Towards Practical Predicate Analysis

Philipp Wendler

2017-11-20

UNIVERSITÄT PASSAU

Fakultät für Informatik und Mathematik

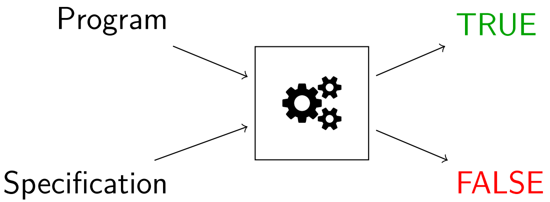
Why Automatic Software Verification?

- Software is critical in today's world
- Software has bugs
- Software is too complex for humans to find all bugs

Why Automatic Software Verification?

- Software is critical in today's world
- Software has bugs
- Software is too complex for humans to find all bugs

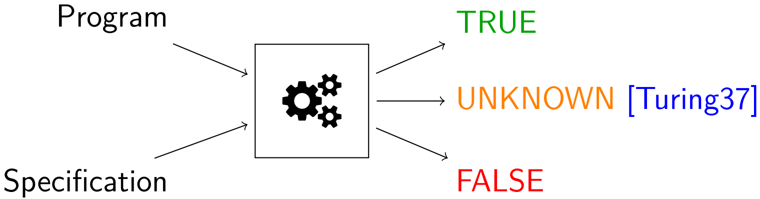
Solution "Model Checking":



Why Automatic Software Verification?

- Software is critical in today's world
- Software has bugs
- Software is too complex for humans to find all bugs

Solution "Model Checking":



Predicate Analysis

Many approaches share common idea:

- Convert parts of program into formulas of first-order logic (satisfiability modulo theories, SMT)
  Example: $x > 0 \wedge y < 10$
- Query solver about satisfiability

We call SMT-based approaches *predicate analyses*.

- Leverage power of modern solvers for model checking
- Used in practice for verification of software

Existing Predicate Analyses

- Predicate Abstraction
- Impact
- Bounded Model Checking
- *k*-Induction
- Property-Directed Reachability
- ...

Existing Predicate Analyses and Tools

- Predicate Abstraction
  (BLAST, SLAM, ...)
- IMPACT
  (IMPACT, WOLVERINE, ...)
- Bounded Model Checking
  (CBMC, ESBMC, ...)
- *k*-Induction
  (ESBMC, 2LS, ...)
- Property-Directed Reachability
  (SEAHORN, VVT, ...)
- ...

Problems with State of the Art

Approaches exist in isolation

- Hard to see differences and commonalities
- Hard to understand key concepts and advantages
- Slows down research

Problems with State of the Art

Approaches exist in isolation

- Hard to see differences and commonalities
- Hard to understand key concepts and advantages
- Slows down research

Approaches implemented in different tools

- ...of varying quality (academic prototypes)
- Experimental comparison of approaches difficult
- Slows down research and adoption in practice

Question at a Recent Workshop

Which do you think is better, i.e., solves more tasks?

k-Induction	Predicate Abstraction

Question at a Recent Workshop

Which do you think is better, i.e., solves more tasks?

(A)	(B)
k-Induction solves 29 % more tasks	Predicate Abstraction solves 3 % more tasks

Question at a Recent Workshop

Which do you think is better, i.e., solves more tasks?

(A)	(B)
k-Induction solves 29 % more tasks	Predicate Abstraction solves 3 % more tasks

Depending on configuration, either (A) or (B) can be true!

Technical details (e.g., choice of SMT theory) influence evaluation of algorithms

Goals

1. Provide a unifying framework for predicate analyses
2. Understand differences and key concepts of approaches
3. Determine potential of extensions and combinations
4. Provide solid platform for experimental research

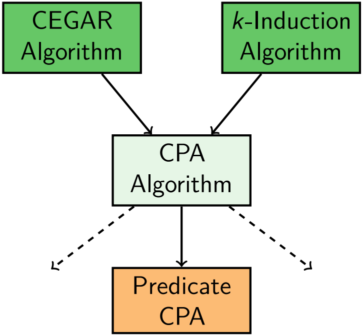
Approach

- Understand, and, if necessary, re-formulate algorithms
- Design a configurable framework for predicate analyses (as configurable program analysis)
- Express existing algorithms using the common framework
  - Predicate abstraction
  - IMPACT
  - Bounded model checking
  - *k*-Induction
- Implement framework (in CPACHECKER)

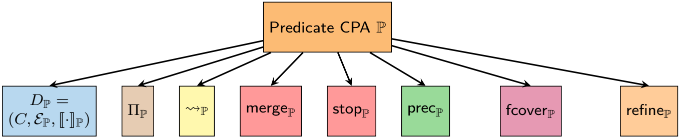
Architecture

Using CPA framework:

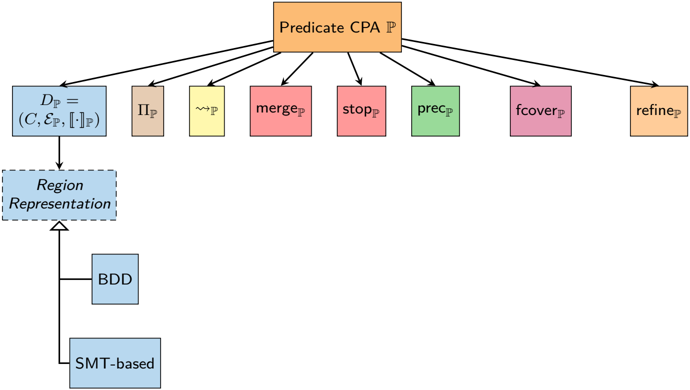
- Reuse existing CPA algorithm for state-space exploration
- CPA algorithm uses operators provided by CPA(s)
- Predicate CPA provides predicate-based operators
- Algorithms for additional features implemented on top of CPA algorithm



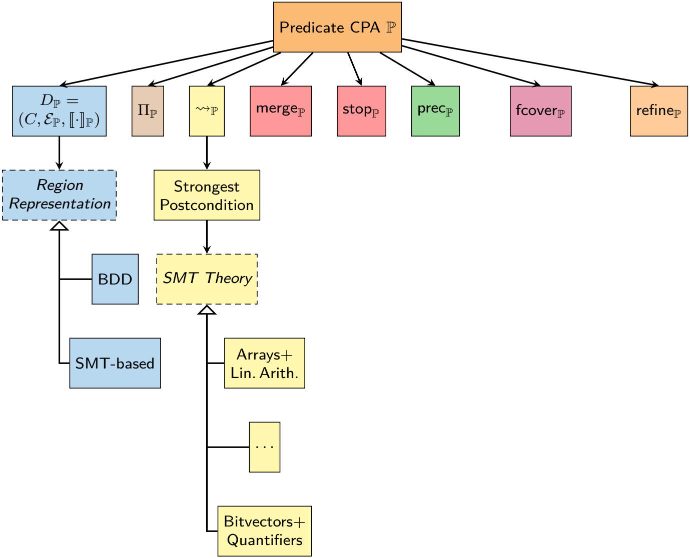
Predicate CPA



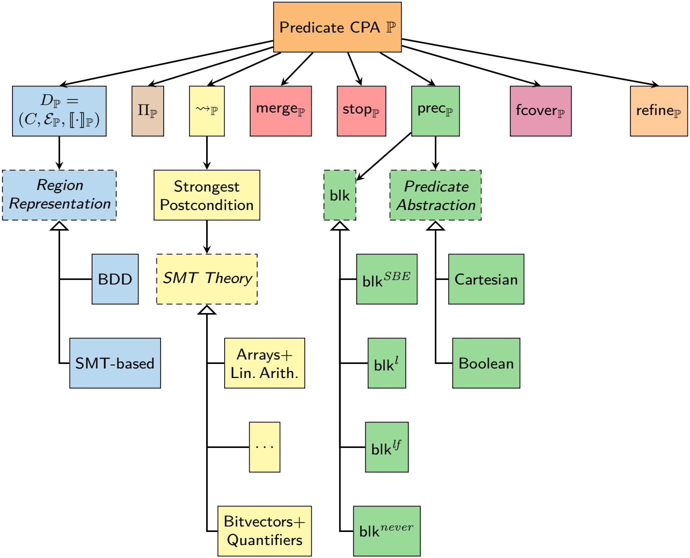
Predicate CPA



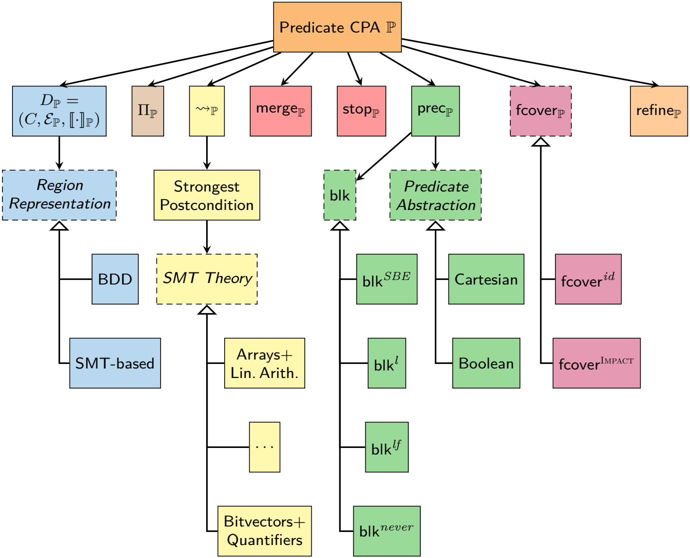
Predicate CPA



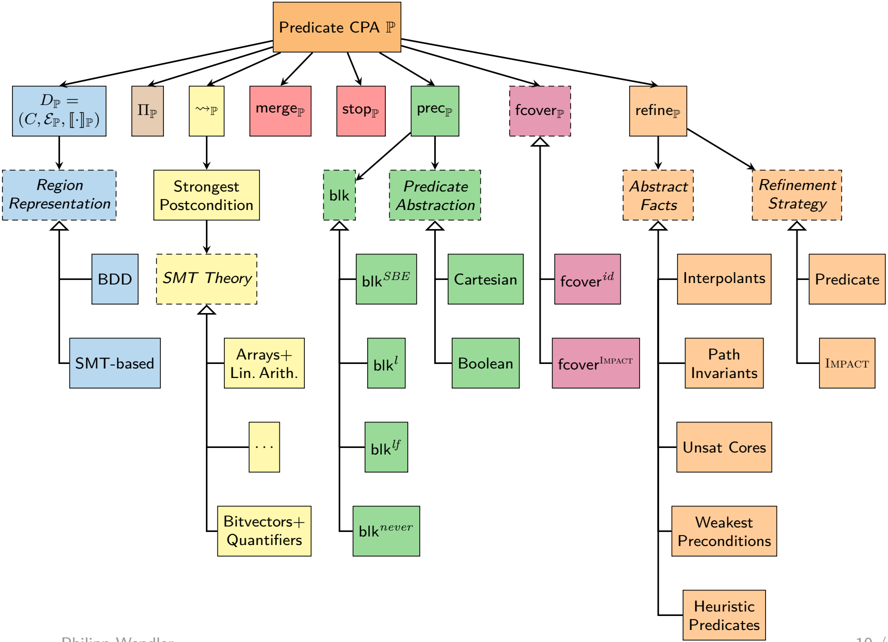
Predicate CPA



Predicate CPA



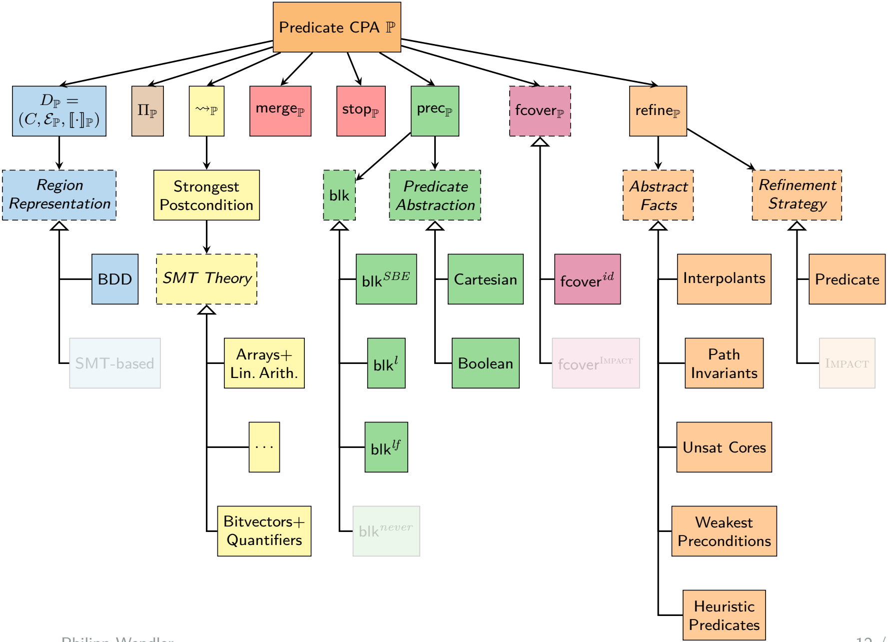
Predicate CPA



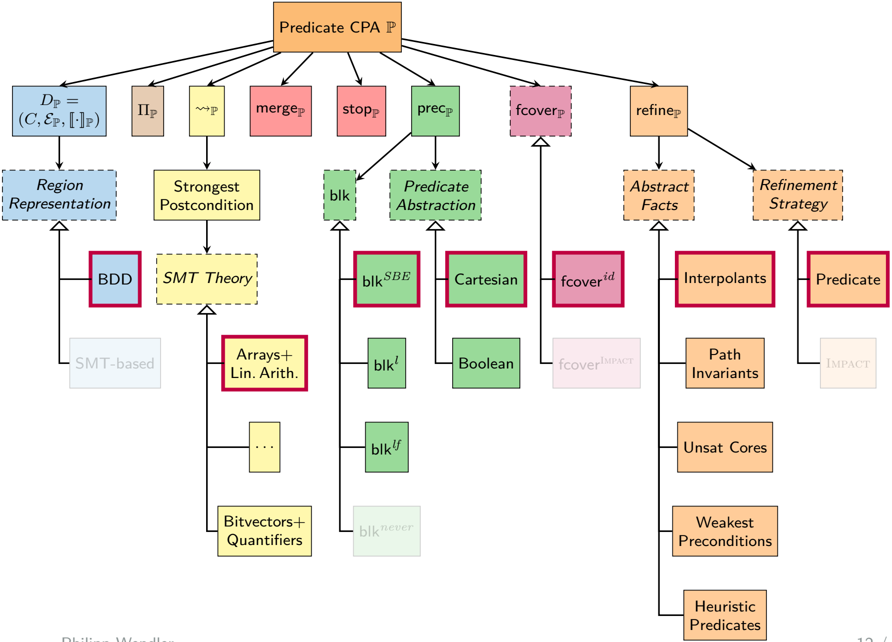
Predicate Abstraction

- [CAV'97, POPL'02, J. ACM'03, POPL'04]
- Abstract-interpretation technique
- Abstract domain constructed from a set of predicates π
- Use CEGAR to add predicates to π (refinement)
- Derive new predicates using Craig interpolation

Predicate CPA for Predicate Abstraction

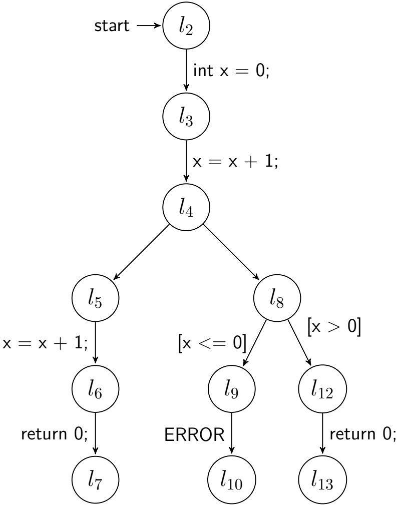


Predicate CPA for Predicate Abstraction



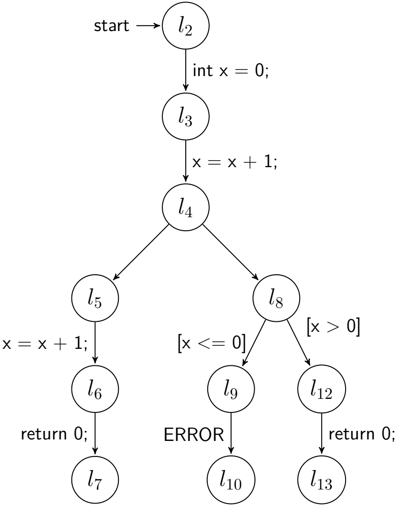
Example Program

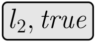
```
int main() {
  int x = 0;
  x = x + 1;
  if (*) {
    x = x + 1;
    return 0;
  } else {
    if (x <= 0) {
      ERROR:
      return 1;
    }
    return 0;
  }
}
```



Predicate Abstraction: Example

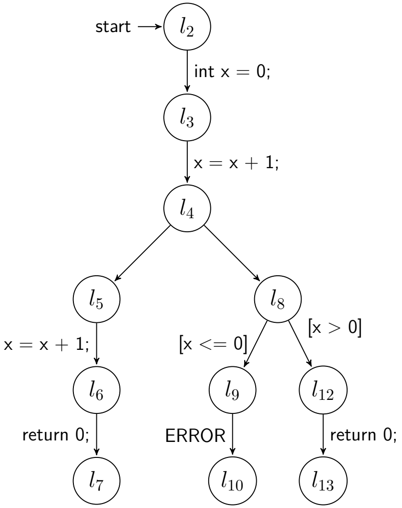
with initial precision: $\pi = \{\}$

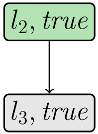




Predicate Abstraction: Example

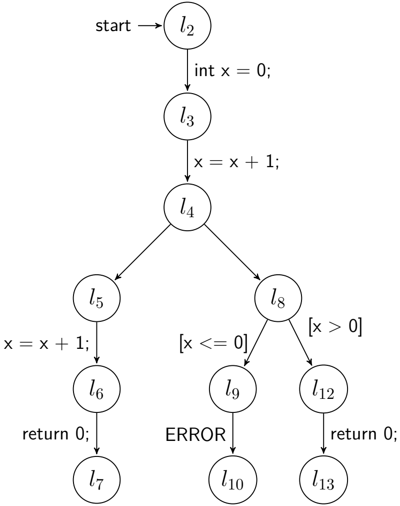
with initial precision: $\pi = \{\}$

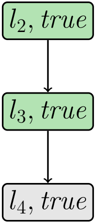




Predicate Abstraction: Example

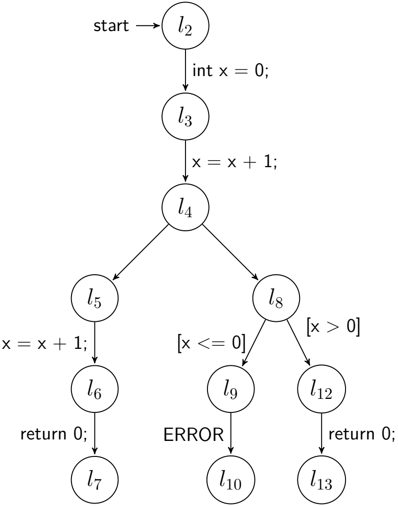
with initial precision: $\pi = \{\}$

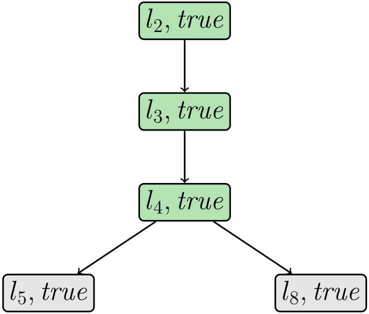




Predicate Abstraction: Example

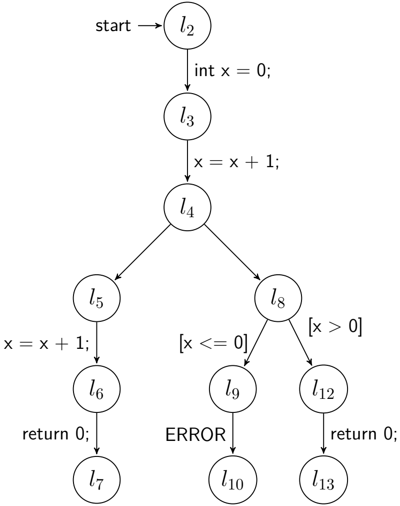
with initial precision: $\pi = \{\}$

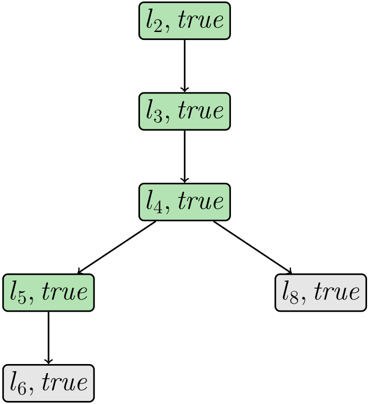




Predicate Abstraction: Example

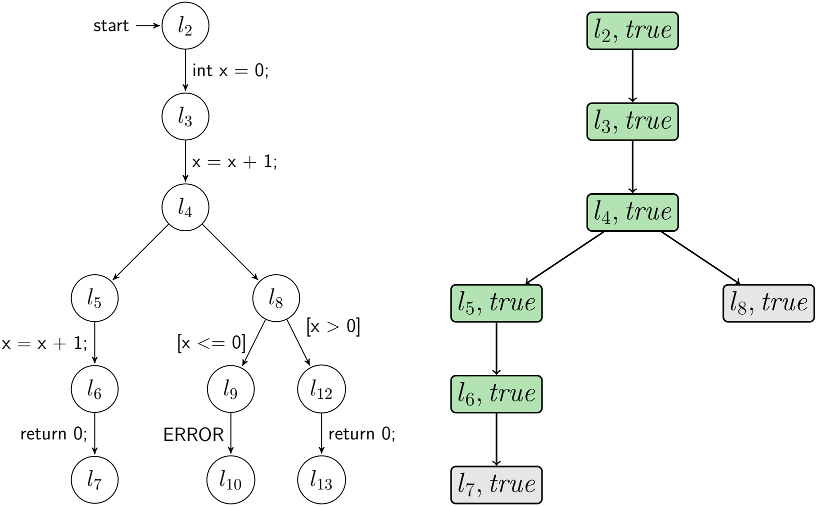
with initial precision: $\pi = \{\}$





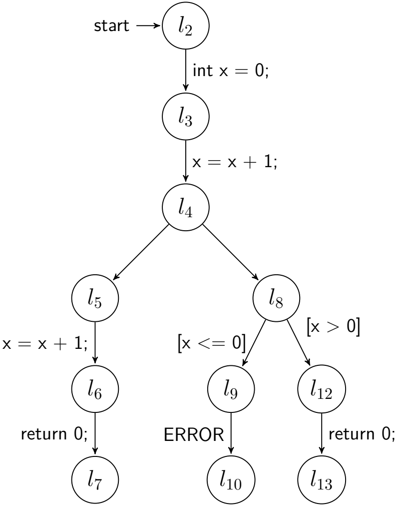
Predicate Abstraction: Example

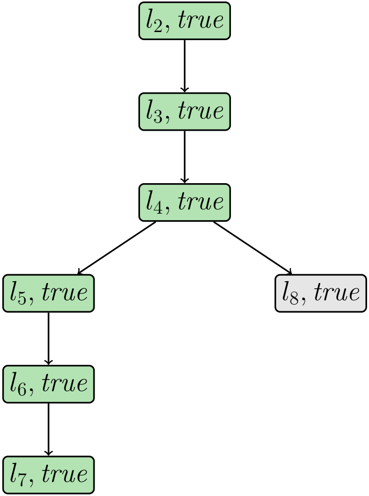
with initial precision: $\pi = \{\}$



Predicate Abstraction: Example

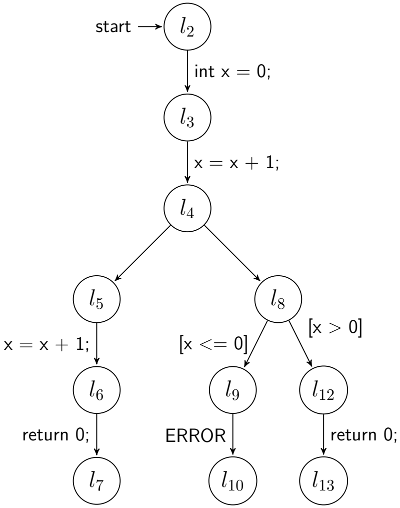
with initial precision: $\pi = \{\}$

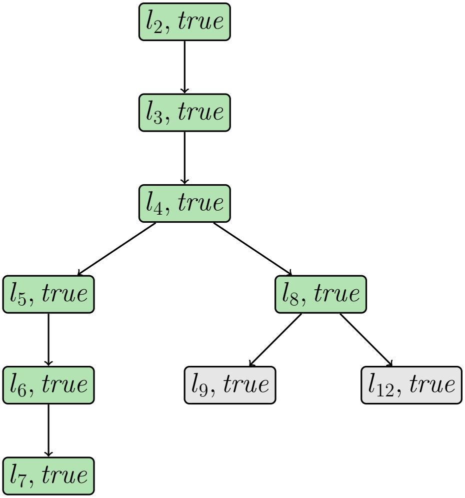




Predicate Abstraction: Example

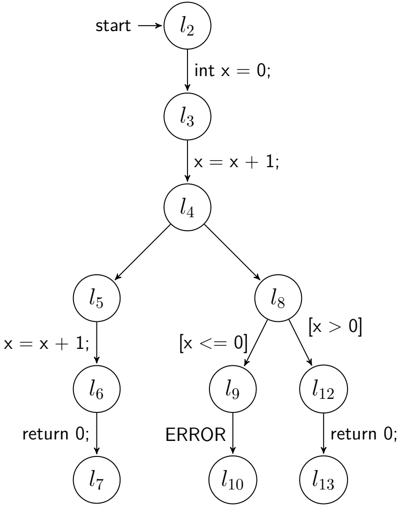
with initial precision: $\pi = \{\}$

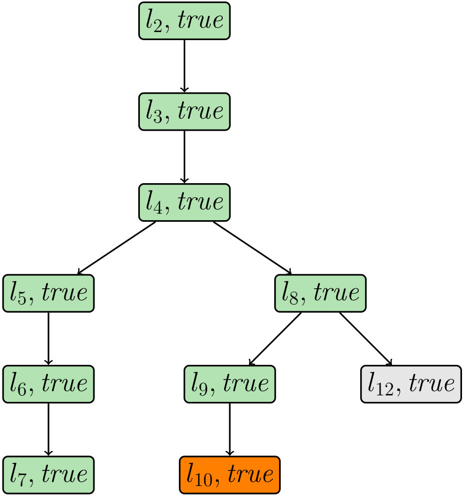




Predicate Abstraction: Example

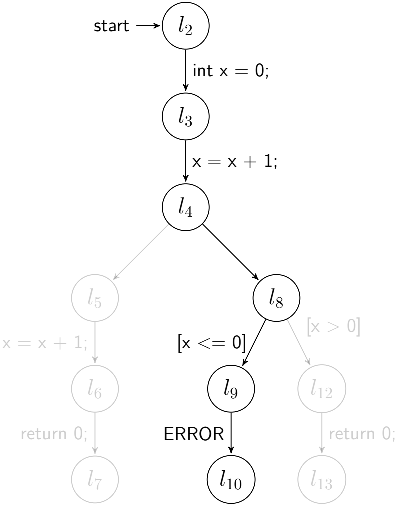
with initial precision: $\pi = \{\}$





Predicate Abstraction: Example

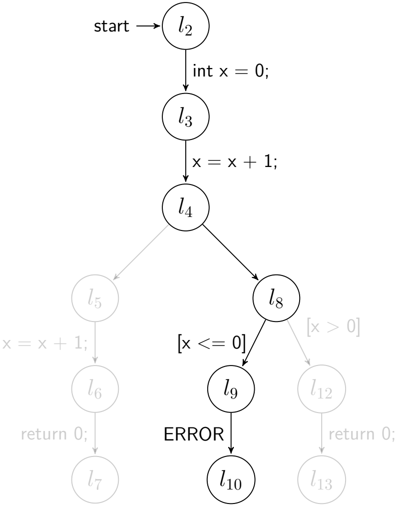
Refinement



1. Convert path $< l_2, \ldots, l_{10} >$ into formula:
   $x_1 = 0 \land x_2 = x_1 + 1 \land x_2 \leq 0$

Predicate Abstraction: Example

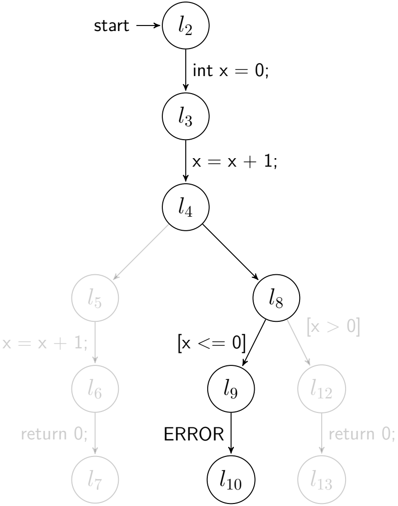
Refinement



1. Convert path $< l_2, \ldots, l_{10} >$ into formula: $x_1 = 0 \land x_2 = x_1 + 1 \land x_2 \leq 0$
2. Check for satisfiability: unsat!

Predicate Abstraction: Example

Refinement



1. Convert path $< l_2, \ldots, l_{10} >$ into formula:
   $x_1 = 0 \wedge x_2 = x_1 + 1 \wedge x_2 \leq 0$
2. Check for satisfiability: unsat!
3. Let solver compute interpolants:

$$x_1 = 0$$

$$x_1 \geq 0$$

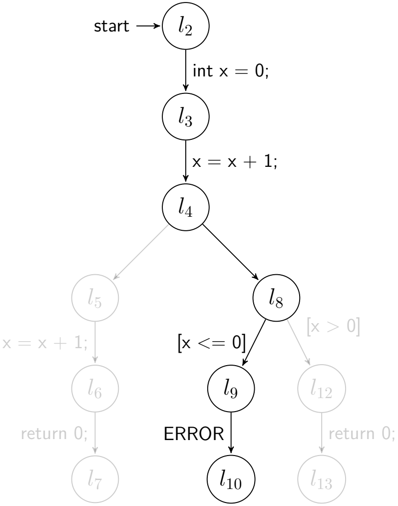
$$x_2 = x_1 + 1$$

$$x_2 > 0$$

$$x_2 \leq 0$$

Predicate Abstraction: Example

Refinement



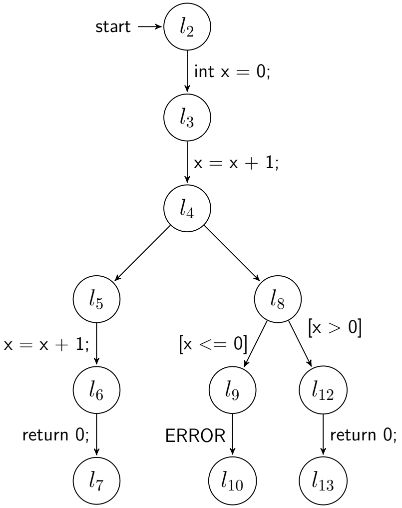
1. Convert path $< l_2, \ldots, l_{10} >$ into formula:
   $x_1 = 0 \wedge x_2 = x_1 + 1 \wedge x_2 \leq 0$
2. Check for satisfiability: unsat!
3. Let solver compute interpolants:

   $$x_1 = 0$$
   $$x_1 \geq 0$$
   $$x_2 = x_1 + 1$$
   $$x_2 > 0$$
   $$x_2 \leq 0$$

4. Add interpolants to precision:
   $\pi = \{x \geq 0, x > 0\}$

Predicate Abstraction: Example

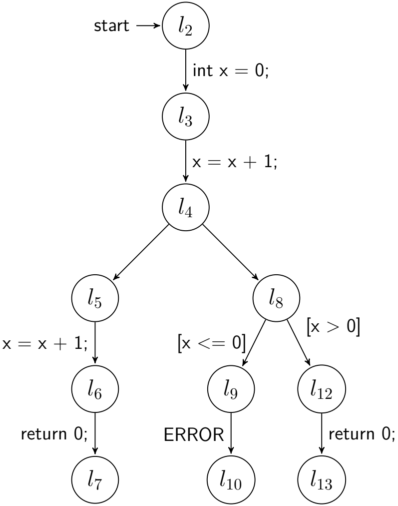
with precision after refinement: $\pi = \{x \geq 0, x > 0\}$

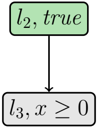


$l_2, true$

Predicate Abstraction: Example

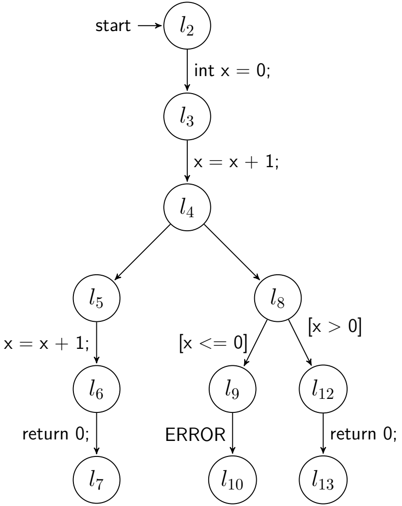
with precision after refinement: $\pi = \{x \geq 0, x > 0\}$

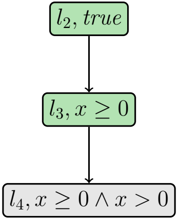




Predicate Abstraction: Example

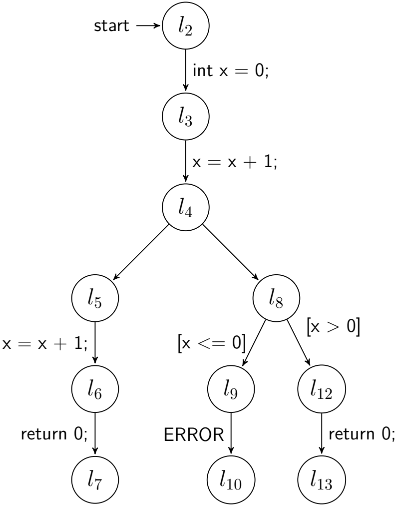
with precision after refinement: $\pi = \{x \geq 0, x > 0\}$

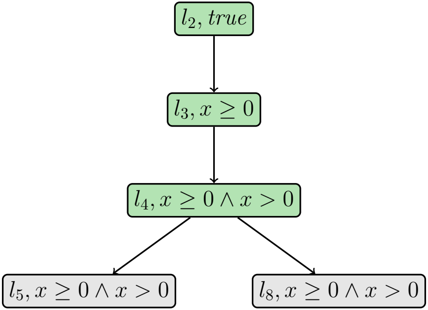




Predicate Abstraction: Example

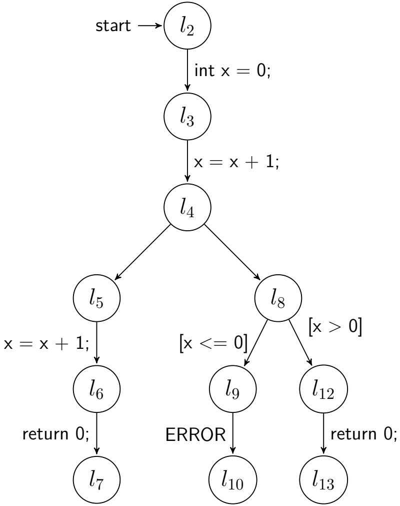
with precision after refinement: $\pi = \{x \geq 0, x > 0\}$

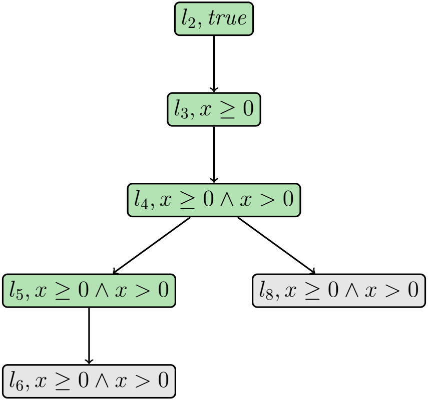




Predicate Abstraction: Example

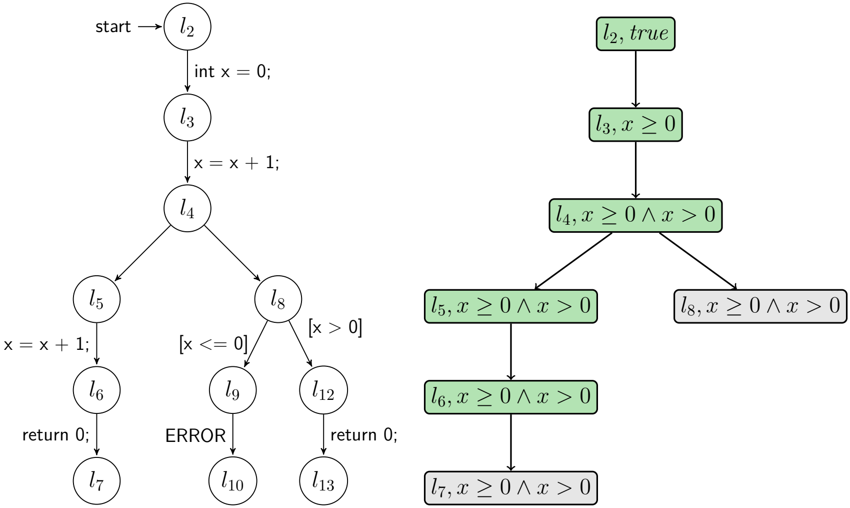
with precision after refinement: $\pi = \{x \geq 0, x > 0\}$





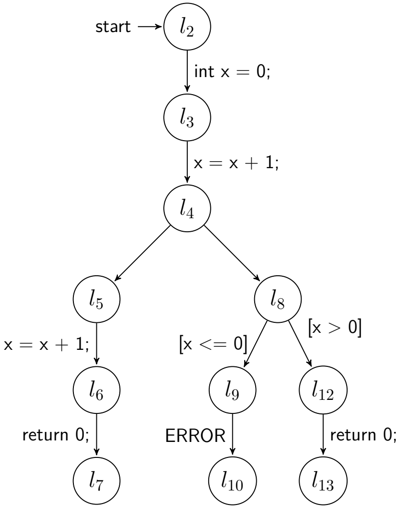
Predicate Abstraction: Example

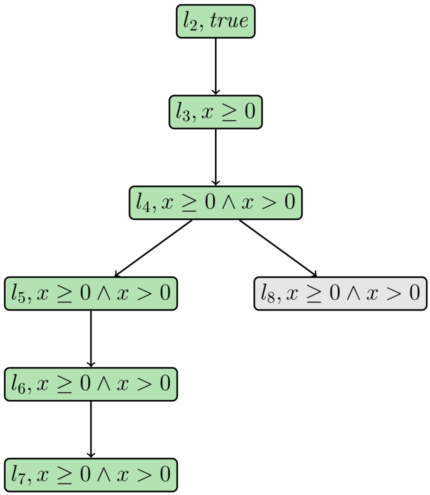
with precision after refinement: $\pi = \{x \geq 0, x > 0\}$



Predicate Abstraction: Example

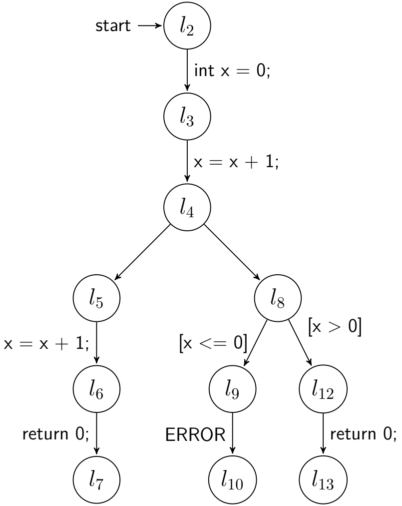
with precision after refinement: $\pi = \{x \geq 0, x > 0\}$

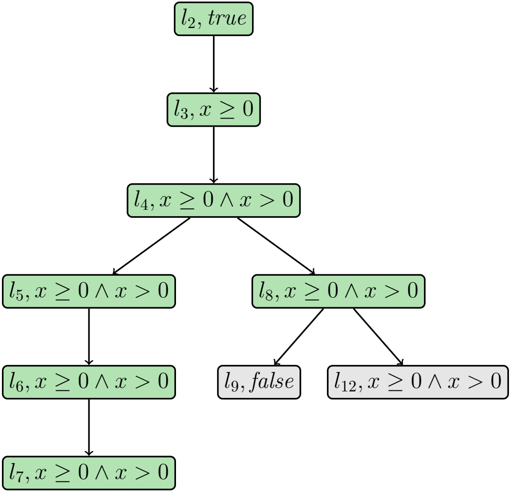




Predicate Abstraction: Example

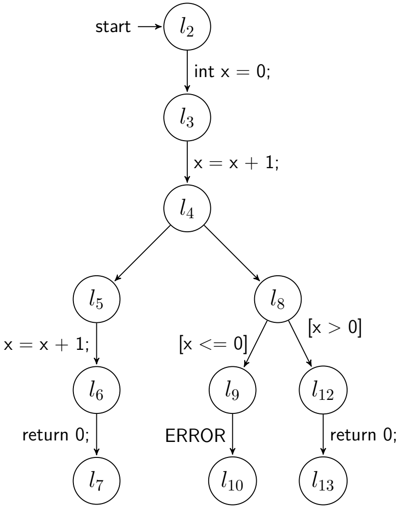
with precision after refinement: $\pi = \{x \geq 0, x > 0\}$

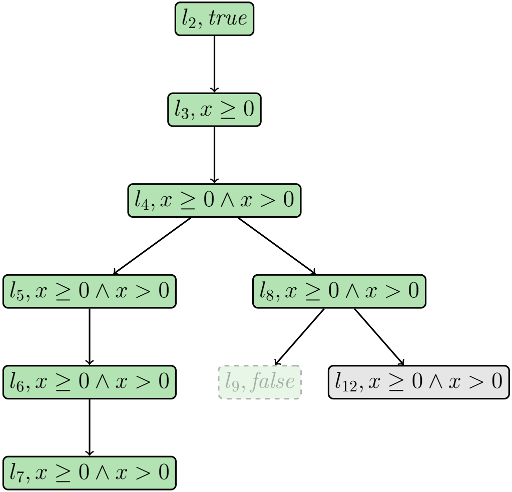




Predicate Abstraction: Example

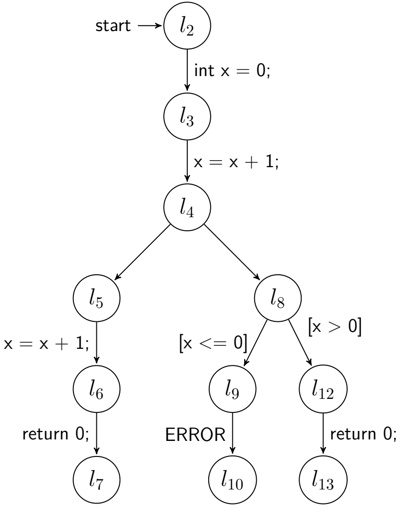
with precision after refinement: $\pi = \{x \geq 0, x > 0\}$

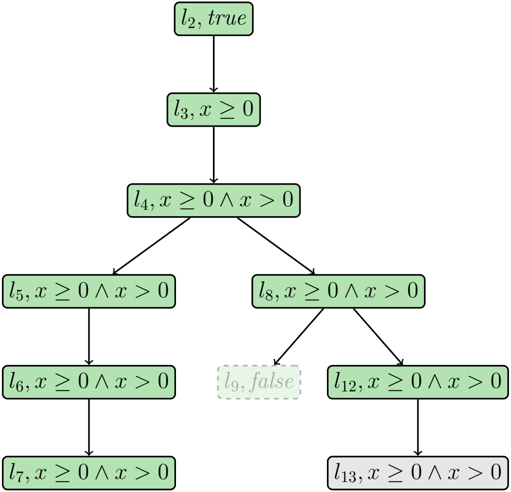




Predicate Abstraction: Example

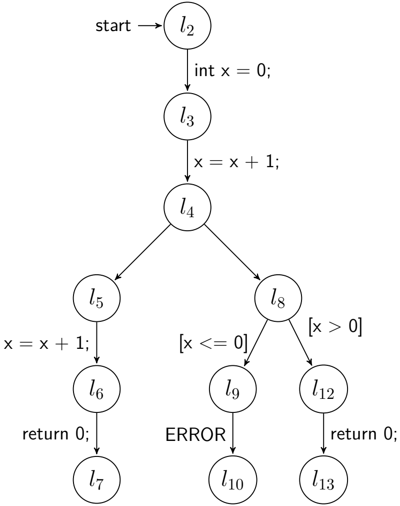
with precision after refinement: $\pi = \{x \geq 0, x > 0\}$

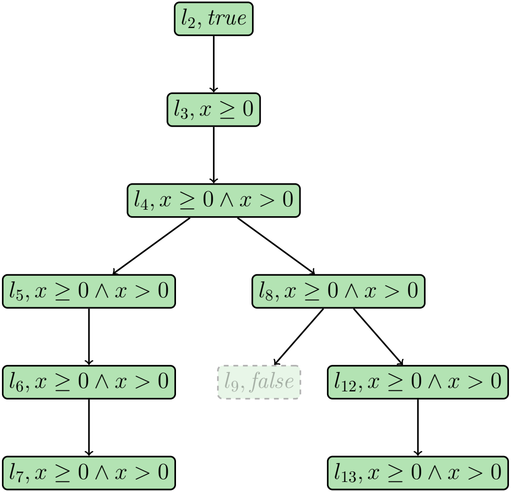




Predicate Abstraction: Example

with precision after refinement: $\pi = \{x \geq 0, x > 0\}$

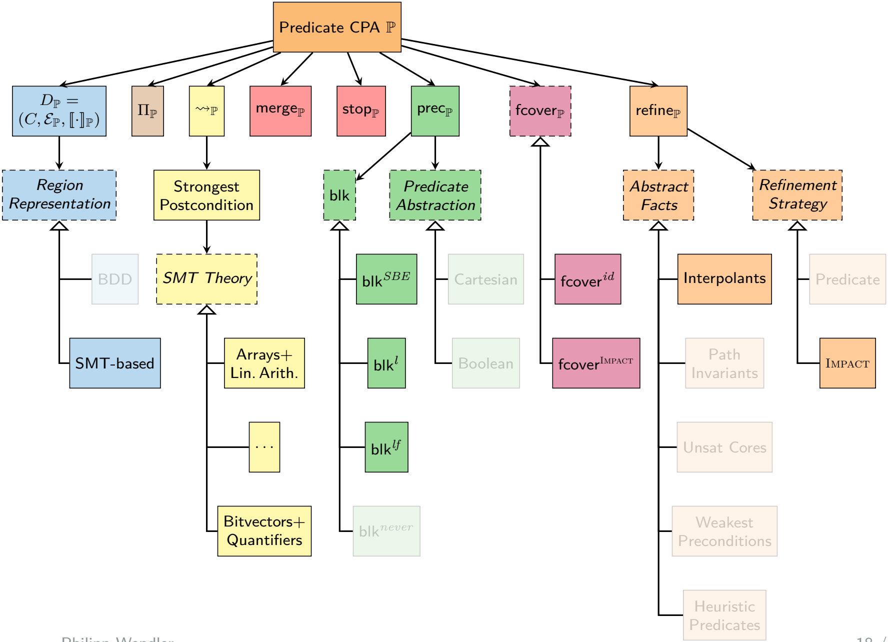




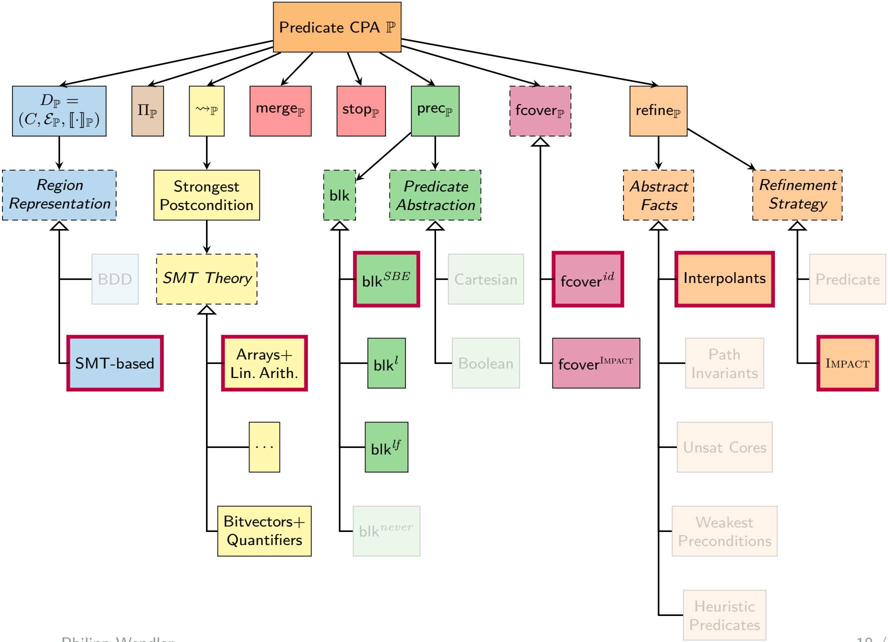
Impact

- "Lazy Abstraction with Interpolants" [CAV'06]
- Abstraction is derived dynamically/lazily
- Solution to avoiding expensive abstraction computations
- Compute fixed point over three operations
  - Expand
  - Refine
  - Cover

Predicate CPA for Impact

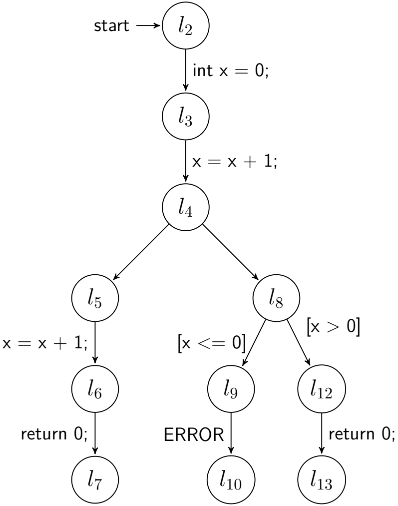


Predicate CPA for Impact



IMPACT: Example

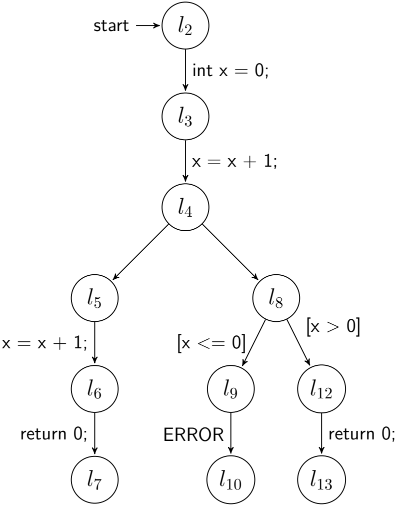
with precision $\pi = \{\}$

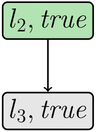


$\boxed{l_2, true}$

Impact: Example

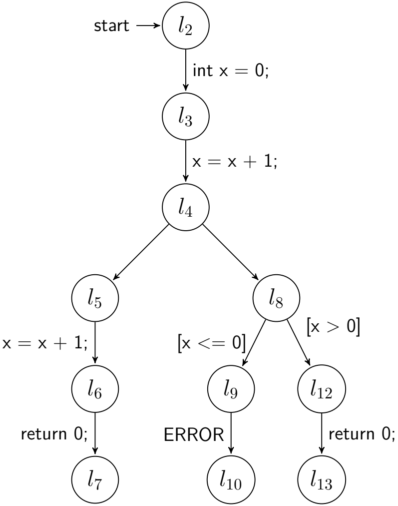
with precision $\pi = \{\}$

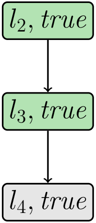




Impact: Example

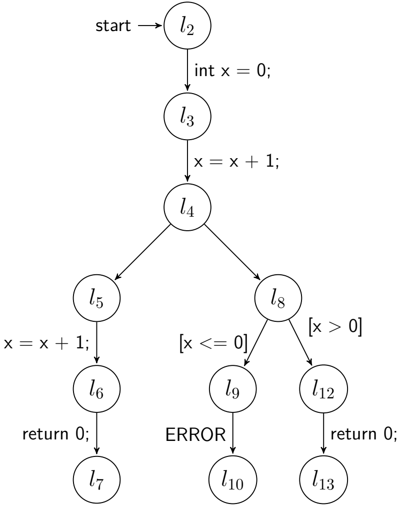
with precision $\pi = \{\}$

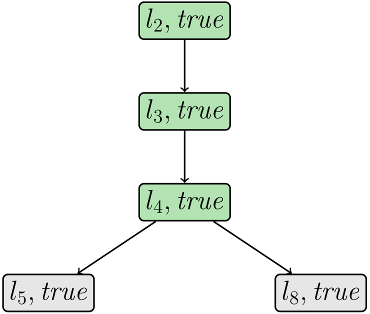




Impact: Example

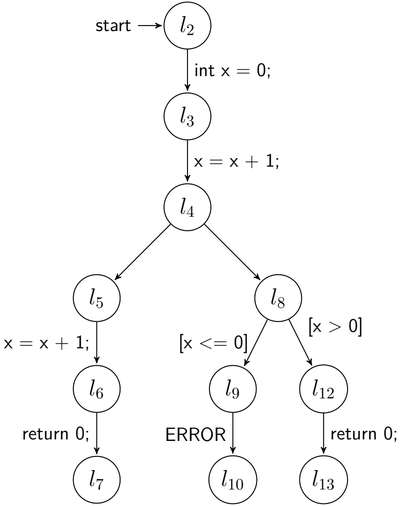
with precision $\pi = \{\}$

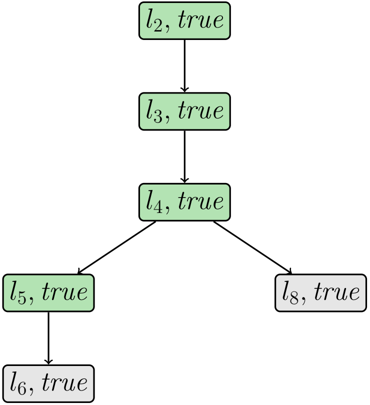




IMPACT: Example

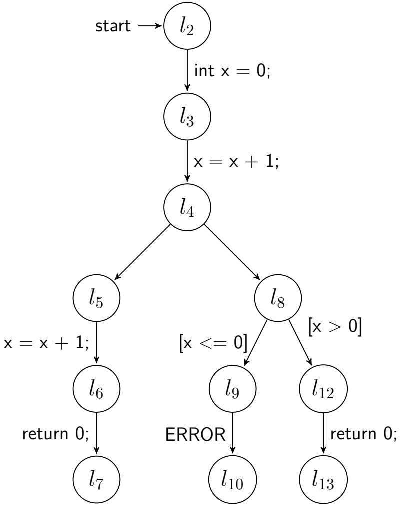
with precision $\pi = \{\}$

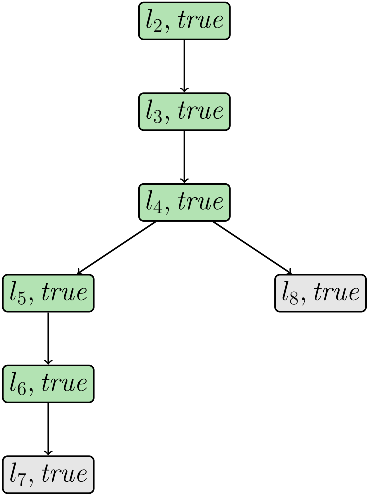




Impact: Example

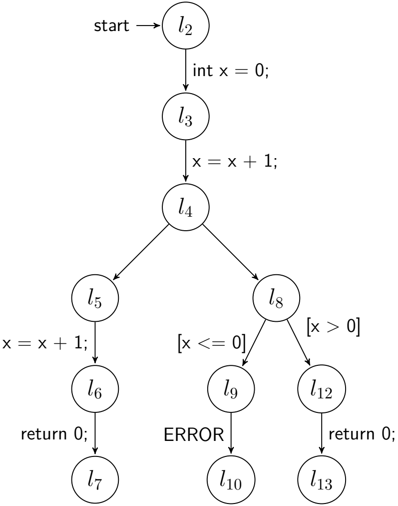
with precision $\pi = \{\}$

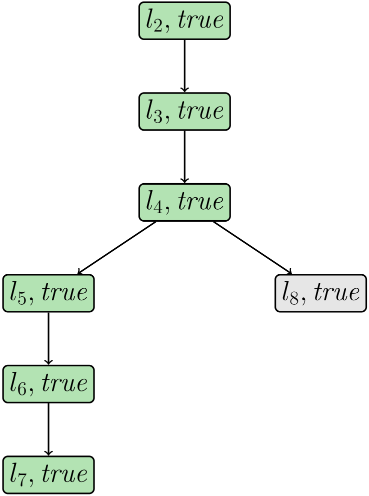




Impact: Example

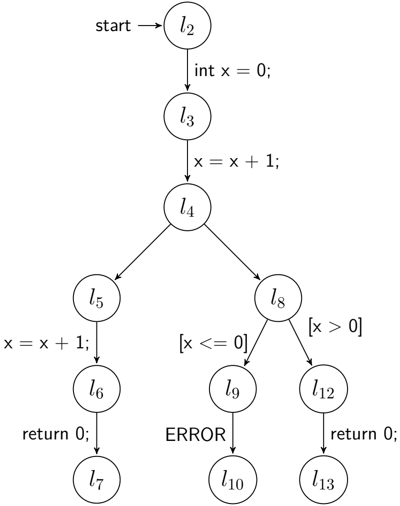
with precision $\pi = \{\}$

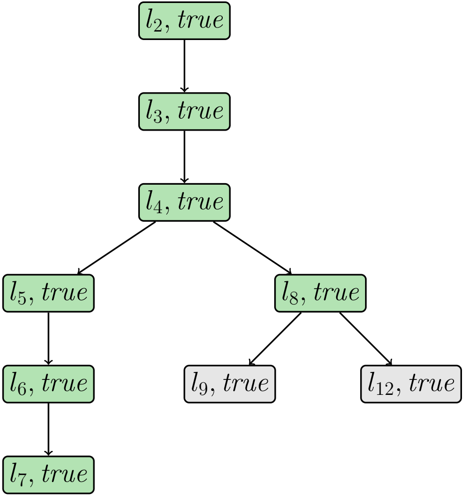




IMPACT: Example

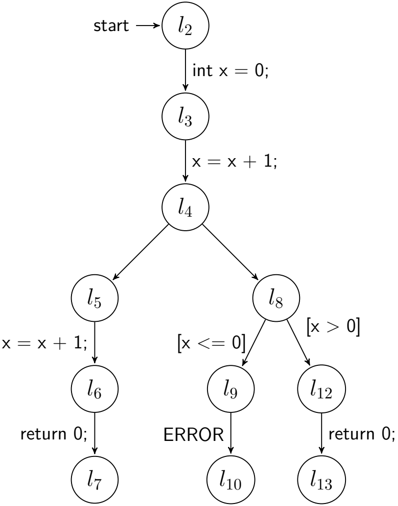
with precision $\pi = \{\}$

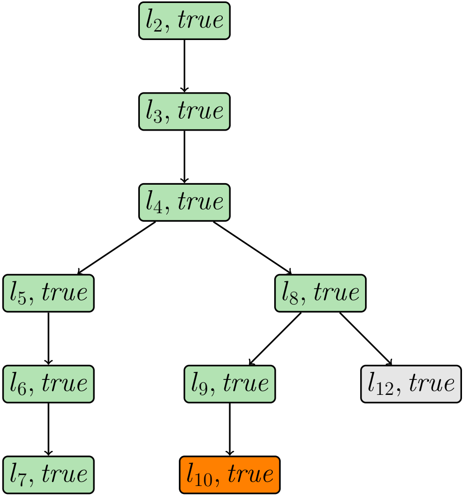




IMPACT: Example

with precision $\pi = \{\}$

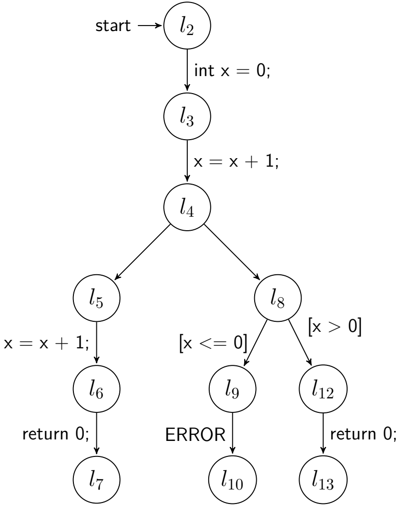


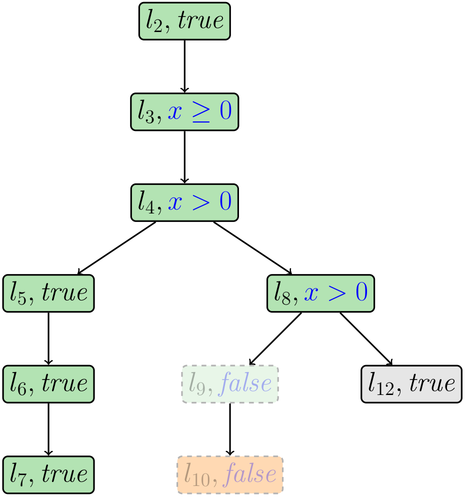


Refinement of path $< l_2, \ldots, l_{10} >$ using interpolation

IMPACT: Example

with precision $\pi = \{\}$

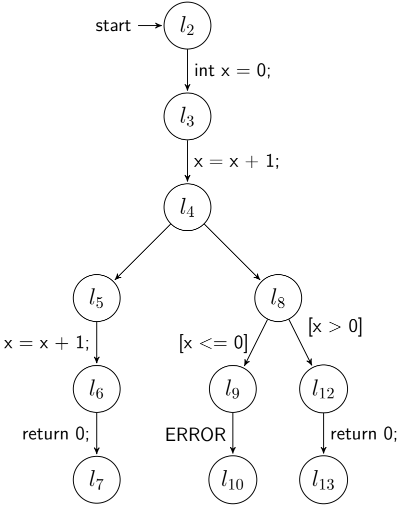


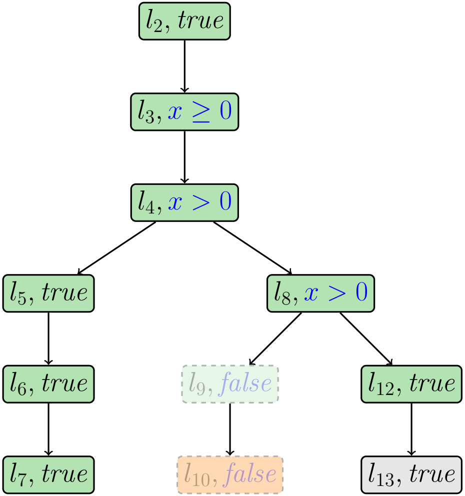


Refinement of path $< l_2, \ldots, l_{10} >$ using interpolation

Impact: Example

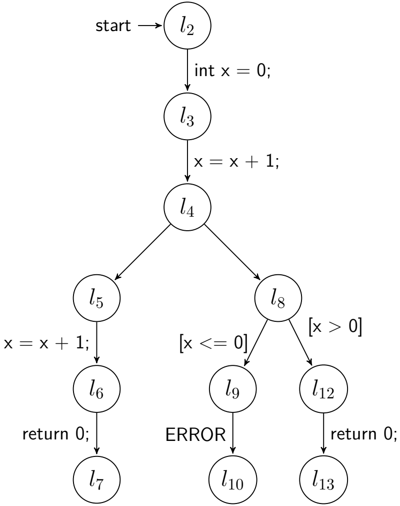
with precision $\pi = \{\}$

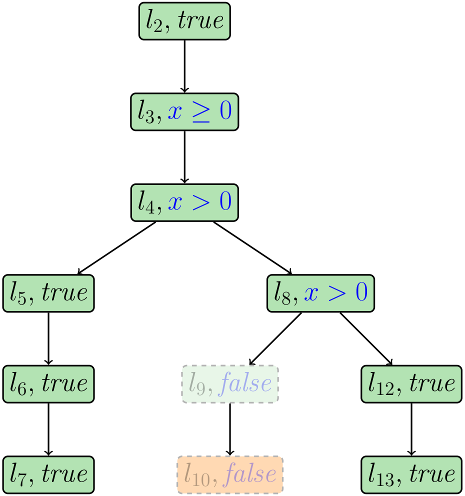




Impact: Example

with precision $\pi = \{\}$





Insights

- Difference between predicate abstraction and IMPACT:
  - Differences explicit via configuration options of framework
  - We know that only these differences are relevant!
  - Predicate abstraction pays for creating more general abstract model
  - IMPACT is lazier but this can lead to many refinements
    $\rightarrow$ forced covering proposed as optimization

Insights

- Difference between predicate abstraction and IMPACT:
  - Differences explicit via configuration options of framework
  - We know that only these differences are relevant!
  - Predicate abstraction pays for creating more general abstract model
  - IMPACT is lazier but this can lead to many refinements $\rightarrow$ forced covering proposed as optimization
- New combinations of features possible
  - Example: IMPACT with adjustable-block encoding reduces high number of refinements

Evaluation: Usefulness of Framework

- 4 existing approaches successfully integrated
- Ongoing projects for integration of further approaches
- Interesting insights learned about these approaches
- High configurability allows new combinations and hybrid approaches

Evaluation: Usefulness of Framework

- 4 existing approaches successfully integrated
- Ongoing projects for integration of further approaches
- Interesting insights learned about these approaches
- High configurability allows new combinations and hybrid approaches
- Already used as base for successful research projects by other researchers, e.g.
  - Block-abstraction memoization [ICFEM'12]
  - Refinement selection [SPIN'15]
  - Local policy iteration [VMCAI'16]
  - ...

Evaluation: Usefulness of Implementation

- Used in other research projects:
  - Conditional model checking [FSE'12]
  - Verifying recursive programs [SAS'14]
  - Verification witnesses [FSE'15, FSE'16]

Evaluation: Usefulness of Implementation

- Used in other research projects:
  - Conditional model checking [FSE'12]
  - Verifying recursive programs [SAS'14]
  - Verification witnesses [FSE'15, FSE'16]
- Enables experimental studies:
  - SMT-based software model checking: An experimental comparison of four algorithms [VSTTE'16]
  - Comparison of SMT solvers and theories
    - 120 different configurations on 5 594 programs
    - Important insights on how SMT solvers and theories can influence benchmark results and skew conclusion

Both studies not possible before!

Evaluation: Comparison with State of the Art

- State of the art visible in Intl. Competition on Software Verification (SV-COMP)
- Implementation won 4 medals in first year (SV-COMP'12)

- Contributed to 40 more medals
- Awarded Gödel medal by Kurt Gödel Society

Conclusion

1. Provide a unifying framework for predicate analyses

2. Understand differences and key concepts of approaches

3. Determine potential of extensions and combinations

4. Provide solid platform for experimental research

Conclusion

1. Provide a unifying framework for predicate analyses ✓
   - Formally defined and used for 4 different analyses: accepted for Journal of Automated Reasoning (JAR)
   - Adopted by others for even more analyses
2. Understand differences and key concepts of approaches
3. Determine potential of extensions and combinations
4. Provide solid platform for experimental research

Conclusion

1. Provide a unifying framework for predicate analyses ✓
   - Formally defined and used for 4 different analyses: accepted for Journal of Automated Reasoning (JAR)
   - Adopted by others for even more analyses
2. Understand differences and key concepts of approaches ✓
   - Interesting insights found
3. Determine potential of extensions and combinations

4. Provide solid platform for experimental research

Conclusion

1. Provide a unifying framework for predicate analyses ✓
   - Formally defined and used for 4 different analyses: accepted for Journal of Automated Reasoning (JAR)
   - Adopted by others for even more analyses
2. Understand differences and key concepts of approaches ✓
   - Interesting insights found
3. Determine potential of extensions and combinations ✓
   - New combinations of features and algorithms immediately available
   - Large potential for future work
4. Provide solid platform for experimental research

Conclusion

1. Provide a unifying framework for predicate analyses ✓
   - Formally defined and used for 4 different analyses: accepted for Journal of Automated Reasoning (JAR)
   - Adopted by others for even more analyses
2. Understand differences and key concepts of approaches ✓
   - Interesting insights found
3. Determine potential of extensions and combinations ✓
   - New combinations of features and algorithms immediately available
   - Large potential for future work
4. Provide solid platform for experimental research ✓
   - Top-ranking implementation provided
   - Used for several experimental studies

Benchmarking

- Performance benchmarking is crucial
  for research of automatic software verification
- Common tools and technologies
  prone to measurement errors of arbitrary size
- Developed new benchmarking framework BenchExec
  based on modern concepts *cgroups* and *namespaces*
- Ensures reliable benchmarking
- Used by SV-COMP and many research teams
- Published in SPIN'15 and STTT

Bounded Model Checking

- [TACAS'99]
- No abstraction
- Unroll loops up to a loop bound k
- Check that P holds in the first k iterations:

$$\bigwedge_{i=1}^{k} P(i)$$

k-Induction

- k-Induction generalizes the induction principle:
- No abstraction
- Base case: Check that P holds in the first k iterations:
  $\rightarrow$ Equivalent to BMC with loop bound k
- Step case: Check that the safety property is k-inductive:

$$\forall n : \left(\left(\bigwedge_{i=1}^{k} P(n+i-1) \right) \Rightarrow P(n+k) \right)$$

- Stronger hypothesis is more likely to succeed
- Add auxiliary invariants [PDMC'11]
- Heavy-weight proof technique

Comparison with SV-COMP'17 Verifiers

- SV-COMP'17 benchmark set:
  5 594 C programs with known result
- Time limit 900 s, memory limit 15 GB (per task)
- Comparison of
  - 4 configurations of CPAchecker with Predicate CPA:
    BMC, *k*-induction, Impact, predicate abstraction
  - 16 participants of SV-COMP'17

Comparison with SV-COMP'17 Verifiers: Results

Number of correctly solved tasks:

- Each configuration of Predicate CPA beats other tools with same approach
- Only 3 tools beat Predicate CPA with *k*-induction:
  - SMACK: guesses results
  - CPA-BAM-BNB, CPA-SEQ: based on Predicate CPA as well

Comparison with SV-COMP'17 Verifiers: Results

Number of correctly solved tasks:

- Each configuration of Predicate CPA beats other tools with same approach
- Only 3 tools beat Predicate CPA with *k*-induction:
  - SMACK: guesses results
  - CPA-BAM-BNB, CPA-SEQ: based on Predicate CPA as well

Number of wrong results:

- Comparable with other tools
- No wrong proofs (sound)

Comparison with SV-COMP'17 Verifiers