	5	87710	214959	206	9524	23383	27	7.63
5	4	--	--	>900	6807	14794	8	>112
5	5	--	--	>900	10346	25263	36	>25
6	4	--	--	>900	7499	16207	9	>100

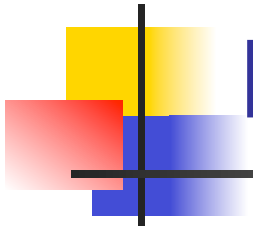


Experimental Results

Graph.remove :

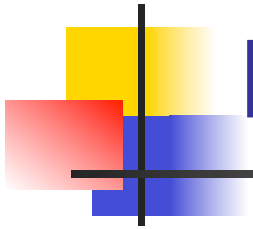
- If given node set is empty, graph nodes aren't changed
- Needs 3 refinement iterations!

Loop Iter	Heap Size	Jalloy			Our Method			Improvement
		#Var	#Clause	Time	#Var	#Clause	Time	
3	3	27147	56298	44	5927	11652	7	6.29
4	4	66661	151489	123	11057	23450	13	9.46
4	5	87803	215129	224	15682	36890	107	2.09
5	4	108016	246914	359	13075	27446	17	20.9
5	5	141087	347466	586	18549	42948	191	3.07



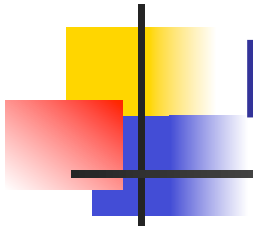
Related Work

- Jalloy
  - Structural properties
  - SAT-based analysis
  - Inlines procedure calls – not scalable
- ESC/Java
  - Based on theorem prover
  - Needs user-provided procedure specification



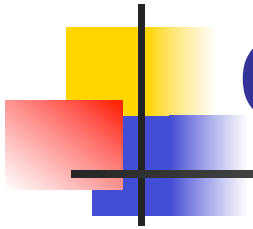
Related Work

- Flanagan's Method
  - Extension to ESC
  - Translates code to CLP
  - Check satisfiability VeriFun – iteratively refined predicate abstraction
- SLAM
  - Predicate abstraction
  - Not structural properties



Related Work

- Bandera
  - Static slicing based on property
  - User-provided data abstraction
  - Intermediate model may be analyzed by a set of model checkers
- Daikon
  - Specification extraction tool
  - Not dependent on context or property
  - Dynamic
  - Unsound (sound over test cases)



Conclusions

- Checks structural properties of the heap
- Exploits modularity from function calls
- Infers context-dependent specs automatically
  - Uses counterexamples to refine specs
  - Uses unsat core to produce small specs