A Light-Weight Approach for Verifying Multi-Threaded Programs with CPAchecker

Dirk Beyer, Karlheinz Friedberger

1 LMU Munich, Germany
2 University of Passau, Germany
Multi-threaded programs appear everywhere!

- several threads per CPU core
- multi-core CPUs
- Linux kernel, device drivers
- internet, web and cloud services, IoT
- ...
- SV-Comp: special category for concurrent programs
Several approaches available:

- direct analysis of all thread interleavings
- program sequentialization
- formula-based encoding of threads
Several approaches available:
- direct analysis of all thread interleavings
- program sequentialization
- formula-based encoding of threads

Combined with some optimization:
- partial order reduction (ample sets, ...)
- iteration order for state-space exploration
- bounded model checking (bounded number of threads, ...)
Several approaches already available in CPAchecker:
(all of them are based on the pthreads library)

- formula-based encoding with predicate analysis
  → very old orphaned branch

- sequentialization of the CFA
  → student’s thesis, needs some work

- ThreadingCPA: handles program locations for multiple threads
  → replaces LocationCPA
  → everything else should work out-of-the-box (really?)
CFA control flow automaton with location nodes (program counter) and edges (statements and assumptions), one CFA per function, all function connected into super-graph of program.
Basics
What every developer of CPAchecker already knows

CFA control flow automaton with location nodes (program counter) and edges (statements and assumptions), one CFA per function, all function connected into super-graph of program

CPA abstract domain: how does an abstract state look alike? transfer relation: how to handle a single edge? merge and stop operator: how are abstract states related?
LocationCPA: one program location per abstract state
**LocationCPA**: one program location per abstract state

**Basic idea**: track *many* instead of *one* program locations
**LocationCPA**: one program location per abstract state

**Basic idea**: track *many* instead of *one* program locations

abstract state: \[ \{ t_1 \mapsto l^{t_1}, t_2 \mapsto l^{t_2}, \ldots \} \]

transfer relation: \( s \xrightarrow{g} s' \) depends on the edge \( g \):

1. *pthread_create*: add a new location for the new thread
2. *pthread_join*: remove the exit location of the joined thread
3. otherwise: just analyze the edge (like LocationCPA, with additional handling of pthread locks)

merge and stop operator: based on equality of abstract states (\( \text{merge}_{sep} \) and \( \text{stop}_{sep} \))

→ can be combined with other CPAs
Example
Program with CFA

```c
#include <pthread.h>

int i = 1, j = 1;

void main() {
    pthread_t id1, id2;
    pthread_create(&id1, 0, t1, 0);
    pthread_create(&id2, 0, t2, 0);
    pthread_join(id1, 0);
    pthread_join(id2, 0);
    assert(j <= 8);
}

void t1() {
    i += j;
    i += j;
}

void t2() {
    j += i;
    j += i;
}
```

Dirk Beyer, Karlheinz Friedberger
ThreadingCPA
Example
CFA and ARG

main
  0: pthread_t id1, id2;
  1: int i=1; j=1
  2: pthread_create(&id1, 0, t1, 0);
  3: pthread_create(&id2, 0, t2, 0);
  4: pthread_join(&id1, 0);
  5: pthread_join(&id2, 0);
  6: assert(j<=8);

thread t1
  A: i+=j;
  B: i+=j;
  C

thread t2
  X: j+=i;
  Y: j+=i;
  Z

main \rightarrow 0
main \rightarrow 1
main \rightarrow 2
main \rightarrow 3
id1 \rightarrow A
id1 \rightarrow B
id1 \rightarrow C
id2 \rightarrow X
id2 \rightarrow Y
id2 \rightarrow Z

main \rightarrow 3

id1 \rightarrow A

id2 \rightarrow X

main \rightarrow 4

id1 \rightarrow B

id2 \rightarrow Y

main \rightarrow 5

id1 \rightarrow C

id2 \rightarrow Z

main \rightarrow 6

main \rightarrow 7

Dirk Beyer, Karlheinz Friedberger
ThreadingCPA
We have to handle several call stacks, one per thread
→ integrate CallstackCPA into ThreadingCPA
We have to handle several call stacks, one per thread
→ integrate CallstackCPA into ThreadingCPA

ValueCPA, BDDCPA, IntervalCPA:
→ track assignments, identify variables as $f::x$
→ problem: same function called in several threads?
→ solution: avoid colliding function names by cloning each function before the analysis
We have to handle several **call stacks**, one per thread
→ integrate CallstackCPA into ThreadingCPA

**ValueCPA, BDDCPA, IntervalCPA:**
→ track assignments, identify variables as \( f::x \)
→ problem: same function called in several threads?
→ solution: avoid colliding function names by *cloning* each
  function before the analysis

**Other CPAs and algorithms:** TODO
→ some small changes required (several locations per state)
→ PredicateCPA: block operator matches thread interleavings?
→ more advanced thread management
Optimization for the ThreadingCPA

Is this simple approach efficient? Not yet!

We need optimization!

- partial order reduction
  → implemented in ThreadingCPA

- bound number of threads
  → implemented in ThreadingCPA

- iteration order
  → implemented as *waitlist order*, like BFS and DFS

- partitioning abstract states based on program location
  → inherit from *Partitionable* and use *PartitionedReachedSet*

- equality for call stack states with different object identities
  ! *CPAchecker* does not use equality for call stacks by default!
Evaluation on the Category "Concurrency", SV-Comp’16
Value Analysis with Optimization Steps

The graph shows the n-th fastest result against CPU time (s) for different optimization steps.

- **Plain value analysis** (red diamond)
- **Partitioning** (blue square)
- **Waitlist order** (brown triangle)
- **POR (opt. VA)** (green circle)

Dirk Beyer, Karlheinz Friedberger

ThreadingCPA
Evaluation on the Category "Concurrency", SV-Comp’16

Different analyses in CPAchecker

CPU time (s)

n-th fastest result

BDD analysis
interval analysis
opt. VA

Dirk Beyer, Karlheinz Friedberger
ThreadingCPA
Evaluation on the Category "Concurrency", SV-Comp’16

Comparison of CPAchecker with other tools

![Graph showing CPU time vs n-th fastest result for different tools: CBMC, VVT, opt. VA.](image-url)
Further Possibilities

CPAchecker is very flexible

Validation Witnesses:

- export counterexamples in Graphml
- extension of the format: include identifiers for threads
Further Possibilities

CPAchecker is very flexible

Validation Witnesses:
- export counterexamples in Graphml
- extension of the format: include identifiers for threads

Deadlock detection:
- for the user: just change the specification
- detail: the *strengthening* operator allows to inform the AutomatonCPA about deadlock found by the ThreadingCPA
Dining Philosophers Problem

Questions before Dinner?

○: Plato, Konfuzius, Socrates, Voltaire and Descartes