Domain-Independent Multi-threaded Software Model Checking

Dirk Beyer and **Karlheinz Friedberger**

LMU Munich, Germany

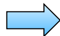
ASE 2018, Montpellier

Software Verification

C Program

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

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

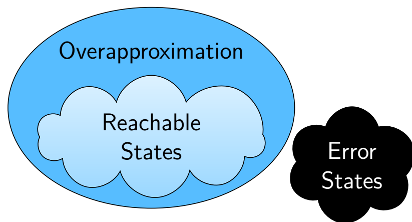


Verification Tool

TRUE
i.e., specification is satisfied

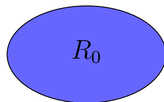
FALSE
i.e., bug found

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



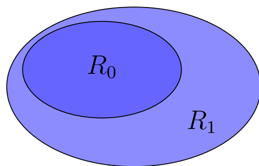
Software Verification by Model Checking
[Clarke/Emerson, Sifakis 1981]

Iterative fixpoint successor computation



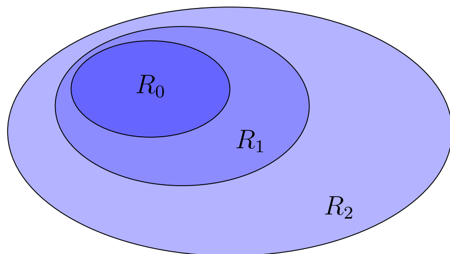
Software Verification by Model Checking
[Clarke/Emerson, Sifakis 1981]

Iterative fixpoint successor computation



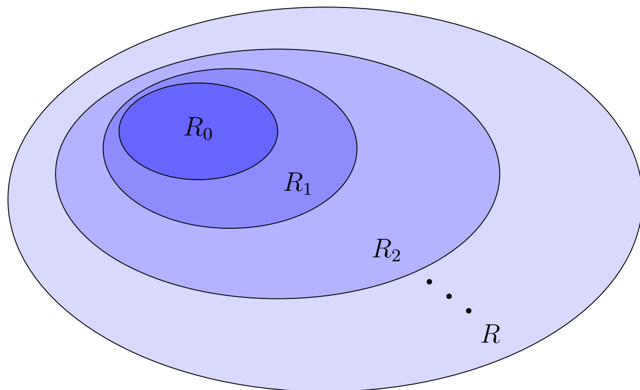
Software Verification by Model Checking
[Clarke/Emerson, Sifakis 1981]

Iterative fixpoint successor computation

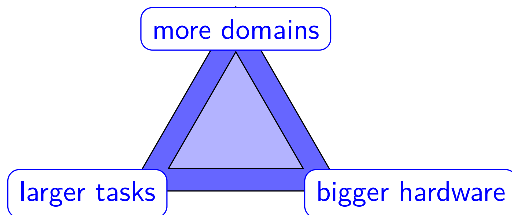


Software Verification by Model Checking
[Clarke/Emerson, Sifakis 1981]

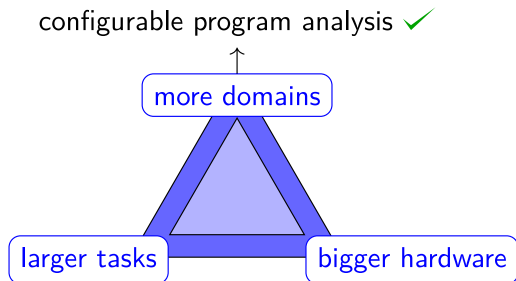
Iterative fixpoint successor computation



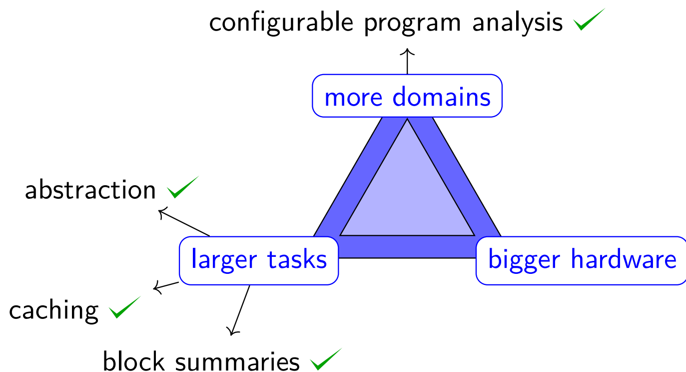
Basic Challenges with Software Verification



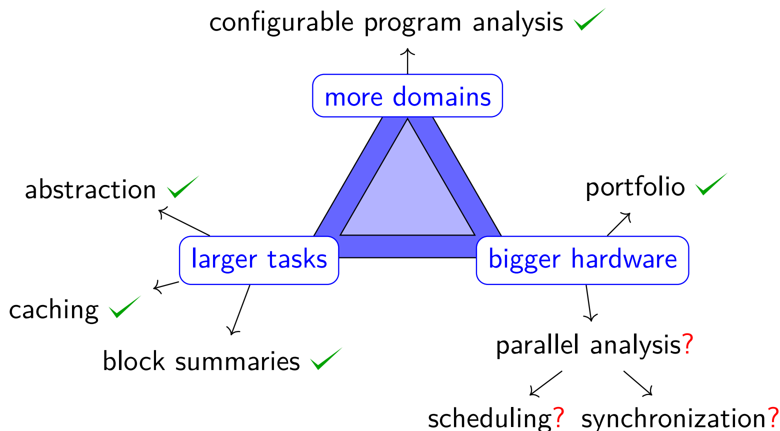
Basic Challenges with Software Verification



Basic Challenges with Software Verification



Basic Challenges with Software Verification



Configurable Program Analysis (CPA)
[Beyer/Henzinger/Théoduloz, 2007]

- CPA algorithm computes a fixed-point based on two sets of abstract states
  - *reached*: already analyzed abstract states
  - *waitlist*: frontier states to be analyzed

Configurable Program Analysis (CPA)

[Beyer/Henzinger/Théoduloz, 2007]

- CPA algorithm computes a fixed-point based on two sets of abstract states
  - *reached*: already analyzed abstract states
  - *waitlist*: frontier states to be analyzed
- Operators defined for specific domain:
  - *transfer*: successor computation
  - *merge*: combination of two abstract states
  - *stop*: coverage of abstract states

Configurable Program Analysis (CPA)

[Beyer/Henzinger/Théoduloz, 2007]

- CPA algorithm computes a fixed-point based on two sets of abstract states
  - *reached*: already analyzed abstract states
  - *waitlist*: frontier states to be analyzed
- Operators defined for specific domain:
  - *transfer*: successor computation
  - *merge*: combination of two abstract states
  - *stop*: coverage of abstract states

✓ Independent of used domain

Configurable Program Analysis (CPA)

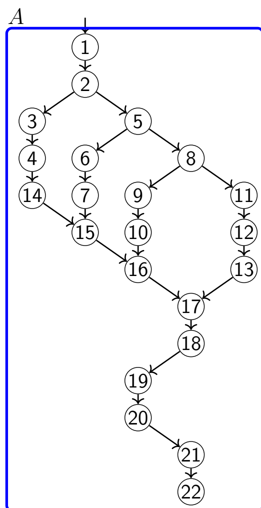
[Beyer/Henzinger/Théoduloz, 2007]

- CPA algorithm computes a fixed-point based on two sets of abstract states
  - *reached*: already analyzed abstract states
  - *waitlist*: frontier states to be analyzed
- Operators defined for specific domain:
  - *transfer*: successor computation
  - *merge*: combination of two abstract states
  - *stop*: coverage of abstract states

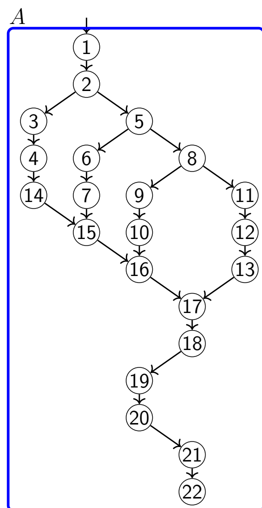
✓ Independent of used domain

X Operators strictly sequential (per analysis!)

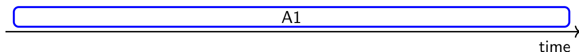
Schematic Example of an Analysis



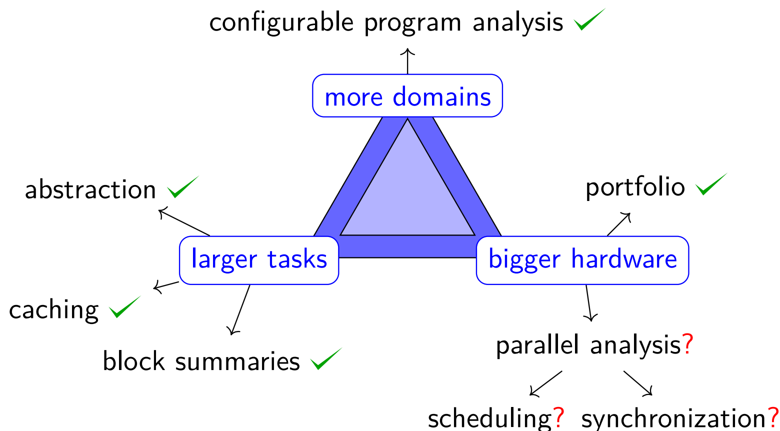
Schematic Example of an Analysis



plain analysis



Basic Challenges with Software Verification



Block Summarization

- Block-abstraction memoization (BAM) defined as CPA [Wonisch/Wehrheim, 2012]
- Split large verification task into smaller problems and solve them separately
- Use CPA algorithm for a domain-specific analysis
- Cache intermediate analysis results

Block Summarization

- Block-abstraction memoization (BAM) defined as CPA [Wonisch/Wehrheim, 2012]
- Split large verification task into smaller problems and solve them separately
- Use CPA algorithm for a domain-specific analysis
- Cache intermediate analysis results

✓ Independent of domain-specific analysis

Block Summarization

- Block-abstraction memoization (BAM) defined as CPA [Wonisch/Wehrheim, 2012]
- Split large verification task into smaller problems and solve them separately
- Use CPA algorithm for a domain-specific analysis
- Cache intermediate analysis results

✓ Independent of domain-specific analysis

x Dependencies between block abstractions

Block Summarization

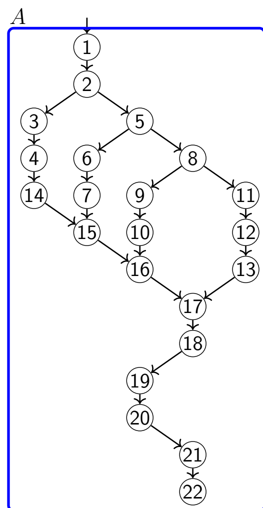
- Block-abstraction memoization (BAM) defined as CPA [Wonisch/Wehrheim, 2012]
- Split large verification task into smaller problems and solve them separately
- Use CPA algorithm for a domain-specific analysis
- Cache intermediate analysis results

✓ Independent of domain-specific analysis

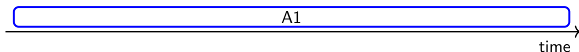
x Dependencies between block abstractions

✓ Configurable block size

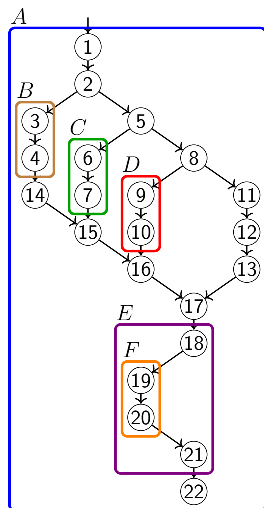
Schematic Example of an Analysis



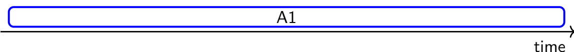
plain analysis



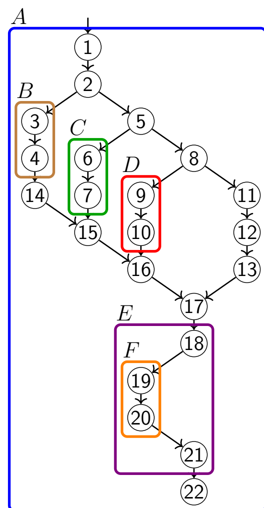
Schematic Example of an Analysis



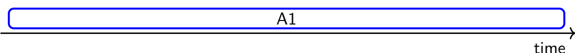
plain analysis



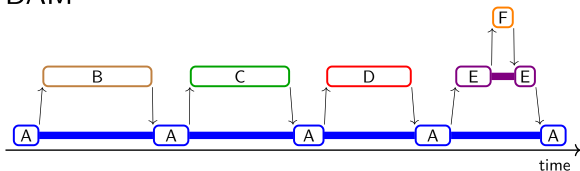
Schematic Example of an Analysis



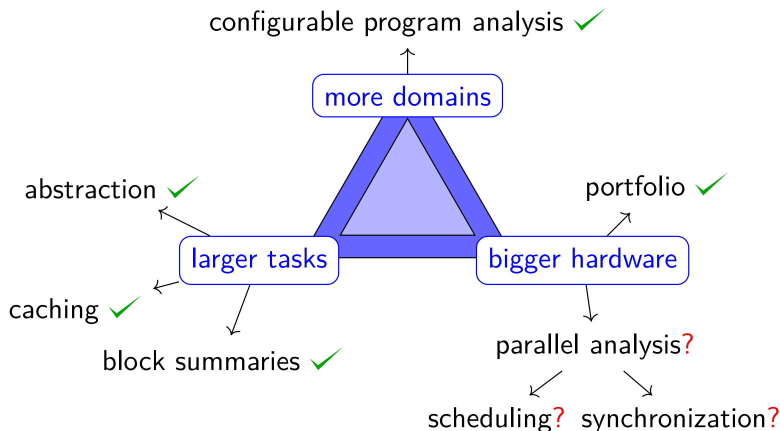
plain analysis



BAM



Basic Challenges with Software Verification



Parallel Block-Abstraction Memoization

Challenges with an efficient parallel algorithm:

- X CPA operators strictly sequential (per analysis!)
- x Dependencies between block abstractions

Parallel Block-Abstraction Memoization

Challenges with an efficient parallel algorithm:

- X CPA operators strictly sequential (per analysis!)
- x Dependencies between block abstractions

Our contribution: Parallel BAM

- Parallel computation of block abstractions (asynchronously)
- Lazy application of computed block abstractions
- Simple dynamic scheduler

Parallel Block-Abstraction Memoization

Challenges with an efficient parallel algorithm:

- X CPA operators strictly sequential (per analysis!)
- x Dependencies between block abstractions

Our contribution: Parallel BAM

- Parallel computation of block abstractions (asynchronously)
- Lazy application of computed block abstractions
- Simple dynamic scheduler

✓ Combines benefits of existing approaches

Parallel Block-Abstraction Memoization

Challenges with an efficient parallel algorithm:

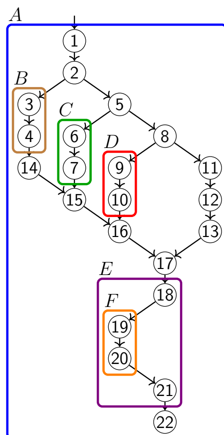
- X CPA operators strictly sequential (per analysis!)
- x Dependencies between block abstractions

Our contribution: Parallel BAM

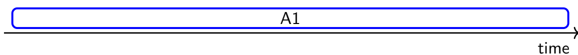
- Parallel computation of block abstractions (asynchronously)
- Lazy application of computed block abstractions
- Simple dynamic scheduler

- ✓ Combines benefits of existing approaches
- ✓ Small synchronization overhead (depends on block size)

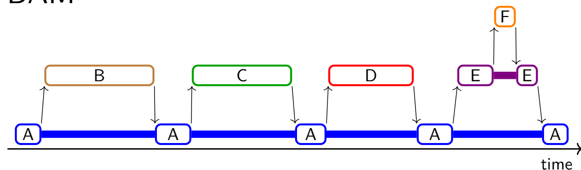
Schematic Example of an Analysis



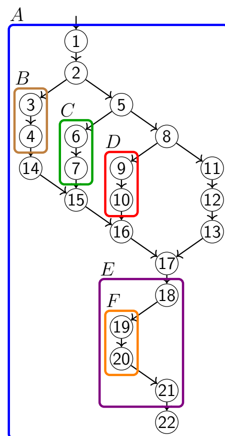
plain analysis



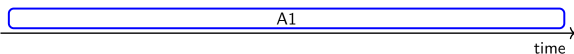
BAM



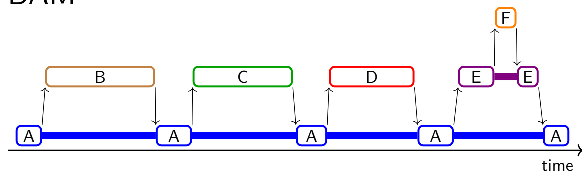
Schematic Example of an Analysis



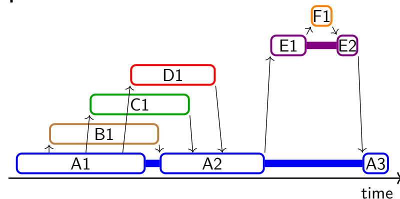
plain analysis



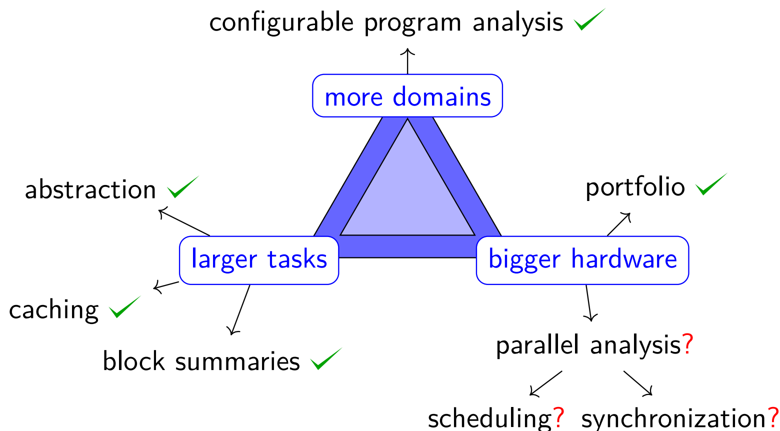
BAM



parallel BAM



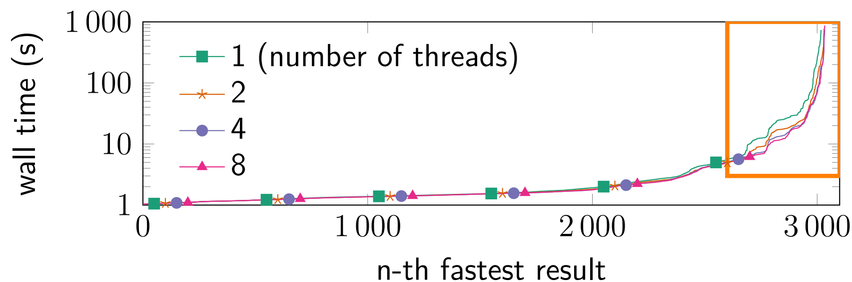
Basic Challenges with Software Verification



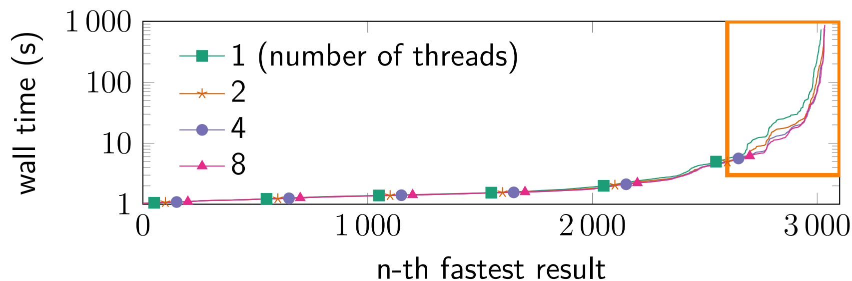
Evaluation

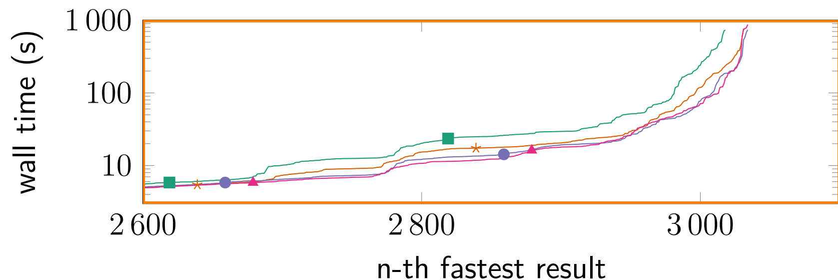
- Configuration
  - CPAchecker `r28809`
  - Explicit Value Domain
- Environment
  - Intel Xeon E3-1230 v5 CPU with 4 physical cores
  - 5400 tasks from SV-COMP benchmark set
- Limitations
  - 15 GB RAM
  - 15 minutes

Evaluation

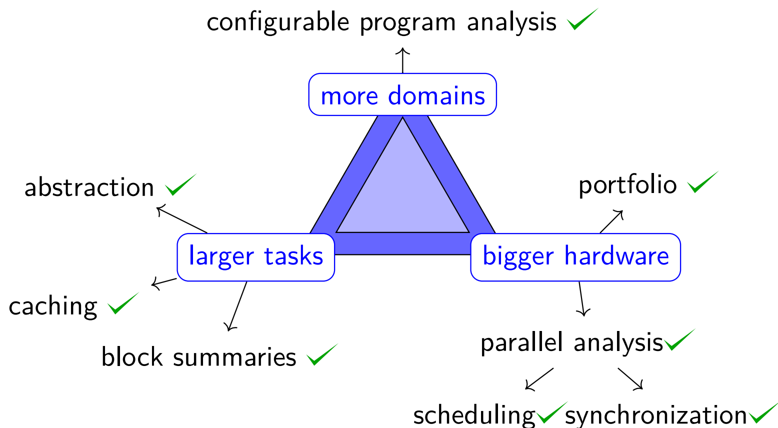


Evaluation





Basic Challenges with Software Verification



Conclusion

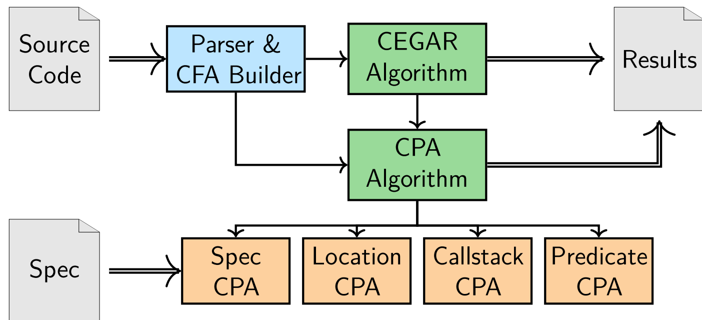
- Configurable program analysis
- Configurable block size
- Small overhead for synchronization in parallel analysis
- Elegant integration into the framework CPAchecker
- No changes necessary to existing analyses and components
  - CEGAR, proof and counterexample witnesses

Future work

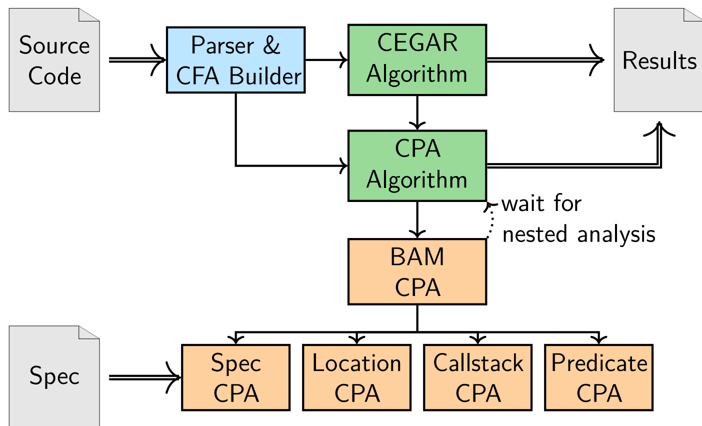
- Scheduling: Prefer parts deeper in the program?
- Processes instead of threads
  - Cluster instead multi-core machine

Questions?

CPAchecker Framework



BAM in CPACHECKER



Parallel BAM in CPAchecker