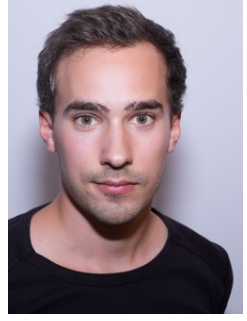
Formale Spezifikation und Verifikation 2
Praktikum

Matthias Dangl, Dr. Marie-Christine Jakobs,
and Thomas Lemberger

SoSy-Lab, LMU Munich, Germany





LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

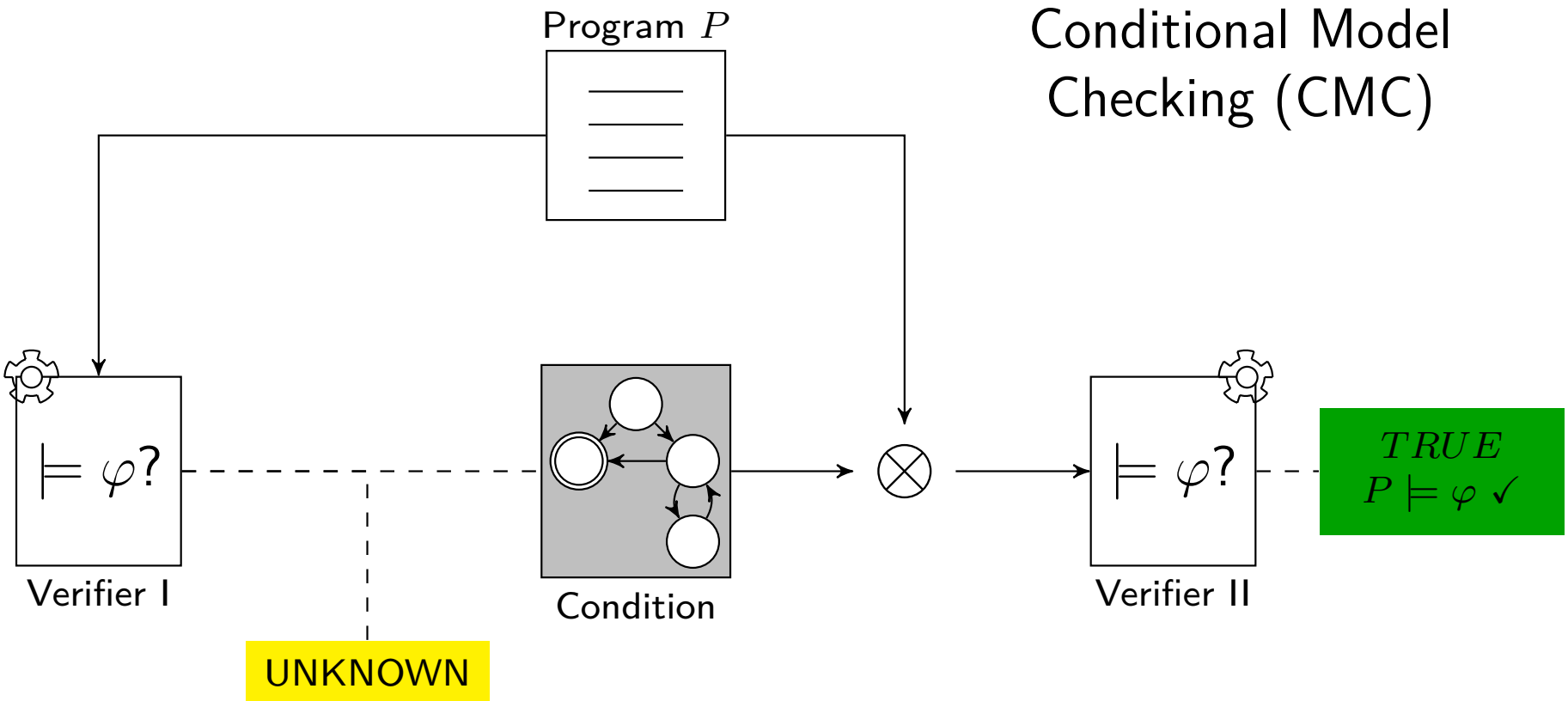
CPA

SoSy-Lab
Software Systems

Dr. Marie-Christine Jakobs

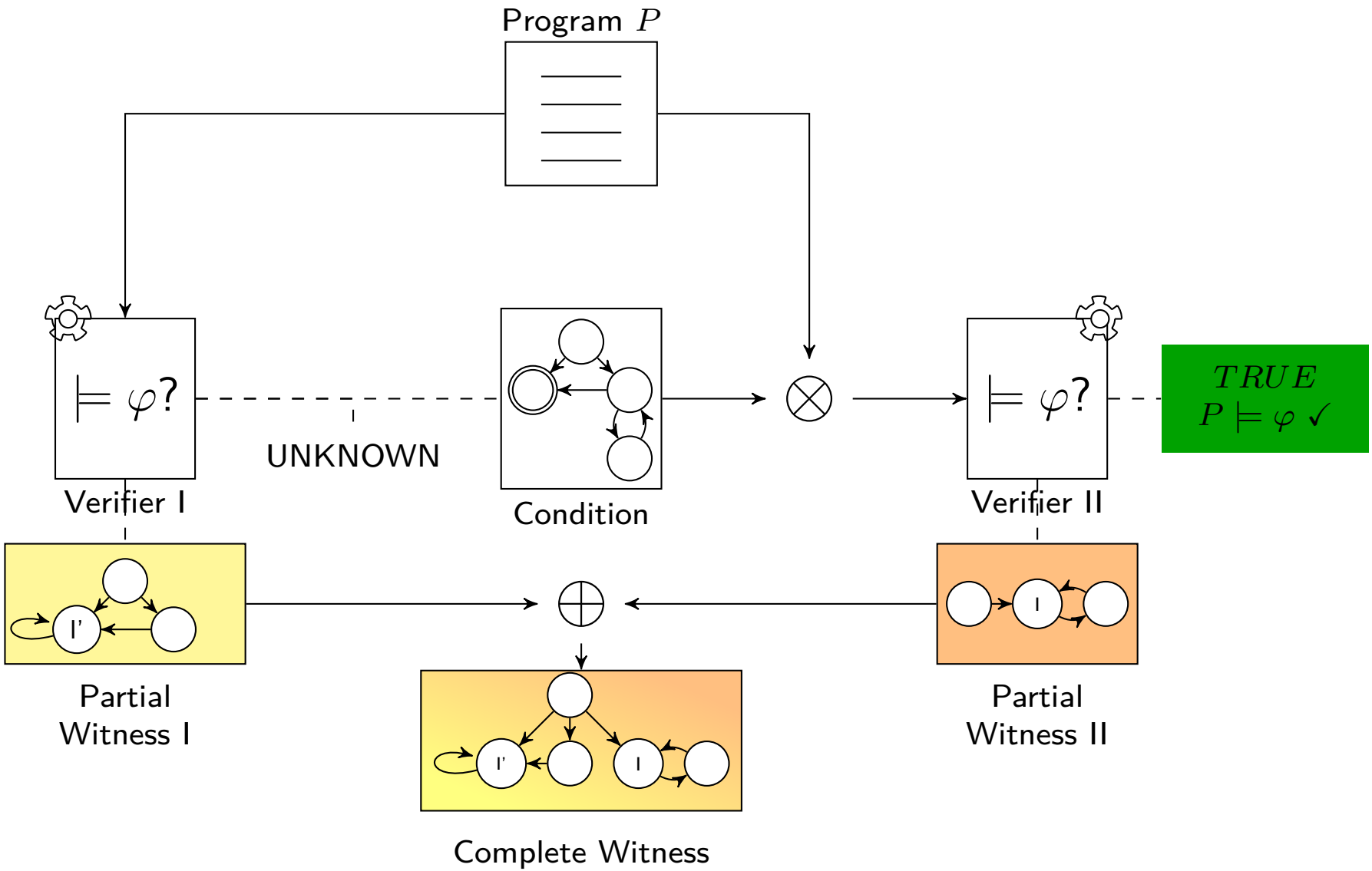
Office: F 007
Email: jakobs@sosy.ifi.lmu.de

Topic 1: Correctness Witness for CMC



Problem: How to witness the final outcome?

Topic 1: Correctness Witness for CMC (Task)



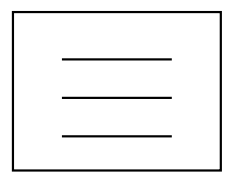
Topic 1: Task Details

1. Correctness witnesses for native CMC
   - Exploration in Verifier II steered by condition
   - Verifier I, Verifier II same tool
2. Correctness witnesses for CMC with residual program
   - Generate residual program from original program and condition
   - Verifier II checks residual program
   - Varying difficulty levels
     - 2.1 Residual program subgraph of original program
       Verifier I, Verifier II same tool
     - 2.2 Residual program subgraph of original program
       Verifier I, Verifier II different tools
     - 2.3 Residual program arbitrary unfolding of original program
       Verifier I, Verifier II same tool
     - 2.4 Residual program arbitrary unfolding of original program
       Verifier I, Verifier II different tool
3. Witnesses for CMC considering more than two verifiers

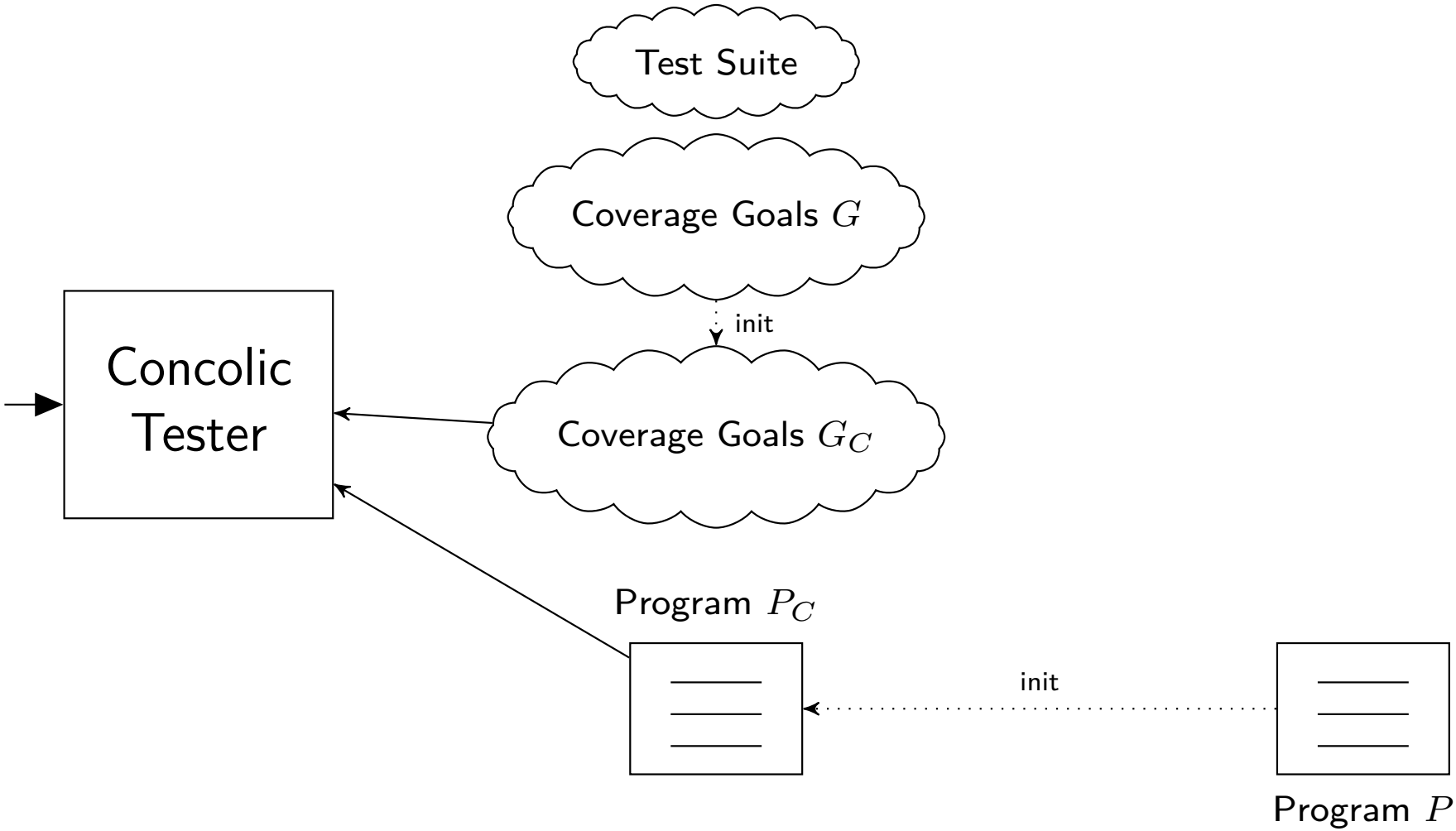
Topic 1: Literature

1. Conditional Model Checking
   - D. Beyer et al.: Conditional model checking: a technique to pass information between verifiers. FSE. ACM (2012).
   - D. Beyer et al.: Conditional Model Checking. ArXive http://arxiv.org/abs/1109.6926 (2011).
2. Residual Program Generation
   - M. Czech et al.: Just Test What You Cannot Verify!. FASE. Springer (2015).
3. Witnesses
   - D. Beyer et al.: Correctness witnesses: exchanging verification results between verifiers. FSE. ACM (2016).
   - D. Beyer et al.: Software Verification and Verifiable Witnesses. TACAS. Springer (2015).
   - M.-C. Jakobs: $PART_{PW}$: From Partial Analysis Results to a Proof Witness. SEFM. Springer (2017).

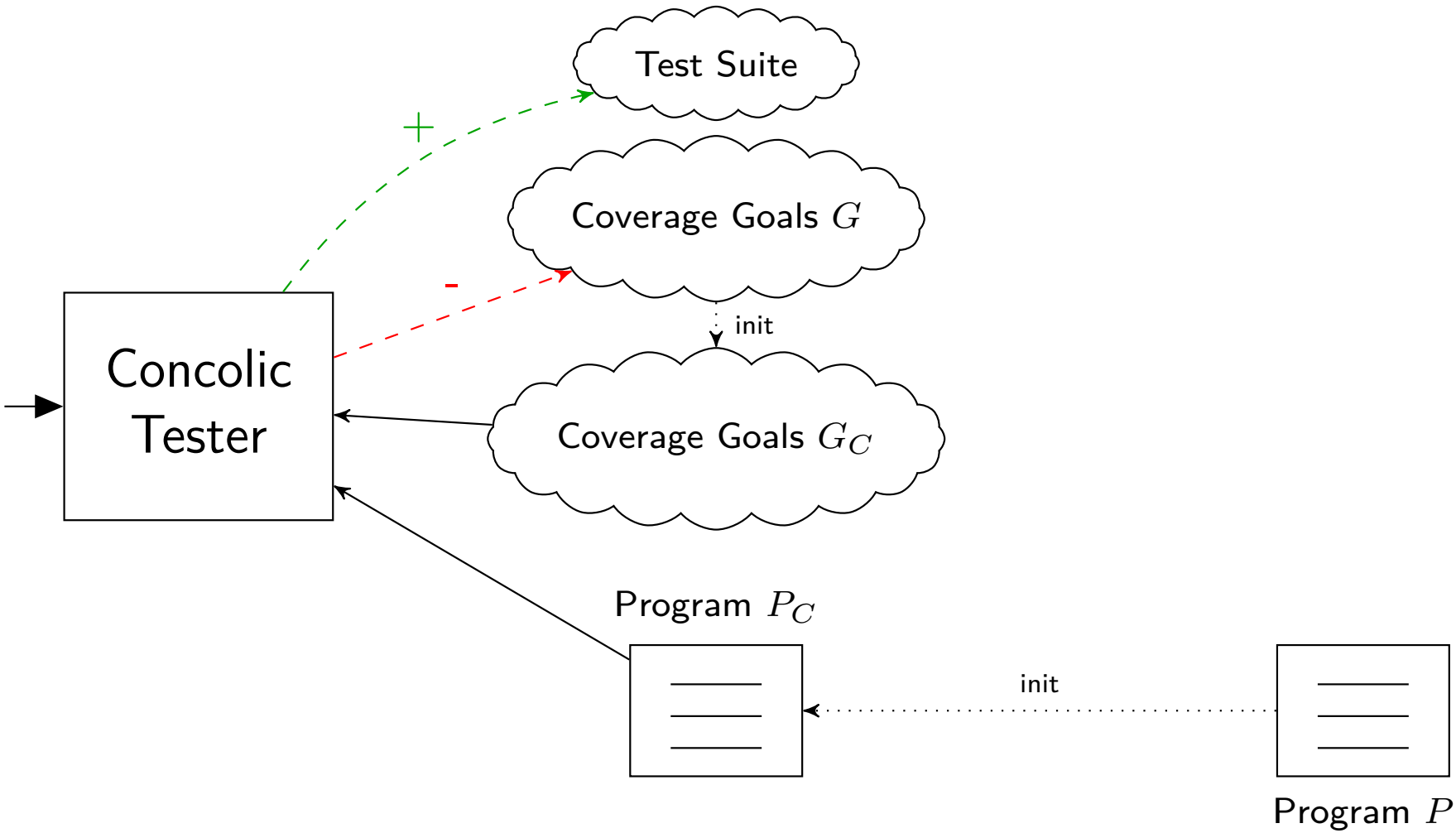
Topic 2: Abstraction-driven Concolic Testing


Program P

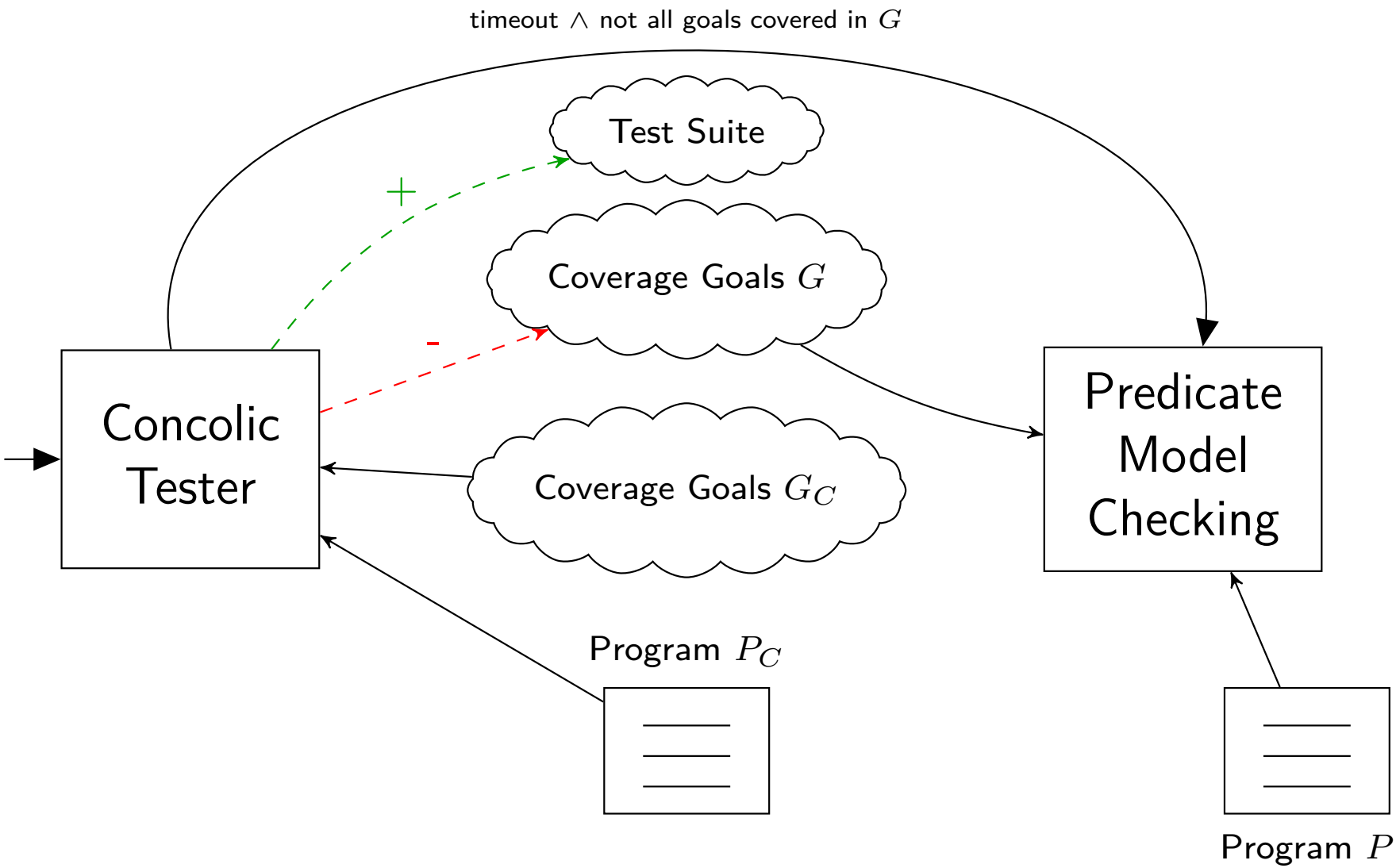
Topic 2: Abstraction-driven Concolic Testing



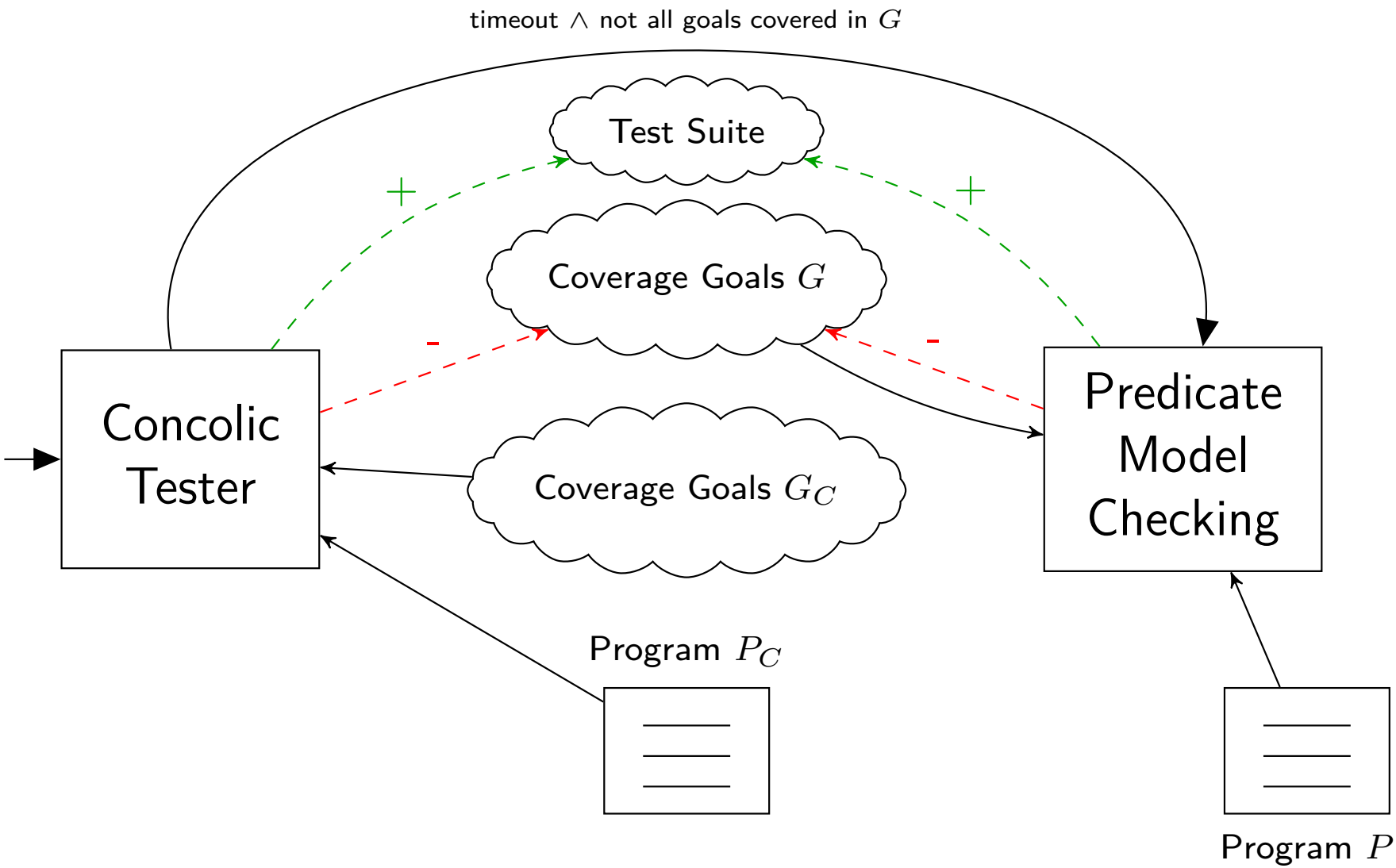
Topic 2: Abstraction-driven Concolic Testing



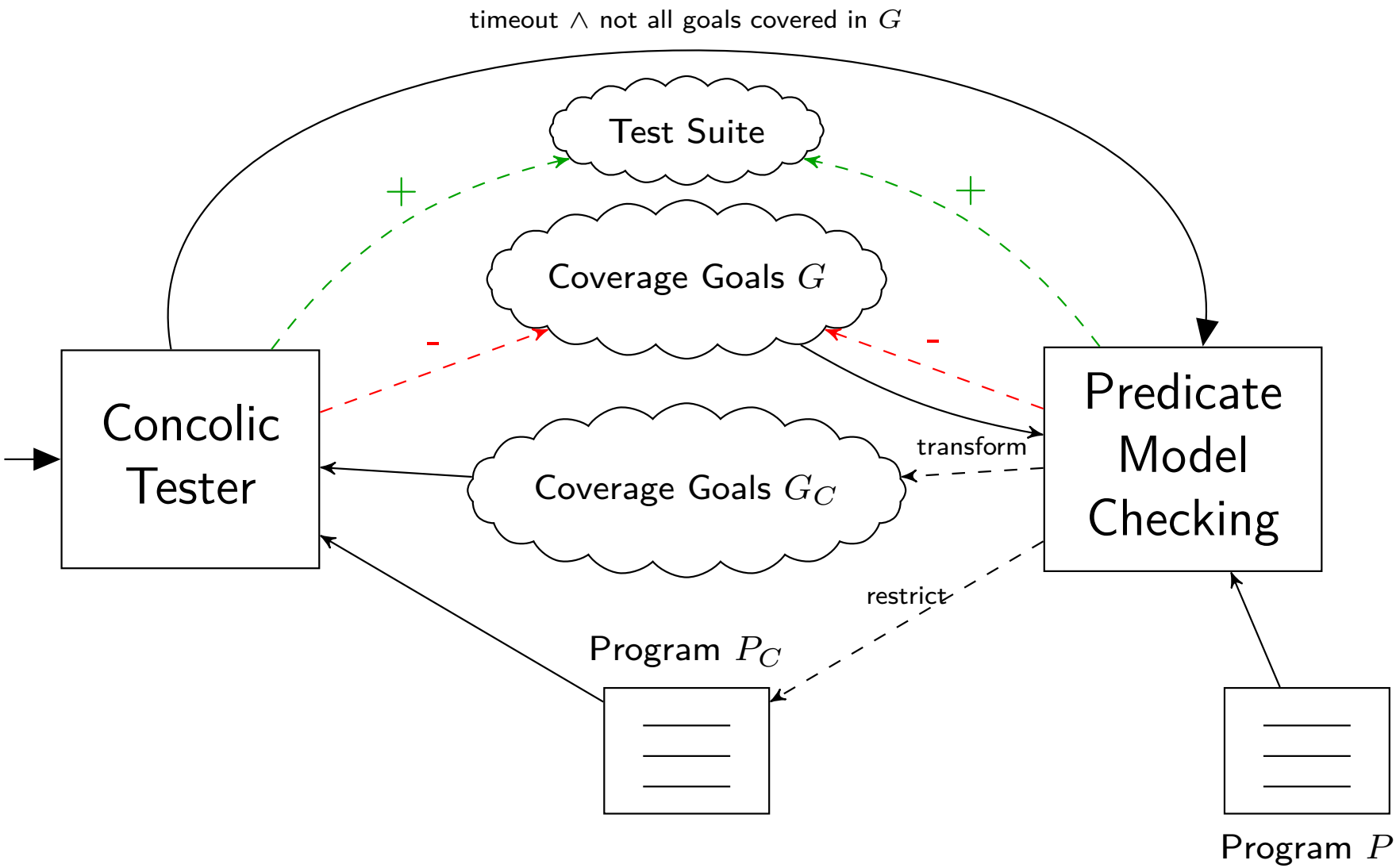
Topic 2: Abstraction-driven Concolic Testing



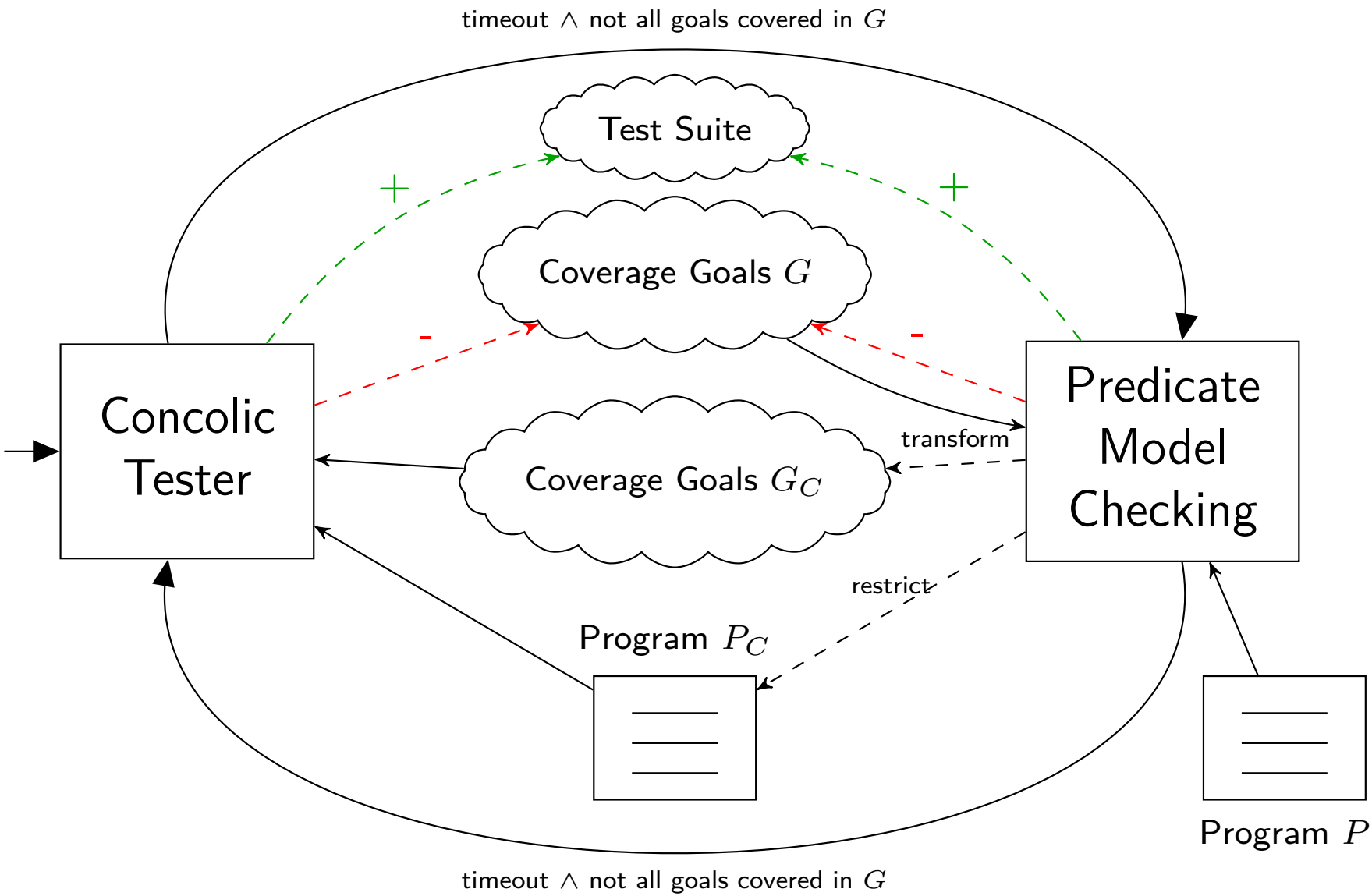
Topic 2: Abstraction-driven Concolic Testing



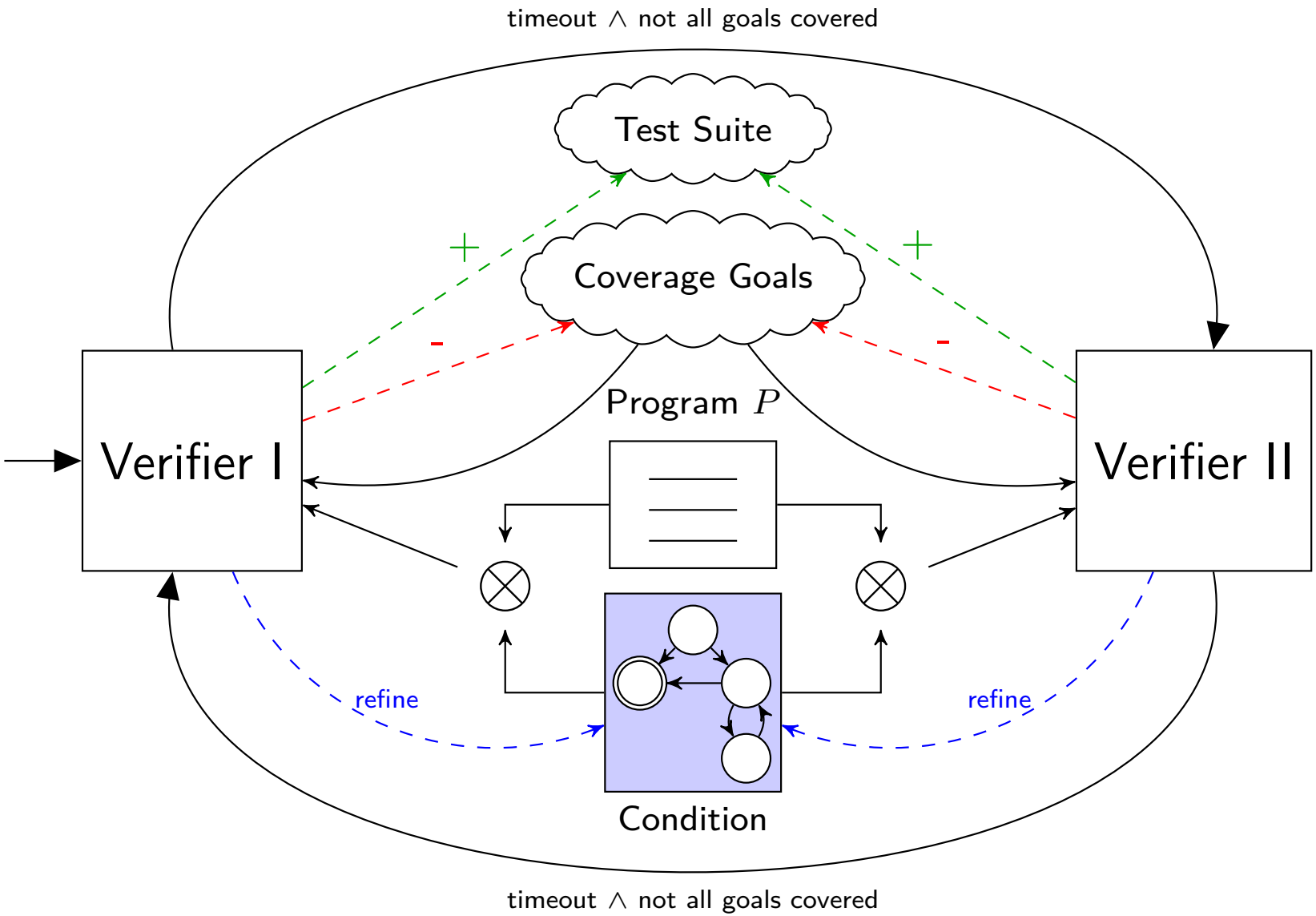
Topic 2: Abstraction-driven Concolic Testing



Topic 2: Abstraction-driven Concolic Testing



Topic 2: CPAchecker-Abstraction-driven Testing



Topic 2: Task Details

1. Interleaved test suite generation
   - Two arbitrary verifiers
   - Only shared goal set (no cooperation via condition)
2. Interleaved, cooperative test suite generation
   - Unidirectional: one verifier creates, the other uses condition
   - Bidirectional: both verifiers create and use condition
   - Evaluate what is best, no cooperation, unidirectional, bidirectional
3. Interleaved test suite generation with information reuse
   - Reuse precision
   - Verifiers continue with ARG from previous execution
   - Comparison against complete restart

Topic 2: Literature

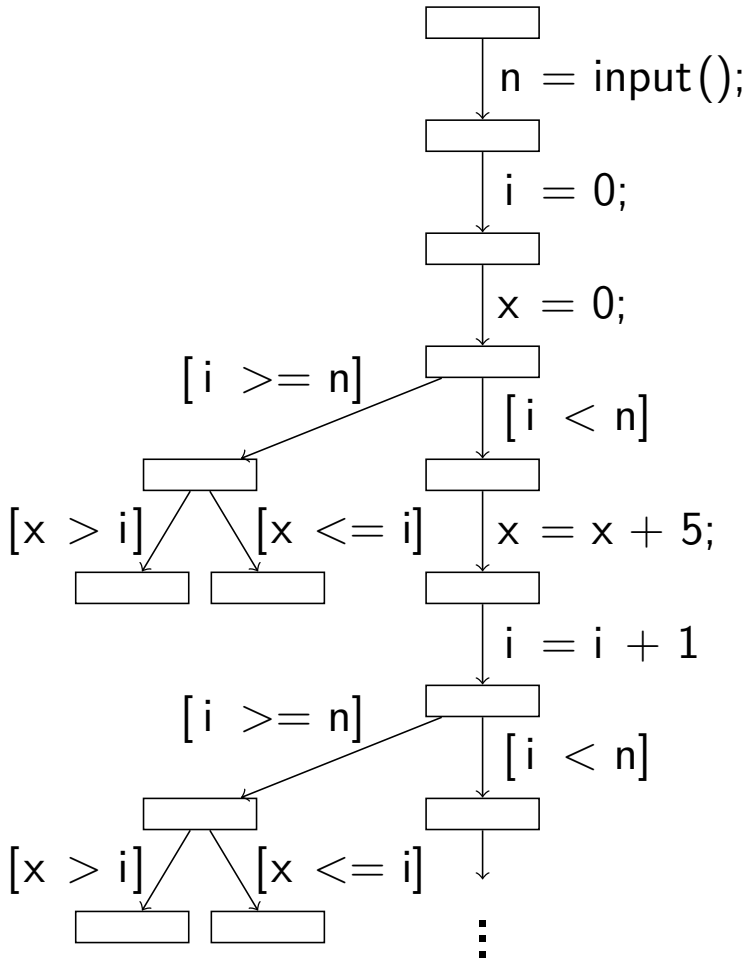
- Abstraction-driven Concolic Testing
  - P. Daca et al..: Abstraction-driven Concolic Testing. VMCAI. Springer (2016).
- Conditional Model Checking
  - D. Beyer et al.: Conditional model checking: a technique to pass information between verifiers. FSE. ACM (2012).
  - D. Beyer et al.: Conditional Model Checking. ArXive http://arxiv.org/abs/1109.6926 (2011).
- Information Reuse
  - T. A. Henzinger et al.: Extreme Model Checking. Verification: Theory and Practice. Springer (2003).
  - D. Beyer et al.: Precision reuse for efficient regression verification. ESEC/FSE. ACM (2013).

Thomas Lemberger

Office:	F 004
Email:	lemberger@sosy.ifi.lmu.de
PGP Public Key:	0x033DE66F
PGP Fingerprint:	BBC4 36E1 F2BA BA4E 8E81 872E 9787 7E1F 033D E66F

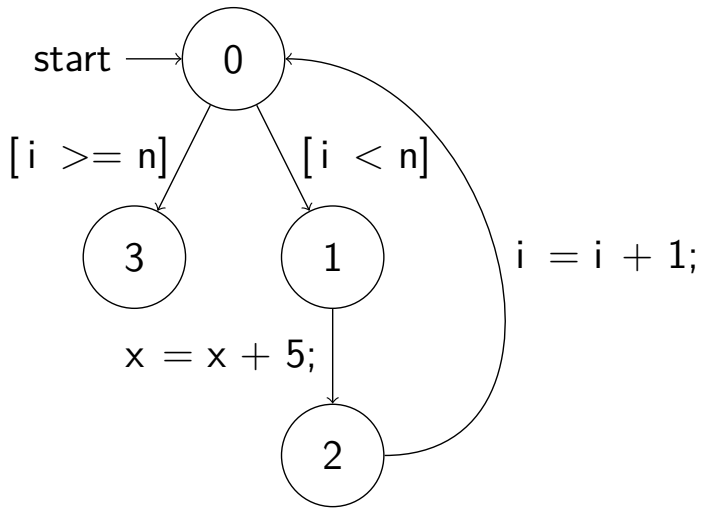
Topic 3: Compact Symbolic Execution

```
int main(void) {
  int n = input();
  int i = 0;
  int x = 0;
  while (i < n) {
    x = x + 5;
    i = i + 1;
  }
  assert(x > i);
  return 0;
}
```



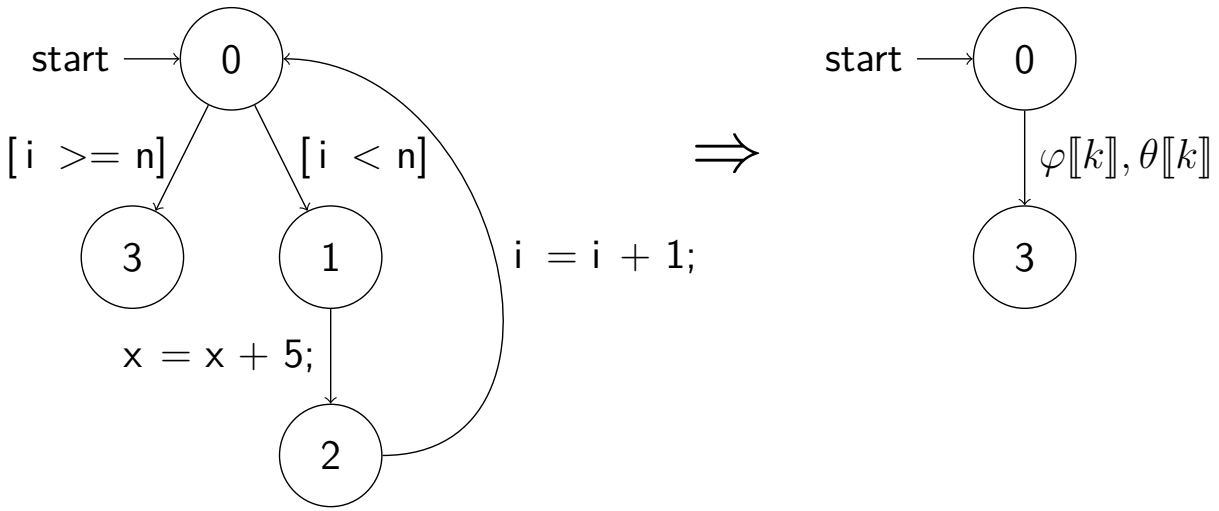
Topic 3: Compact Symbolic Execution

```
int main(void) {
  int n = input();
  int i = 0;
  int x = 0;
  while (i < n) {
    x = x + 5;
    i = i + 1;
  }
  assert(x > i);
  return 0;
}
```



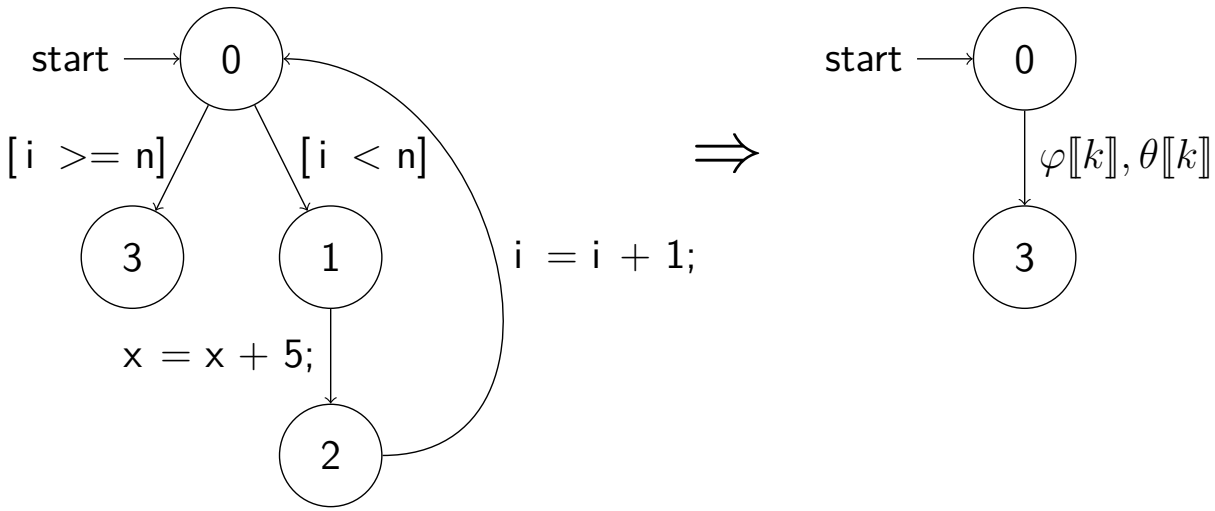
Topic 3: Compact Symbolic Execution

```
int main(void) {
  int n = input();
  int i = 0;
  int x = 0;
  while (i < n) {
    x = x + 5;
    i = i + 1;
  }
  assert(x > i);
  return 0;
}
```



Topic 3: Compact Symbolic Execution

```
int main(void) {
  int n = input();
  int i = 0;
  int x = 0;
  while (i < n) {
    x = x + 5;
    i = i + 1;
  }
  assert(x > i);
  return 0;
}
```



$$\varphi[\![k]\!] = k \geq 0 \wedge \forall \tau (0 \leq \tau < k \Rightarrow i + \tau < n_0) \wedge i + k \geq n_0$$

$$\theta[\![k]\!](i) = i_0 + k,$$
$$\theta[\![k]\!](x) = x_0 + 5k,$$
$$\theta[\![k]\!](n) = n_0$$

Topic 3: Task Details

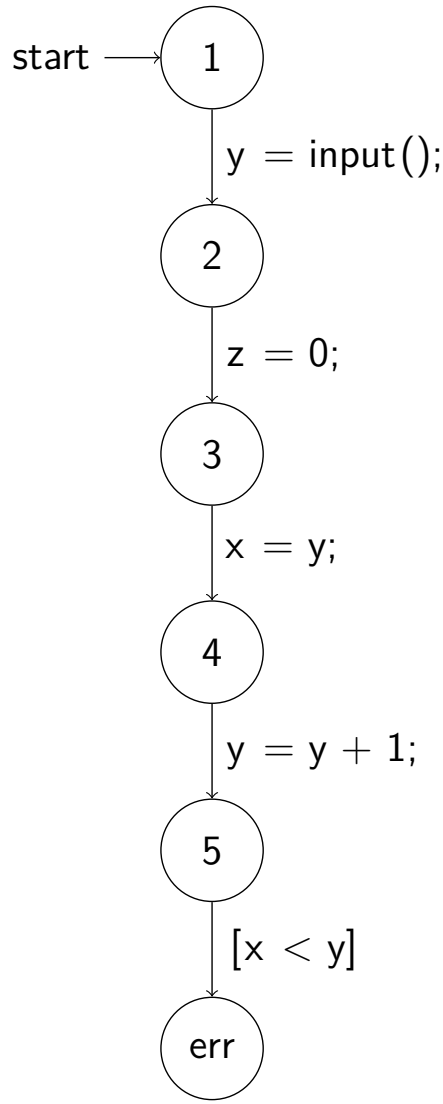
1. Implement algorithm in CPAchecker
2. Evaluate on large set of benchmarks, compare with plain symbolic execution and symbolic execution + CEGAR
3. Identify and explain strengths/weaknesses of algorithm
4. (Mitigate weaknesses)

Topic 3: Literature

- King: Symbolic Execution and Program Testing, Comm. o. ACM 1976.
- Slaby. Strejcek, Trtik: Compact Symbolic Execution, ATVA 2013.
- Beyer, Lemberger: Symbolic Execution with CEGAR, ISoLA 2016.

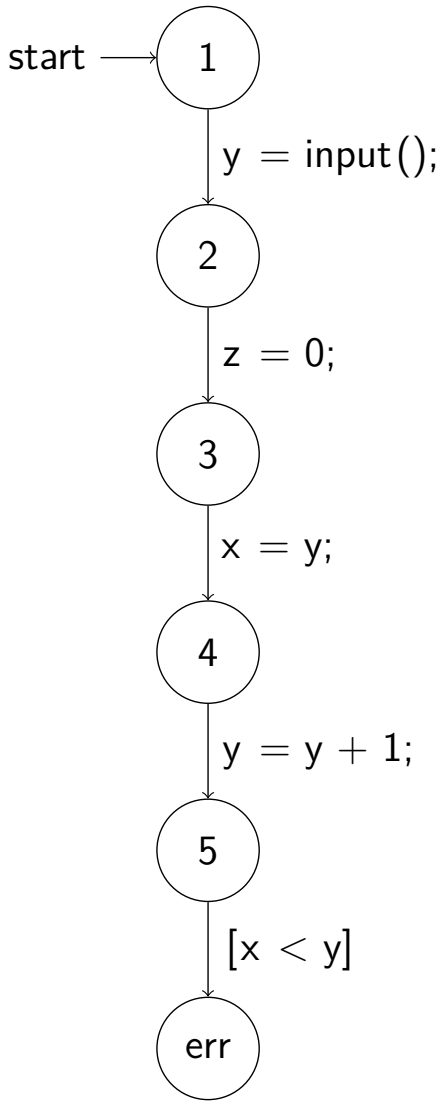
Topic 4: MaxSAT for Fault Localization

```
int main(void) {
    int y = input();
    int z = 0;
    int x = y;
    y++;

    assert (x >= y);
}
```



Topic 4: MaxSAT for Fault Localization

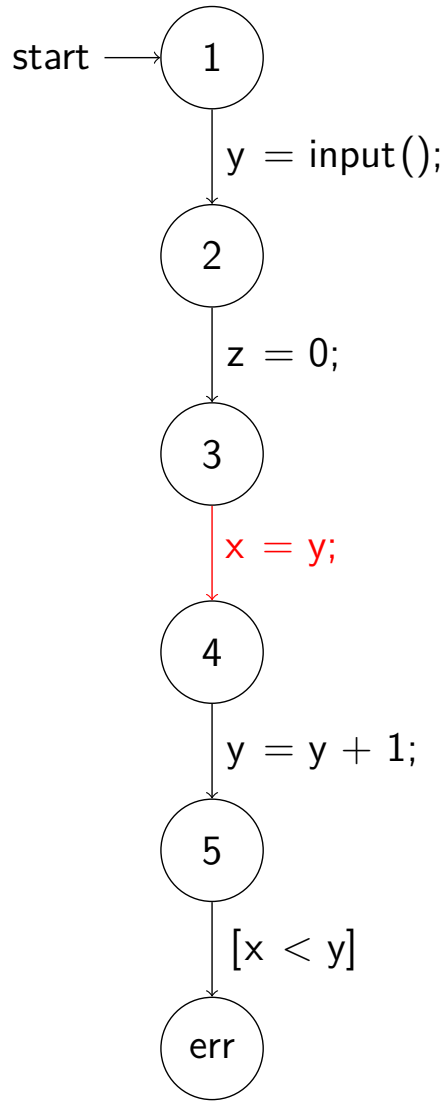
```
int main(void) {
    int y = input();
    int z = 0;
    int x = y;
    y++;

    assert (x >= y);
}
```



$$\mathbf{y = 0} \wedge z = 0 \wedge x = y \wedge y' = y + 1 \wedge \mathbf{x \geq y'}$$

Topic 4: MaxSAT for Fault Localization

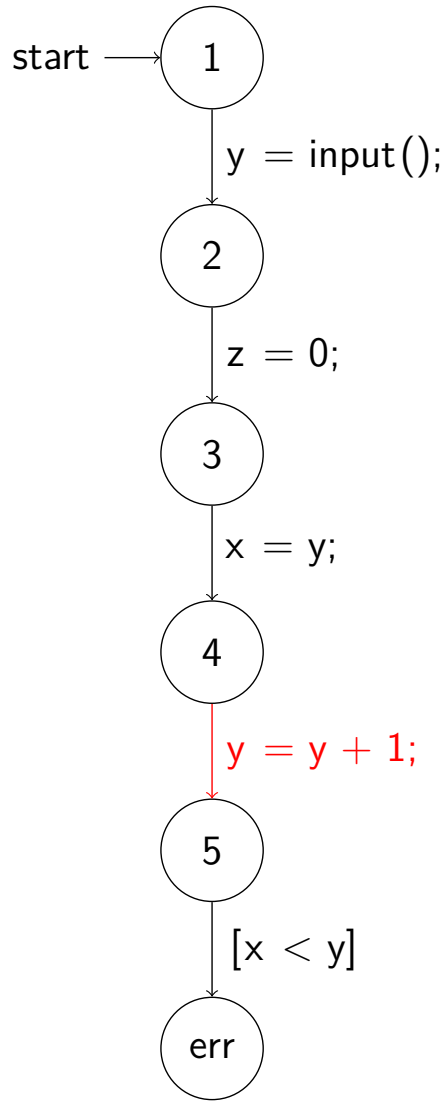
```
int main(void) {
    int y = input();
    int z = 0;
    int x = y;
    y++;

    assert (x >= y);
}
```



$$\mathbf{y = 0} \wedge z = 0 \wedge x = y \wedge y' = y + 1 \wedge \mathbf{x \geq y'}$$

Topic 4: MaxSAT for Fault Localization

```
int main(void) {
    int y = input();
    int z = 0;
    int x = y;
    y++;

    assert (x >= y);
}
```



$$\mathbf{y = 0} \wedge z = 0 \wedge x = y \wedge y' = y + 1 \wedge \mathbf{x \geq y'}$$

Topic 4: Task Details

1. Implement approach in CPAchecker
2. Design evaluation strategy (e.g., using known bugs, bug seeding)
3. Evaluate approach (strengths/weaknesses)

Topic 4: Literature

- Jose, Majumdar: Bug-Assist: Assisting Fault Localization in ANSI-C Programs, CAV 2011.
- Jose, Majumdar: Cause Clue Clauses: Error Localization using Maximum Satisfiability, PLDI 2011.
- Marques-Silva, Planes: On Using Usatisfiability for Solving Maximum Satisifiability, 2007.

Matthias Dangl

Office:	F 010
Email:	dangl@sosy.ifi.lmu.de
PGP Public Key:	0xD8BF1A0B19A1702D
PGP Fingerprint:	36E9 C2E7 CF98 ACF5 BE5C 6EC2 D8BF 1A0B 19A1 702D

Topic 5: Loop-Invariant Generation with SymPy

```
int main(void) {
  int i = 0;
  int x = 3;
  while (i < 100) { // x == 5 * i + 3
    x = x + 5;
    i = i + 1;
  }
  return 0;
}
```

Topic 5: Task Details

1. Integrate SymPy recurrence solver into CPAchecker using Jython
2. Evaluate technique on a large set of benchmarks
3. Integrate technique with existing algorithms (e.g. k-induction)
4. Evaluate effects on effectiveness and efficiency

Topic 5: Literature

1. Kreuzer, Robbiano: Computational commutative algebra 1.
2. Loop-invariant generation
   - Kovács: Invariant generation for p-solvable loops with assignments. CSR 2008.
   - Kovács, Voronkov: Finding loop invariants for programs over arrays using a theorem prover. FASE 2009.
3. Application of invariant generation to k-induction:
   - Beyer, Dangl, Wendler: Boosting k-induction with continuously-refined invariants. CAV 2015.
   - Brain, Joshi, Kröning, Schrammel: Safety verification and refutation by k-invariants and k-induction. SAS 2015.

Topic 6: Impact Refinement with PDR

```
int main(void) {
  int x = 1;
  int y = 1;
  while (*) {
    y = y + x;
    x = x + 1;
    assert(x >= 1); // trivially inductive
  }
}
```

Topic 6: Impact Refinement with PDR

```
int main(void) {
  int x = 1;
  int y = 1;
  while (*) {
    y = y + x;
    x = x + 1;
    assert(y >= 1); // requires x >= 0
  }
}
```

Topic 6: Task Details

1. Adopt clause-learning strategy of PDR as alternative to interpolation during abstraction refinement in the PredicateCPA
2. Conduct extensive experimental evaluation of new strategy, old strategies, and combinations on a large set of benchmarks

Topic 6: Literature

1. PDR (aka IC3):
   - Bradley: SAT-Based Model Checking without Unrolling. VMCAI 2011.
   - Bradley: Understanding IC3. SAT 2012.
2. IMPACT:
   McMillan: Lazy abstraction with interpolants. CAV 2006.
3. PredicateCPA:
   Beyer, Dangl, Wendler: A Unifying View on SMT-Based Software Verification. JAR 2017.
4. Tree-IC3:
   Cimatti, Griggio: Software model checking via IC3. CAV 2012.

Course Organization

Dates

Presentation:	25 min + 5 min	08.03.2018 12:15:00 CET
Implementation:	code in trunk	18.03.2018 23:59:59 AoE
Paper:	ca. 6 to 12 pages	31.03.2018 23:59:59 AoE

Grading Weight

Presentation:	30 %
Implementation:	50 %
Paper:	20 %

Assignment of Topics

- One or two students per topic
- Choose until Tuesday, 12.12.2017, 23:59:59 CET
- As an ordered list of three topics
- To lemberger@sosy.ifi.lmu.de
- FCFS decides same-priority conflicts