Improving Static Analysis Precision by Minimal Program Refinement

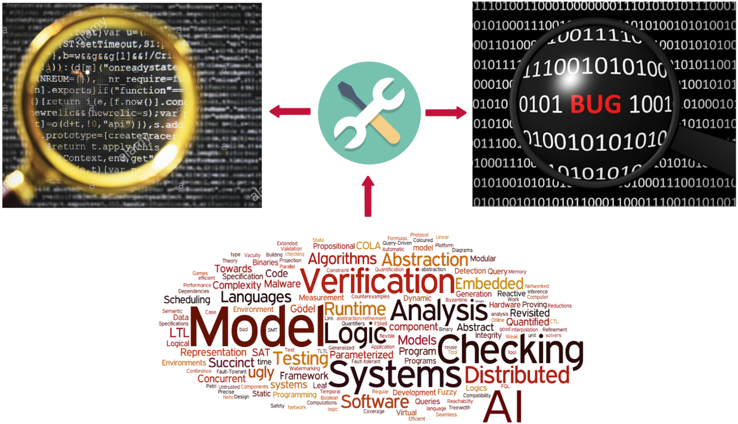
Charles Babu M[1] Sebastien Bardin[1] Matthieu Lemerre[1]
Jean-Yves Marion[2]

CEA, List, Université Paris-Saclay

Université de Lorraine, CNRS, LORIA

(Work in Progress)

Context

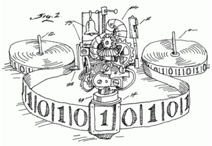


0101 BUG 1001
Model
Verification
Checking
Systems
Analysis
Logic
Runtime
Abstraction
Algorithms
Distributed
AI
Software
Testing
Embedded
Languages
Scheduling
Complexity
Malware
Towards
Code
Specification
Binaries
Propositional
COLA
Modular
Detection
Query
Reactive
Generation
Hardware
Proving
Revisited
Quantified
Abstract
Integrity
component
Models
Program
Programs
Parameterized
Representation
SAT
Succinct
time
Concurrent
ugly
Framework
Leaf
systems
Static
Programming
Development
Fuzzy
Queries
Virtual
Logics
LTL
Logical
Data
Environments
Environment
Measurement
Dynamic
Gödel

(Sound) Static Analysis

```
int i=i0;
while (i<100)
    j=0;
    while(j<i)
        j=j+1
    i=i+1;
assert(i==100);
```

⟹

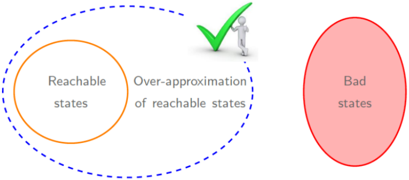


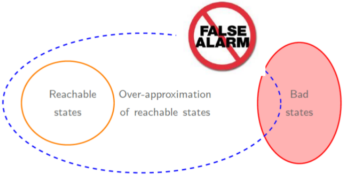
⟹

```
{i0>=0}
int i=i0;
{i=i0, i0>=0}
while (i<100)
    {i=i0, i<100, i0>=0}
    j=0;
    {i=i0, i<100, j=0, i0>=0}
    while(j<i)
        {i=i0, i<100, j<i, i0>=0}
        j=j+1
        {i=i0, i<100, j<=i, i0>=0}
    {i=i0, i<100, j=i, i0>=0}
    i=i+1;
    {i+1=i0, i<=100, j=i+1, i0>=0}

{i=100, i0>=0}
assert(i==100);
{i=100, i0>=0}
```

(Sound) Static Analysis

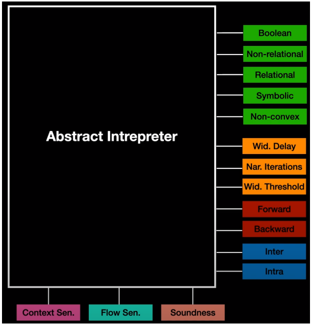




Imprecision is a common phenomenon which raises false alarms

Goal: To improve precision of a static analysis

Problem: Improving Precision is tricky



- Many parameters to tune
- Creating new domains requires lots of efforts and expertise

Hence the need for **automatic** precision refinement

Automatic Refinement

Goal: To design an automatic refinement framework for static analysis which is:

1. Generic: Independent from the abstract domain
2. Principled: General strategy to control disjunctions

Automatic Refinement

Goal: To design an automatic refinement framework for static analysis which is:

1. Generic: Independent from the abstract domain
2. Principled: General strategy to control disjunctions

		Generic	Principled
Model-Checking		✗ Specialized E.g. CEGAR	✓
Static Analysis	Trace Partitioning	✓	✗ Heuristics
	Others	✗ Specialized X Rival. in Shape analysis, etc	✓

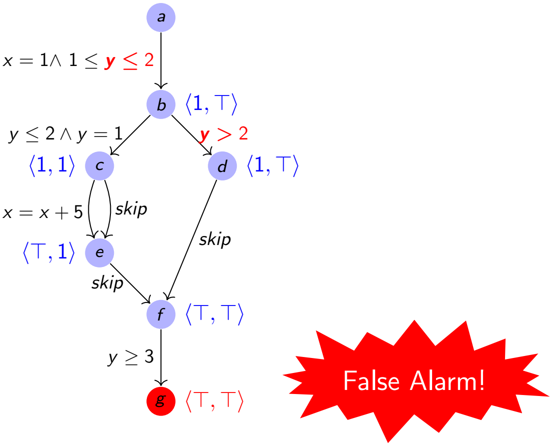
Contribution (Work in Progress)

Contributions

- A new automatic refinement framework
  - Refinements are CFG-based
  - Independent of any abstract domain ($\sqcup, \sqcap, \mathtt{post}, [\![\cdot]\!]^*$)
- Identify desired properties of the framework
  - Correctness, Completeness, Minimality
  - Interface contracts
-
  - Optimal instance with the above all 3 desired properties
  - Optimizations
    - ✓ E.g state-space Exploration, Incremental computation
- Handle non-linear reasoning (widening)

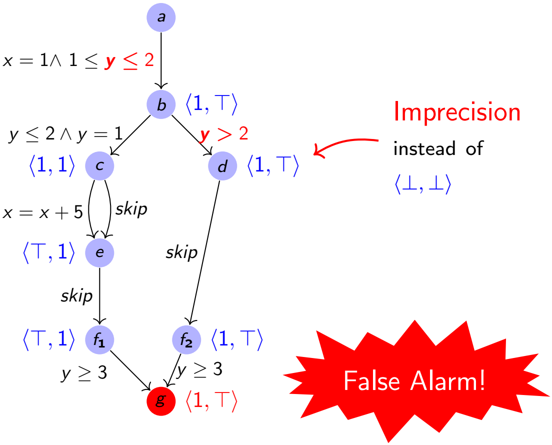
Program Refinement

Constant propagation analysis with abstract states $\{\langle x, y\rangle\}$



Imprecision at g. Can we remove it?

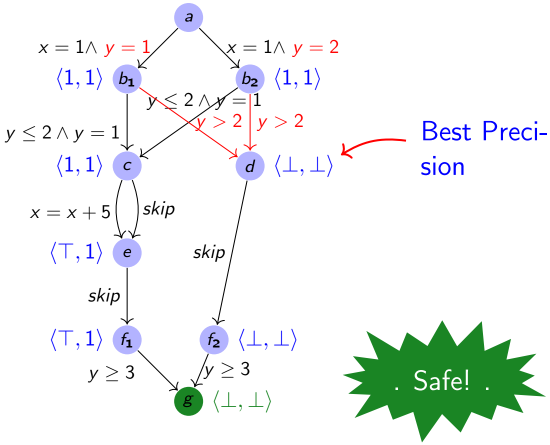
Constant propagation analysis with abstract states $\{\langle x, y \rangle\}$



Split at f reduces imprecision at g. Can we reduce more?

Program Refinement

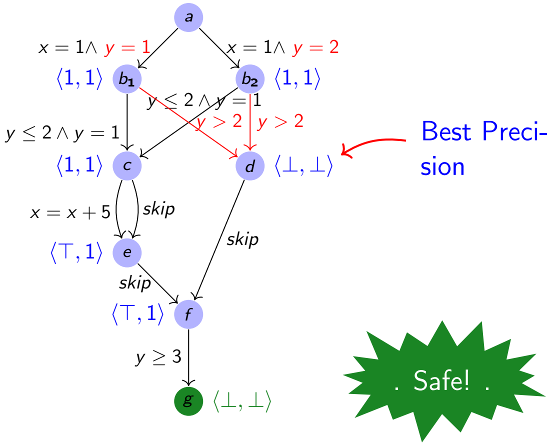
Constant propagation analysis with abstract states $\{\langle x, y\rangle\}$



Can some of the splits be merged?

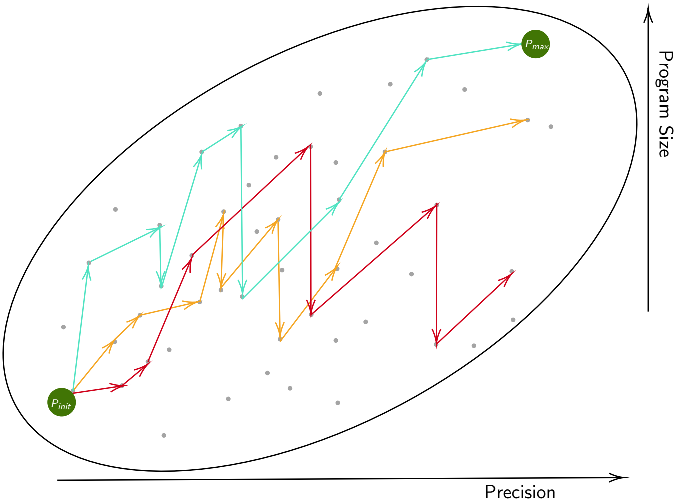
Program Refinement

Constant propagation analysis with abstract states $\{\langle x, y\rangle\}$

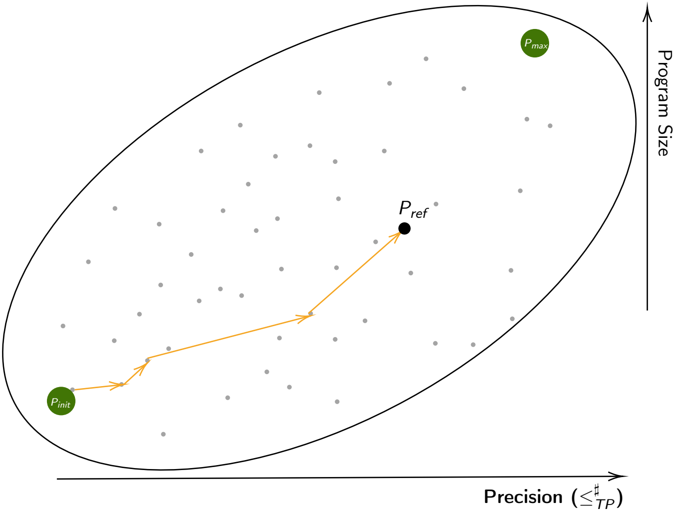


Yes, and there is no loss of precision

Generic Refinement Techniques

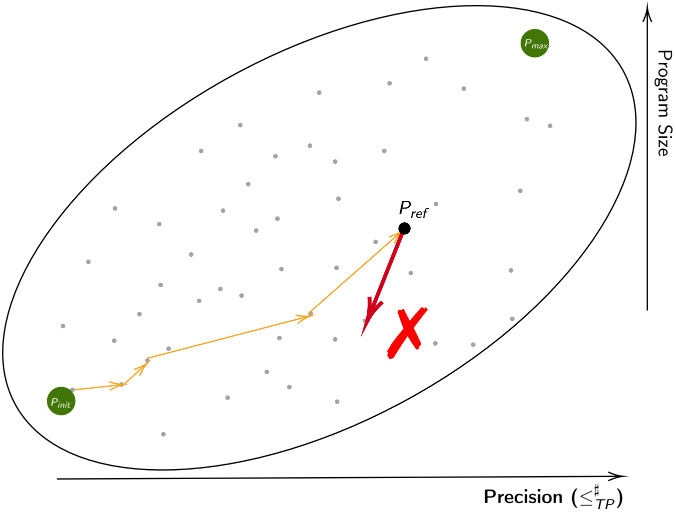


Problem. Decreasing size leads to loss of precision.



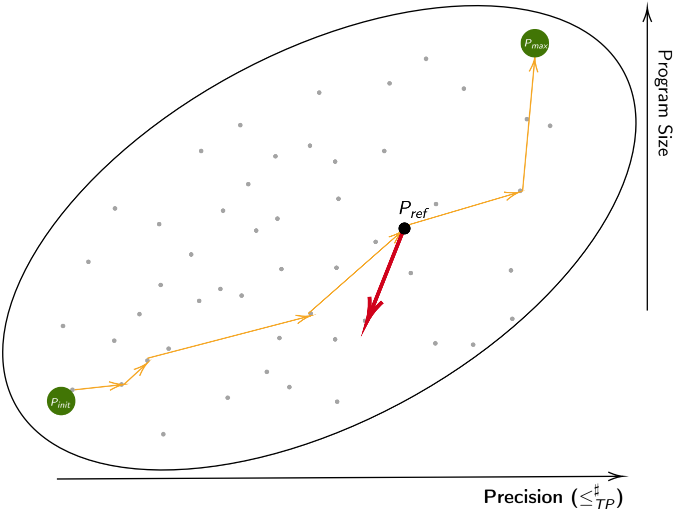
Problem of Standard Trace Partitioning Framework 1

Problem. Decreasing size leads to loss of precision.



Problem of Standard Trace Partitioning Framework 1

Problem. Decreasing size leads to loss of precision.



Problem. Decreasing size leads to loss of precision.

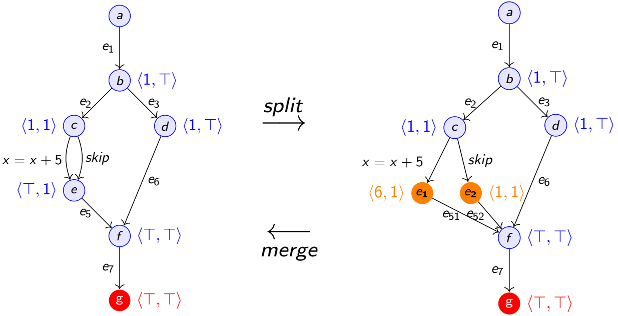


Figure: P_1 Figure: P_2

Problem of Standard Trace Partitioning Framework 2

Problem. Decreasing size leads to loss of precision.

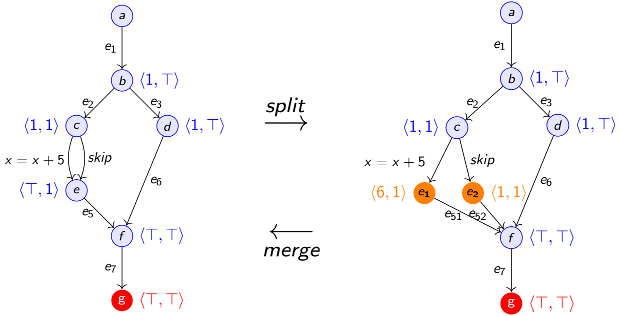


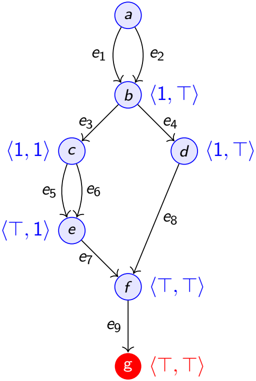
Figure: P_1

Figure: P_2

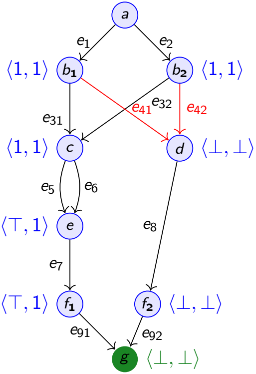
Solution. Choose *locations of interest* $\mathcal{L}$.

✓ E.g. For $\mathcal{L} = \{g\}$, merge does not lose precision at g

Refinements as $n-$tuples

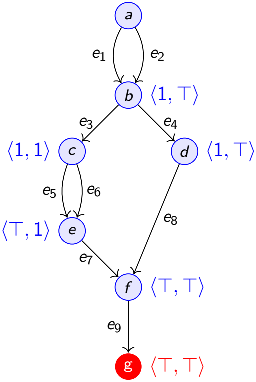


$$t = [(e_7, e_8, 1), (e_1, e_2, 1)]$$

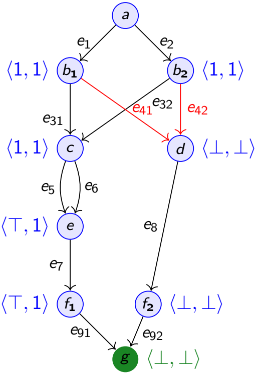


- $t \in \mathbb{N}^n$
- Partial Orderings on n-tuples:
  1. Component-wise ordering $\leq$
  2. Precision ordering $\leq^{\sharp}_{\mathcal{L}}$

Refinements as $n-$tuples

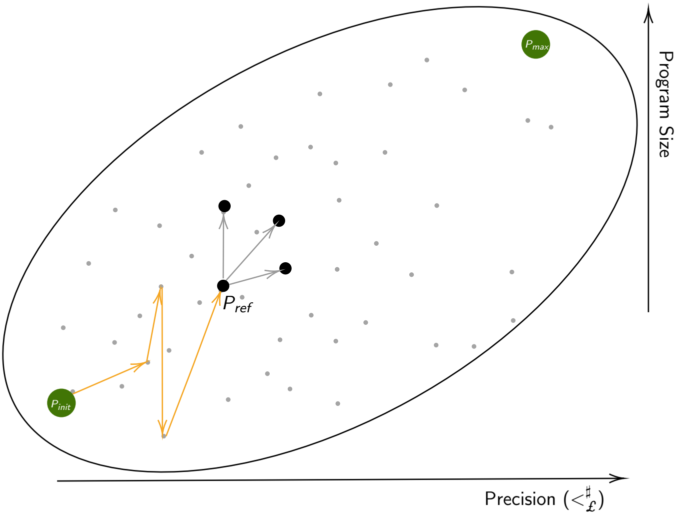


$t = (1, 1)$

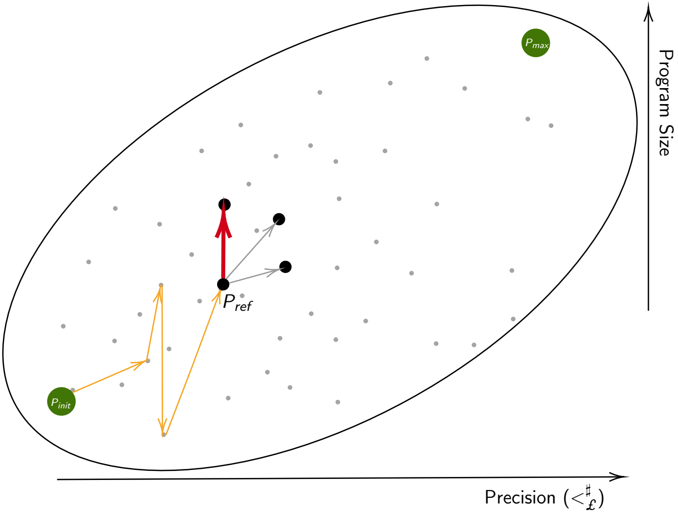


- $t \in \mathbb{N}^n$
- Partial Orderings on n-tuples:
    1. Component-wise ordering $\leq$
    2. Precision ordering $\leq^{\sharp}_{\mathcal{L}}$

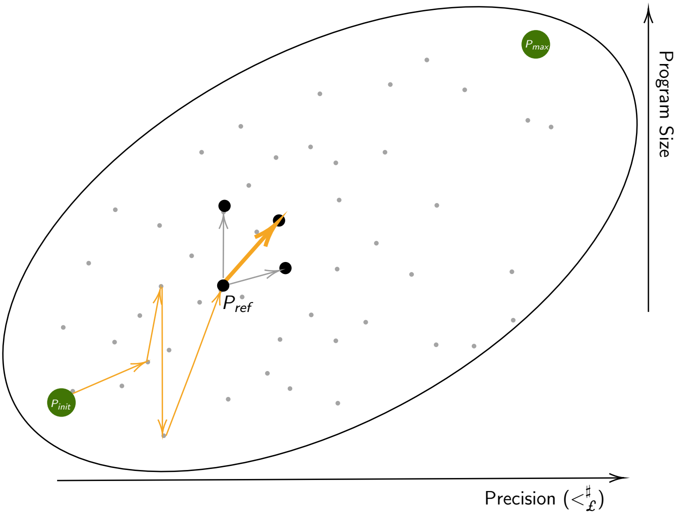
Generic Refinement Framework: **next** method



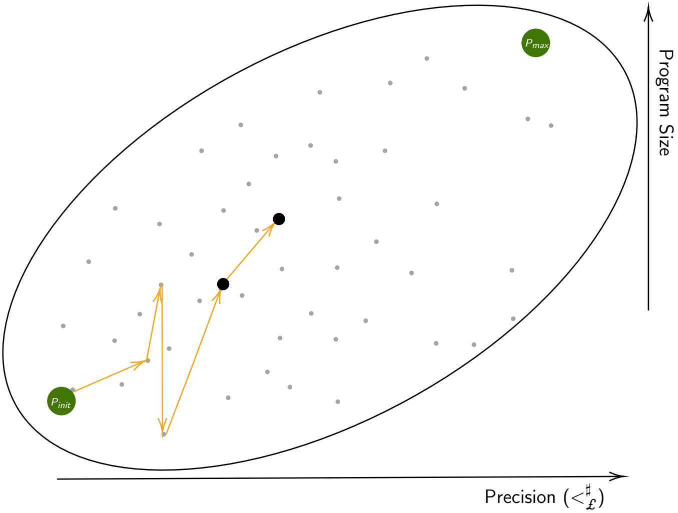
Generic Refinement Framework: **next** method



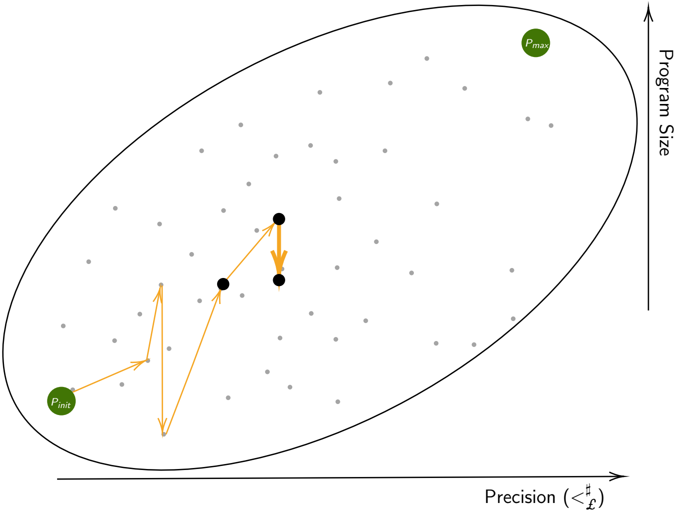
Generic Refinement Framework: **next** method



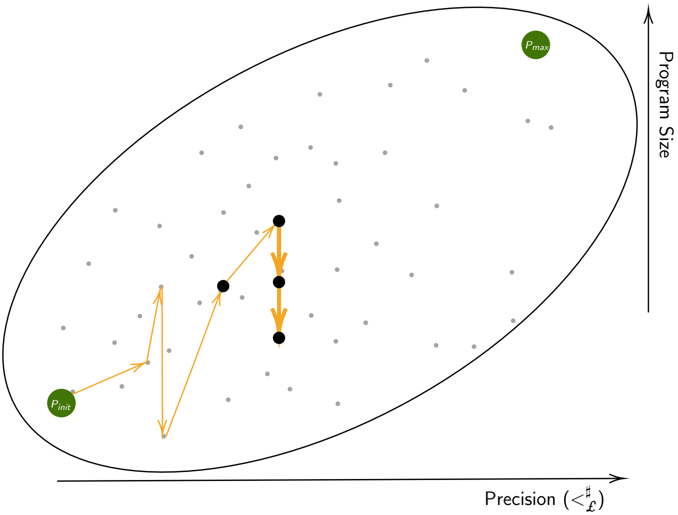
Generic Refinement Framework: **collapseSplits** method



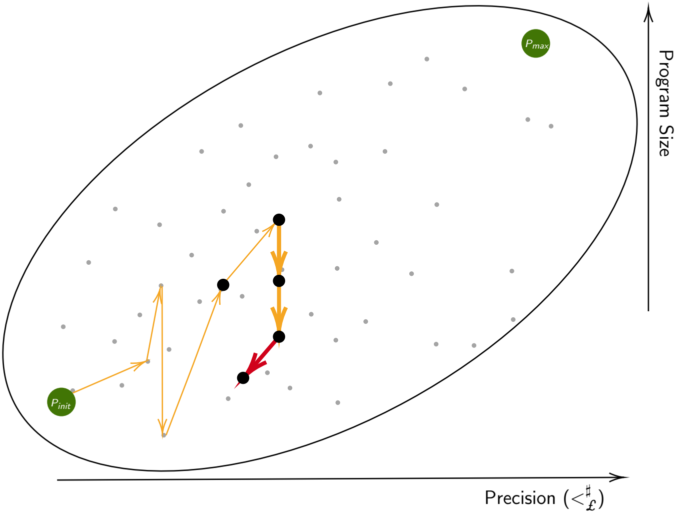
Generic Refinement Framework: **collapseSplits** method



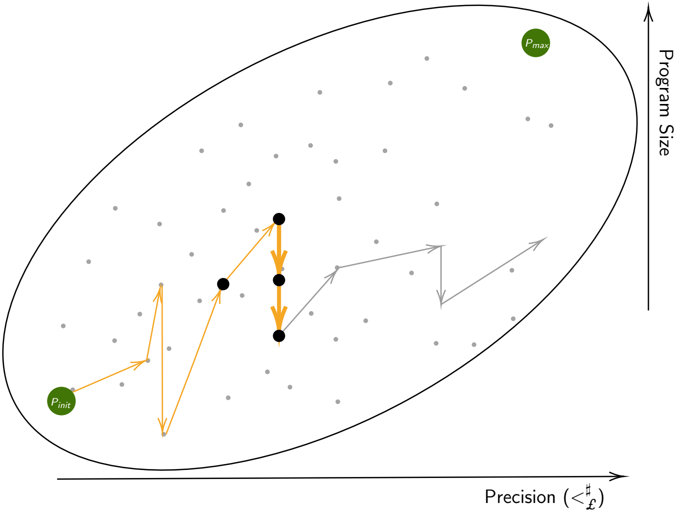
Generic Refinement Framework: **collapseSplits** method



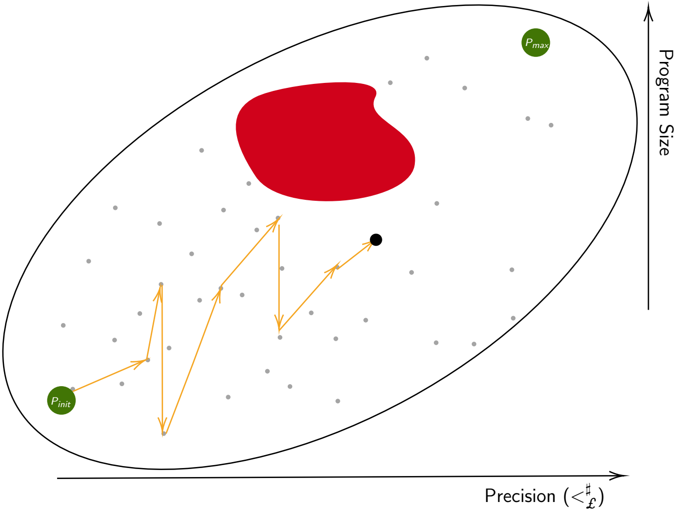
Generic Refinement Framework: **collapseSplits** method



Generic Refinement Framework: **collapseSplits** method

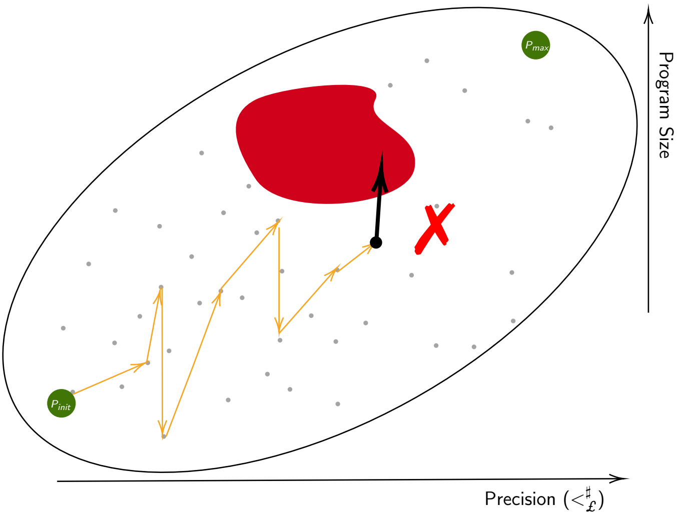


Generic Refinement Framework: Optimizing Exploration



`next` prunes search space with region set `AVOID`

Generic Refinement Framework: Optimizing Exploration



`next` prunes search space with region set `AVOID`

Generic Refinement Framework

Input : Program P, $D^\sharp$, max dist k, loc $\pounds$
Output: Tuple with maximum precision

$t_{prev} = t_{best} = (0, \ldots, 0)$, AVOID $= \emptyset$;
while (**true**) **do**
  t_{cur} := next(t_{prev}, t_{best}, AVOID) ;
  if t_{cur} == **null then** **break;** ;
  else
    if $t_{cur} <^\sharp_{\pounds} t_{best}$ **then**
      t_{best} = collapseSplit(t_{best}, t_{cur});
    else
      AVOID = AVOID $\cup$ newPruneSet(t_{cur}, t_{best}, i, AVOID);
    $t_{prev} = t_{cur}$;
return t_{best};

Variables.

1. Previous tuple t_{prev}
2. Current tuple t_{cur}
3. Best tuple t_{best}
4. Pruning set AVOID

Properties

Definition (Correctness, global improvement)

Termination + Global Improvement

Definition (Completeness)

Finds a maximally precise refinement

Definition (Minimality)

When do we have Algorithm properties?

	Correctness	Completeness	Minimality
`next`	**Monotonicity** (Global Imp)	-	-
`collapseSplits`	Merge-Lossless (Termination)	Merge-Lossless	Merge-Minimal
`newPruneSet`	-	Non-Improving	-

Comparison

	Correctness	Completeness	Minimality
Naive GSR	✓	✗	✗
Trace partitioning	✓	✗	✗
Optimal GSR	✓	✓	✓
Intermediate GSR	✓	✓	✗

Problem with Monotonicity Assumption

Widening is non-monotone:

$$[1,40]\triangledown[1,40]=[1,40] \qquad \overset{\textit{splits}}{\Longrightarrow} \qquad [1,1]\triangledown[1,40]=[1,+\infty]$$

Monotonicity Forcing Trick

Key Idea. Take previous refinement fixpoint map and force monotonicity.

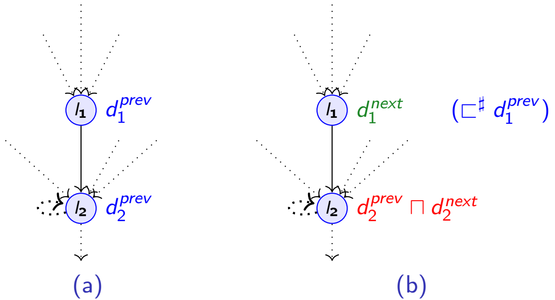


Figure: a) Previous Refinement; b) Forcing monotonicity in the new refinement

Both d_2^{prev} and d_2^{next} are over-approximation of reachable states.
So, take a meet $\sqcap$ of them

Conclusion

Within static analysis we provide a new refinement framework that is automatic, generic and principled.

Thanks for your attention!