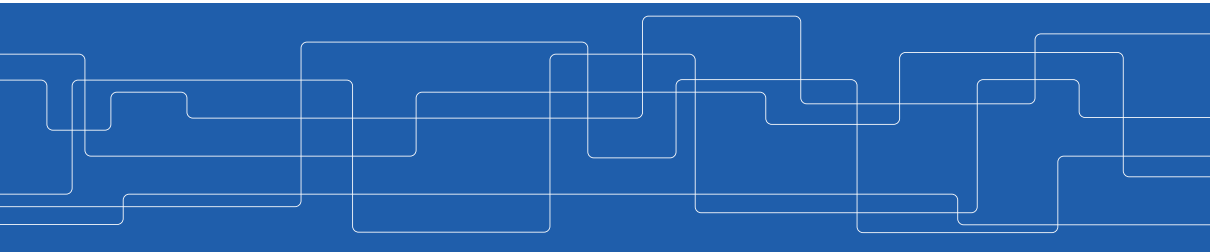
An exercise in mind reading: Automatic contract inference for Frama-C

Jesper Amilon[1], Zafer Esen[2], Dilian Gurov[1], Christian Lidström[1], Philipp Rümmer[2,3]

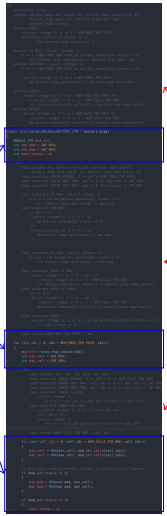
[1] KTH University, Sweden [2] Uppsala University, Sweden [3] Regensburg University, Germany

Frama-C workshop Paris June 14, 2024



Automating deductive verification

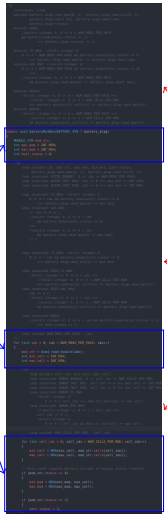
Automating deductive verification



Code

Annotations

Automating deductive verification



Code

Annotations

Specification inference techniques

- Dynamic
- Strongest post-condition
- Abstract Interpretation
- Property-guided
- etc

Contract inference – for helper functions

```
1
2
3  void helper1() f ... g
4
5
6
7  void helper2() f ... g
8
9  /@ requires P;
10     ensures Q; /
11 void main() f
12     ...
13     helper1();
14     helper2();
15     ...
16 g
```

Idea

- Manually specify the main function.
- Infer contracts for all *helper* functions.
- The Saida plugin

Contract inference – for helper functions

```
1
2
3  void helper1() f ... g
4
5
6
7  void helper2() f ... g
8
9  /@ requires P;
10     ensures Q; /
11 void main() f
12     ...
13     helper1();
14     helper2();
15     ...
16 g
```

Where's my contract?

Idea

- Manually specify the main function.
- Infer contracts for all *helper* functions.
- The Saida plugin

Contract inference – for helper functions

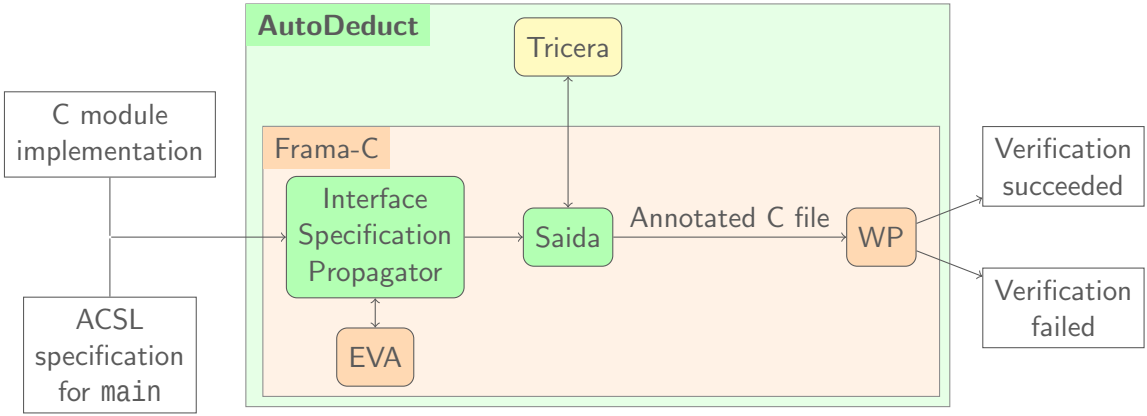
Here they are!
(automatically inferred)

```
1  /*@ requires ...;
2      ensures  ...; */
3  void helper1() f ... g
4
5  /*@ requires ...;
6      ensures  ...; */
7  void helper2() f ... g
8
9  /*@ requires P;
10     ensures Q; */
11 void main() f
12     ...
13     helper1();
14     helper2();
15     ...
16 g
```

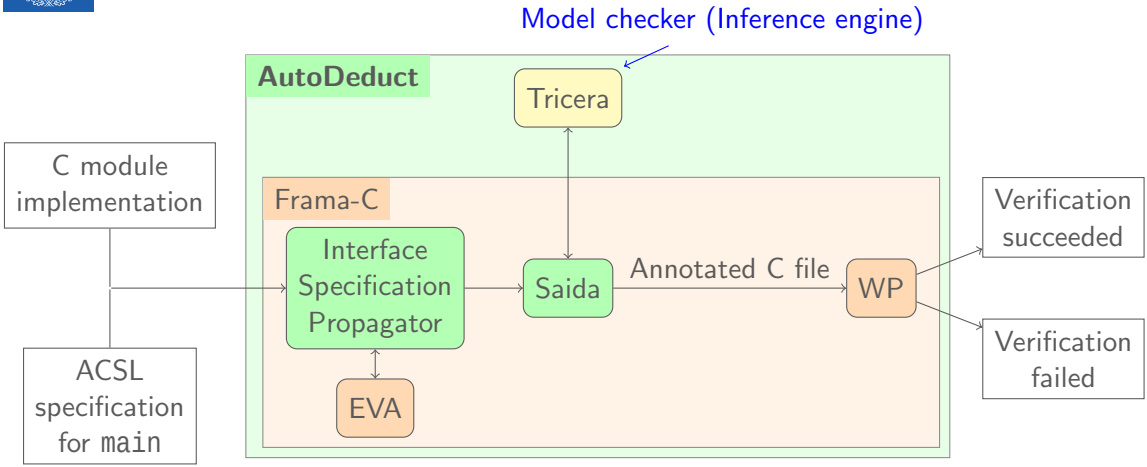
Idea

- Manually specify the main function.
- Infer contracts for all *helper* functions.
- The Saida plugin

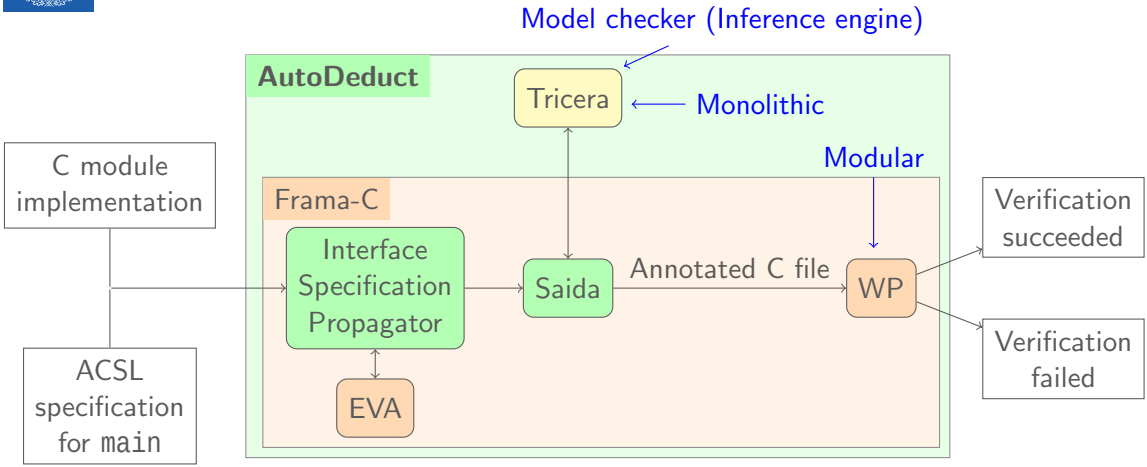
Workflow and toolchain



Workflow and toolchain



Workflow and toolchain



The TriCera model checker

```
1  int mc91(int n) {
2    if (n > 100) {
3      return n - 10;
4    } else {
5      return mc91(mc91(n + 11));
6    }
7  }
8
9  void harness() {
10   int x,y = *; // nondet init
11   assume(x <= 100);
12   y = mc91(x);
13   assert(y==91);
14 }
```

- Assertion-based
- Example, McCarthy 91:

$$mc91(n) = \begin{cases} n - 10 & \text{if } n > 100 \\ mc91(mc91(n+11)) & \text{if } n \leq 100 \end{cases}$$

The TriCera model checker

```
1  int mc91(int n) {
2    if (n > 100) {
3      return n - 10;
4    } else {
5      return mc91(mc91(n + 11));
6    }
7  }
8
9  void harness() {
10   int x,y = *; // nondet init
11   assume(x <= 100);
12   y = mc91(x);
13   assert(y==91);
14 }
```

- Assertion-based
- Example, McCarthy 91:

$$mc91(n) = \begin{cases} n - 10 & \text{if } n > 100 \\ mc91(mc91(n+11)) & \text{if } n \leq 100 \end{cases}$$

Satisfies property:

$$mc91(n) = \begin{cases} n - 10 & \text{if } n > 100 \\ 91 & \text{if } n \leq 100 \end{cases}$$

Contract inference with TriCera

```
1  int mc91(int n) {
2    if (n > 100) {
3      return n - 10;
4    } else {
5      return mc91(mc91(n + 11));
6    }
7  }
8
9  void harness() {
10   int x,y = *; // nondet init
11   assume(x <= 100);
12   y = mc91(x);
13   assert(y==91);
14 }
```

Horn clause encoding CHCs (uninterpreted predicates):

mc91:

$mc91_{post}(n; n - 10) \quad mc91_{pre}(n) \wedge n > 100$

$mc91_{pre}(n + 11) \quad mc91_{pre}(n) \wedge n \leq 100$

$mc91_{post}(n; r) \quad mc91_{post}(n + 11; r^{0}) \wedge mc91_{post}(r^{0}; r)$

harness:

$harness1(x; y) \quad x \leq 100$

$mc91_{pre}(x) \quad harness1(x; y)$

$harness2(r; y) \quad mc91_{post}(x; r) \wedge harness1(x; y)$

$false \quad harness2(x; y) \wedge y \neq 91$

Contract inference with TriCera

Verification, find a solution:

$harness1(x; y) = ?$

$harness2(x; y) = ?$

$harness3(x; y) = ?$

$mc91_{pre}(n) = ?$

$mc91_{post}(n; r) = ?$

Horn clause encoding CHCs (uninterpreted predicates):

mc91:

$mc91_{post}(n; n \quad 10) \quad mc91_{pre}(n) \wedge n > 100$

$mc91_{pre}(n + 11) \quad mc91_{pre}(n) \wedge n \quad 100$

$mc91_{post}(n; r) \quad mc91_{post}(n + 11; r^{\theta}) \wedge$
$mc91_{post}(r^{\theta}; r)$

harness:

$harness1(x; y) \quad x \quad 100$

$mc91_{pre}(x) \quad harness1(x; y)$

$harness2(r; y) \quad mc91_{post}(x; r) \wedge harness1(x; y)$

$false \quad harness2(x; y) \wedge y \not= 91$

Contract inference with TriCera

Verification, find a solution:

$harness1(x;y) = :::$

$harness2(x;y) = :::$

$harness3(x;y) = :::$

$mc91_{pre}(n) = 111 \quad n$

$mc91_{post}(n;r) = 111 \quad n \wedge (r = 91\_$
$n \quad r \quad 10 \wedge r = 92)$

Horn clause encoding CHCs (uninterpreted predicates):

mc91:

$mc91_{post}(n;n \quad 10) \quad mc91_{pre}(n) \wedge n > 100$

$mc91_{pre}(n+11) \quad mc91_{pre}(n) \wedge n \quad 100$

$mc91_{post}(n;r) \quad mc91_{post}(n+11;r^{\theta}) \wedge$
$mc91_{post}(r^{\theta};r)$

harness:

$harness1(x;y) \quad x \quad 100$

$mc91_{pre}(x) \quad harness1(x;y)$

$harness2(r;y) \quad mc91_{post}(x;r) \wedge harness1(x;y)$

$false \quad harness2(x;y) \wedge y \not= 91$

Contract inference with TriCera

Verification, find a solution:

$harness1(x; y) = :::$

$harness2(x; y) = :::$

$harness3(x; y) = :::$

$mc91_{pre}(n) = 111 \quad n$

$mc91_{post}(n; r) = 111 \quad n \wedge (r = 91\_$

$n \quad r \quad 10 \wedge r = 92)$

Horn clause encoding CHCs (uninterpreted predicates):

mc91:

$mc91_{post}(n; n \quad 10) \quad mc91_{pre}(n) \wedge n > 100$

$mc91_{pre}(n + 11) \quad mc91_{pre}(n) \wedge n \quad 100$

$mc91_{post}(n; r) \quad mc91_{post}(n + 11; r^{\theta}) \wedge$

$mc91_{post}(r^{\theta}; r)$

harness:

$harness1(x; y) \quad x \quad 100$

$mc91_{pre}(x) \quad harness1(x; y)$

$harness2(r; y) \quad mc91_{post}(x; r) \wedge harness1(x; y)$

$false \quad harness2(x; y) \wedge y \not= 91$

Contract inference with TriCera

Verification, find a solution:

$harness1(x; y) = :::$

$harness2(x; y) = :::$

$harness3(x; y) = :::$

$mc91_{pre}(n) = 111 \quad n$

$mc91_{post}(n; r) = 111 \quad n \wedge (r = 91\_$

$n \quad r \quad 10 \wedge r = 92)$

```
/*@ requires 111 >= n;
    ensures 111 >= \old(n) &&
     (\result == 91 || (\old(n)
    -
      \result >= 10 &&
     \result >= 92)); */
```

Horn clause encoding CHCs (uninterpreted predicates):

mc91:

$mc91_{post}(n; n \quad 10) \quad mc91_{pre}(n) \wedge n > 100$

$mc91_{pre}(n + 11) \quad mc91_{pre}(n) \wedge n \quad 100$

$mc91_{post}(n; r) \quad mc91_{post}(n + 11; r^{\theta}) \wedge$

$mc91_{post}(r^{\theta}; r)$

harness:

$harness1(x; y) \quad x \quad 100$

$mc91_{pre}(x) \quad harness1(x; y)$

$harness2(r; y) \quad mc91_{post}(x; r) \wedge harness1(x; y)$

$false \quad harness2(x; y) \wedge y \not= 91$

Contract inference for Frama-C (Saida)

1. Create Harness
2. Verify Harness *and* Program in TriCera
3. Extract contracts from solution

```
1  int mc91(int n) {
2    if (n > 100) {
3      return n - 10;
4    } else {
5      return mc91(mc91(n + 11));
6    }
7  }
8  /*@ requires x <= 100;
9      ensures \result == 2*91;*/
10 int entry(int x, int y) {
11   int tmp = mc91(x);
12   return 2*tmp;
13 }
```

Contract inference for Frama-C (Saida)

1. Create Harness
2. Verify Harness *and* Program in TriCera
3. Extract contracts from solution

```
1  int mc91(int n) {
2    if (n > 100) {
3      return n - 10;
4    } else {
5      return mc91(mc91(n + 11));
6    }
7  }
8  /*@ requires x <= 100;
9      ensures \result == 2*91;*/
10 int entry(int x, int y) {
11   int tmp = mc91(x);
12   return 2*tmp;
13 }
```

```
void harness() {
  int x,y = _; //nondet init
  assume((x <= 100)); //requires
  int res = entry(x, y);
  assert((2*res == 91));  //ensures
}
```

Contract inference for Frama-C (Saida)

```
/*@ requires 111 >= n;
    ensures 111 >= \old(n) &&
     (\result == 91 || (\old(n) -
      \result >= 10 &&
      \result >= 92)); */
```

1. Create Harness
2. Verify Harness *and* Program in TriCera
3. Extract contracts from solution

```
int mc91(int n) {
  if (n > 100) {
    return n - 10;
  } else {
    return mc91(mc91(n + 11));
  }
}
/*@ requires x <= 100;
    ensures \result == 2*91; */
int entry(int x, int y) {
  int tmp = mc91(x);
  return 2*tmp;
}
```

```
void harness() {
  int x,y = _; //nondet init
  assume((x <= 100)); //requires
  int res = entry(x, y);
  assert((2*res == 91));  //ensures
}
```

Contract inference for Frama-C (Saida)

```
/*@ requires 111 >= n;
    ensures 111 >= \old(n) &&
     (\result == 91 || (\old(n) -
      \result >= 10 &&
      \result >= 92)); */
```

1. Create Harness
2. Verify Harness *and* Program in TriCera
3. Extract contracts from solution

```
int mc91(int n) {
  if (n > 100) {
    return n - 10;
  } else {
    return mc91(mc91(n + 11));
  }
}
/*@ requires x <= 100;
    ensures \result == 2*91; */
int entry(int x, int y) {
  int tmp = mc91(x);
  return 2*tmp;
}
```

```
void harness() {
  int x,y = _; //nondet init
  assume((x <= 100)); //requires
  int res = entry(x, y);
  assert((2*res == 91));  //ensures
}
```

Contract inference for Frama-C (Saida)

```
/*@ requires 111 >= n;
    ensures 111 >= \old(n) &&
     (\result == 91 || (\old(n) -
      \result >= 10 &&
      \result >= 92)); */
```

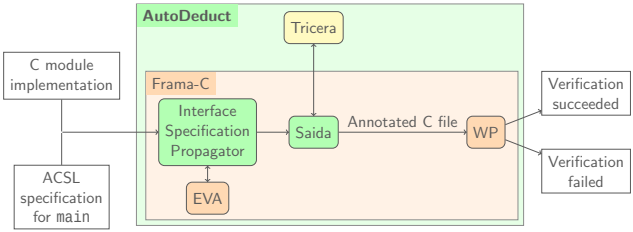
```
int mc91(int n) {
  if (n > 100) {
    return n - 10;
  } else {
    return mc91(mc91(n + 11));
  }
}
/*@ requires x <= 100;
    ensures \result == 91; */
int entry(int x, int y) {
  int tmp = mc91(x);
  return 2*tmp;
}
```

1. Create Harness
2. Verify Harness *and* Program in TriCera
3. Extract contracts from solution
4. Verify program with inferred contracts using WP

```
void harness() {
  int x,y = _; //nondet init
  assume((x <= 100)); //requires
  int res = entry(x, y);
  assert((2*res == 91));  //ensures
}
```

Why back to Frama-C?

- Frama-C more powerful
- Partial inference
- Relax constraints on TriCera



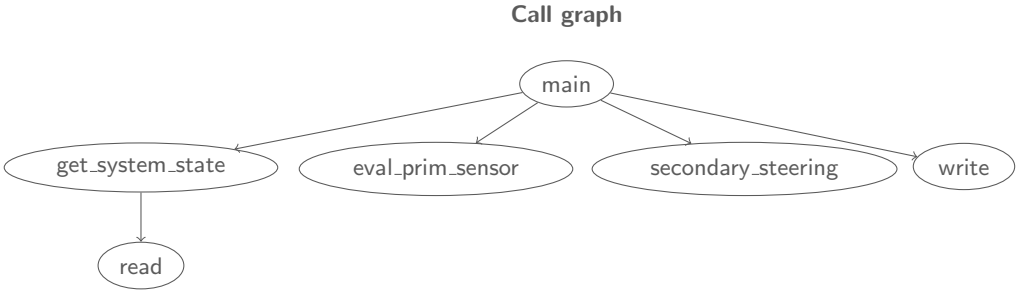
Loop invariants

```
1  /@
2    requires N > 0 && x > y;
3    ensures x > y;
4  @/
5  int main(int N, int x, int y) {
6    int i = 0;
7
8    /@@loop invariant N >= 1 && x - y >= 1 && i >= 0;@/
9    while (i < N) {
10     x = x@2;
11     y = y@2;
12     i++;
13   }
14 }
```

Demo

- Simplified version of module controlling secondary steering
- Five requirements, specified at top-level (main)

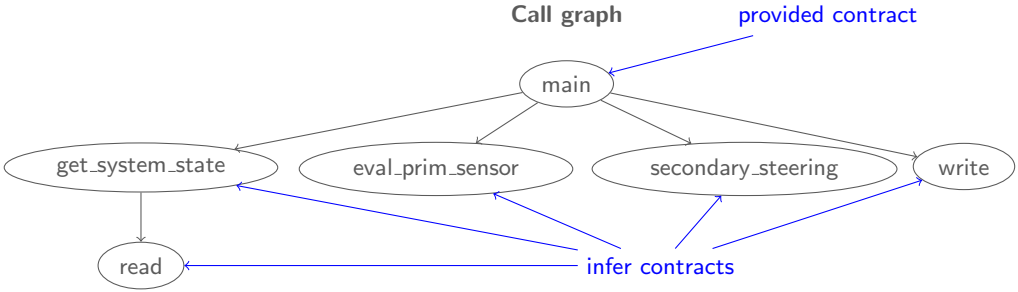
Demo



Demo

- Simplified version of module controlling secondary steering
- Five requirements, specified at top-level (main)

Demo



Outlook

- Show industrial-scale utility
- TriCera ACSL support
- Partial contract inference
- Library/API functions
- Support more language features

Saida Andersson (mind reader)

Saida: https://github.com/rse-verification/saida
TriCera: https://github.com/uuverifiers/tricera