Configurable Software Model Checking

CPAchecker

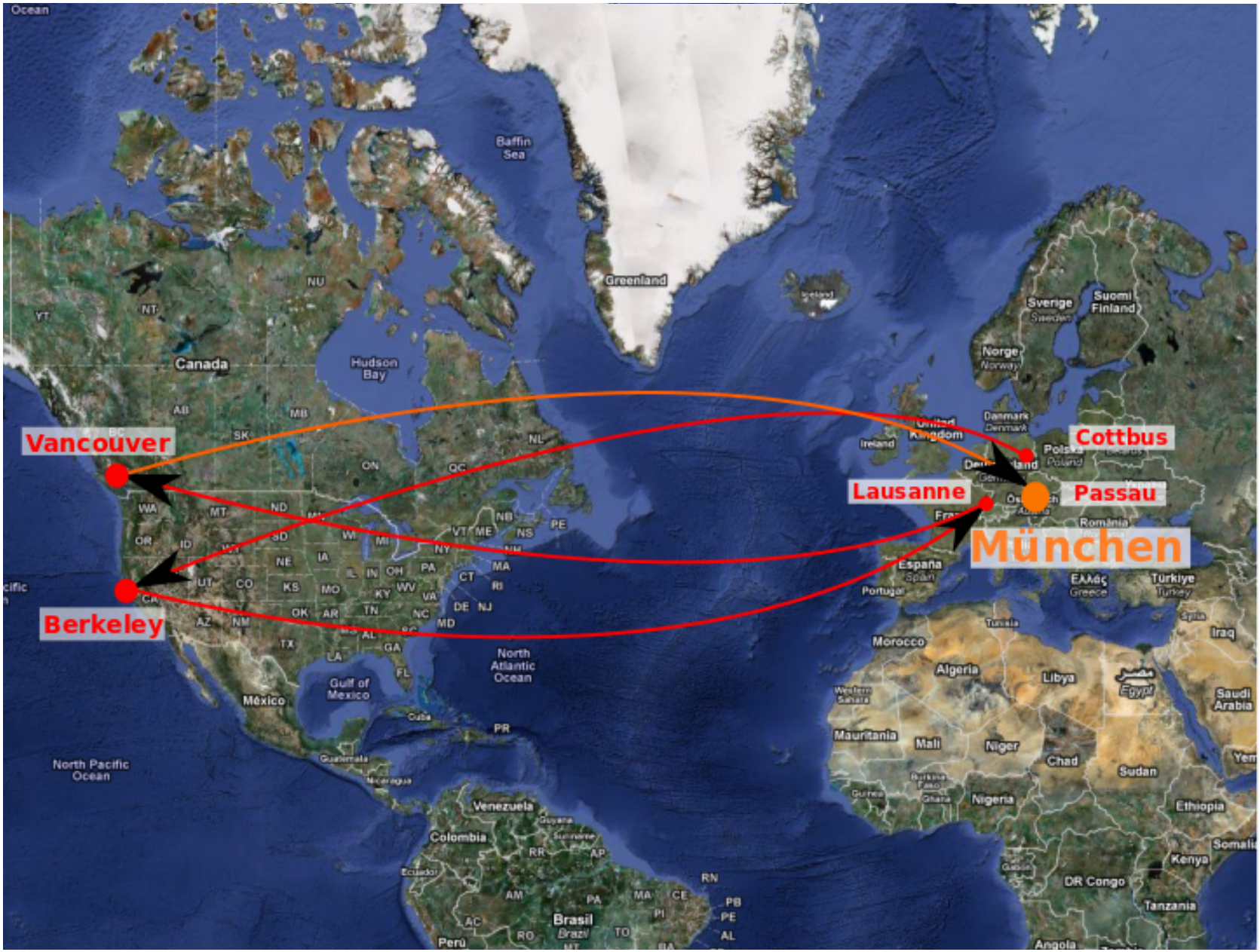
Dirk Beyer
LMU Munich, Germany

SEFM 2022 Summer School, Sept. 21–24, 2022

About the Speaker: Short CV

- 1998–2002 PhD on Verification of Hybrid Systems
- 2003 PostDoc at University of California, Berkeley
- 2004 PostDoc at EPFL Lausanne
- 2006 Assistant Professor at Simon Fraser University
- 2009 Professor at University of Passau
- 2016 Professor at LMU Munich

Trace

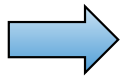

Vancouver
Berkeley
Cottbus
Lausanne
Passau
München
Canada
Greenland
Baffin Sea
Hudson Bay
North Atlantic Ocean
North Pacific Ocean
Gulf of Mexico
México
Venezuela
Colombia
Brasil
Perú
Sverige
Suomi Finland
Norge
Ireland
España
Portugal
Ελλάς
Greece
Türkiye
Turkey
Morocco
Algeria
Libya
Egypt
Mauritania
Mali
Niger
Chad
Sudan
Nigeria
Ethiopia
Kenya
DR Congo
Tanzania
Saudi Arabia
Iraq
Romania

Software Verification

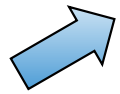
C Program

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

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



Verification Tool



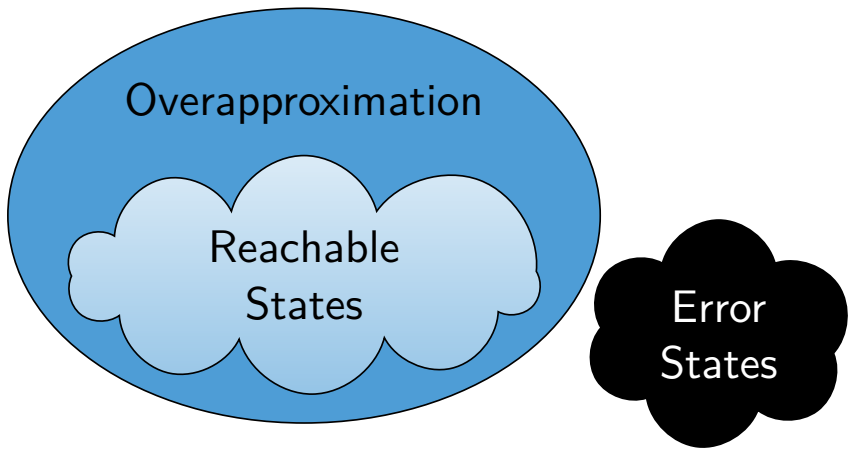
TRUE

i.e., specification is satisfied

FALSE

i.e., bug found

General method:
Create an overapproximation of the program states / compute program invariants



CPAchecker History

- 2002: BLAST with lazy abstraction refinement [9, 28]
- 2003: Multi-threading support [26]
- 2004: Test-case generation, interpolation, spec. lang. [9, 1]
- 2005: Memory safety, predicated lattices [25, 8]
- 2006: Lazy shape analysis [11]
- Maintenance and extensions became extremely difficult because of design choices that were not easy to revert
- 2007: Configurable program analysis [12, 13],
  CPAchecker was started
  as complete reimplementation from scratch [14]

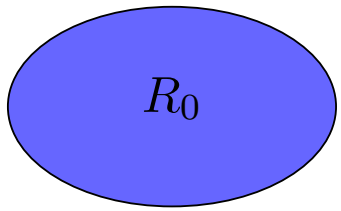
CPAchecker History (2)

- 2009: Large-block encoding [2, FMCAD '09]
- 2010: Adjustable-block encoding [15, FMCAD '10]
- 2012: Conditional model checking [10, FSE '12],
  PredAbs vs. Impact [21, FMCAD '12]
- 2013: Explicit-state MC [17, FASE '13],
  BDDs [20, STTT '14],
  precision reuse [18, FSE '13]
- ...

Software Verification by Model Checking

[24, 32, Clarke/Emerson, Queille/Sifakis 1981]

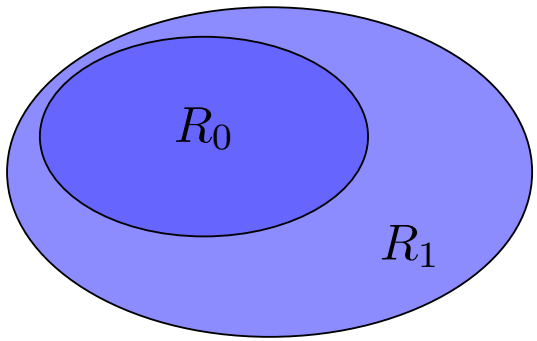
Iterative fixpoint (forward) post computation



Software Verification by Model Checking

[24, 32, Clarke/Emerson, Queille/Sifakis 1981]

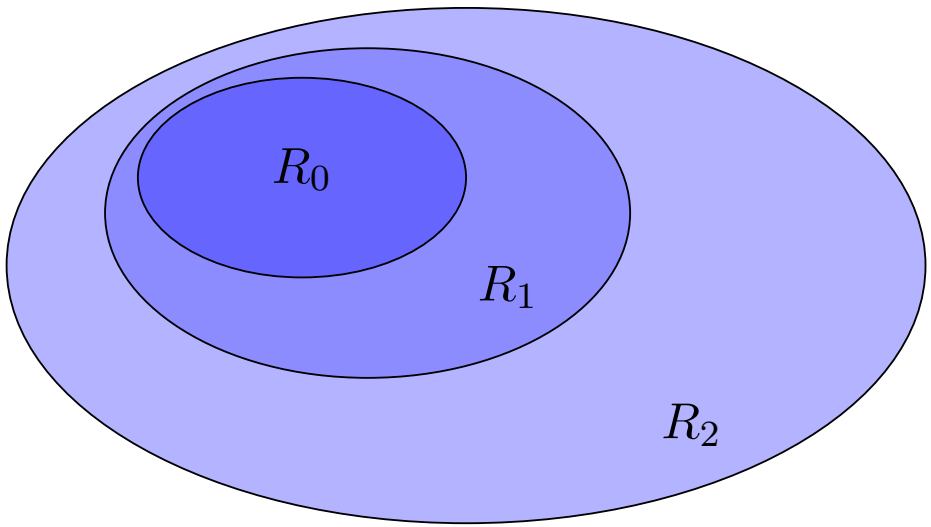
Iterative fixpoint (forward) post computation



Software Verification by Model Checking

[24, 32, Clarke/Emerson, Queille/Sifakis 1981]

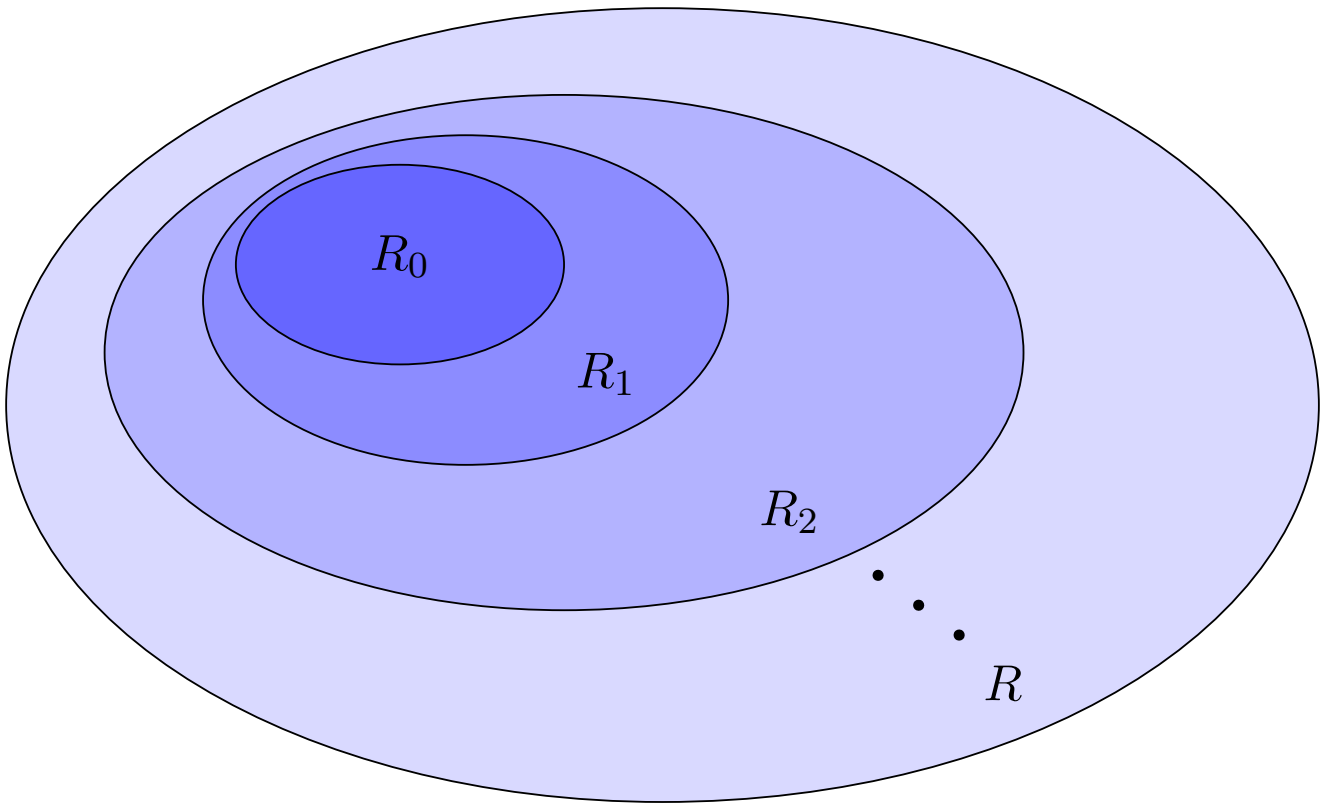
Iterative fixpoint (forward) post computation



Software Verification by Model Checking

[24, 32, Clarke/Emerson, Queille/Sifakis 1981]

Iterative fixpoint (forward) post computation



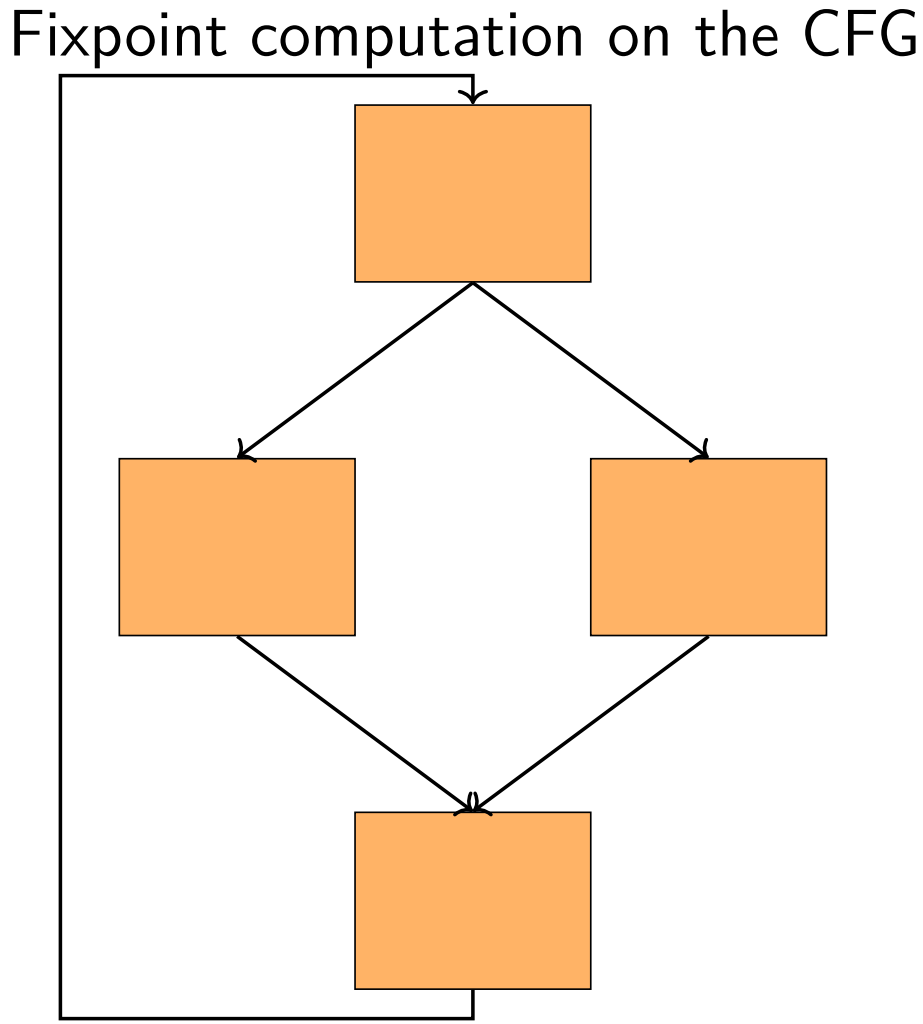
Software Model Checking

Reached, *Frontier* := $\{e_0\}$
while *Frontier* $\neq \emptyset$ **do**
  remove e from *Frontier*
  for all e' $\in$ **post**(e) **do**

    if $\neg$**stop**$(e', \mathit{Reached})$ **then**
      add e' to *Reached*, *Frontier*
return *Reached*

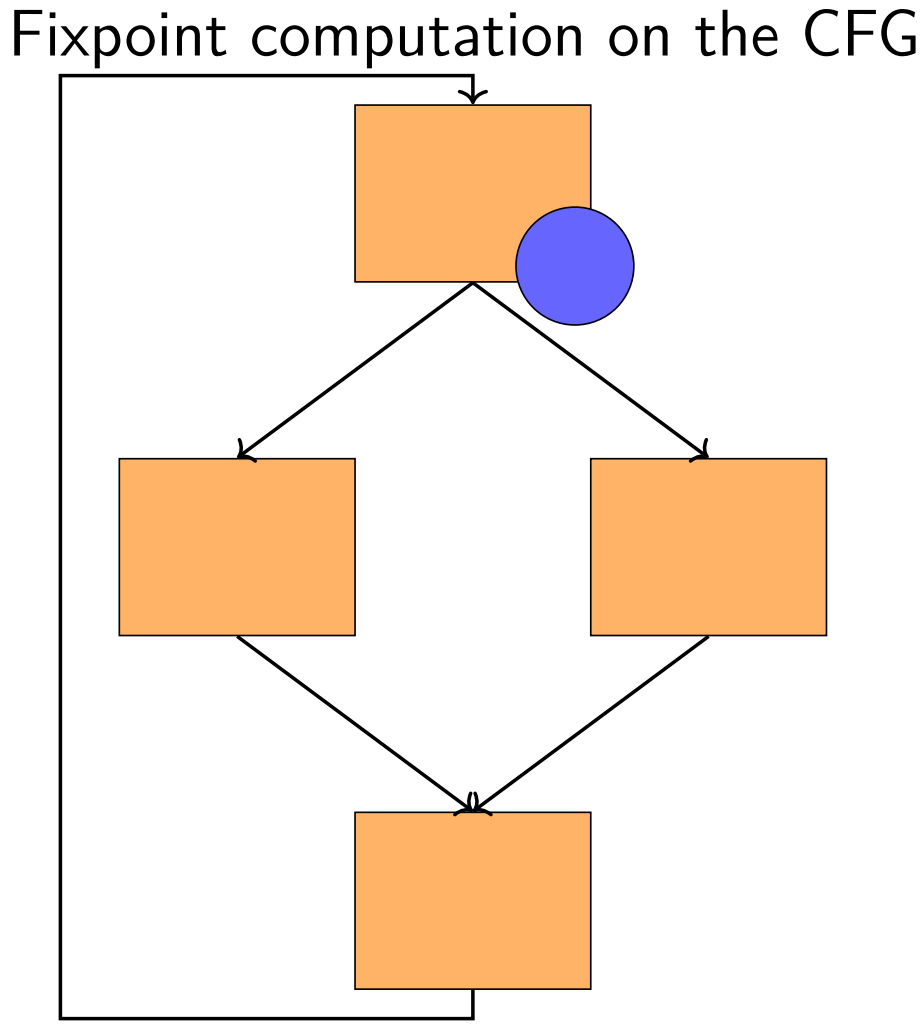
Software Verification by Data-Flow Analysis

[30, Kildall 1973]



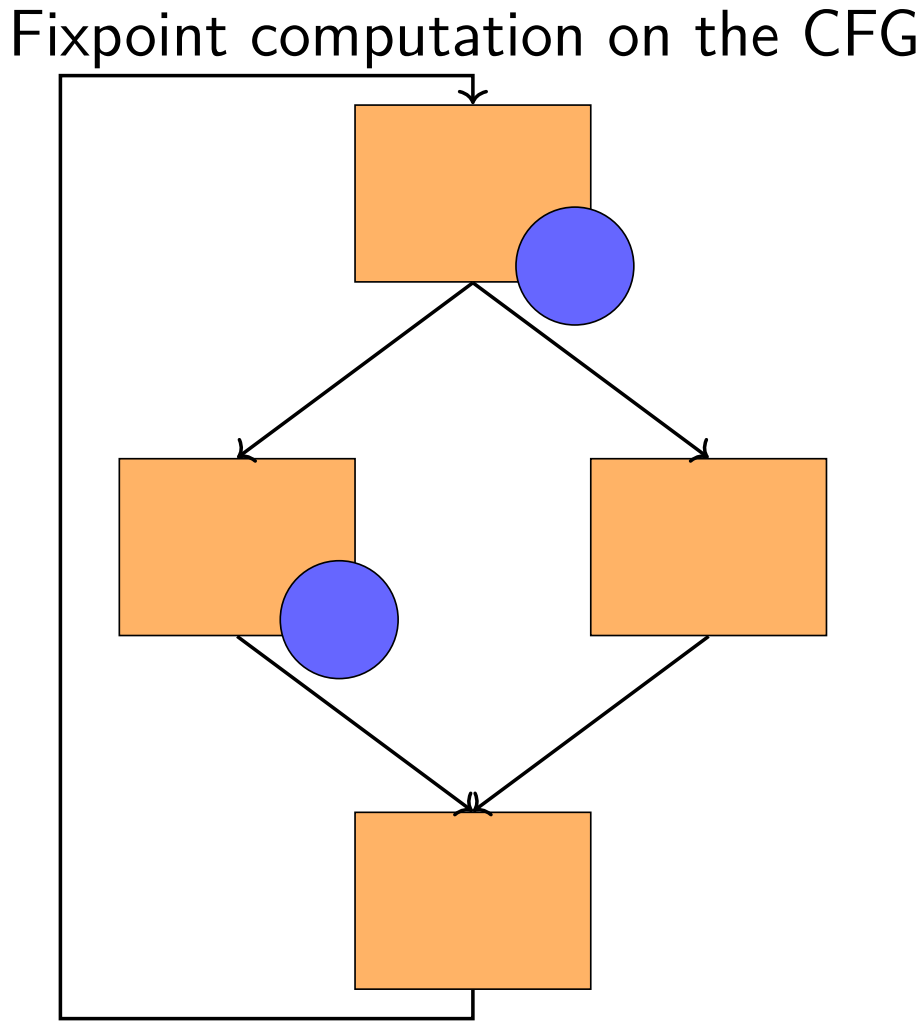
Software Verification by Data-Flow Analysis

[30, Kildall 1973]



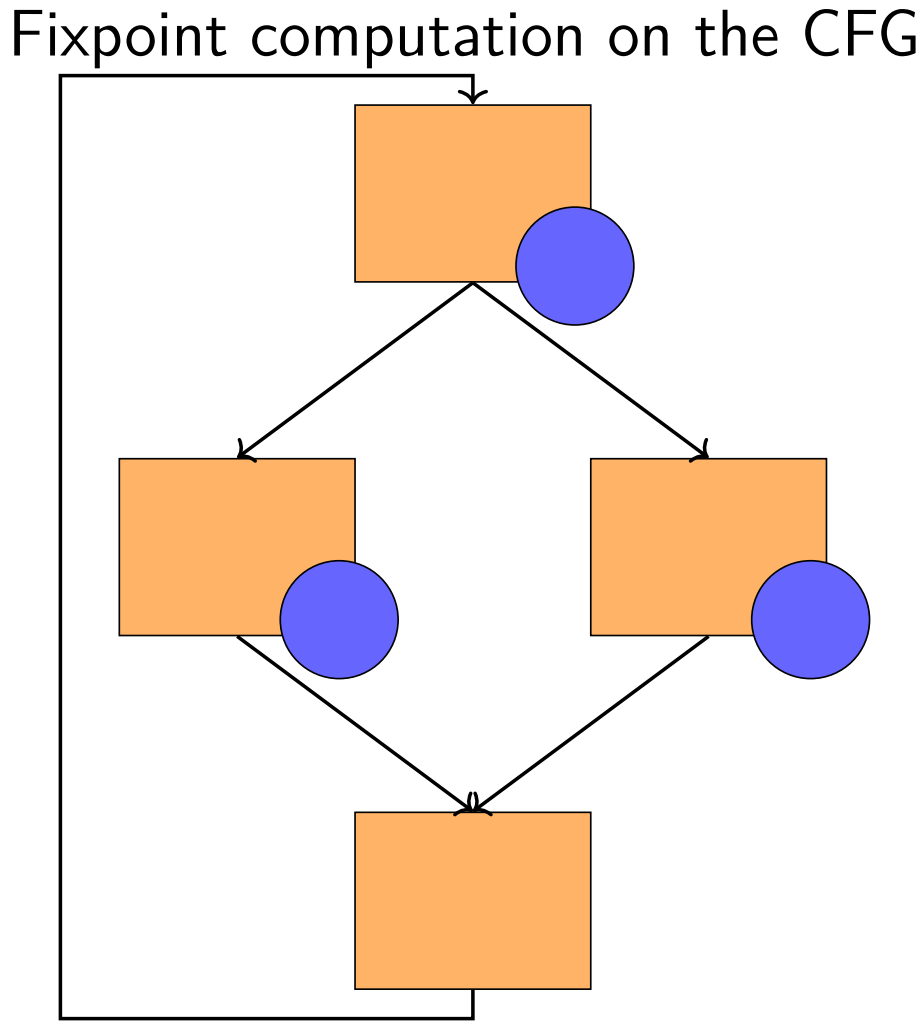
Software Verification by Data-Flow Analysis

[30, Kildall 1973]



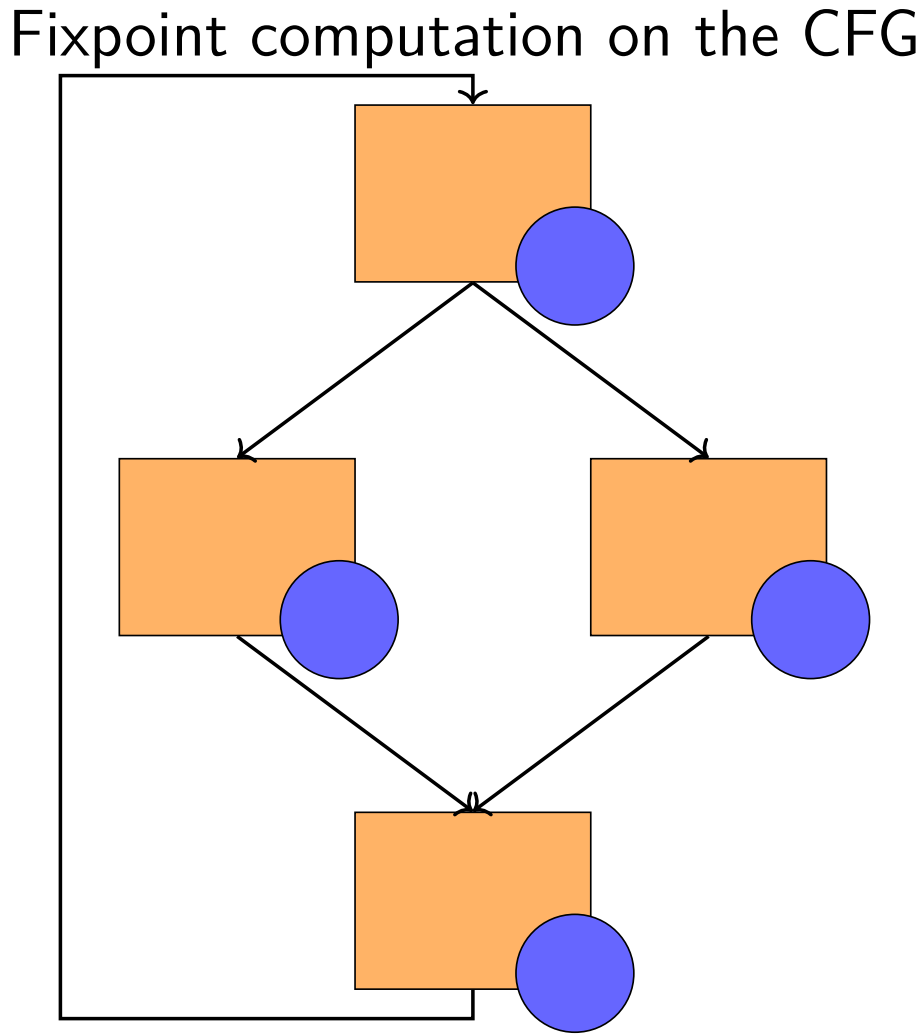
Software Verification by Data-Flow Analysis

[30, Kildall 1973]



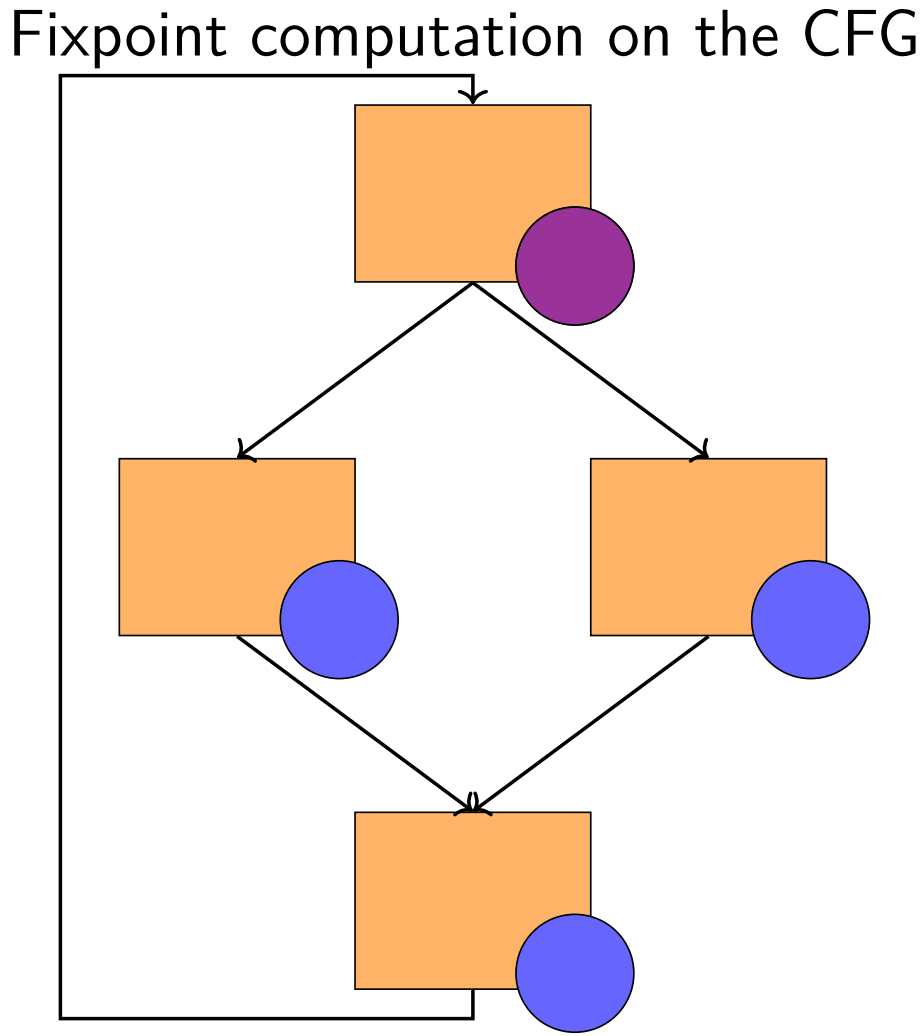
Software Verification by Data-Flow Analysis

[30, Kildall 1973]



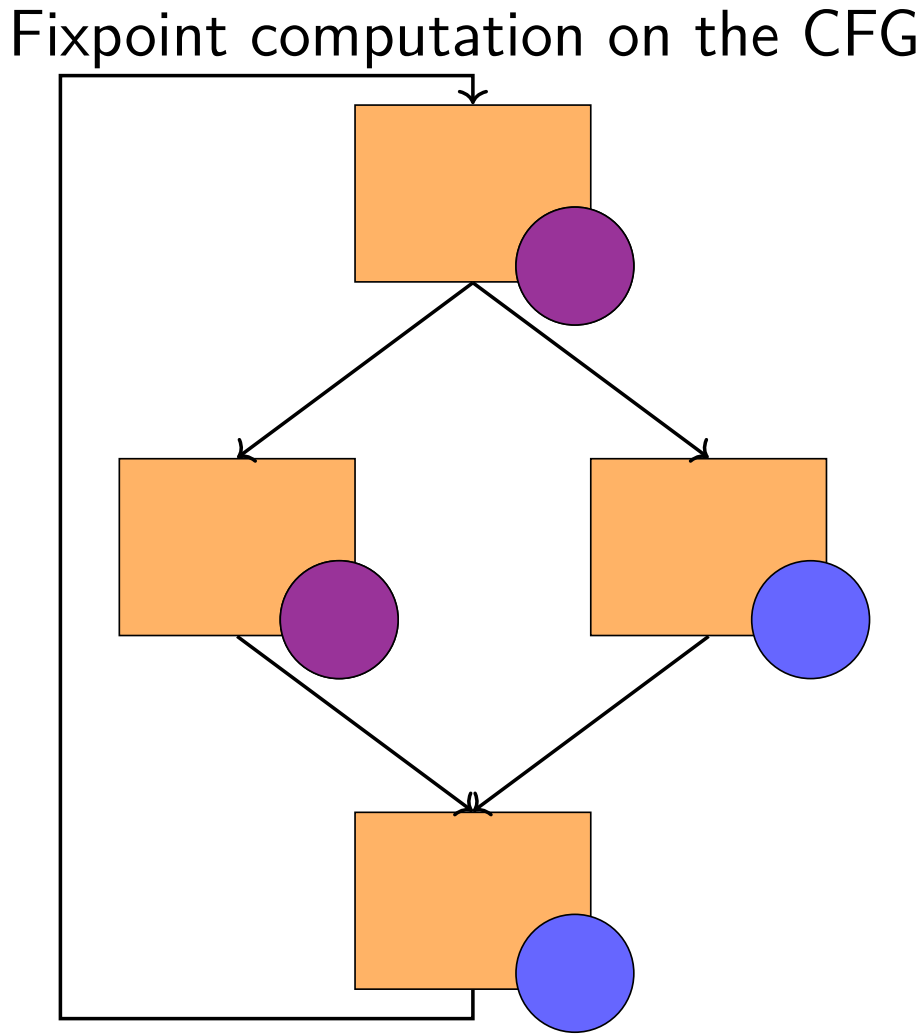
Software Verification by Data-Flow Analysis

[30, Kildall 1973]



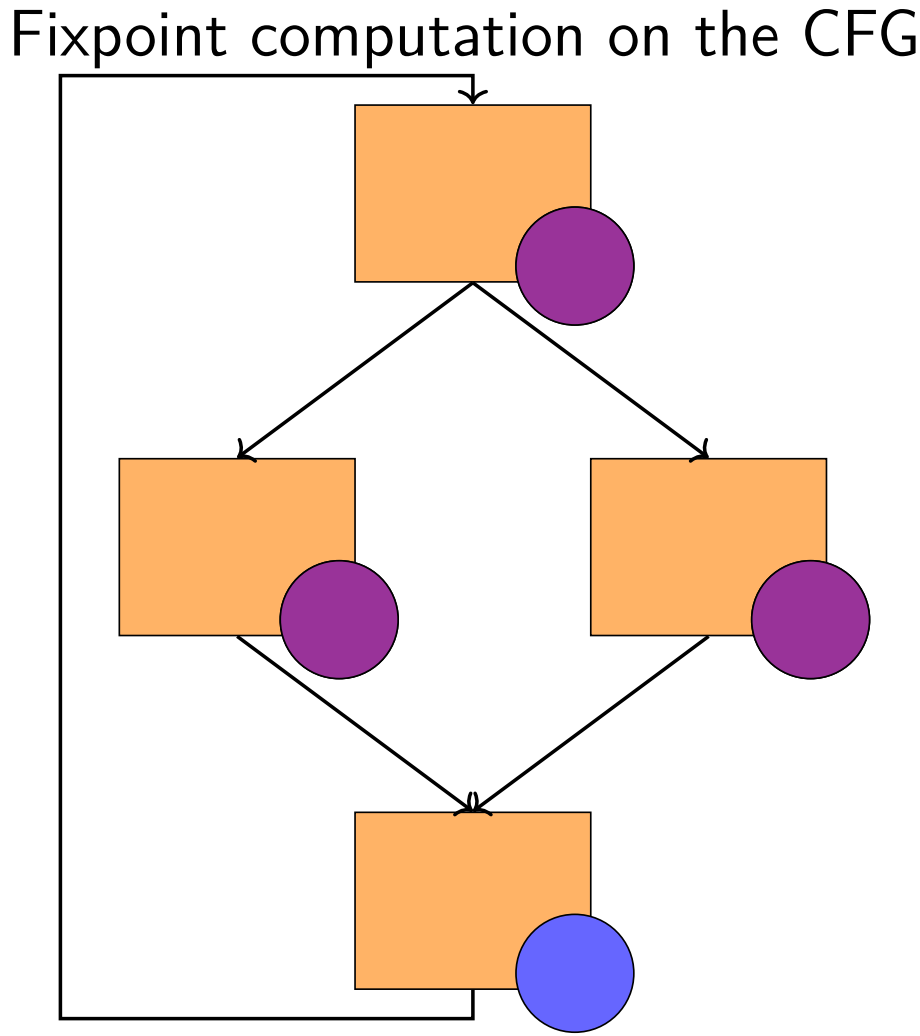
Software Verification by Data-Flow Analysis

[30, Kildall 1973]



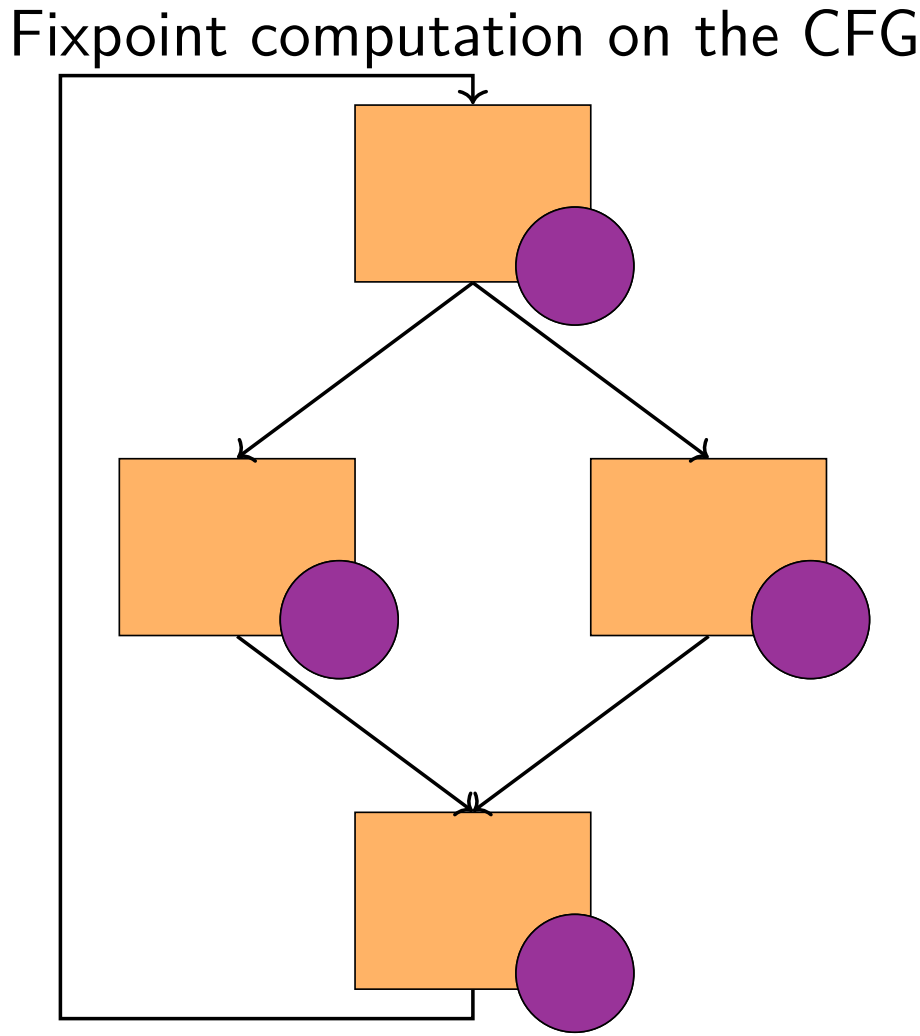
Software Verification by Data-Flow Analysis

[30, Kildall 1973]



Software Verification by Data-Flow Analysis

[30, Kildall 1973]



Software Model Checking

Reached, *Frontier* := $\{e_0\}$
while *Frontier* $\neq \emptyset$ **do**
  remove e from *Frontier*
  for all e' $\in$ **post**(e) **do**

    if $\neg$**stop**$(e', \mathit{Reached})$ **then**
      add e' to *Reached*, *Frontier*
return *Reached*

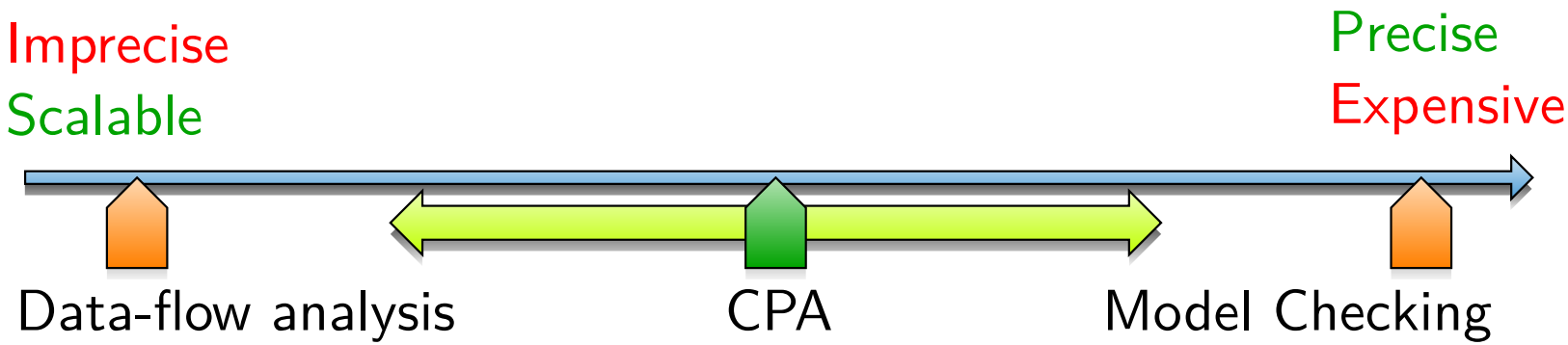
Configurable Program Analysis

Reached, *Frontier* := $\{e_0\}$
while *Frontier* $\neq \emptyset$ **do**
  remove e from *Frontier*
  for all e' $\in$ **post**(e) **do**
    for all 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**$(e', \mathit{Reached})$ **then**
      add e' to *Reached*, *Frontier*
return *Reached*

Configurable Program Analysis

[12, Beyer/Henzinger/Theoduloz CAV '07]

- Better combination of abstractions
  $\rightarrow$ Configurable Program Analysis



Unified framework that enables intermediate algorithms

Dynamic Precision Adjustment

Lazy abstraction refinement: [27, Henzinger/Jhala/Majumdar/Sutre POPL '02]

- Different predicates per location and per path
- Incremental analysis instead of restart from scratch after refinement

Dynamic Precision Adjustment

Better fine tuning of the precision of abstractions

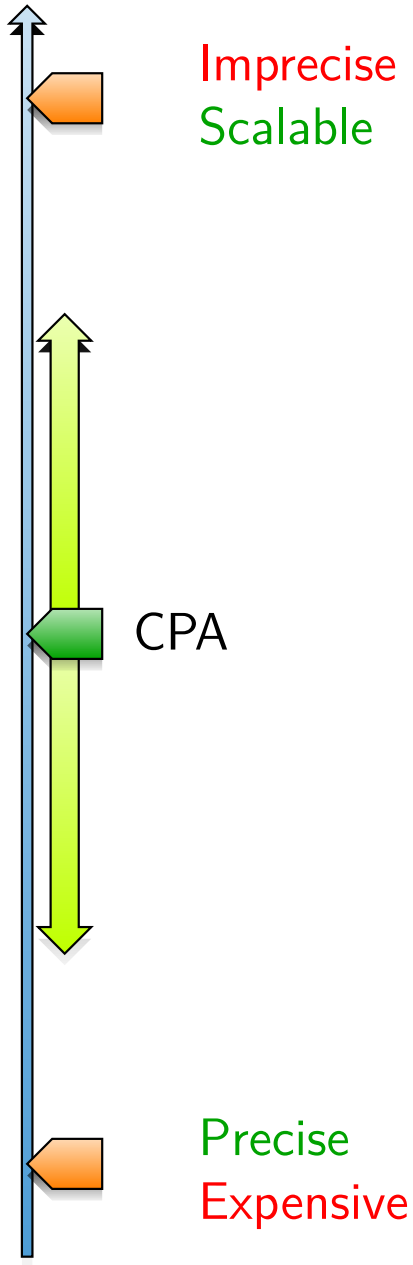
$\rightarrow$ Adjustable Precision

[13, Beyer/Henzinger/Theoduloz ASE'08]

Unified framework enables:

- switch on and off different analysis, and can
- adjust each analysis separately

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

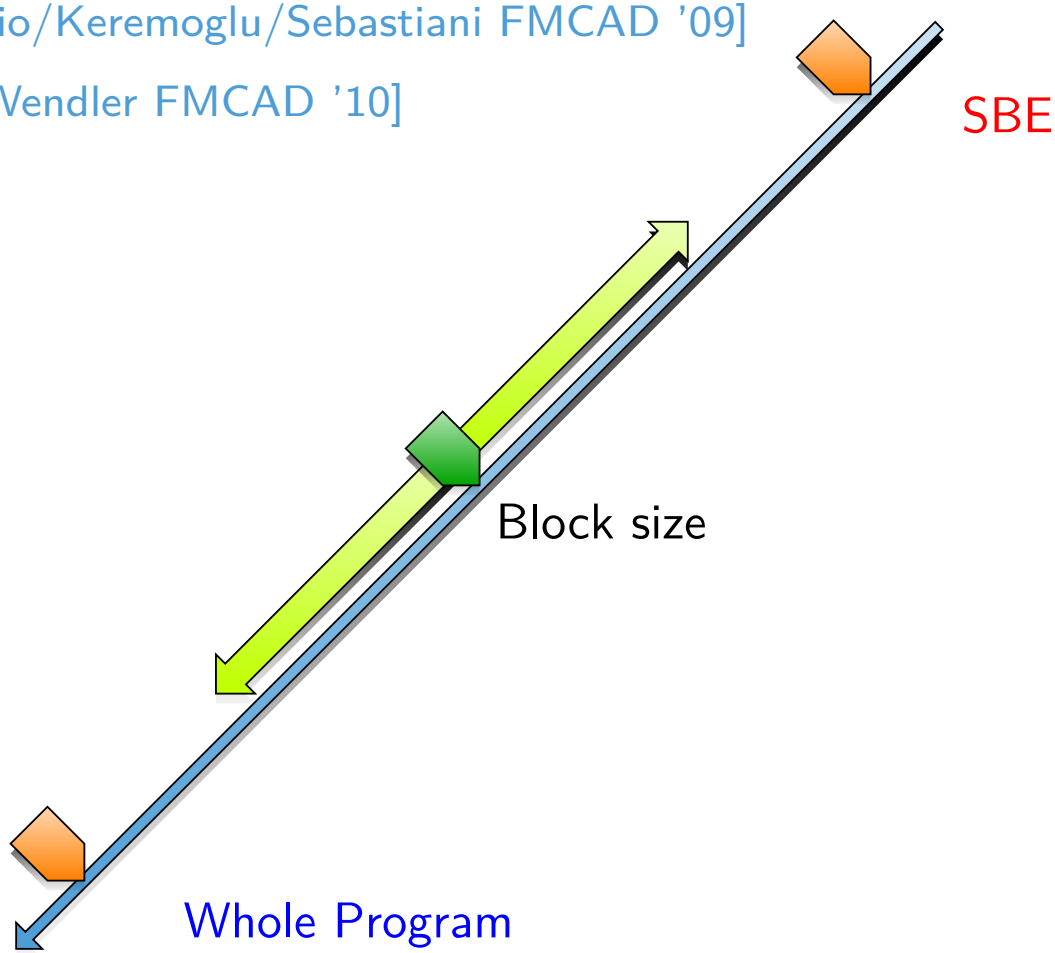


Adjustable Block-Encoding

- Handle loop-free blocks of statements at once
- Abstract only between blocks
  (less abstractions, less refinements)

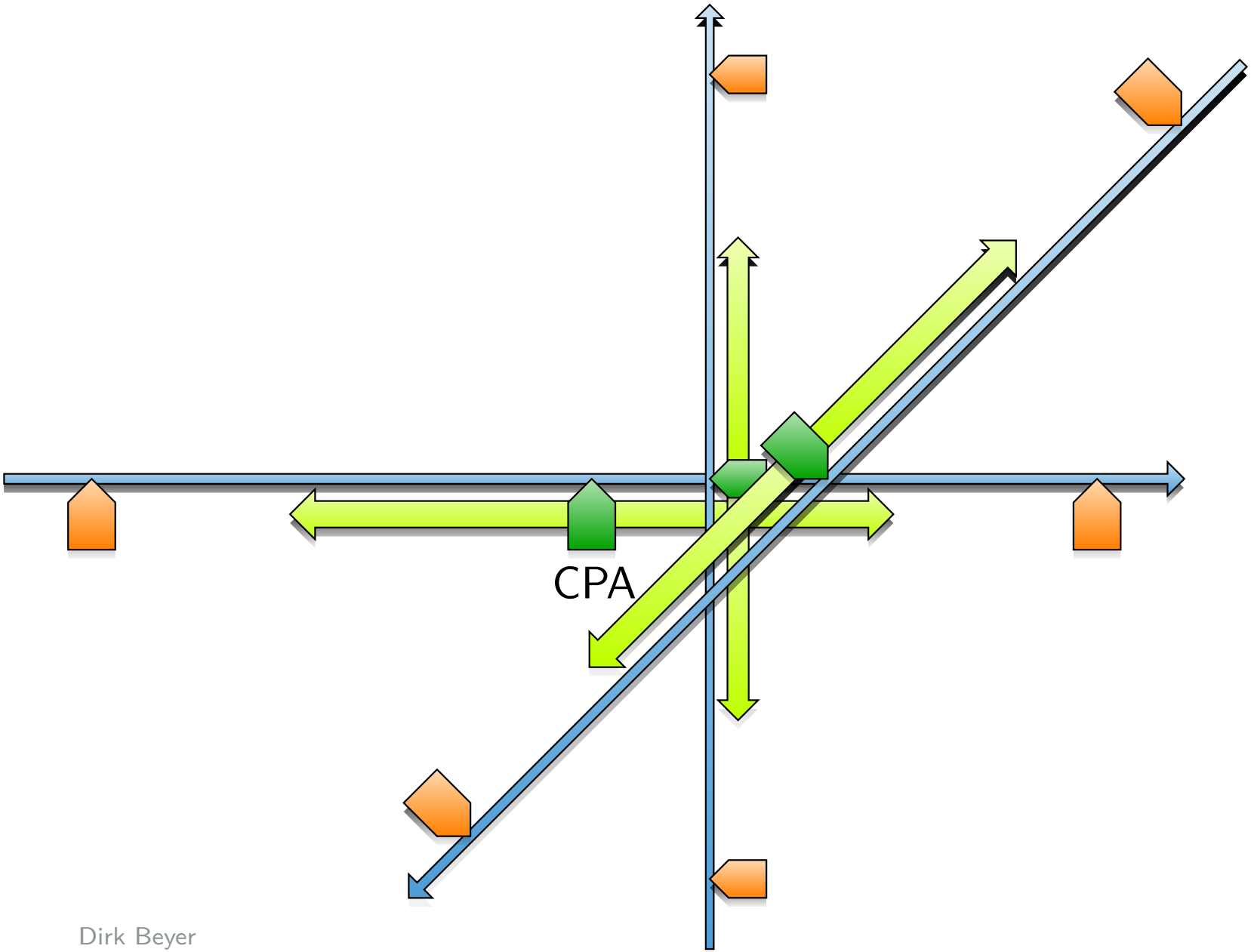
  [2, Beyer/Cimatti/Griggio/Keremoglu/Sebastiani FMCAD '09]

  [15, Beyer/Keremoglu/Wendler FMCAD '10]



CPAchecker

[14, Beyer/Keremoglu CAV '11]



CPA – Summary

- Unification of several approaches
  $\rightarrow$ reduced to their essential properties
- Allow experimentation with new configurations
  that we could never think of
- Flexible implementation CPAchecker

CPAchecker

- Framework for Software Verification — current status
  - Written in Java
  - Open Source: Apache 2.0 License
  - ~80 contributors so far from 15 universities/institutions
  - 470.000 lines of code
    (300.000 without blank lines and comments)
  - Started 2007

https://cpachecker.sosy-lab.org

CPAchecker: Features

- Input language C (experimental: Java)
- Web frontend available:
  https://vcloud.sosy-lab.org/cpachecker/webclient/run
- Counterexample output with graphs
- Benchmarking infrastructure available
  (with large cluster of machines)
- Cross-platform: Linux, Mac, Windows

CPAchecker: Achievements

- Among world's best software verifiers:
  https://sv-comp.sosy-lab.org/2021/results/
- Continuous success in competition since 2012
  (66 medals: 19x gold, 22x silver, 25x bronze)
- Awarded Gödel medal
  by Kurt Gödel Society

- Used for Linux driver verification
  with dozens of real bugs found and fixed in Linux [29, 19]

CPAchecker: Concepts

- Included Concepts:
  - CEGAR [22] Interpolation [17, 7]
  - Configurable Program Analysis [12, 13]
  - Adjustable-block encoding [15]
  - Conditional model checking [10]
  - Verification witnesses [5, 4]
  - Various abstract domains: predicates, intervals, BDDs, octagons, explicit values
- Available analyses approaches:
  - Predicate abstraction [2, 15, 13, 18]
  - Impact algorithm [31, 21, 7]
  - Bounded model checking [23, 7]
  - k-Induction [6, 7]
  - IC3/Property-directed reachability [3]
  - Explicit-state model checking [17]
  - Interpolation-based model checking [16]

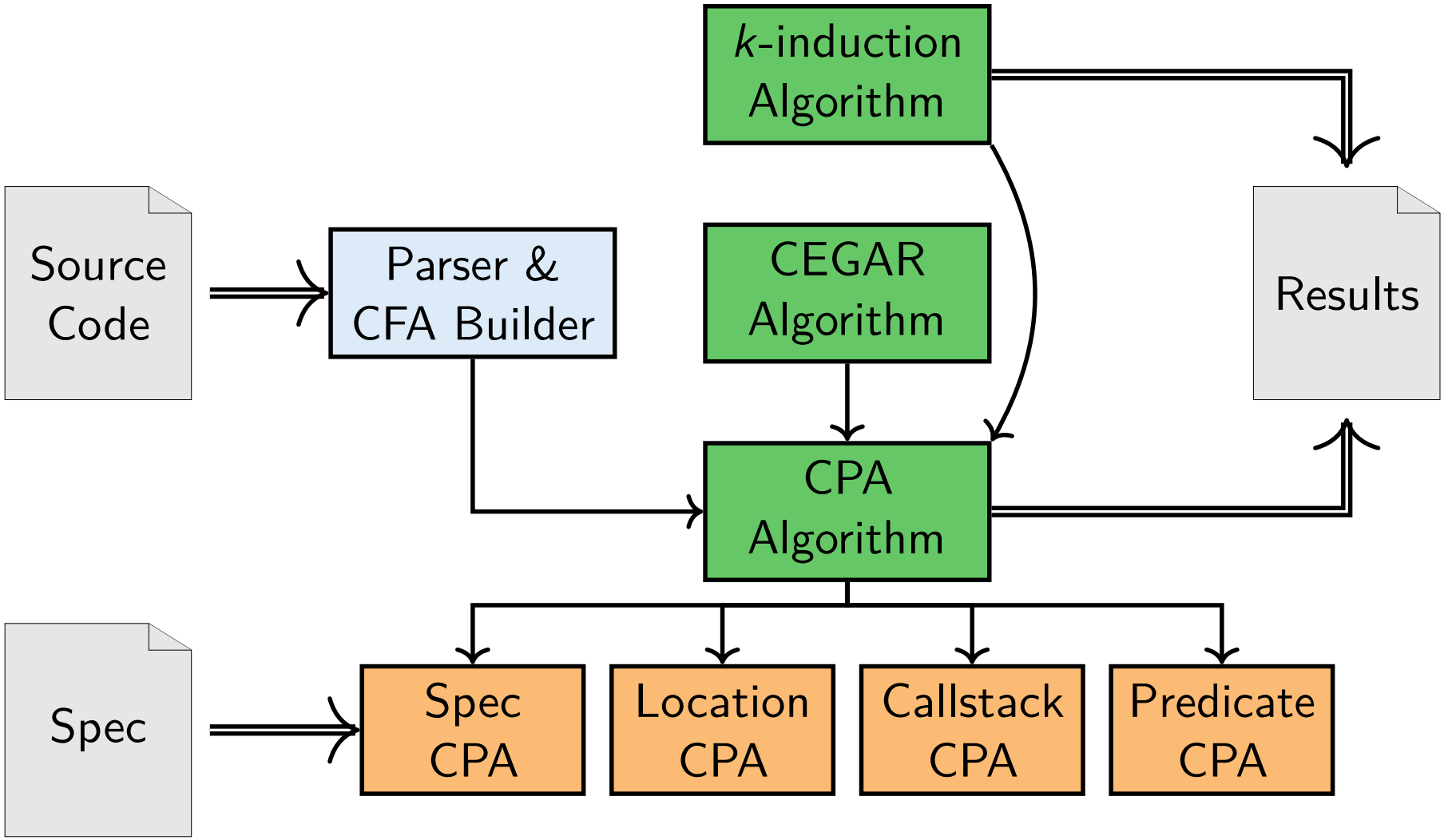
CPAchecker: Concepts

- 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, memory graphs, 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: Architecture



Try CPAchecker

- Online at SoSy-Lab VerifierCloud:
  https://vcloud.sosy-lab.org/cpachecker/webclient/run
- Download for Linux/Windows:
  https://cpachecker.sosy-lab.org
  - Run `scripts/cpa.sh` | `scripts\cpa.bat`
  - `-default <FILE>`
  - Windows/Mac need to disable bitprecise analysis:
    ```
    -predicateAnalysis-linear
    -setprop solver.solver=smtinterpol
    -setprop analysis.checkCounterexamples=false
    ```
- Open graphical report in browser: `output/*.html`
- Open `.dot` files with dotty / xdot (www.graphviz.org/)

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: LTL-based formulas
  (e.g., reachability of call to `reach_error()`)
- General: Monitor automata

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

The CPA framework is flexible:

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

Conclusions

- Flexible framework
- Open source; large group of contributors
- Most state-of-the-art approaches integrated
- Decent performance (see competitions)
- Many papers decribing its concepts

References I

[1] Beyer, D., Chlipala, A.J., Henzinger, T.A., Jhala, R., Majumdar, R.: Generating tests from counterexamples. In: Proc. ICSE. pp. 326–335. IEEE (2004). https://doi.org/10.1109/ICSE.2004.1317455

[2] Beyer, D., Cimatti, A., Griggio, A., Keremoglu, M.E., Sebastiani, R.: Software model checking via large-block encoding. In: Proc. FMCAD. pp. 25–32. IEEE (2009). https://doi.org/10.1109/FMCAD.2009.5351147

[3] Beyer, D., Dangl, M.: Software verification with PDR: An implementation of the state of the art. In: Proc. TACAS (1). pp. 3–21. LNCS 12078, Springer (2020). https://doi.org/10.1007/978-3-030-45190-5_1

[4] Beyer, D., Dangl, M., Dietsch, D., Heizmann, M.: Correctness witnesses: Exchanging verification results between verifiers. In: Proc. FSE. pp. 326–337. ACM (2016). https://doi.org/10.1145/2950290.2950351

[5] Beyer, D., Dangl, M., Dietsch, D., Heizmann, M., Stahlbauer, A.: Witness validation and stepwise testification across software verifiers. In: Proc. FSE. pp. 721–733. ACM (2015). https://doi.org/10.1145/2786805.2786867

[6] Beyer, D., Dangl, M., Wendler, P.: Boosting k-induction with continuously-refined invariants. In: Proc. CAV. pp. 622–640. LNCS 9206, Springer (2015). https://doi.org/10.1007/978-3-319-21690-4_42

References II

[7] Beyer, D., Dangl, M., Wendler, P.: A unifying view on SMT-based software verification. J. Autom. Reasoning **60**(3), 299–335 (2018). https://doi.org/10.1007/s10817-017-9432-6

[8] Beyer, D., Henzinger, T.A., Jhala, R., Majumdar, R.: Checking memory safety with Blast. In: Proc. FASE. pp. 2–18. LNCS 3442, Springer (2005). https://doi.org/10.1007/978-3-540-31984-9_2

[9] Beyer, D., Henzinger, T.A., Jhala, R., Majumdar, R.: The software model checker Blast. Int. J. Softw. Tools Technol. Transfer **9**(5-6), 505–525 (2007). https://doi.org/10.1007/s10009-007-0044-z

[10] Beyer, D., Henzinger, T.A., Keremoglu, M.E., Wendler, P.: Conditional model checking: A technique to pass information between verifiers. In: Proc. FSE. ACM (2012). https://doi.org/10.1145/2393596.2393664

[11] Beyer, D., Henzinger, T.A., Théoduloz, G.: Lazy shape analysis. In: Proc. CAV. pp. 532–546. LNCS 4144, Springer (2006). https://doi.org/10.1007/11817963_48

[12] Beyer, D., Henzinger, T.A., Théoduloz, G.: Configurable software verification: Concretizing the convergence of model checking and program analysis. In: Proc. CAV. pp. 504–518. LNCS 4590, Springer (2007). https://doi.org/10.1007/978-3-540-73368-3_51

References III

[13] Beyer, D., Henzinger, T.A., Théoduloz, G.: Program analysis with dynamic precision adjustment. In: Proc. ASE. pp. 29–38. IEEE (2008). https://doi.org/10.1109/ASE.2008.13

[14] Beyer, D., Keremoglu, M.E.: CPAchecker: A tool for configurable software verification. In: Proc. CAV. pp. 184–190. LNCS 6806, Springer (2011). https://doi.org/10.1007/978-3-642-22110-1_16

[15] Beyer, D., Keremoglu, M.E., Wendler, P.: Predicate abstraction with adjustable-block encoding. In: Proc. FMCAD. pp. 189–197. FMCAD (2010), https://www.sosy-lab.org/research/pub/2010-FMCAD.Predicate_Abstraction_with_Adjustable-Block_Encoding.pdf

[16] Beyer, D., Lee, N.Z., Wendler, P.: Interpolation and SAT-based model checking revisited: Adoption to software verification. arXiv/CoRR **2208**(05046) (July 2022). https://doi.org/10.48550/arXiv.2208.05046

[17] Beyer, D., Löwe, S.: Explicit-state software model checking based on CEGAR and interpolation. In: Proc. FASE. pp. 146–162. LNCS 7793, Springer (2013). https://doi.org/10.1007/978-3-642-37057-1_11

[18] Beyer, D., Löwe, S., Novikov, E., Stahlbauer, A., Wendler, P.: Precision reuse for efficient regression verification. In: Proc. FSE. pp. 389–399. ACM (2013). https://doi.org/10.1145/2491411.2491429

References IV

[19] Beyer, D., Petrenko, A.K.: Linux driver verification. In: Proc. ISoLA. pp. 1–6. LNCS 7610, Springer (2012). https://doi.org/10.1007/978-3-642-34032-1_1

[20] Beyer, D., Stahlbauer, A.: BDD-based software verification: Applications to event-condition-action systems. Int. J. Softw. Tools Technol. Transfer **16**(5), 507–518 (2014). https://doi.org/10.1007/s10009-014-0334-1

[21] Beyer, D., Wendler, P.: Algorithms for software model checking: Predicate abstraction vs. IMPACT. In: Proc. FMCAD. pp. 106–113. FMCAD (2012)

[22] Clarke, E.M., Grumberg, O., Jha, S., Lu, Y., Veith, H.: Counterexample-guided abstraction refinement for symbolic model checking. J. ACM **50**(5), 752–794 (2003). https://doi.org/10.1145/876638.876643

[23] Clarke, E.M., Kröning, D., Lerda, F.: A tool for checking ANSI-C programs. In: Proc. TACAS. pp. 168–176. LNCS 2988, Springer (2004). https://doi.org/10.1007/978-3-540-24730-2_15

[24] Clarke, E.M., Emerson, E.A.: Design and synthesis of synchronization skeletons using branching-time temporal logic. In: Proc. Logic of Programs 1981. pp. 52–71. LNCS 131, Springer (1982). https://doi.org/10.1007/BFb0025774

[25] Fischer, J., Jhala, R., Majumdar, R.: Joining data flow with predicates. In: Proc. FSE. pp. 227–236. ACM (2005). https://doi.org/10.1145/1081706.1081742

References V

[26] Henzinger, T.A., Jhala, R., Majumdar, R., Qadeer, S.: Thread-modular abstraction refinement. In: Proc. CAV. pp. 262–274. LNCS 2725, Springer (2003). https://doi.org/10.1007/978-3-540-45069-6_27

[27] Henzinger, T.A., Jhala, R., Majumdar, R., Sutre, G.: Lazy abstraction. In: Proc. POPL. pp. 58–70. ACM (2002). https://doi.org/10.1145/503272.503279

[28] Henzinger, T.A., Jhala, R., Majumdar, R., Sutre, G.: Software verification with Blast. In: Proc. SPIN, pp. 235–239. LNCS 2648, Springer (2003). https://doi.org/10.1007/3-540-44829-2_17

[29] Khoroshilov, A.V., Mutilin, V.S., Petrenko, A.K., Zakharov, V.: Establishing Linux driver verification process. In: Proc. Ershov Memorial Conference. pp. 165–176. LNCS 5947, Springer (2009). https://doi.org/10.1007/978-3-642-11486-1_14

[30] Kildall, G.A.: A unified approach to global program optimization. In: Proc. POPL. pp. 194–206. ACM (1973). https://doi.org/10.1145/512927.512945

[31] McMillan, K.L.: Lazy abstraction with interpolants. In: Proc. CAV. pp. 123–136. LNCS 4144, Springer (2006). https://doi.org/10.1007/11817963_14

[32] Queille, J.P., Sifakis, J.: Specification and verification of concurrent systems in CESAR. In: Proc. Symposium on Programming. pp. 337–351. LNCS 137, Springer (1982). https://doi.org/10.1007/3-540-11494-7_22