Synthesis of Synchronization using Uninterpreted Functions*

October 22, 2014

Roderick Bloem, Georg Hofferek, Bettina Könighofer, Robert Könighofer, Simon Außerlechner, and Raphael Spörk

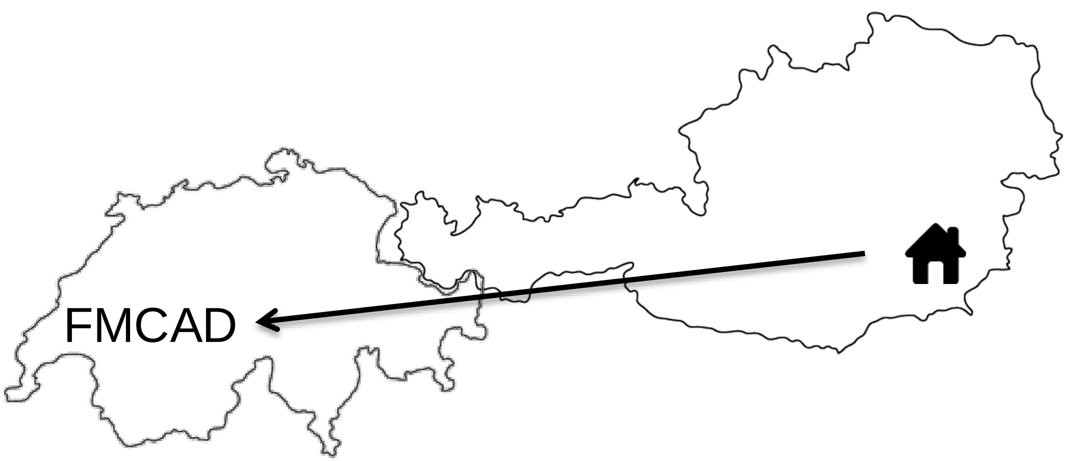
* This work was supported in part by the Austrian Science Fund (FWF) through the national research network RiSE (S11406-N23) and the project QUAINT (I774-N23).

What is Synthesis?

- Specification: **What?**
  - **From:** Graz, Inffeldgasse
  - **To:** Lausanne, 6pm

Synthesis →

- Implementation: **How?**
  - Walk to Moserhofgasse
  - Tram 6 to Jakominiplatz
    - Buy tram ticket
  - Tram 3 to train station Graz
  - Buy train ticket
  - Train to Salzburg
  - Train to Zürich
  - Train to Launsanne
  - Walk to Lausanne Fon
  - And so on …

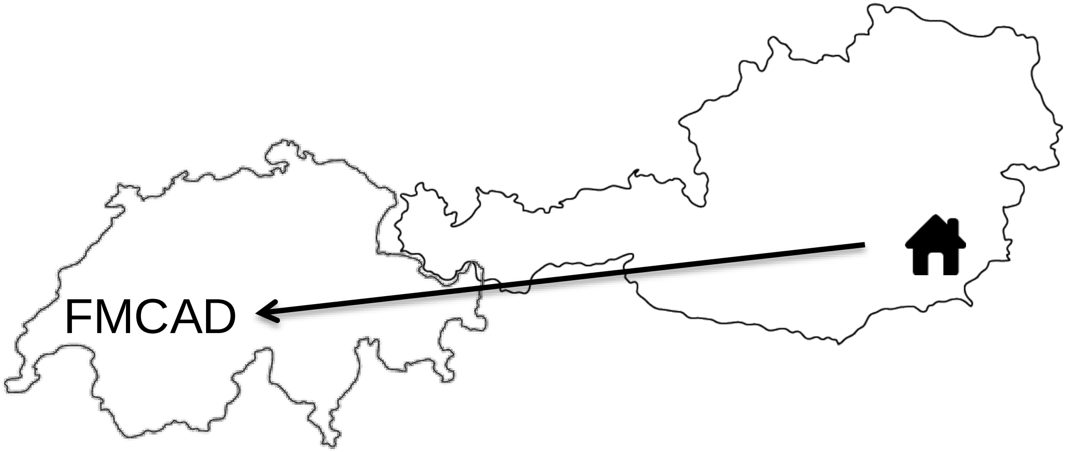


What is Synthesis?

- Specification: **What?**
  - **From:** Graz, Inffeldgasse
  - **To:** Lausanne, 6pm

Synthesis

- Implementation: **How?**
  - Walk to Moserhofgasse
  - Tram **???** to Jakominiplatz
    - Buy tram ticket
  - Tram 3 to train station Graz
  - Buy train ticket
  - Train to **???**
  - Train to Zürich
  - Train to Launsanne
  - Walk to Lausanne Fon
  - And so on …



Concurrent Programs

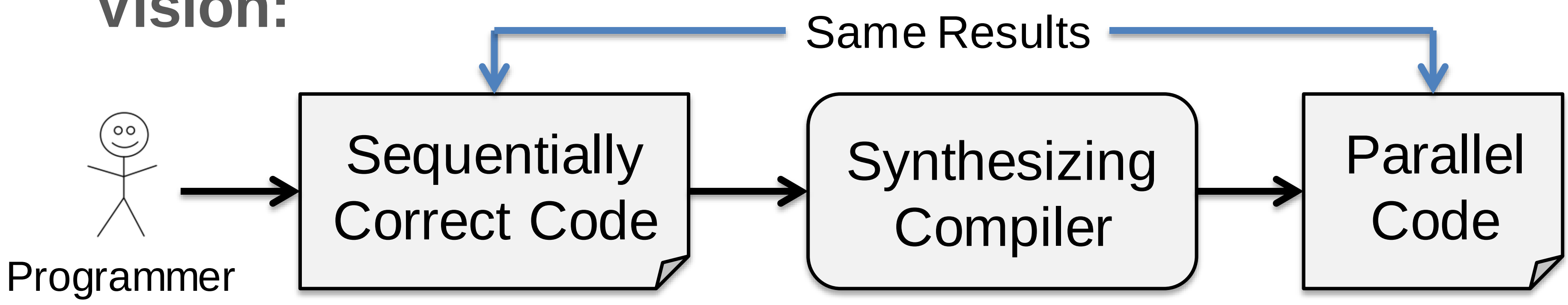
Functionality:
- Hard to specify
- Easy to implement

→ Implement manually

Concurrent Correctness:
- Easy to specify
  - Same result
- Hard to implement

→ Synthesize

Vision:



Synthesizing Atomic Sections

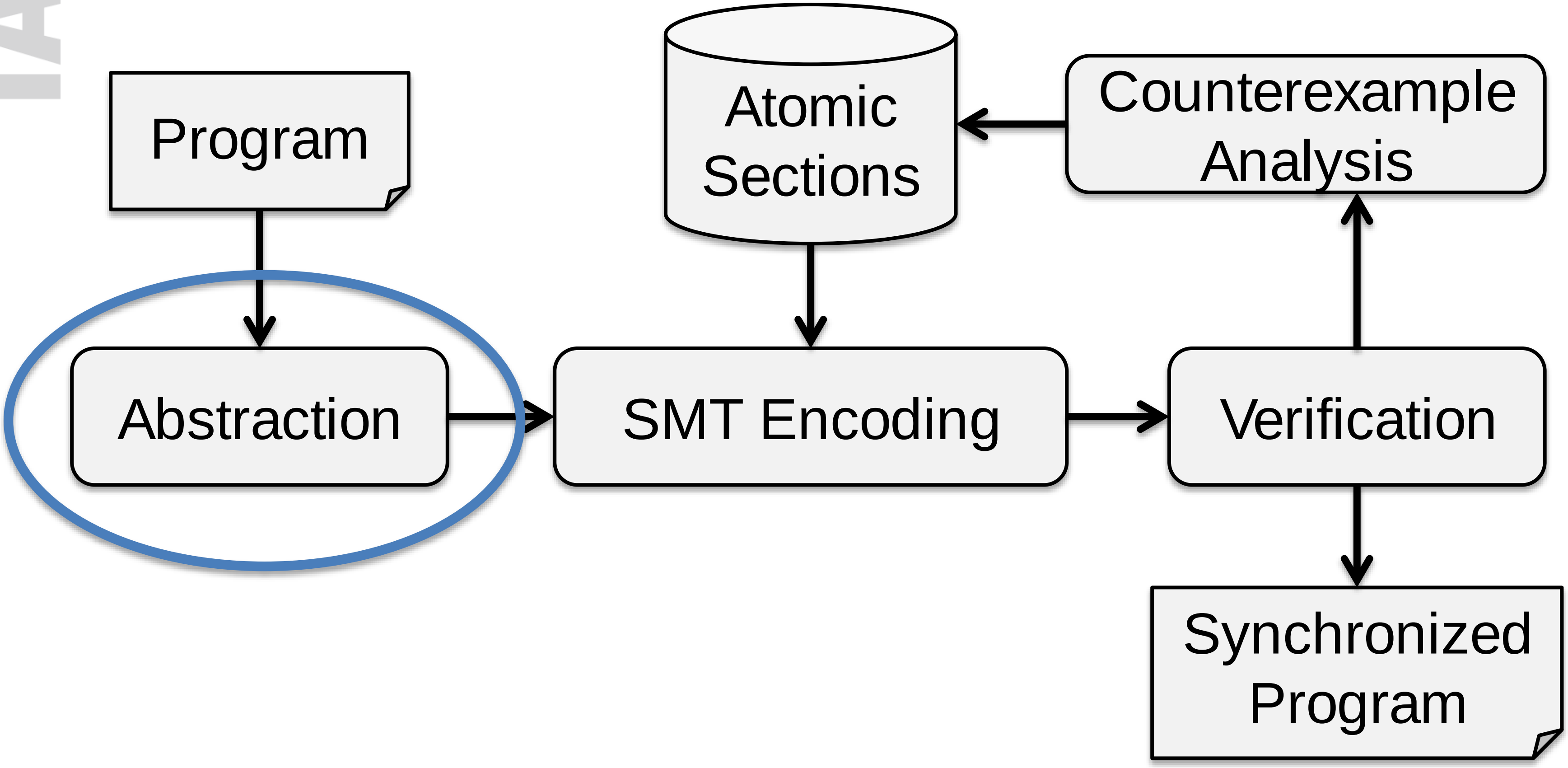
Example:

- RSA decryption using Chinese Remainder Theorem
  - Goal: $m = c^d \bmod (p*q)$
  - Faster: $m_p = c^d \bmod p$ $m_q = c^d \bmod q$ $m = crt(m_p, m_q)$
- Parallelization:

```
thread1() {
  m_p := c^d mod p;
  fin_1 := true;
  if(!merged && fin_2)
    merged := true;
    m_p := crt(m_p, m_q);
}
```

```
thread2() {
  m_q := c^d mod q;
  fin_2 := true;
  if(!merged && fin_1)
    merged := true;
    m_p := crt(m_p, m_q);
}
```

Flow


Program
Atomic Sections
Counterexample Analysis
Abstraction
SMT Encoding
Verification
Synchronized Program

Abstraction

Challenge: Complicated arithmetic

- Synchronization should not depend on arithmetic
- → Abstract using **uninterpreted functions**

```
thread1() {
  m_p := c^d mod p;
  fin_1 := true;
  if(!merged && fin_2)
    merged := true;
    m_p := crt(m_p, m_q);
}
```

```
thread2() {
  m_q := c^d mod q;
  fin_2 := true;
  if(!merged && fin_1)
    merged := true;
    m_p := crt(m_p, m_q);
}
```

Abstraction

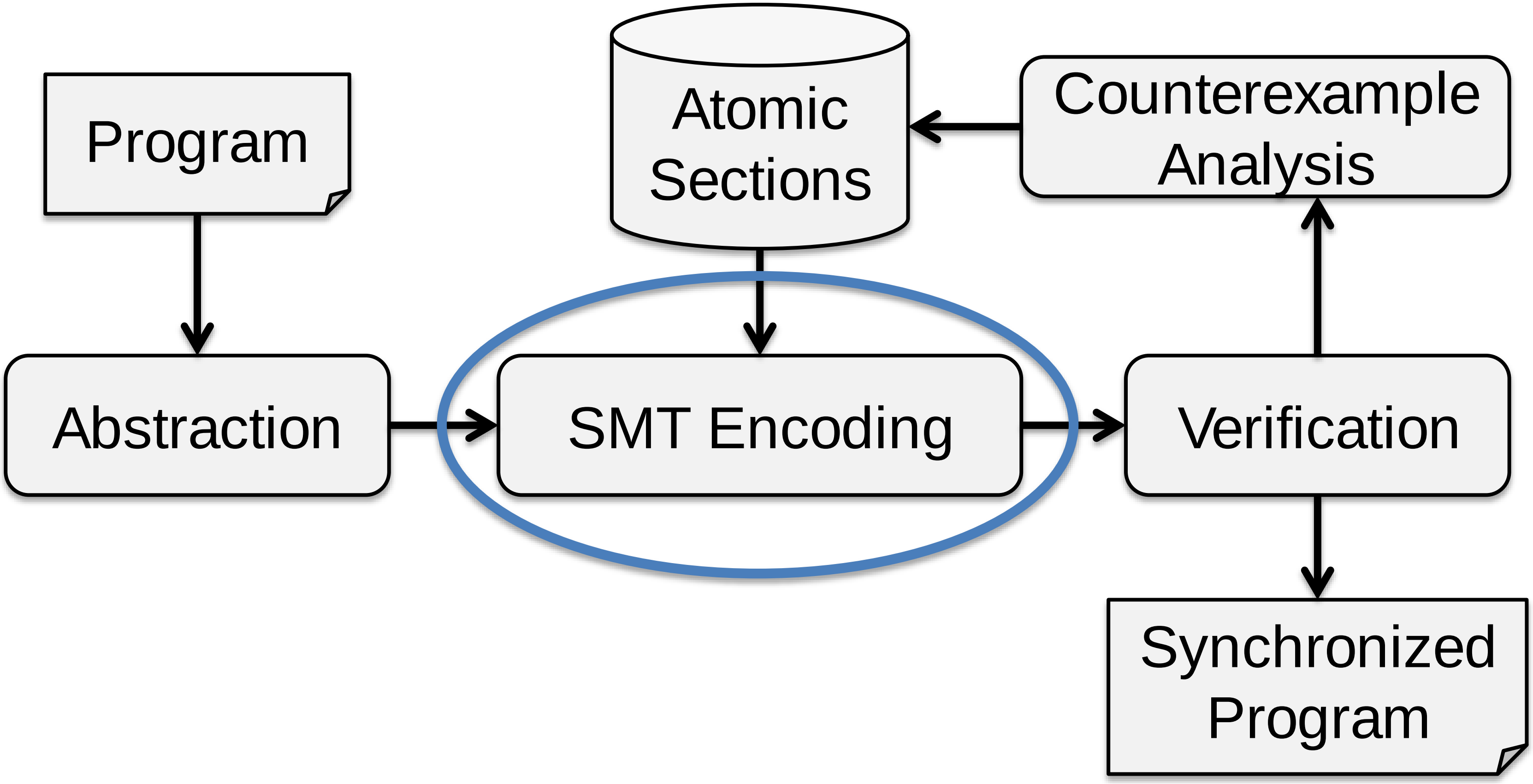
Challenge: Complicated arithmetic

- Synchronization should not depend on arithmetic
- → Abstract using **uninterpreted functions**
  - All arithmetic operations: +,-,*, …
  - Calls of functions without side-effects

```
thread1() {
  m_p := f_me(c, d, p);
  fin_1 := true;
  if(!merged && fin_2)
    merged := true;
    m_p := f_crt(m_p, m_q);
}
```

```
thread2() {
  m_q := f_me(c, d, q);
  fin_2 := true;
  if(!merged && fin_1)
    merged := true;
    m_p := f_crt(m_p, m_q);
}
```

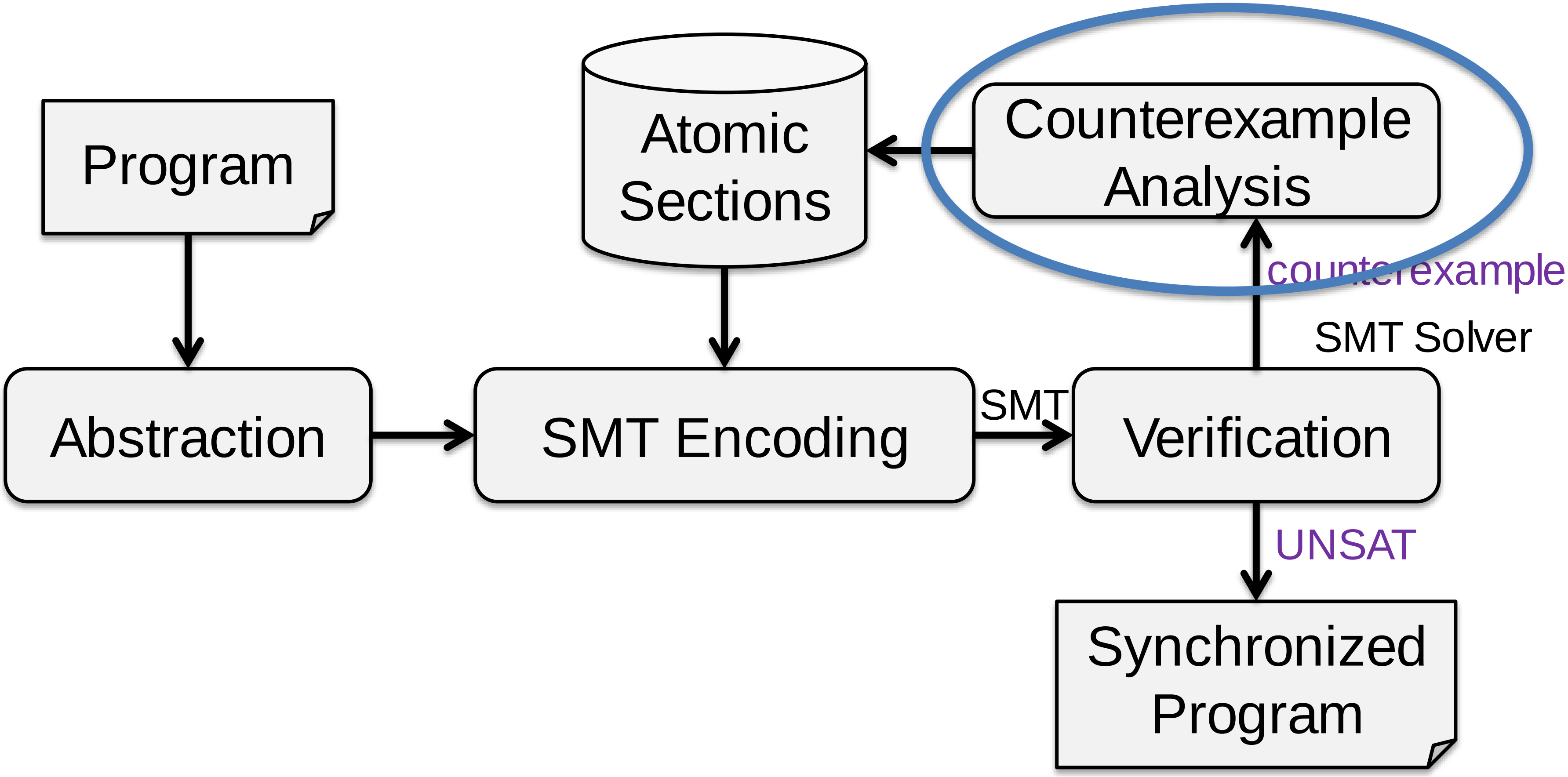
Flow


Program
Atomic Sections
Counterexample Analysis
Abstraction
SMT Encoding
Verification
Synchronized Program

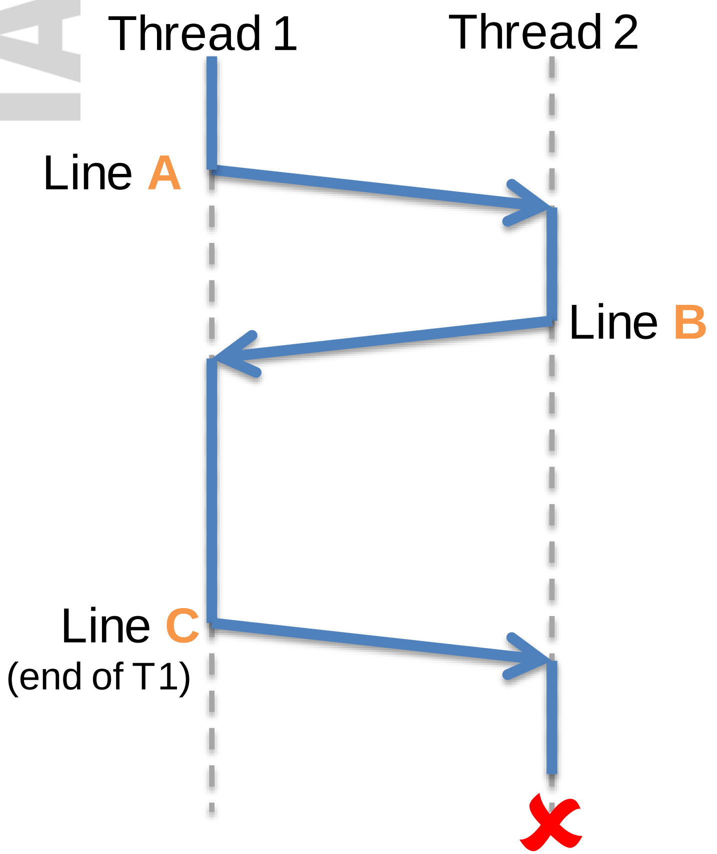
SMT Encoding

- Implicit specification
  - result(Thread1 || Thread2) = result(Thread1 ○ Thread2) or result(Thread2 ○ Thread1)
  - result(): global variables at termination
  - Often called "serializability" or "linearizability"
- Construct SMT formula:
  - incorrect(inputs, scheduling)
  - Satisfying assignment = incorrect execution
- Approach based on Bounded Model Checking [CAV'05]
  - Loops are unrolled
  - Function calls are inlined (or abstracted)

Flow

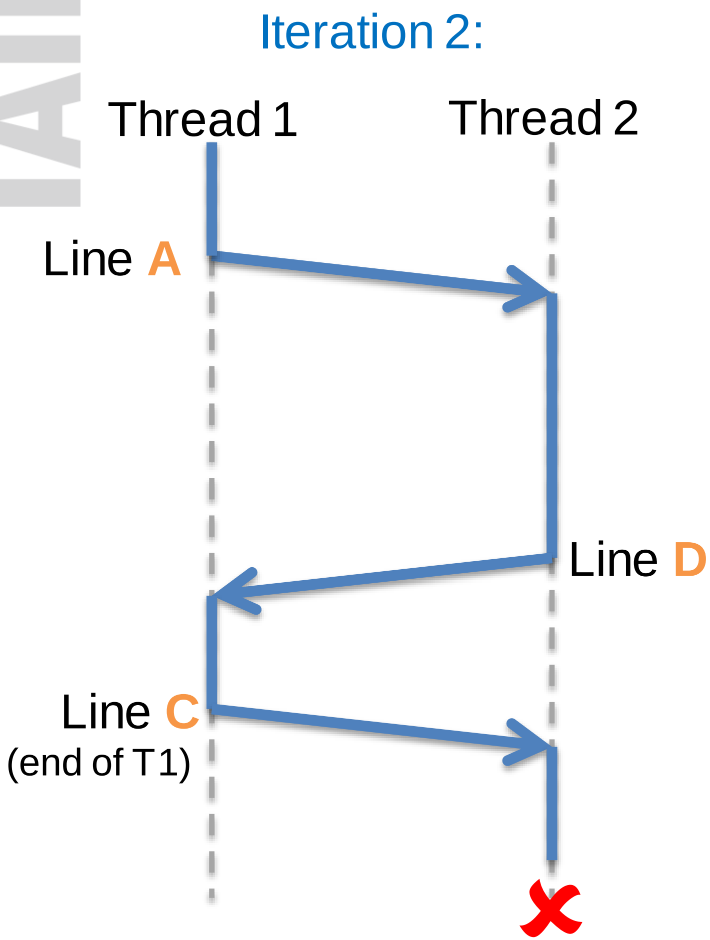

Program
Atomic Sections
Counterexample Analysis
counterexample
SMT Solver
Abstraction
SMT Encoding
SMT
Verification
UNSAT
Synchronized Program

Counterexample Analysis: Method 1 [POPL'10]



- Eliminate counterexample:
  - Atomic section at $A \lor B$

Counterexample Analysis: Method 1 [POPL'10]



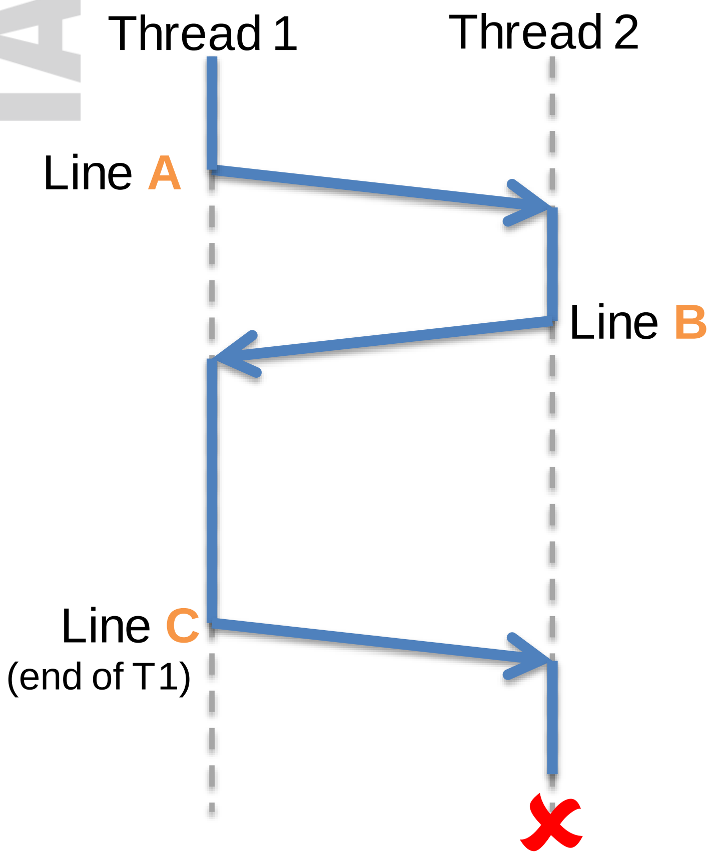
- Eliminate counterexample:
  - Atomic section at $A \vee B$
  - Atomic section at $A \vee D$

Counterexample Analysis: Method 1 [POPL'10]

Iteration 3:

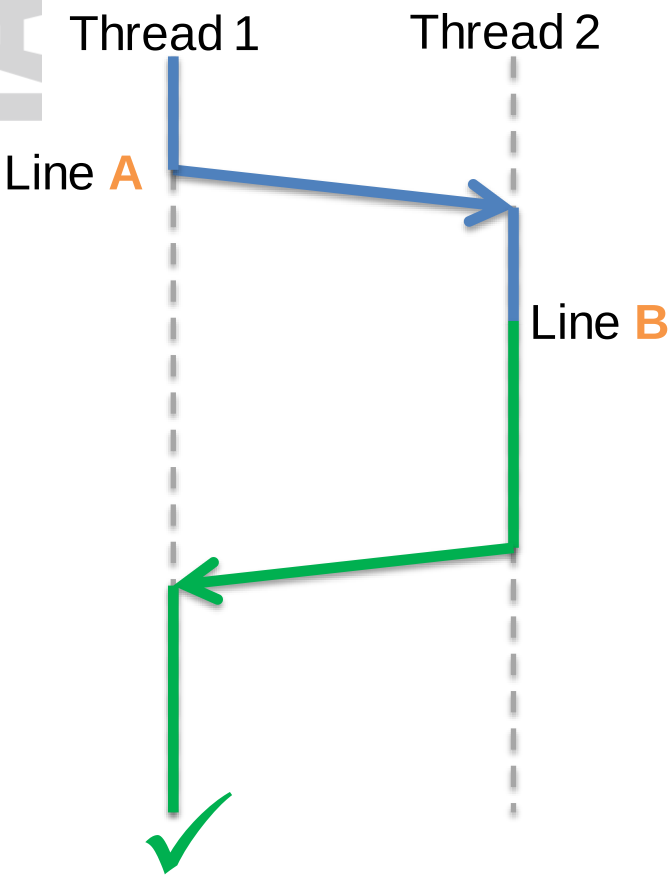
- Eliminate counterexample:
  - Atomic section at $A \vee B$
  - Atomic section at $A \vee D$
- Minimal satisfying assignment
  - → Atomic section at A

Counterexample Analysis: Method 2



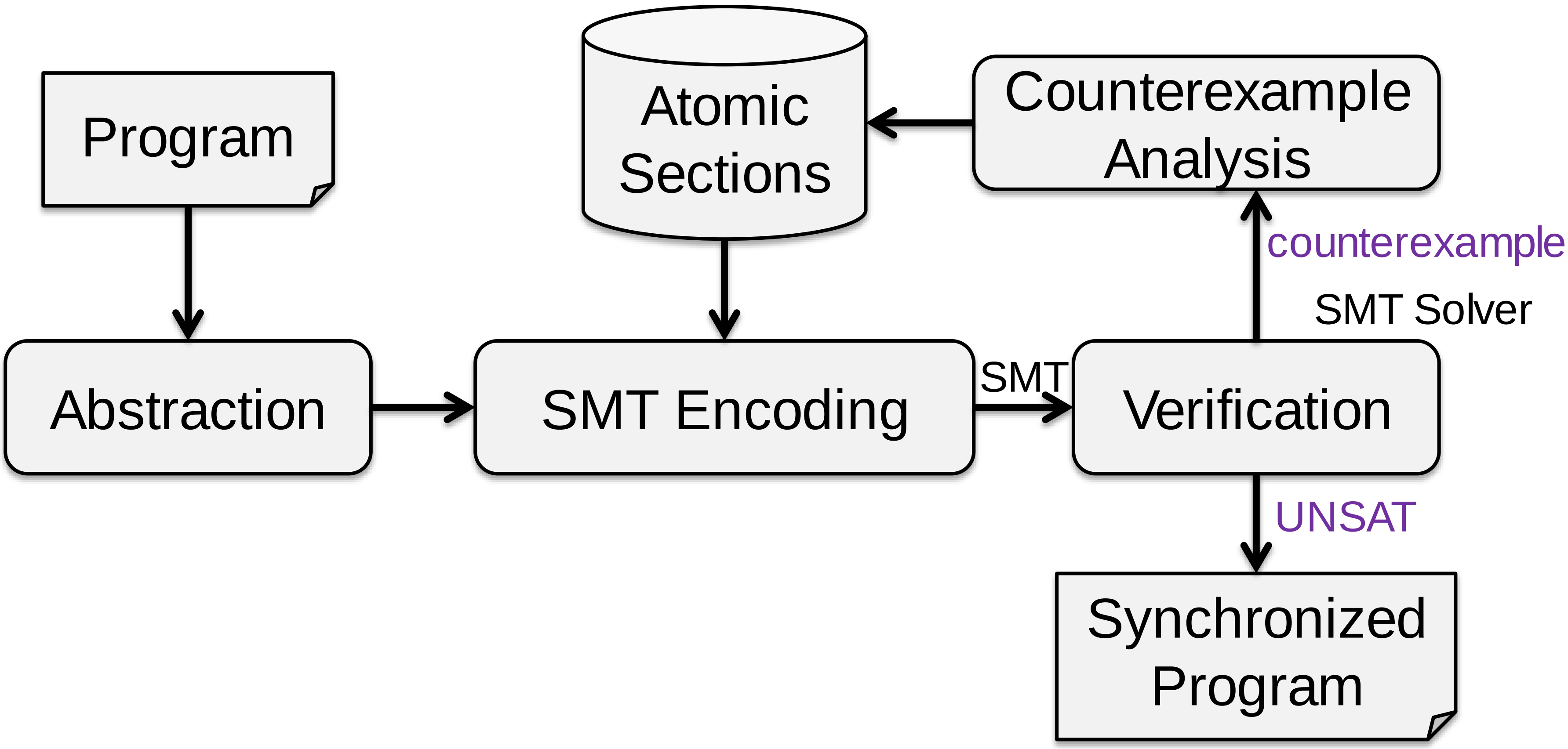
- Start with last (non-mandatory) thread switch **B**
- Can we build a valid run from **B** on?

Counterexample Analysis: Method 2



- Start with last (non-mandatory) thread switch **B**
- Can we build a valid run from **B** on?
  - No? Problem already before
    - Investigate **A** in the same way
  - Yes? **B** is suspicious.
    - Add atomic section at **B**
- This is a heuristic!
  - May not find the minimal solution

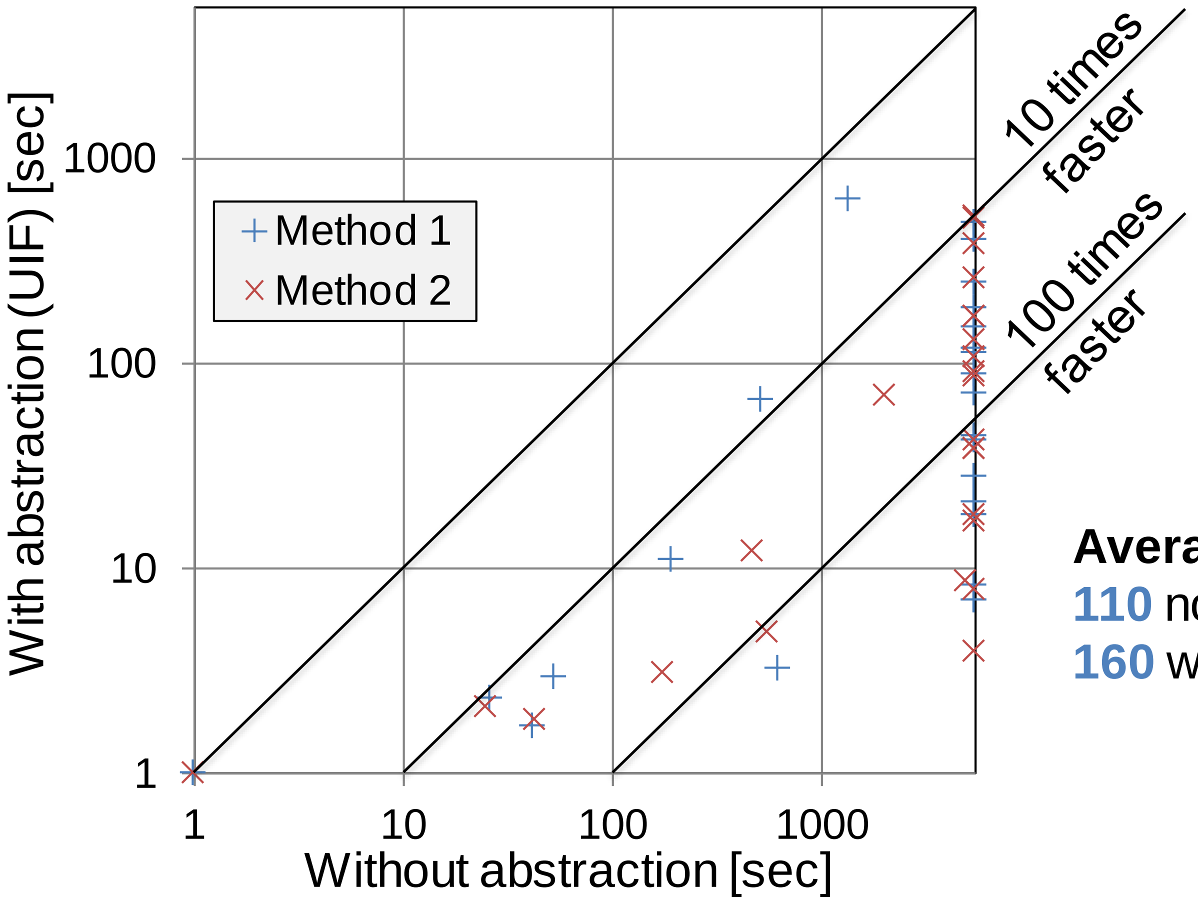
Flow


Program
Atomic Sections
Counterexample Analysis
counterexample
SMT Solver
Abstraction
SMT Encoding
SMT
Verification
UNSAT
Synchronized Program

Experimental Results

- Prototype tool for (simple) C programs
- Toy examples:
  - linEq:
    - Given: linear equation 4a + 3b + 9c -4d = 6
    - Given: assignment a=100, b=0, c=3, d=12
    - Program performs parallelized check
    - Abstraction: **+,*** → **$f_+()$, $f_*()$**
  - VecPrime:
    - Counts prime numbers in a vector
    - Abstraction: **isPrime()** → **$f_p()$**

Experimental Results: Toy Examples
Speedup due to Abstraction



Average speedup factor:
110 not counting time-outs
160 when counting time-outs

Experimental Results

- Real-world examples:
  - CVE-2014-0196 bug in Linux TTY driver
    - Race condition can produce buffer overflow

Experimenta

- Real-world examples:
  - CVE-2014-0196 bug in Lin
    - Race condition can proc

```
int tty_size;
int tty_offset;
int OPOST_tty;
int STATE = 1;
void thread1() {
  int c = 0;
  int nr = 22;
  int b = 77;
  int true_int = 1;
  while(true_int == 1) {
    if(OPOST_tty) {
      STATE = 2;
      while(nr > 0) {
        int num = nr + 3;
        b = b + num;
        nr = nr - num;
        if(nr != 0){
          c = b;
          b = b + 1;
          nr = nr - 1;
        }
      }
    } else {
      STATE = 3;
      while(nr > 0) {
        int tmpOffset = tty_offset;
        int tty_space_left =  tty_size - tmpOffset;
        if( tty_space_left - nr >= 0 )
          c = nr;
        else
          c = tty_space_left;
        tmpOffset = tty_offset;
        tmpOffset = tmpOffset + c;
        tty_offset =  tmpOffset;
        if(c>0){
          b = b + c;
          nr = nr - c;
        }
      }
    }
  }
}
```

atomic section

Experimental Results

- Real-world examples:
  - CVE-2014-0196 bug in Linux TTY driver
    - Race condition can produce buffer overflow
  - Race condition in iio-subsystem of linux-kernel
    - Variable that counts the number of running threads
  - Race condition in broadcom tigon3 ethernet driver
    - Statistics can get inconsistent

Experimental Results: Real-World Bugs

- TTY and Tigon3:
  - Our tool finds exactly the suggested fix
- IIO:
  - Our tool finds a slightly different fix
- No user-defined specification necessary
  - Serialzability as implicit specification is enough
- Execution times [sec]:

	Without Abstraction		With Abstraction	
	Method 1	Method 2	Method 1	Method 2
TTY	11	13	4.1	5.8
IIO	1.1	1.3	0.9	1.1
Tigon3	17	21	9.8	13

Summary and Conclusions

Highlights:

- No manual specifications → usability
- Abstraction with uninterpreted functions → scalability

- Proof-of-concept implementation
  - http://www.iaik.tugraz.at/content/research/design verification/atoss/

Future work:

- Abstraction refinement (e.g., associativity, commutativity), other abstractions, loops, …

References

[CAV'05] I. Rabinovitz and O. Grumberg. Bounded model checking of concurrent programs. In CAV'05, LNCS 3576. Springer, 2005.

[POPL'10] M. T. Vechev, E. Yahav, and G. Yorsh. Abstraction-guided synthesis of synchronization. In POPL'10. ACM, 2010.