# Software Model Checking

Benedikt Maderbacher

23.6.2022

# Overview

- Verifying Software
- Bounded Model Checking
- Predicate Abstraction
- Software Development Workflow at AWS

# Verifying Software

# Example Program

$l_1 : x := 0;$
$l_2 : y := 0;$
$l_3 :$ **while** $x \neq 10$ **do**
$l_4 :$ $\quad x := x + 1;$
$l_5 :$ $\quad y := y + 1$
**end while**
$l_6 :$ **assert** $x = y;$
$l_7 :$

Figure from [1]

# Specification

- User-defined Assertions
- Null pointers, our-of-bounds access, ...
- Integer overflows

# Control-Flow Graphs

$l_1 : x := 0;$
$l_2 : y := 0;$
$l_3 :$ **while** $x \neq 10$ **do**
$l_4 :$ $\quad x := x + 1;$
$l_5 :$ $\quad y := y + 1$
**end while**
$l_6 :$ **assert** $x = y;$
$l_7 :$

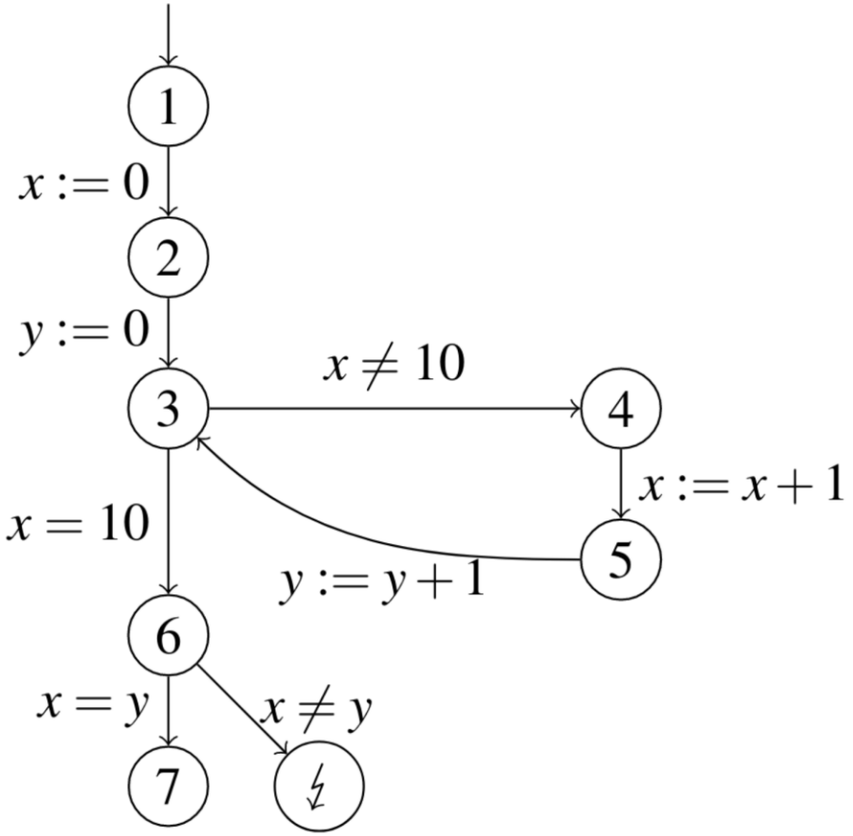


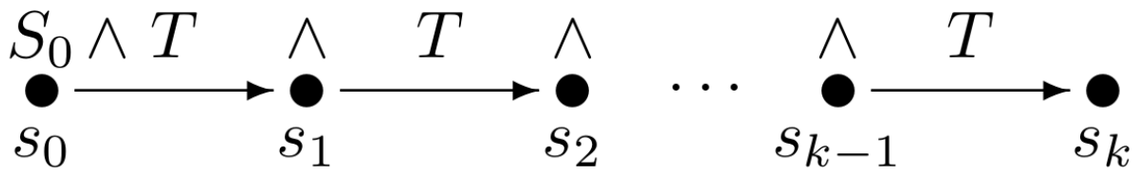

Figure from [1]

# Bounded Model Checking

[2-5]

# BMC

- set of states $S$,
- a set of initial states $S_0 \subset S$, and
- a transition relation $T \subset (S \times S)$.




$$S_0(s_0) \wedge \bigwedge_{i=0}^{k-1} T(s_i, s_{i+1})$$

Figure from [3]

# BMC

Check if $AG\,p$ holds:

$$S_0(s_0) \wedge \bigwedge_{i=0}^{k-1} T(s_i, s_{i+1}) \quad \wedge \quad \bigvee_{i=0}^{k} \neg p(s_i)$$

# Transition Relation for Software

First Idea (Monolithic Encoding)

- Add program counter (pc)
- State is defined by all variables plus pc
- Transition relation defines next state (after executing a statement) for every possible state.

# Monolithic Encoding

Example:

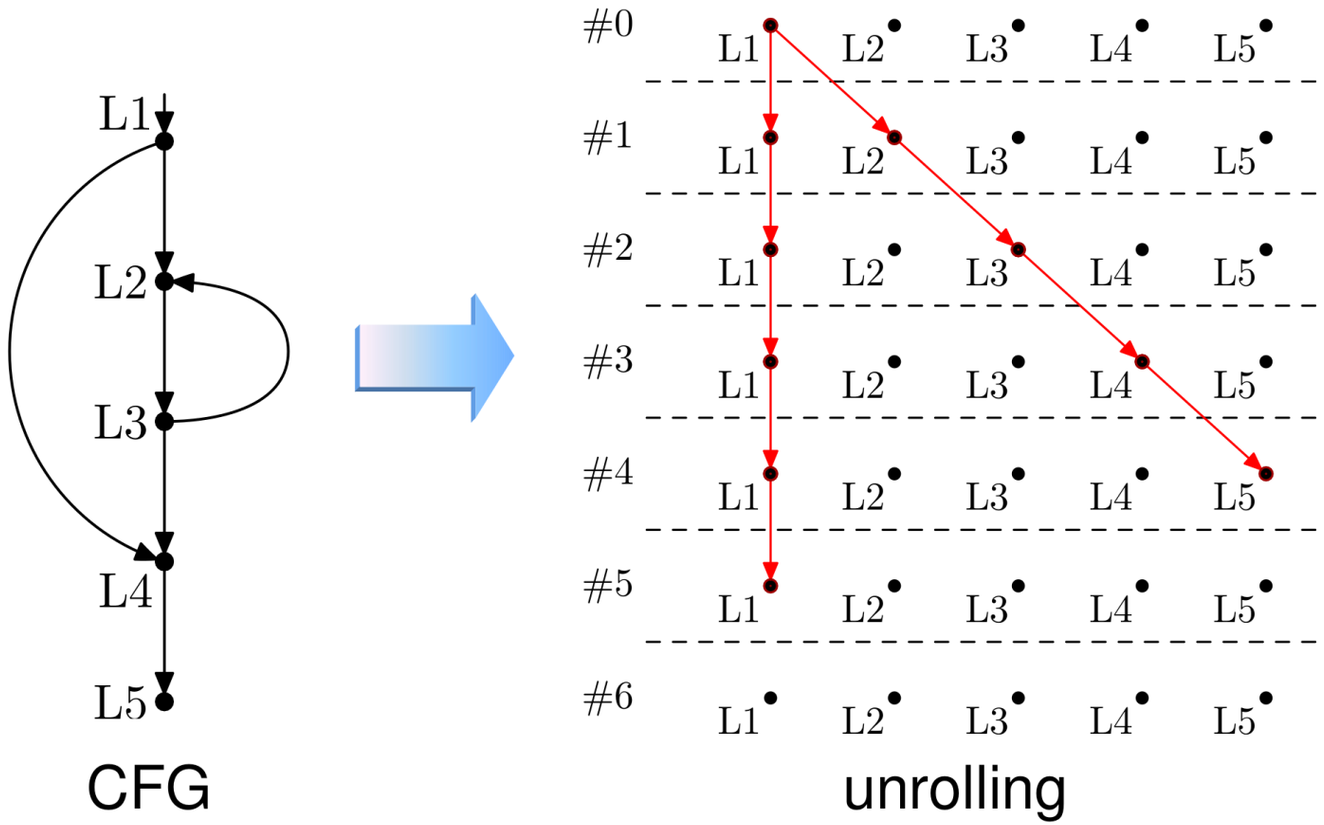



Figure from [3]

# Monolithic Encoding

## Optimization:

Don't encode unreachable states!

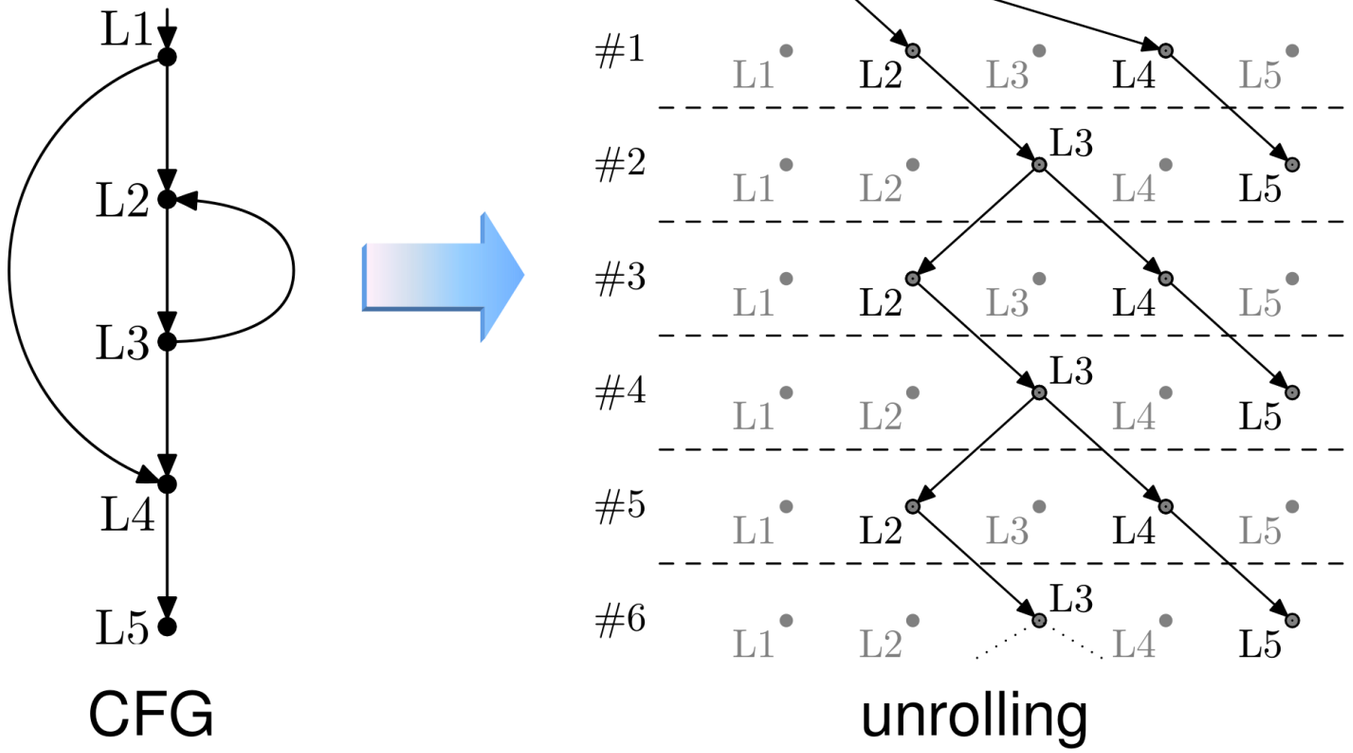



Figure from [3]

# Monolithic Encoding

Still a lot of repetition:

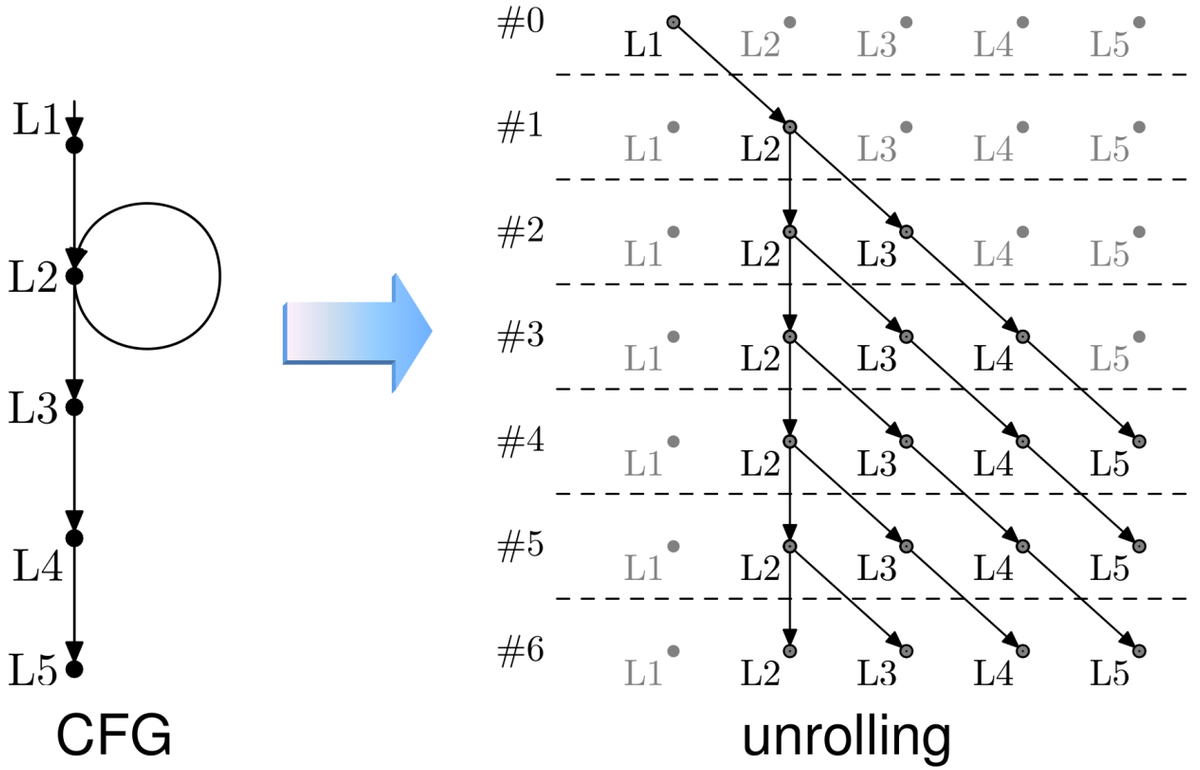
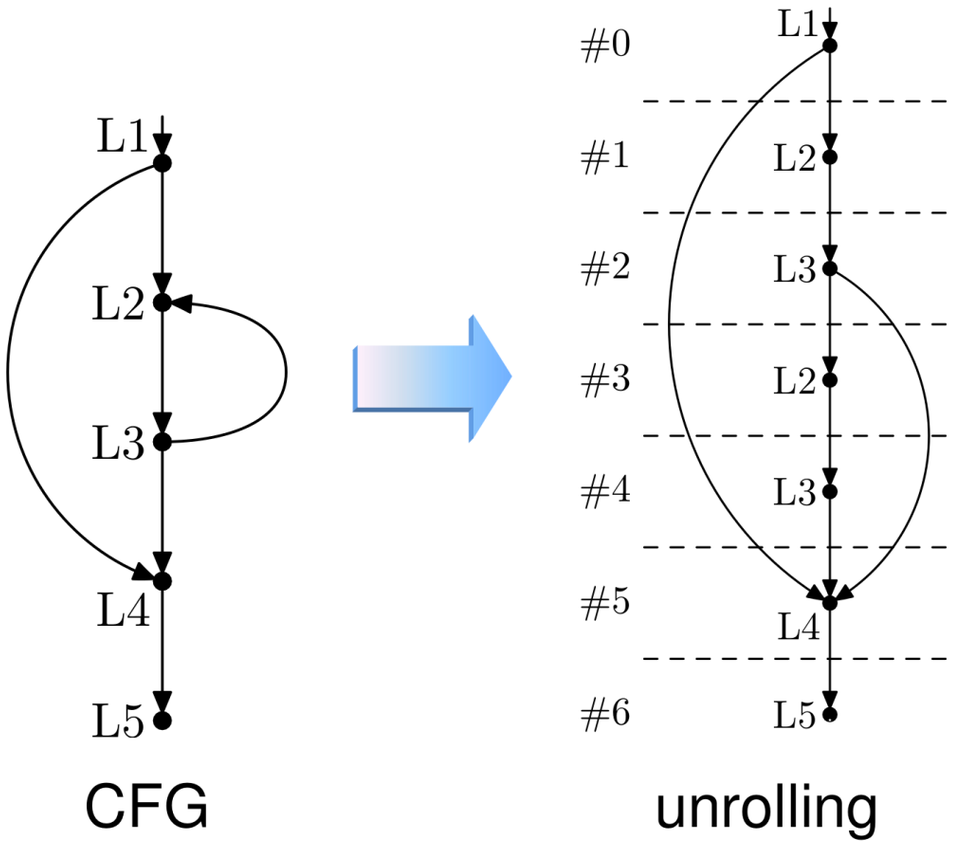



Figure from [3]

# Unrolling Loops

Alternative:

unroll loops instead of statements.

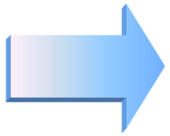



The bound k is for the loop unrollings and not the number of executed statements.

Figure from [3]

# Unrolling loops

```
while(cond)
    Body;
```



```
if(cond) {
    Body;
    if(cond) {
        Body;
        if(cond) {
            Body;
            while(cond)
                Body;
        }
    }
}
```

Figure from [3]

# Completeness Threshold

When have we unrolled enough?

Can we proof a worst-case execution time?

# Completeness Threshold

Add an assertion to guarantee that the loop will terminate within the current bound.

```
if(cond) {
    Body;
    if(cond) {
        Body;
        if(cond) {
            Body;
            assert(!cond);

        }
    }
}
```

Figure from [3]

# Finding the Completeness Threshold

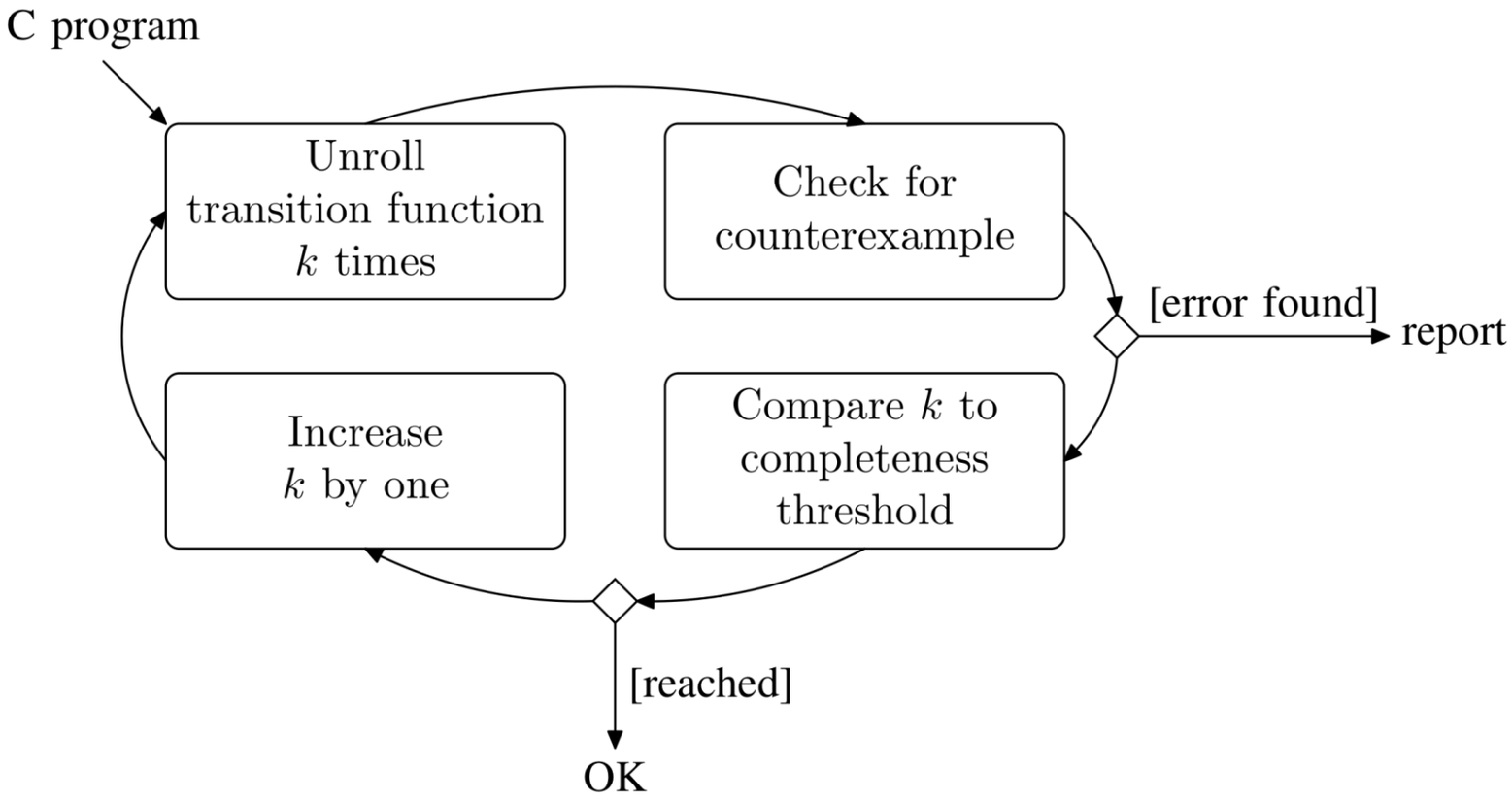



Figure from [4]

# Single Static Assignment

```
#define DLE 16
#define STX 2
#define ETX 3
uchar nd_uchar();
int main (void) {
  uchar in[6] = {DLE, STX, nd_uchar(),
                 DLE, ETX, '\0'};
  uchar out[6];
  int i = 0;
  int j = 0;
  while (in[i] != '\0') {
    switch (in[i]) {
      case (DLE):
        if (in[i+1]==STX || in[i+1]==ETX) {
          out[j] = in[i];
        } else {
          out[j] = in[i];
          out[++j] = DLE;
        };
        break;
      default:
        out[j] = in[i];
        break;
    }
    i++;
    j++;
  }
  out[j] = '\0';
  assert(out[4]==ETX || out[5]==ETX);
  return 0;
}
```

```
in1 == {16, 2, nd_uchar1, 16, 3, 0}
i1 == 0
j1 == 0
out1 == (out0 WITH [0:=16])
i2 == 1
j2 == 1
out2 == (out1 WITH [1:=2])
i3 == 2
j3 == 2
g1 == (nd_uchar1 != 0)
g2 == !(nd_uchar1 == 16)
out3 == (out2 WITH [2:=nd_uchar1])
j4 == 3
out4 == (out3 WITH [3:=16])
out5 == out2
j5 == j3
out6 == (out5 WITH [j5:=nd_uchar1])
out7 == (!g2 ? out4 : out6)
j6 == (!g2 ? j4 : j5)
i4 == 3
j7 == 1 + j6
out8 == (out7 WITH [j7:=16])
i5 == 4
j8 == 1 + j7
out9 == (out8 WITH [j8:=3])
i6 == 5
j9 == 1 + j8
out10 == (!g1 ? out2 : out9)
i7 == (!g1 ? i3 : i6)
j10 == (!g1 ? j3 : j9)
out11 == (out10 WITH [j10:=0])
```

Figure from [6]

# Verification Conditions

Add formula of the negated property to the state unrolling.

SAT if the property is violated.

This needs to also include enough guards to ensure that the violated property is reachable.

# Encoding Integers

Bitvector theory or Bitblasting

# Arrays, Pointers, and Memory

These can be encoded using the Array theory of a SMT solver.

# CBMC Demo

# Predicate Abstraction

[1,4,6]

# Predicate Abstraction

Many programs cannot be verified using BMC, because the required bound would be too large.

Use abstraction instead!

# Predicate Abstraction

Compute a sound over-approximation of the reachable states.

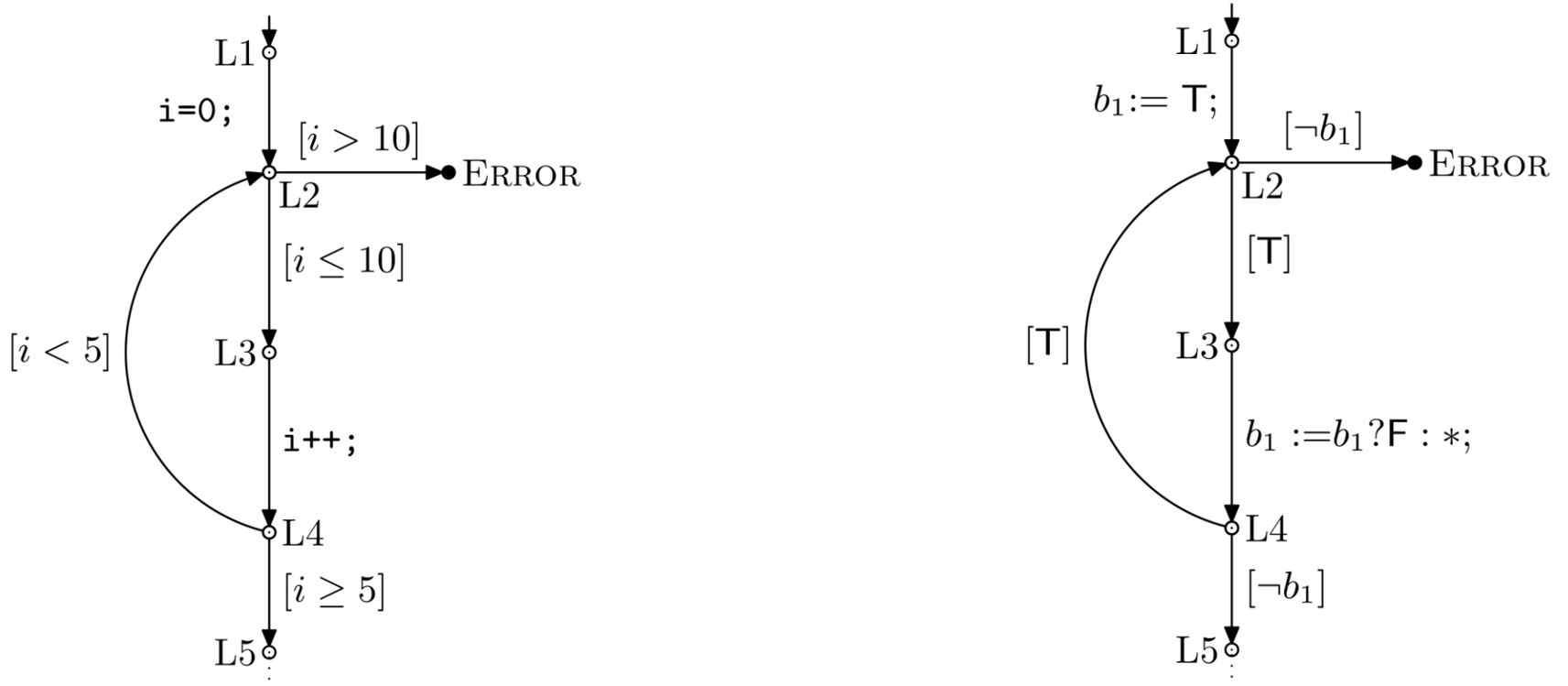
Only track Boolean predicates instead of actual states.

If the abstract program is safe the original is also safe.
The other way does not hold.

# Predicate Abstraction

b1: i == 0

## Example:

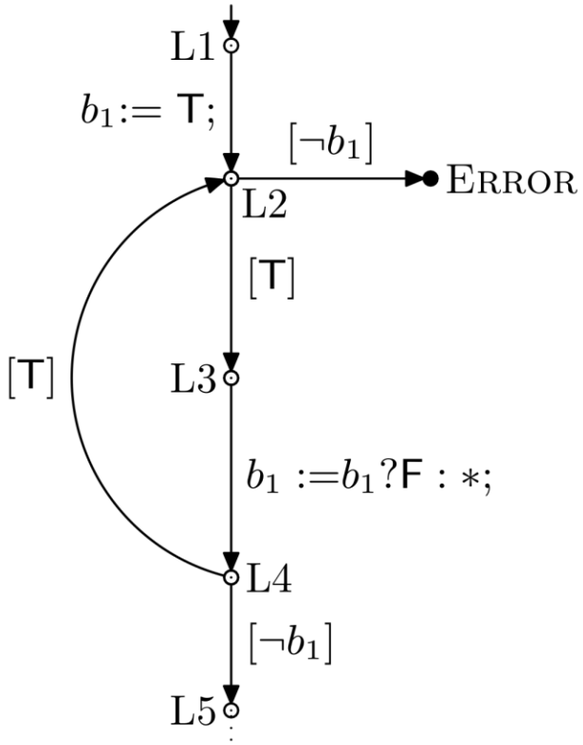



(a) Concrete Program

(b) Abstraction for $P = \{b_1\}$

Figure from [4]

# Predicate Abstraction




(b) Abstraction for $P = \{b_1\}$

$\mathrm{b1: i} == 0$

Error reached via:
L1, L2, L3, L4, L2, Error

Check against the real program.
Counter example is spurious.

Figure from [4]

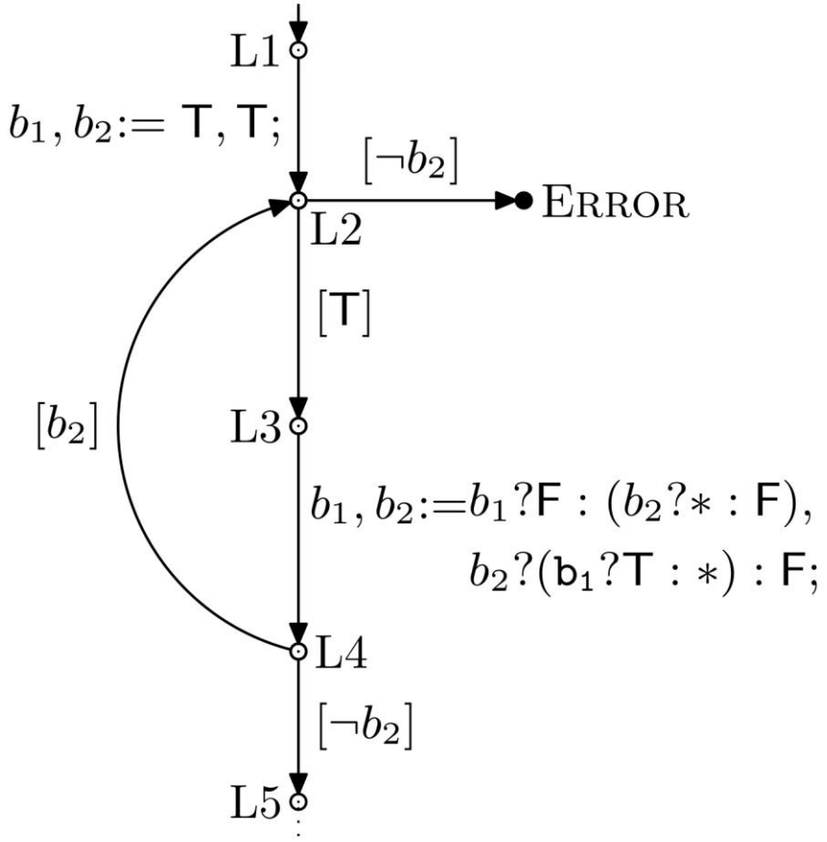
# Predicate Abstraction

A different set of predicates:

b1: i==0

b2: $i < 5$

Program is correct.




(c) Abstraction for $P = \{b_1, b_2\}$

Figure from [4]

# Counter Example Guided Abstraction Refinement (CEGAR)

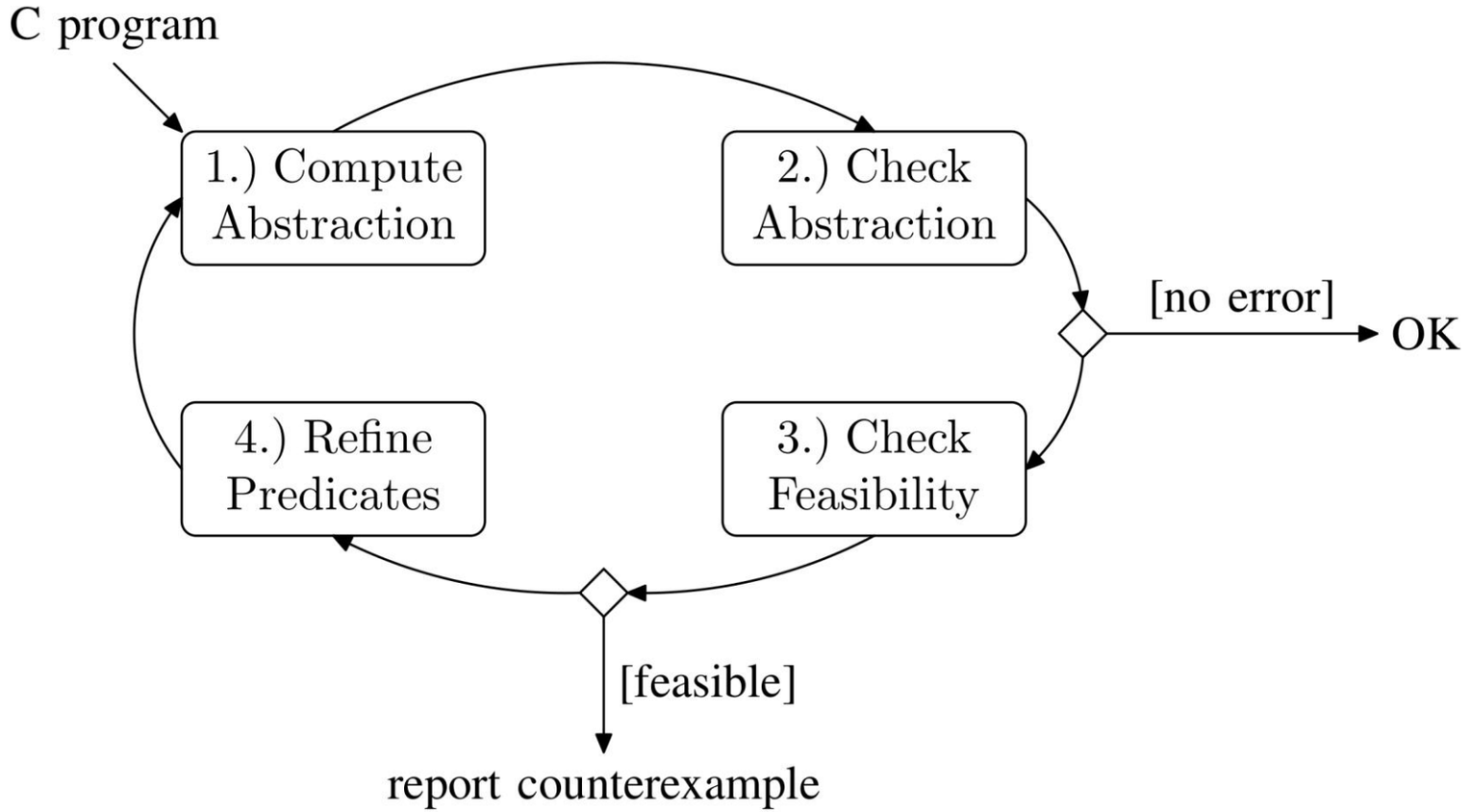



Figure from [4]

This is covered in more detail in the course Verification and Testing.

# Other Areas in Software Model Checking

- Concurrency
  - All possible interleavings need to be checked.
  - ⑩ Partial order reduction can be used as an optimization.

- Termination Checking
  - Show that there is not infinite computation
  - Can be done using CEGAR by abstracting the transition relation and demonstrating that there is no infinite chain. [6]

# Industry Applications

[7-9]

# Code-level model checking at Amazon Web Services

Amazon uses CBMC to verify some of their C code.

This includes:

- AWC C Common — cross platform data structures, error handling, ...
- FreeRTOS — a real time operating system
- s2n — implementation of TSL/SSL used in e.g. S3
- AWS Encryption SDK for C

[7]

# Verified Properties

- User defined assertions
- Function contracts (pre-/post-conditions)

- Memory Safety
- Arithmetic overflows, division by zero
- ...

[7]

# Methodology

- Make specifications explicit in source code
- Write proof harnesses in declarative style
- Integrate proof artifacts into the development workflow
- Fix bugs instead of just reporting them

[7]

# Example

```c
/* Resizable array implementation. */
struct aws_array_list {
    struct aws_allocator *alloc;
    size_t current_size;
    size_t length;
    size_t item_size;
    void *data;
};
```

```c
/**
 * Copies the memory address of the element at index to *val.
 * If element does not exist, AWS_ERROR_INVALID_INDEX will be raised.
 */
int aws_array_list_get_at_ptr(
      const struct aws_array_list *list,
      void **val,
      size_t index)
{
  if (aws_array_list_length(list) > index) {
      *val = (void *)((uint8_t *)list->data + (list->item_size * index));
      return AWS_OP_SUCCESS;
  }
  return aws_raise_error(AWS_ERROR_INVALID_INDEX);
}
```

[7]

# Proof Harness

```
1  /* CBMC-proof harness for aws_array_list_get_at_ptr function. */
2  void aws_array_list_get_at_ptr_harness()
3  {
4      /* initialization */
5      struct aws_array_list list;
6      __CPROVER_assume(aws_array_list_is_bounded(&list));
7      ensure_array_list_has_allocated_data_member(&list);
8
9      /* generate unconstrained inputs */
10     void **val = can_fail_malloc(sizeof(void *));
11     size_t index;
12
13     /* preconditions */
14     __CPROVER_assume(aws_array_list_is_valid(&list));
15     __CPROVER_assume(val != NULL);
16
17     /* call function under verification */
18     if(!aws_array_list_get_at_ptr(&list, val, index)) {
19       /* If aws_array_list_get_at_ptr is successful,
20        * i.e. ret==0, we ensure the list isn't
21        * empty and index is within bounds */
22        assert(list.data != NULL);
23        assert(list.length > index);
24     }
25
26     /* postconditions */
27     assert(aws_array_list_is_valid(&list));
28     assert(val != NULL);
29 }
```

[7]

# Data Invariant

```
1  /* Invariants for aws_array_list data structure. */
2  bool aws_array_list_is_valid(const struct aws_array_list *list)
3  {
4    /* Object must be valid. */
5    if (list != NULL) return false;
6
7    /* Length and item size must not lead to overflows. */
8    size_t required_size = 0;
9    bool required_size_is_valid =
10           (aws_mul_size_checked(list->length,
11                                 list->item_size,
12                                 &required_size)
13            == AWS_OP_SUCCESS);
14
15   /* Current size must be enough to store all elements. */
16   bool current_size_is_valid =
17           (list->current_size >= required_size);
18
19   /* Underlying pointer must have current_size allocated positions. */
20   bool data_is_valid =
21           ((list->current_size == 0 && list->data == NULL)
22           || AWS_MEM_IS_WRITABLE(list->data, list->current_size));
23
24   /* Item size must not be zero. */
25   bool item_size_is_valid = (list->item_size != 0);
26
27   return required_size_is_valid
28          && current_size_is_valid
29          && data_is_valid && item_size_is_valid;
30 }
```

[7]

# Function Contract

```
1  int aws_array_list_get_at_ptr(
2          const struct aws_array_list* list,
3          void **val,
4          size_t index)
5  {
6      /* Contracts. */
7      AWS_PRECONDITION(aws_array_list_is_valid(list));
8      AWS_PRECONDITION(val != NULL);
9
10     if (aws_array_list_length(list) > index) {
11         *val = (void *)((uint8_t *)list->data +
12                         (list->item_size * index));
13         AWS_POSTCONDITION(aws_array_list_is_valid(list));
14         return AWS_OP_SUCCESS;
15     }
16
17     /* Contracts. */
18     AWS_POSTCONDITION(aws_array_list_is_valid(list));
19     return aws_raise_error(AWS_ERROR_INVALID_INDEX);
20 }
```

[7]

# Results

- Increased proof speed
- Increased rated of bugs found and fixed
- Active developer engagement with proofs
- Increase in lines of specifications written by developers

[7]

# Microsoft's Static Driver Verifier (SDV)

Pioneered CEGAR for Software Verification (SLAM)

Verification for Windows kernel-mode drivers

[8]

# Facebook's Infer Tool

Another large-scale application of formal methods for software.

Not model checking, instead uses static analysis and separation logic

Also integrated in CI

Focus on fast bug finding with few false positives.

[9]

# SV Comp

Yearly competition for software verification tools.

https://sv-comp.sosy-lab.org/2022/

# Summary

Bounded Model Checking for Software

Predicate Abstraction

Industry Applications of Software Model Checking

# Sources

[1] Clarke et al.: "Model Checking" MIT Press (2018), Chapter 14

[2] Clarke, Kroening, Lerda: "A Tool for Checking ANSI-C Programs" TACAS (2004)

[3] Kroening : "CBMC Slides" (2010) https://www.cprover.org/cbmc/doc/cbmc-slides.pdf

[4] D'Silva, Kroening, Weissenbacher: "A Survey of Automated Techniques for Formal Software Verification" IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. (2008)

[5] Cordeiro, Fischer, Marques-Silva: "SMT-Based Bounded Model Checking for Embedded ANSI-C Software". IEEE Trans. Software Eng. (2012)

[6] Ranjit Jhala, Andreas Podelski, Andrey Rybalchenko: "Predicate Abstraction for Program Verification". Handbook of Model Checking 2018: Chapter 15

[7] Chong et al. : "Code-level model checking in the software development workflow at Amazon Web Services". Softw. Pract. Exp. (2021)

[8] Ball et al.: "SLAM and Static Driver Verifier: Technology Transfer of Formal Methods inside Microsoft" IFM (2004)

[9] Distefano et al.: "Scaling static analyses at Facebook". Commun. ACM (2019)