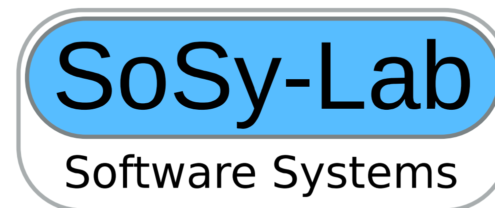
Configurable Software-Verification

Dirk Beyer

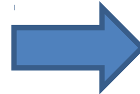


Software Verification

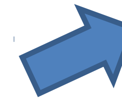
C program

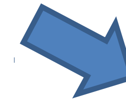
```
int main() {
  int a = foo();
  int b = bar(a);

  assert(a == b);
}
```



Verification Tool



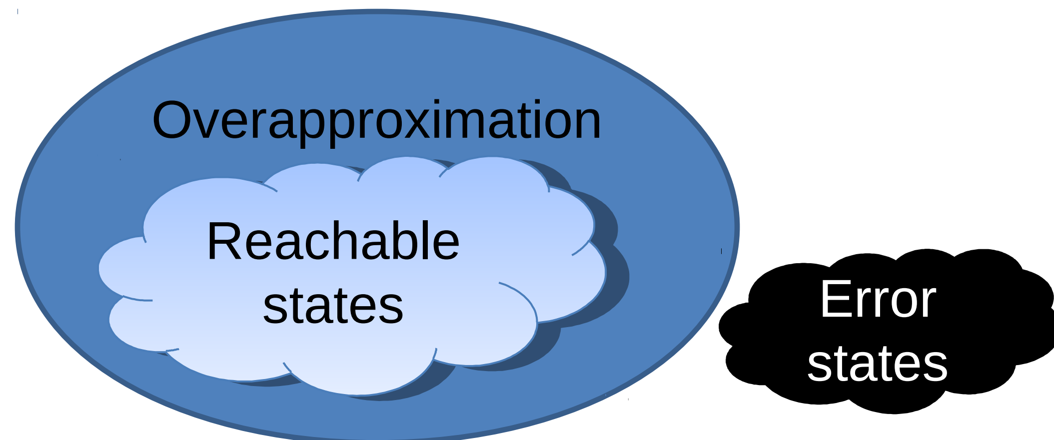


SAFE

i.e. assertions cannot be violated

UNSAFE

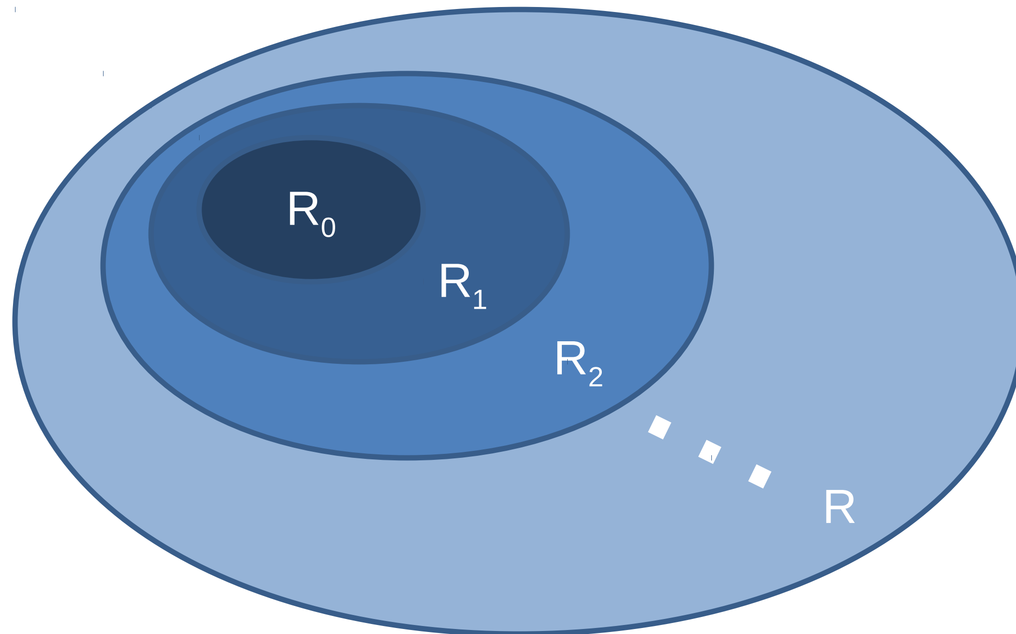
General method:
Create an overapproximation of the program states



Software Verification by Model Checking

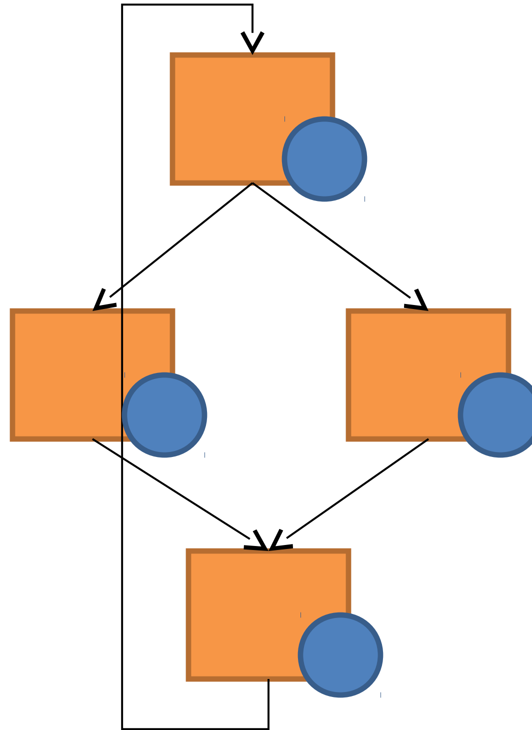
[Clarke/Emerson, Sifakis 1981]

Iterative fixpoint (forward) post computation



Software Verification by Data-Flow Analysis

Fixpoint computation on the CFG



Software Model Checking

Reached, *Frontier* := { e_0 }

while *Frontier* ≠ ∅ do

remove *e* from *Frontier*

for each *e'* ∈ **post**(*e*) do

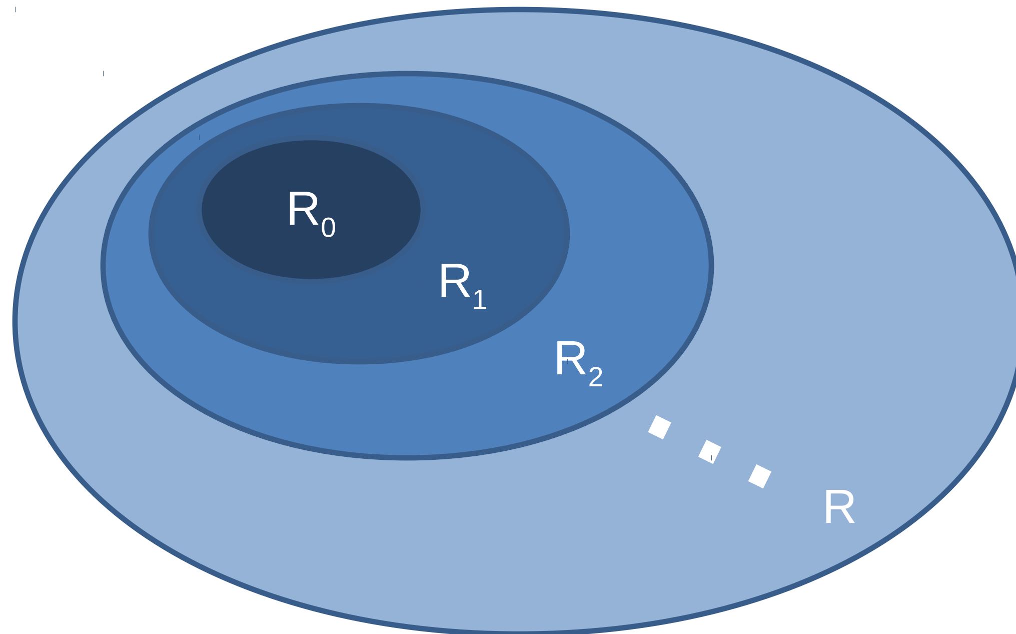
if ¬ **stop**(*e'*, *Reached*) add *e'* to *Reached*, *Frontier*

return *Reached*

Software Verification by Model Checking

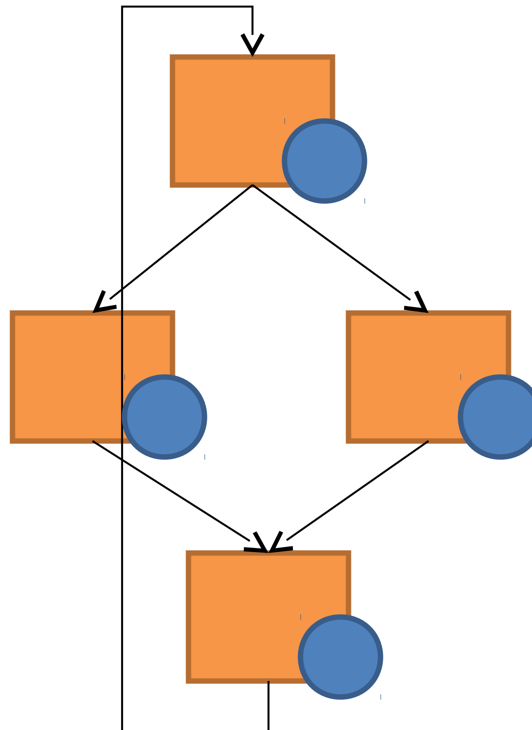
[Clarke/Emerson, Sifakis 1981]

Iterative fixpoint (forward) post computation



Software Verification by Data-flow Analysis

Fixpoint computation on the CFG



Software Model Checking

Reached, *Frontier* := { e_0 }

while *Frontier* $\neq \emptyset$ do

  remove *e* from *Frontier*

  for each *e'* $\in$ **post**(*e*) do

    if $\neg$ **stop**(*e'*, *Reached*) add *e'* to *Reached*, *Frontier*

return *Reached*

Configurable Program Analysis

[CAV 2007]

Reached, *Frontier* := { e_0 }

while *Frontier* $\neq \emptyset$ do

  remove *e* from *Frontier*

  for each *e'* $\in$ **post**(*e*) do

    for each *e''* $\in$ *Reached* do

      e''_{new} := **merge**(*e'*, *e''*)

      if $e''_{new} \neq e''$ then

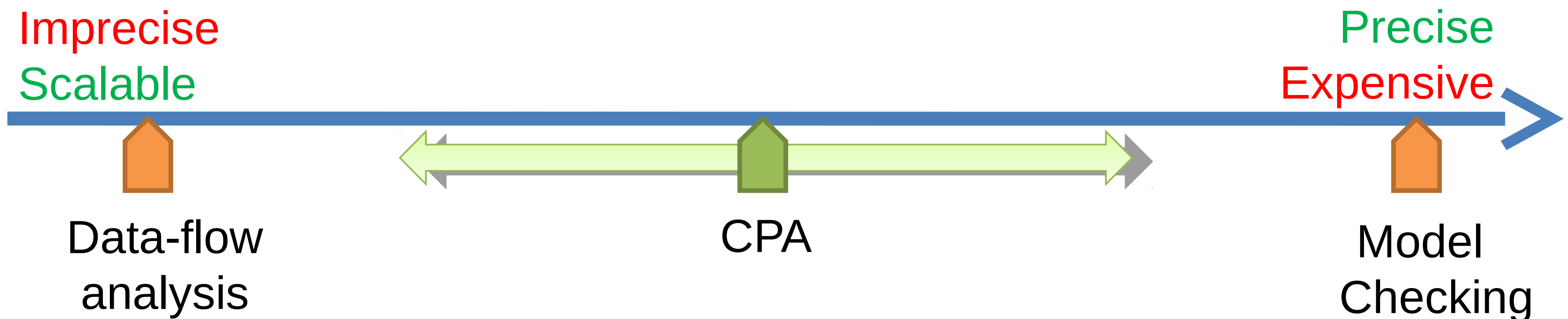
        replace *e''* in *Reached*, *Frontier* by e''_{new}

    if $\neg$ **stop**(*e'*, *Reached*) add *e'* to *Reached*, *Frontier*

return *Reached*

Configurable Program Analysis

- Better combination of abstractions
  ➔ Configurable Program Analysis [CAV07]



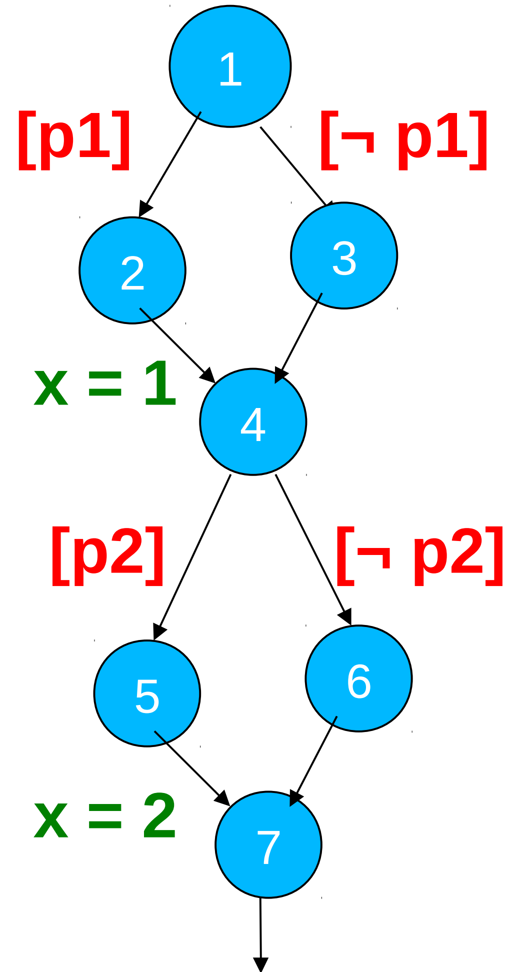
Unified framework that enables intermediate algorithms

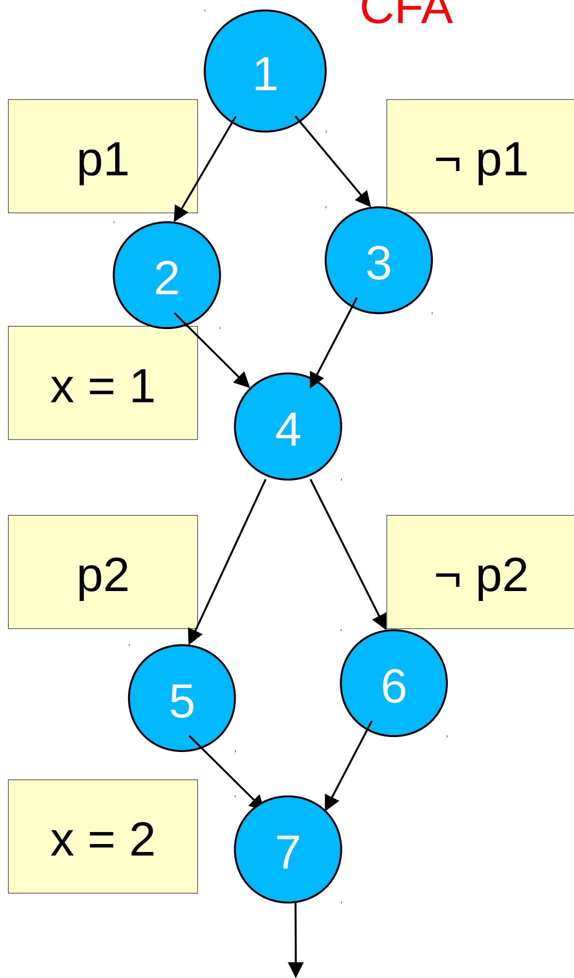
Example Domain:

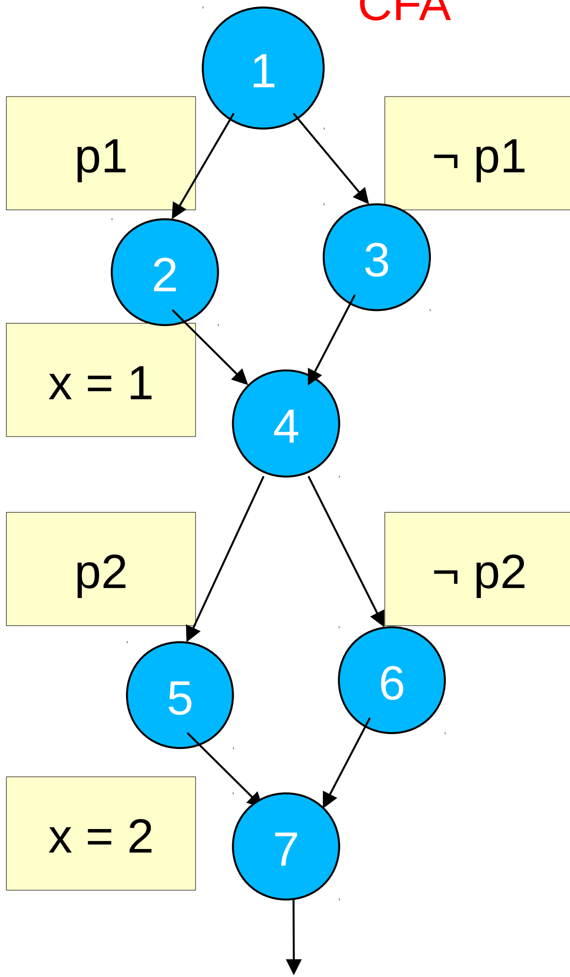
Predicate Analysis with Late Abstraction

Control-Flow Automaton

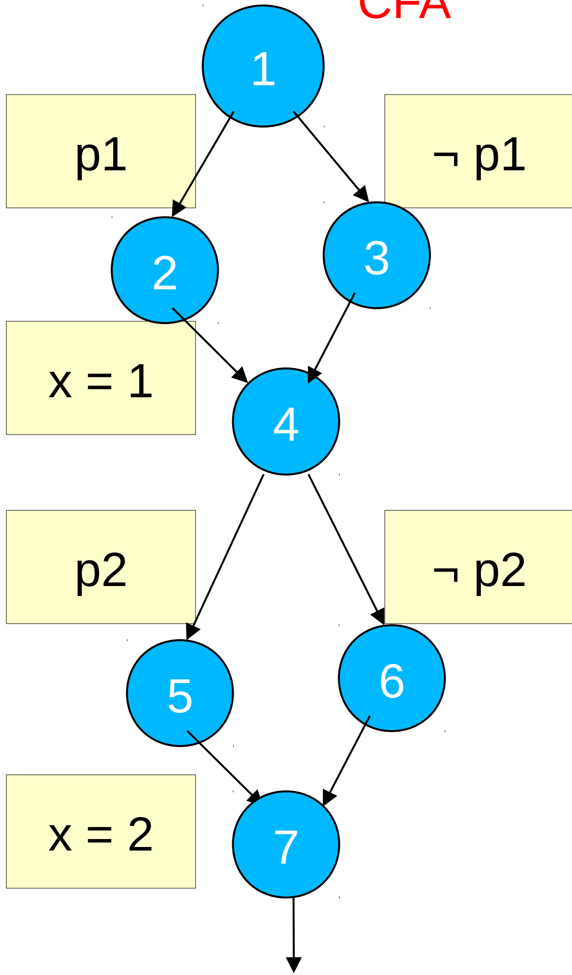
```
if (p1) {
    x = 1;
}
if (p2) {
    x = 2;
}
  ...
if (pN) {
    x = N;
}
```

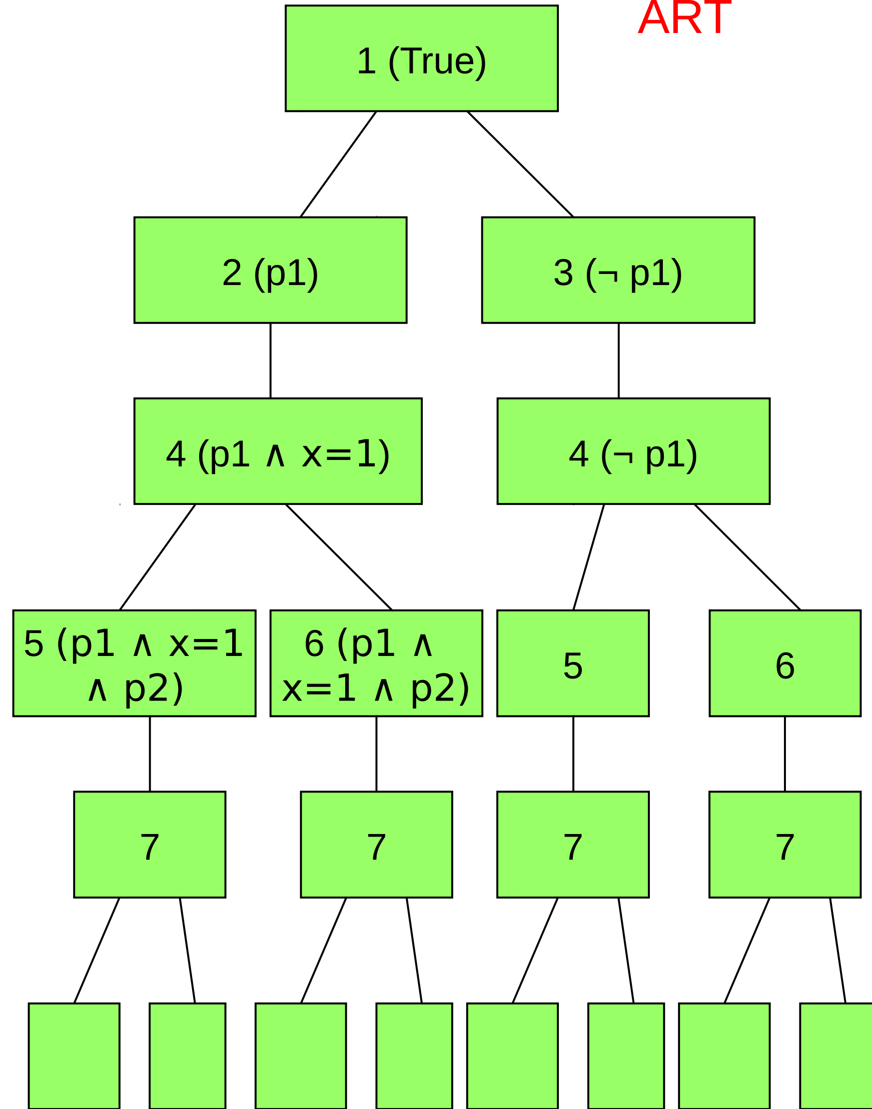



CFA
1
p1
¬ p1
2
3
x = 1
4
p2
¬ p2
5
6
x = 2
7



Precision = Predicate list:
p1, ¬ p1, x=1, p2, ...





Precision = Predicate list:
p1, ¬ p1, x=1, p2, ...

Abstract Successors

Abstract state: (Φ, ψ)

Φ: Strongest Post ψ: Abstract Formula

Example:

Precision: { $x > 0$ }

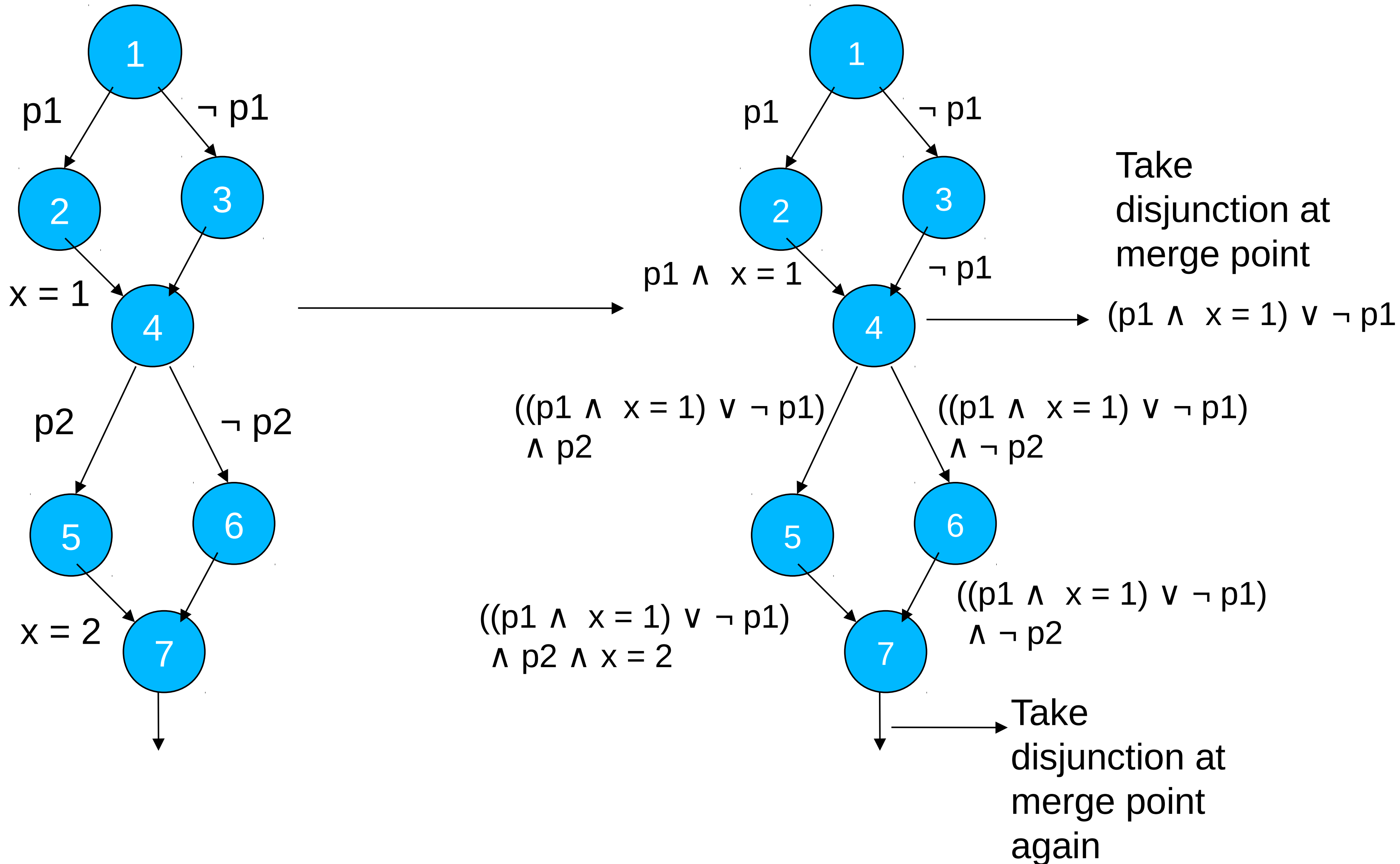
Current abstract state: (true, $x > 0$)

CFA edge: $x := 1$

Successor abstract state: ($x = 1$, $x > 0$)

After predicate abstraction: (true, $x > 0$)

Symbolic Approach FMCAD'10



Abstract Successors

Abstract state: (Φ, ψ)

Φ: Strongest Post ψ: Abstract Formula

Example:

Precision: { $x > 0$ }

Current abstract state: (true, $x > 0$)

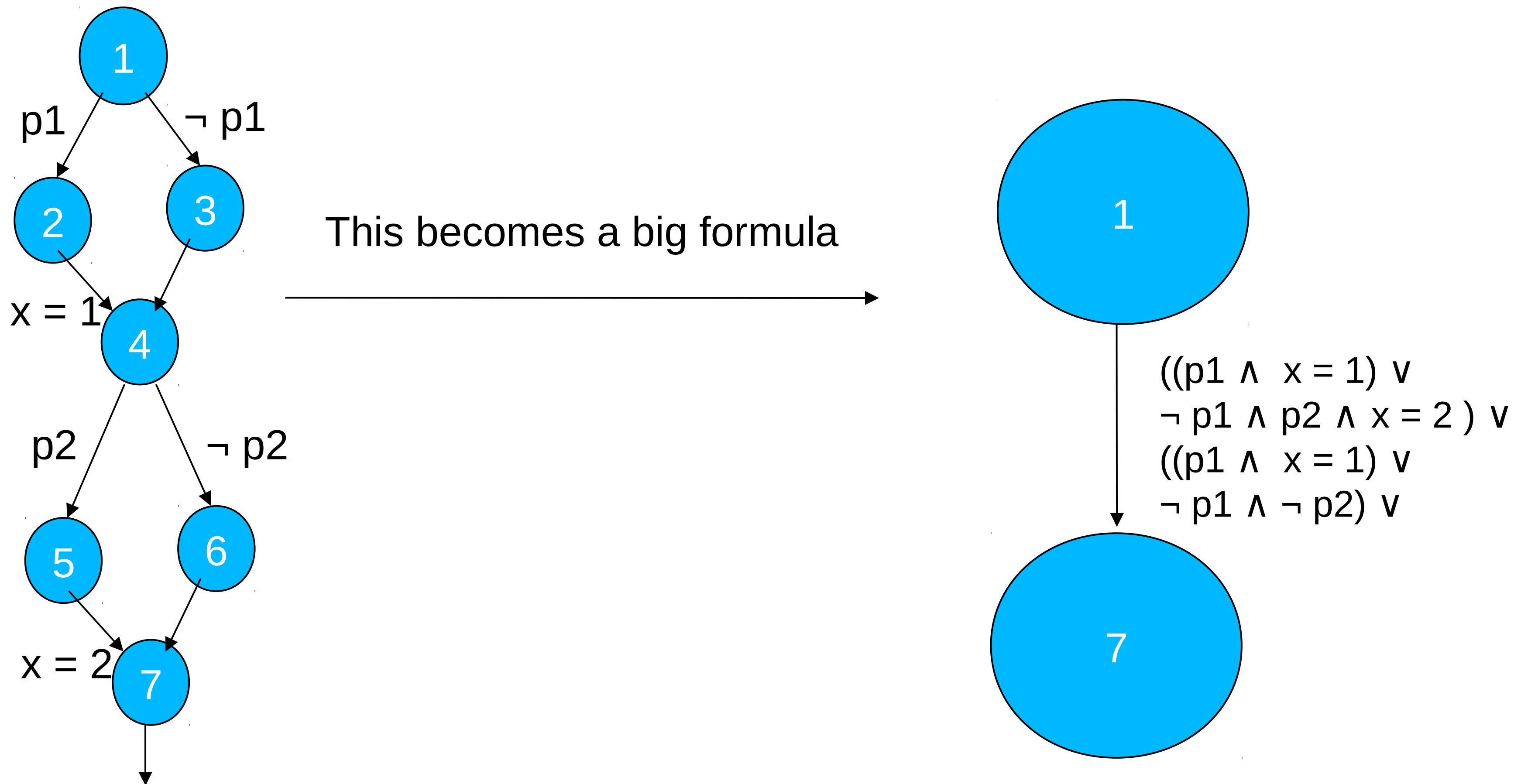
CFA edge: $x := 1$

Successor abstract state: ($x = 1$, $x > 0$)

After predicate abstraction: (true, $x > 0$)

From a Different Viewpoint

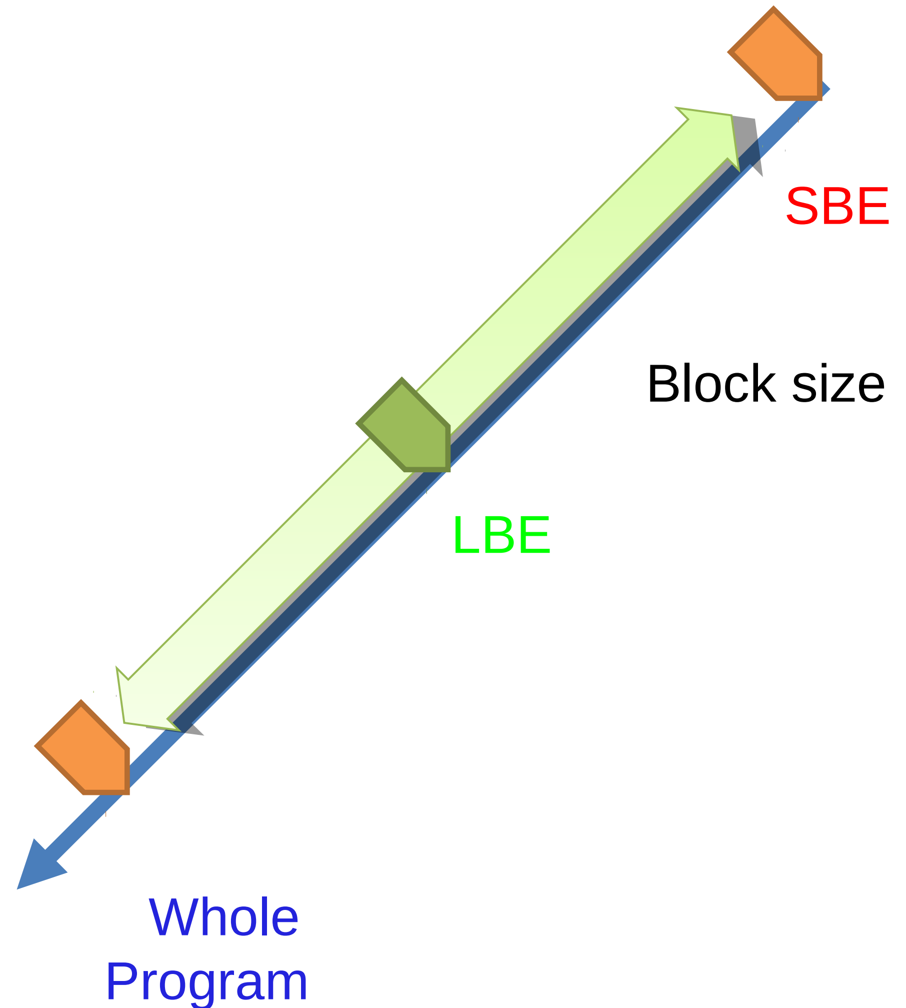
These are purely syntactical operations in a number of steps linear in n.



Adjustable-Block Encoding

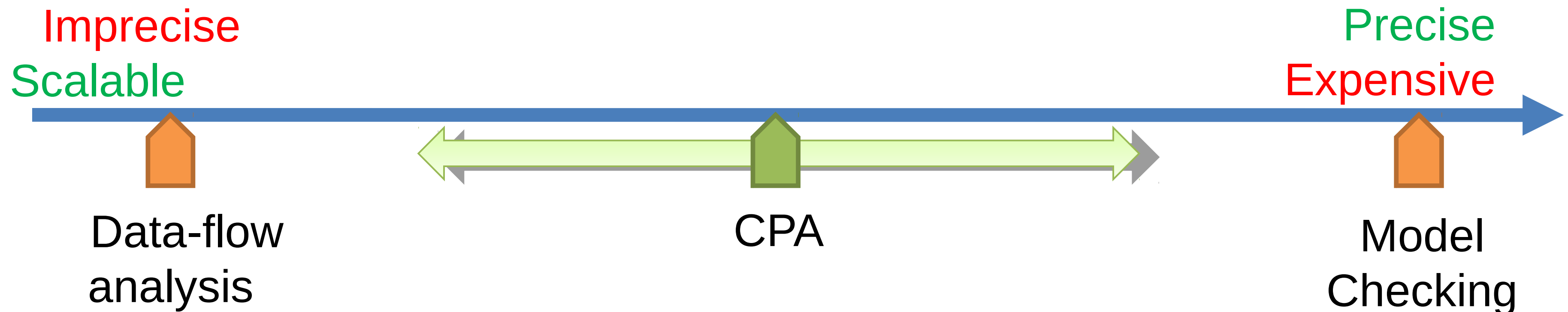
- We can use more power of SMT
- Disjunctions not handled explicitly
  ART not forced to grow exponentially
- Precise boolean abstraction
- Reduced number of abstractions
- Reduced number of refinements

Adjustable Block Size



Configurable Program Analysis

- Better combination of abstractions
  ➔ Configurable Program Analysis
  [B/Henzinger/Theoduloz CAV'07]



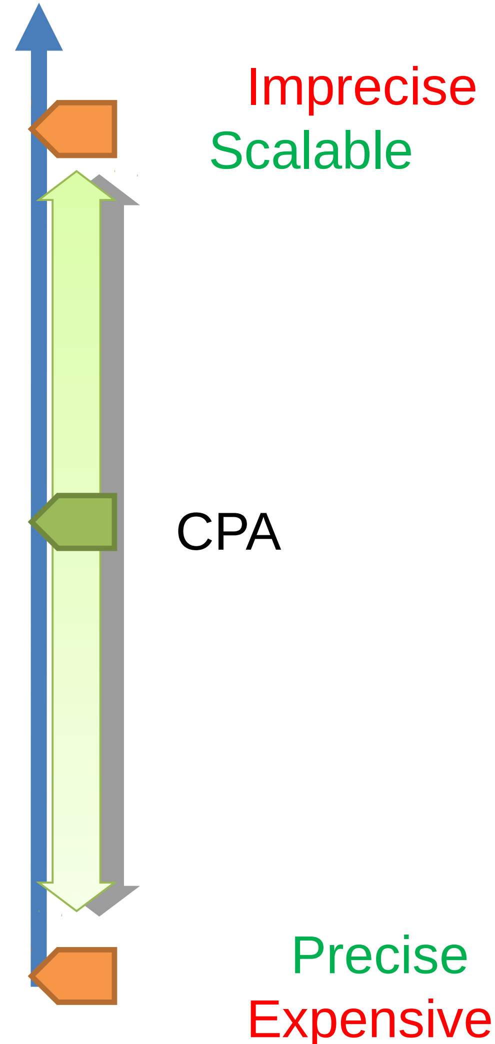
Unified framework that enables intermediate algorithms

Dynamic Precision Adjustment

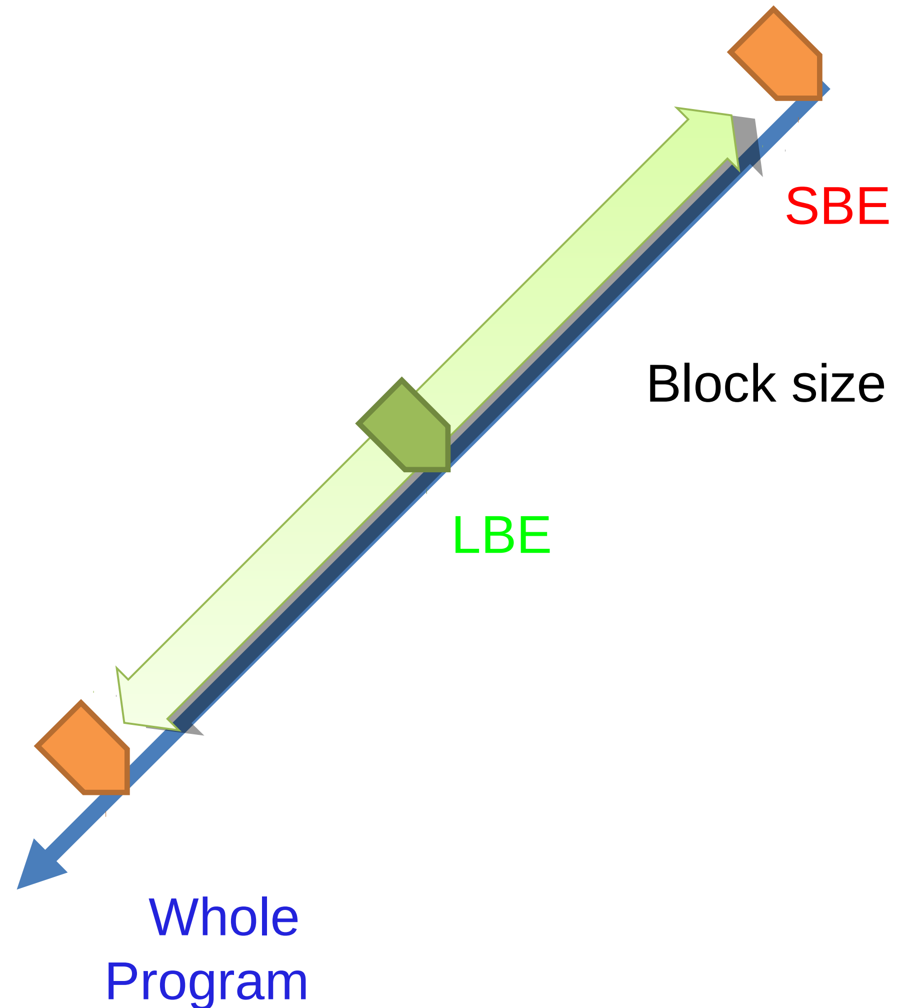
Better fine tuning of the precision of abstractions
➔ Adjustable Precision [B/Henzinger/Theoduloz ASE'08]

Unified framework enables:
- switch on and off different analyses, and can
- adjust each analysis separately

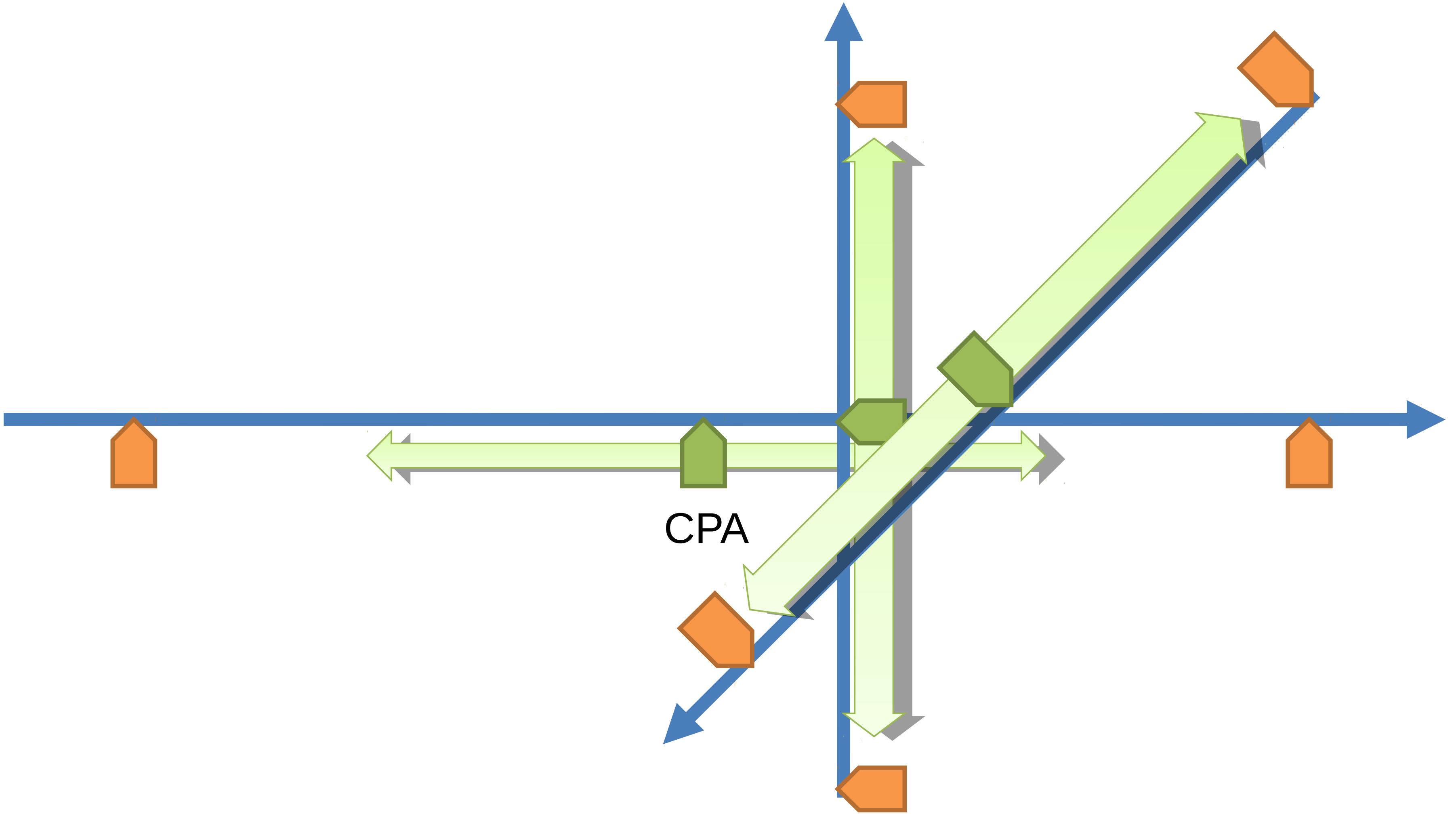
- Not only **refine**, also **abstract**!



Adjustable Block Size



CPAchecker



CPA – Intermediate Summary

- Unification of several approaches
  → reduced to their essential properties
- Allow experimentation with new configurations that we would never think of
- Flexible implementation CPAchecker

CPAchecker

- Framework for Software Verification
  - Written in Java
  - Open Source: Apache 2.0 License
  - ~40 contributors so far
    from 7 universities/institutions
  - 335.000 lines of code
    (205.000 without blank lines and comments)
  - Started 2007

http://cpachecker.sosy-lab.org

CPAchecker: Features

- Input language C (experimental: Java)
- Web frontend available: http://cpachecker.appspot.com
- Error path output with graphs
- Benchmarking infrastructure available (with large cluster of machines)
- Cross-platform: Linux, Mac, Windows, AppEngine, (Android)

CPAchecker

- Included Concepts:
  - CEGAR
  - Interpolation
  - Adjustable-block encoding
- Further available analyses:
  - IMPACT algorithm
  - Bounded model checking
  - k-Induction
  - Conditional Model Checking

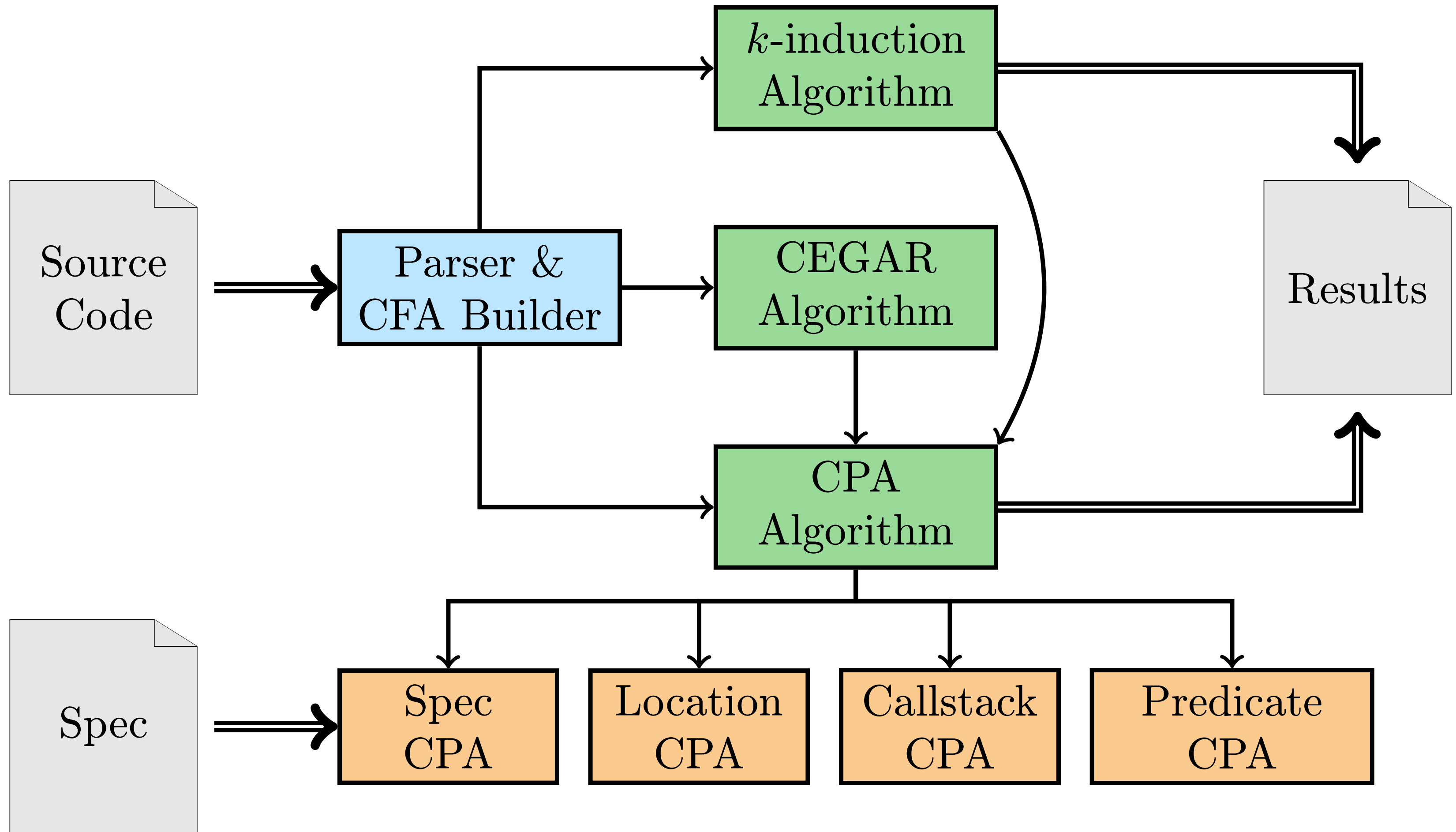
CPAchecker: Concepts

- Completely modular,
  and thus flexible and easily extensible
- Every abstract domain is implemented as a "Configurable Program Analysis" (CPA)
- E.g. predicate abstraction, explicit-value analysis, intervals, octagon, BDDs, and more
- Algorithms are central and implemented only once
- Separation of concerns
- Combined with Composite pattern

CPAchecker: Algorithms

- CPAAlgorithm is the core algorithm for reachability analysis / fixpoint iteration
- Other algorithms can be added if desired, e.g.,
  - CEGAR
  - Double-checking counterexamples
  - Sequential combination of analyses

CPAchecker



Specification

- Model Checkers check only what you specified
- CPAchecker's default:
  - Label `ERROR`
  - Calling function `__assert_fail()`
  - `assert(pred)` needs to be pre-processed
- SV-COMP:
  - Calling function `__VERIFIER_error()`
  - `-spec sv-comp-reachability`

CPAchecker for Developers

Want to implement your own analysis?

- Easy, just write a CPA in Java
- Implementations for 10 interfaces needed
- But for 8, we have default implementations
  - ➔ Minimal configuration:
    abstract state and
    abstract post operator

CPAchecker for Developers

The CPA framework is flexible:

- Many components are provided as CPAs:
  - Location / program counter tracking
  - Callstack tracking
  - Type information
  - Specification input (the automata)
- CPAs can be combined, so your analysis doesn't need to care about these things