

# FShell: Systematic Test Case Generation for Dynamic Analysis and Measurement



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

forlsybel

*Joint work with Andreas Holzer,  
Christian Schallhart, and Helmut Veith*

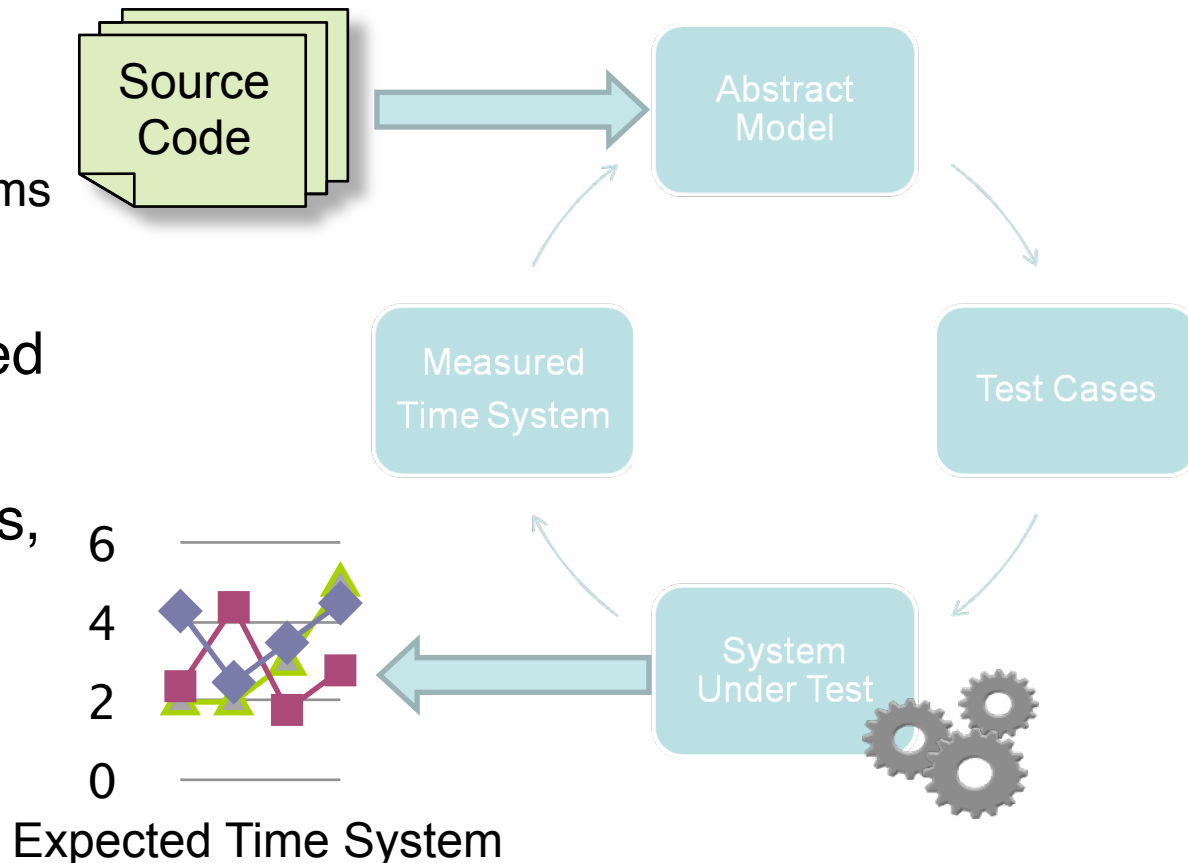
# Does anybody need testing (any more)?



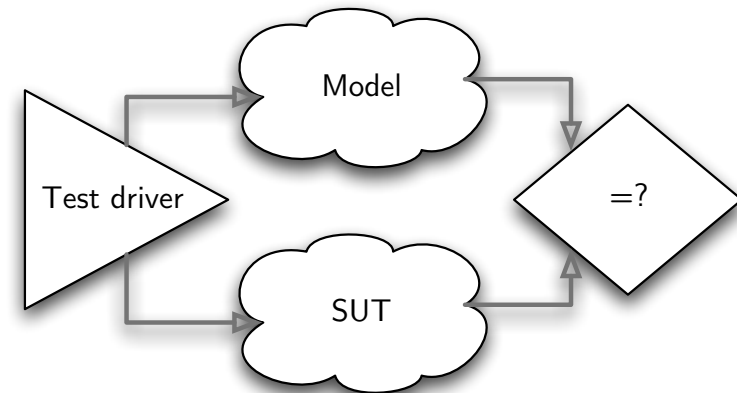
- Software developers are accustomed to the use of testing
- Testing does not require complete specifications
- The purpose of testing is not only falsification
  - Test data required for execution time measurement
  - Tests as certificates

# FORTAS (Formal Timing Analysis Suite)

- Execution time analysis in a white box setting
  - C source code
  - Focus on embedded systems
- Abstracts from platform
- Execution times obtained through measurements
- Requires large data sets, possibly with code coverage



- Automotive safety certifications require test cases as certificates of proper operation
- In model-driven settings, I/O conformance (ioco) testing is a popular approach
- Requires generating **good** test cases
  - random testing does not easily reach crucial parts of the model
  - coverage guarantees



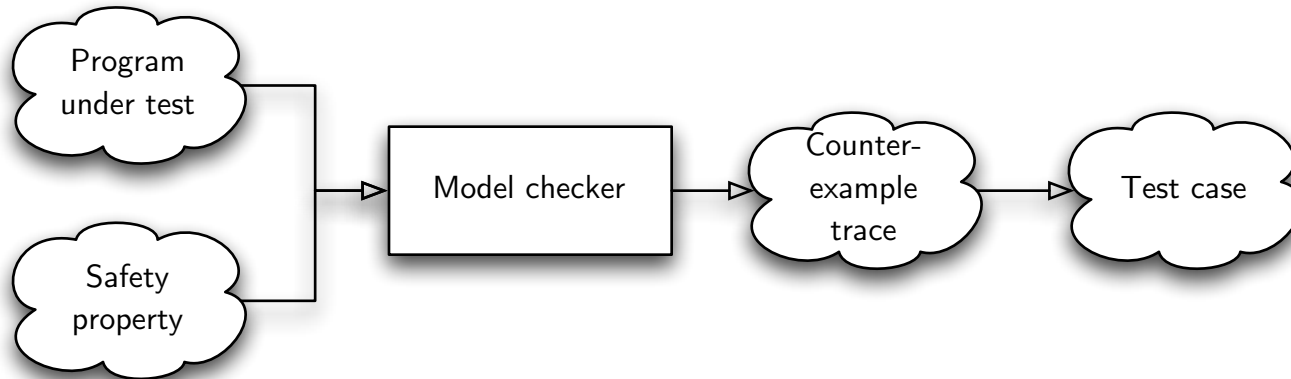
# White Box Test Case Generation

- In model-driven development, several tools are available
- Insufficient tool support for C/C++/Java
- (Directed) random testing: DART
- Concolic testing: CUTE
- C/Java: Software model checkers as test case generators

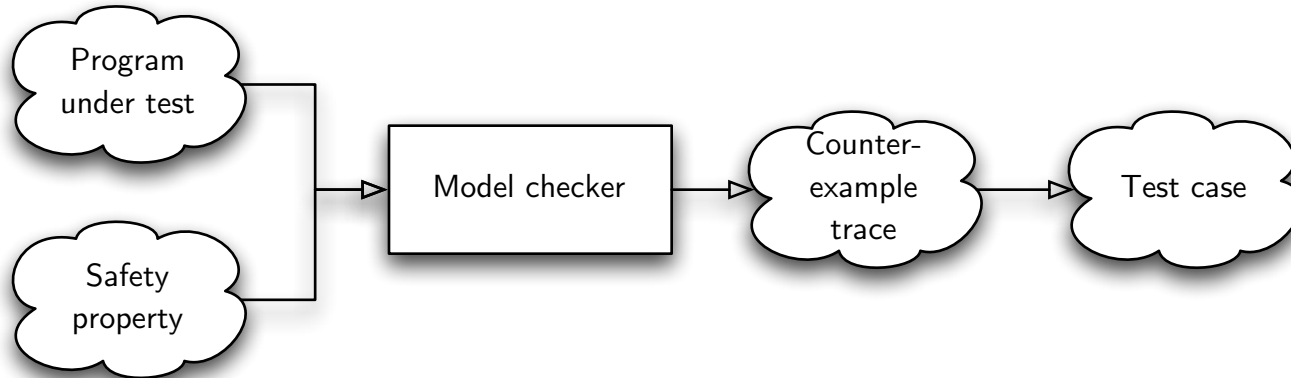
# Test Case Generation Using Model Checkers



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

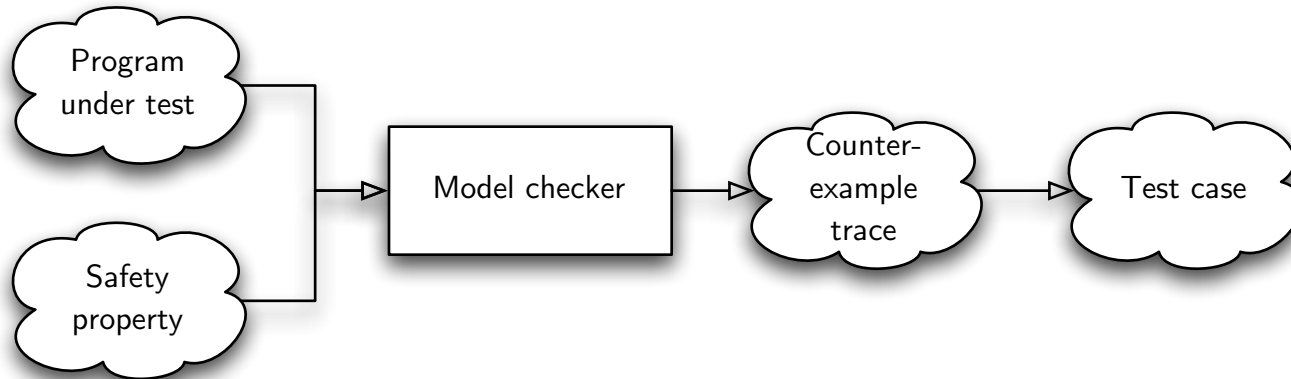


# Test Case Generation Using Model Checkers



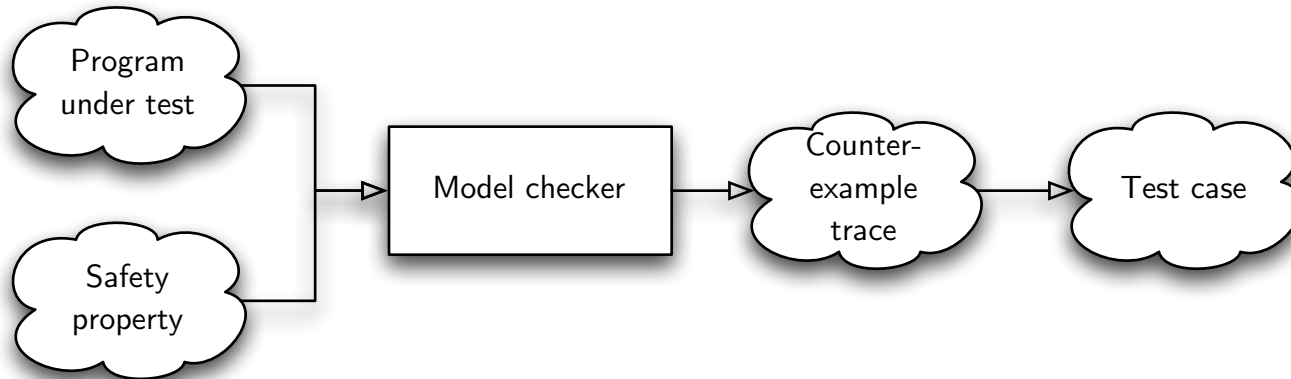
- Naive approach requires manual annotation of assertions

# Test Case Generation Using Model Checkers



- Naive approach requires manual annotation of assertions
- MC neither tailored for coverage nor efficient enumeration

# Test Case Generation Using Model Checkers



- Naive approach requires manual annotation of assertions
  - MC neither tailored for coverage nor efficient enumeration
  - BLAST 2.0:
    - BLAST Query Language as a means of specification
    - First automated test suite generation using model checkers
    - Basic block coverage hard coded
    - Coverage may be invalidated by spurious counterexamples
- Abstraction may be undesirable for test case generation

# White Box Test Case Generation

---



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

- Random testing is fast, easily generates large data sets
- Model checkers provide high quality test cases
- Is temporal logic and assert/assume appropriate for test suite specifications?

---

# FShell: Requirements

---



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

---

# FShell: Requirements

---



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

Fast generation of

---

# FShell: Requirements

---



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

Fast generation of  
high quality test suites

---

# FShell: Requirements

---



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

Fast generation of  
high quality test suites  
in a white box setting

# FShell: Requirements

---



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

Fast generation of  
high quality test suites  
in a white box setting  
for real-world C code.

# FShell: Requirements



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

Fast generation of  
high quality test suites  
in a white box setting  
for real-world C code.

Usable without expertise in formal methods

---

# Formal Foundations

---

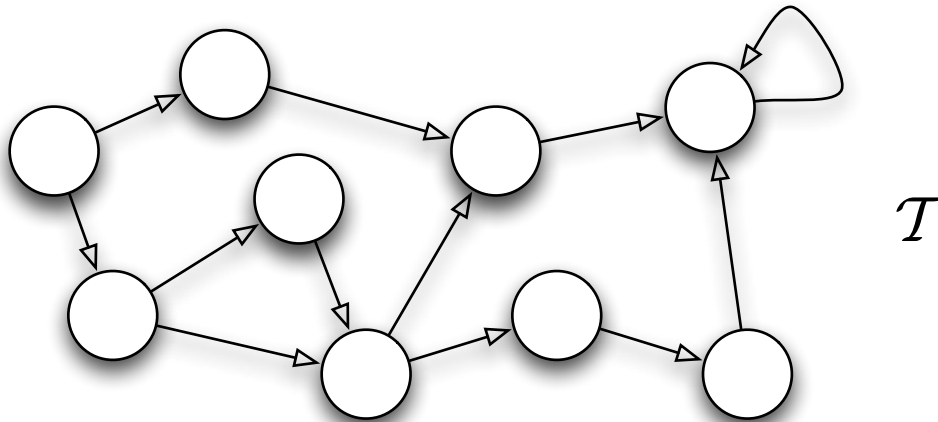


TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

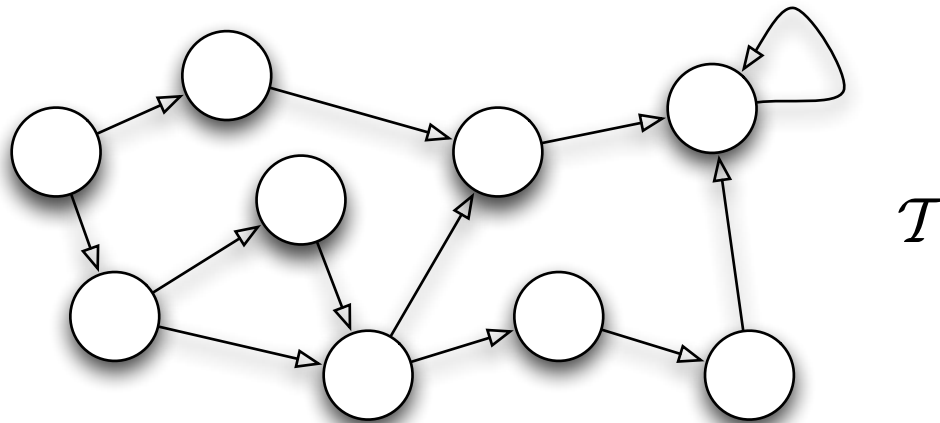


- Program represented as transition system  $\mathcal{T}$

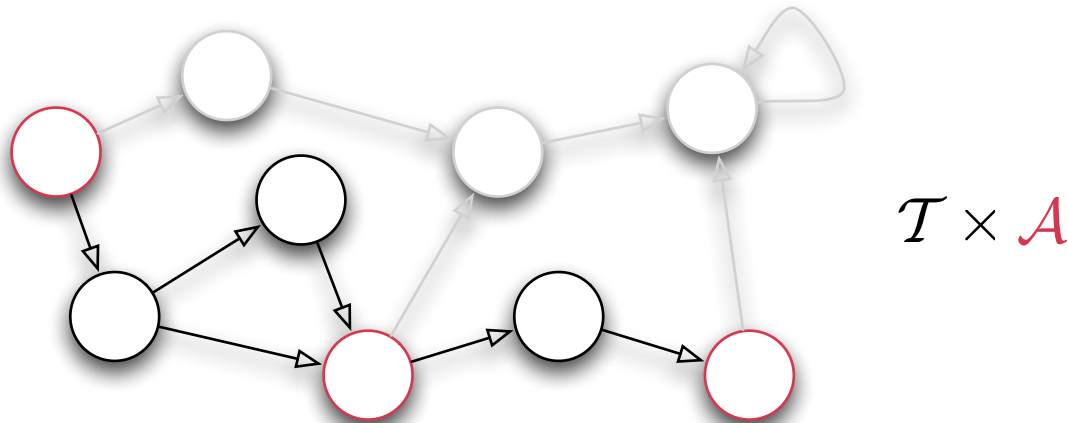
- Program represented as transition system  $\mathcal{T}$



- Program represented as transition system  $\mathcal{T}$
- Test case specification is translated to **path automaton**  $\mathcal{A}$
- Intersected with transition system to obtain new TS  $\mathcal{T} \times \mathcal{A}$



- Program represented as transition system  $\mathcal{T}$
- Test case specification is translated to **path automaton**  $\mathcal{A}$
- Intersected with transition system to obtain new TS  $\mathcal{T} \times \mathcal{A}$



---

# Formal Foundations

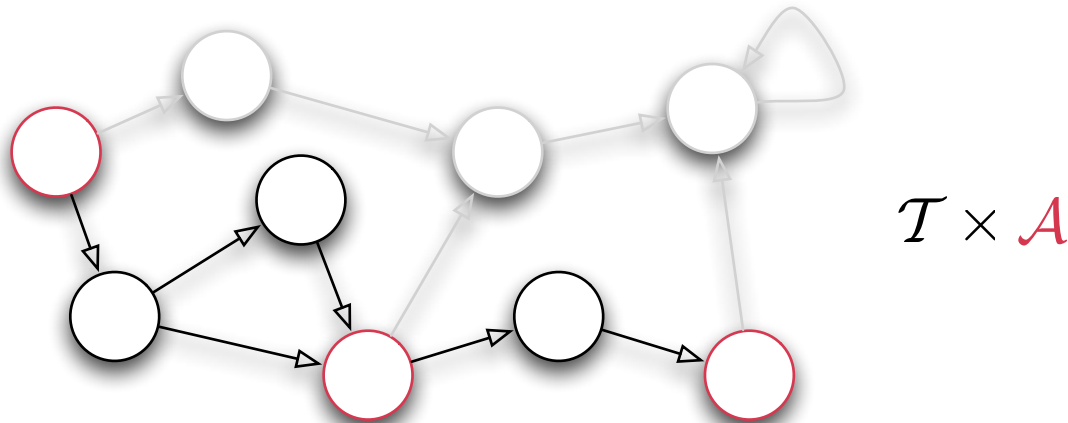
---



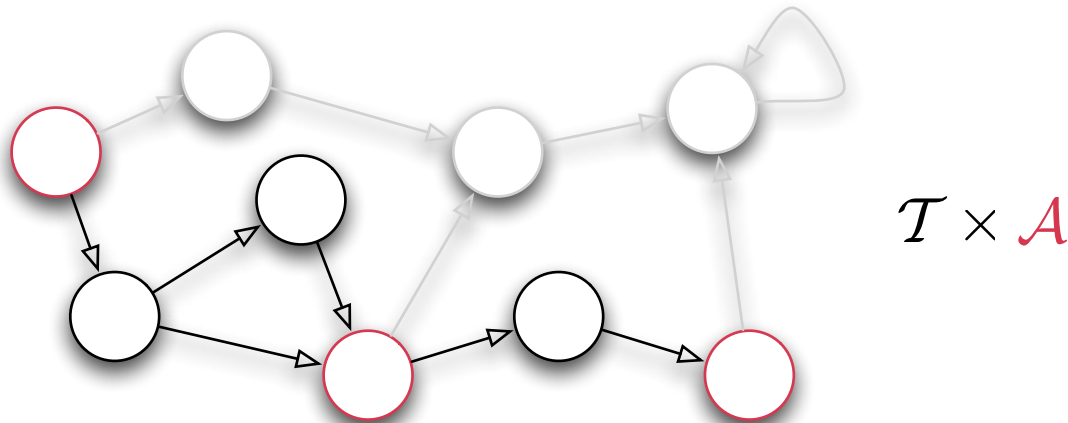
TECHNISCHE  
UNIVERSITÄT  
DARMSTADT



- A **test case** is a finite path (specification)  $\pi \in \mathcal{T} \times \mathcal{A}$
- A **test suite** is a set of paths  $\Gamma \subseteq \{\pi \mid \pi \in \mathcal{T} \times \mathcal{A}\}$

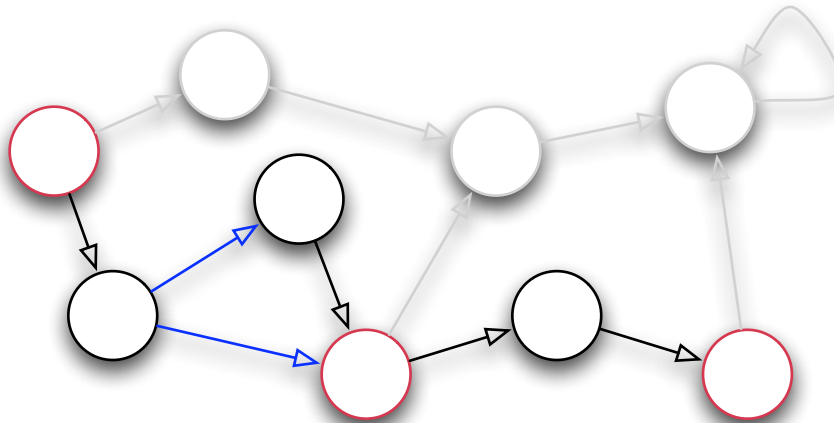


- A **test case** is a finite path (specification)  $\pi \in \mathcal{T} \times \mathcal{A}$
- A **test suite** is a set of paths  $\Gamma \subseteq \{\pi \mid \pi \in \mathcal{T} \times \mathcal{A}\}$
  
- Coverage is a property of a test suite





- A **test case** is a finite path (specification)  $\pi \in \mathcal{T} \times \mathcal{A}$
- A **test suite** is a set of paths  $\Gamma \subseteq \{\pi \mid \pi \in \mathcal{T} \times \mathcal{A}\}$
- Coverage is a property of a test suite
- **Coverage criterion**: Boolean formula  $\Phi_{\mathcal{A}}^{\mathcal{T}}$  over **path predicates**

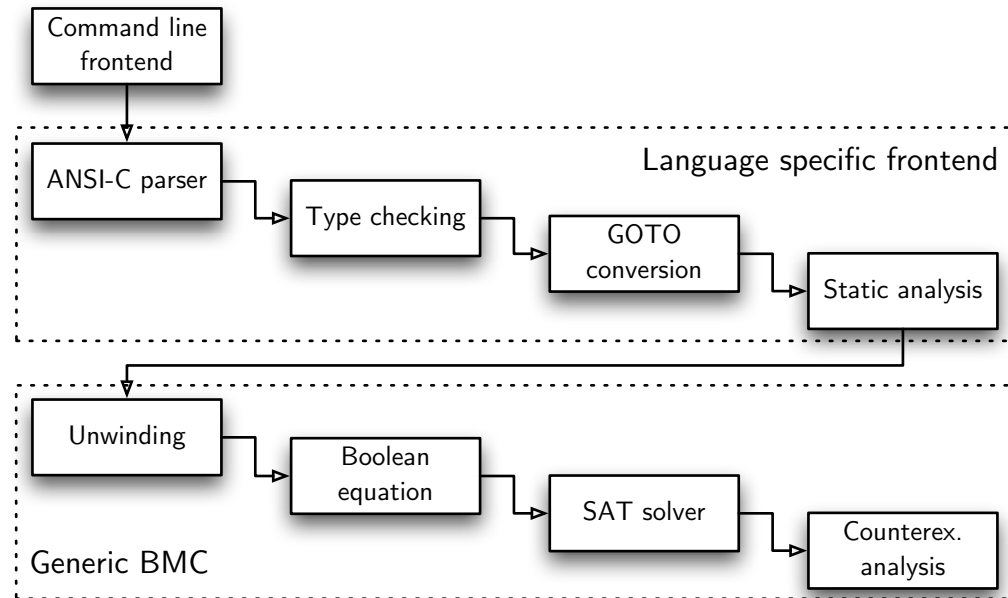


$$\mathcal{T} \times \mathcal{A} \wedge \Phi_{\mathcal{A}}^{\mathcal{T}}$$

# Background: Kröning's CBMC

## *C Bounded Model Checker*

- Written in C++, clean and stable code base
- Full ANSI-C support
- SAT solver used as decision procedure





# FShell: Test Case Generation Workflow



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

```
void bubble(int a[], int N) {
    int i, j, t;
    for (i = N - 1; i >= 0; i--)
        for (j = 1; j <= i; j++)
            if (a[j - 1] > a[j])
                SWAP(a[j-1],a[j]);
}
void main() {int a[20]; bubble(a, 20);}
```

# FShell: Test Case Generation Workflow



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

> ADD SOURCECODE "bubble.c"

```
void bubble(int a[], int N) {
    int i, j, t;
    for (i = N - 1; i >= 0; i--)
        for (j = 1; j <= i; j++)
            if (a[j - 1] > a[j])
                SWAP(a[j-1],a[j]);
}
void main() {int a[20]; bubble(a, 20);}
```

# FShell: Test Case Generation Workflow



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

> ADD SOURCECODE "bubble.c"

bubble.c

CBMC C parser

AST  
(bubble.c)

```
void bubble(int a[], int N) {
    int i, j, t;
    for (i = N - 1; i >= 0; i--)
        for (j = 1; j <= i; j++)
            if (a[j - 1] > a[j])
                SWAP(a[j-1],a[j]);
}
void main() {int a[20]; bubble(a, 20);}
```

# FShell: Test Case Generation Workflow



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

```
> ADD SOURCECODE "bubble.c"
```

bubble.c

CBMC C parser

AST  
(bubble.c)

```
> q := PATH PROC main TO FILE  
"bubble.c" LINE 3 COVERAGE  
PREDICATE TO PROC main  
EXITPOINTS
```

```
void bubble(int a[], int N) {  
    int i, j, t;  
    for (i = N - 1; i >= 0; i--)  
        for (j = 1; j <= i; j++)  
            if (a[j - 1] > a[j])  
                SWAP(a[j-1],a[j]);  
}  
void main() {int a[20]; bubble(a, 20);}
```

# FShell: Test Case Generation Workflow



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

```
> ADD SOURCECODE "bubble.c"
```

bubble.c

CBMC C parser

AST  
(bubble.c)

```
> q := PATH PROC main TO FILE  
"bubble.c" LINE 3 COVERAGE  
PREDICATE TO PROC main  
EXITPOINTS
```

```
void bubble(int a[], int N) {  
    int i, j, t;  
    for (i = N - 1; i >= 0; i--)  
        for (j = 1; j <= i; j++)  
            if (a[j - 1] > a[j])  
                SWAP(a[j-1],a[j]);  
}  
void main() {int a[20]; bubble(a, 20);}
```

# FShell: Test Case Generation Workflow



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

```
> ADD SOURCECODE "bubble.c"
```

bubble.c

CBMC C parser

AST  
(bubble.c)

```
> q := PATH PROC main TO FILE  
"bubble.c" LINE 3 COVERAGE  
PREDICATE TO PROC main  
EXITPOINTS
```

```
void bubble(int a[], int N) {  
    int i, j, t;  
    for (i = N - 1; i >= 0; i--)  
        for (j = 1; j <= i; j++)  
            if (a[j - 1] > a[j])  
                SWAP(a[j-1],a[j]);  
}  
void main() {int a[20], bubble(a, 20),}
```

# FShell: Test Case Generation Workflow



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

```
> ADD SOURCECODE "bubble.c"
```

bubble.c

CBMC C parser

AST  
(bubble.c)

```
> q := PATH PROC main TO FILE  
"bubble.c" LINE 3 COVERAGE  
PREDICATE TO PROC main  
EXITPOINTS
```

```
void bubble(int a[], int N) {  
  int i, j, t;  
  for (i = N - 1; i >= 0; i--)  
    for (j = 1; j <= i; j++)  
      if (a[j - 1] > a[j])  
        SWAP(a[j-1], a[j]);  
}  
void main() {int a[20]; bubble(a, 20);}
```

*predicate coverage*

# FShell: Test Case Generation Workflow



```
> ADD SOURCECODE "bubble.c"
```

bubble.c

CBMC C parser

AST  
(bubble.c)

```
> q := PATH PROC main TO FILE  
"bubble.c" LINE 3 COVERAGE  
PREDICATE TO PROC main  
EXITPOINTS
```

```
void bubble(int a[], int N) {  
    int i, j, t;  
    for (i = N - 1; i >= 0; i--)  
        for (j = 1; j <= i; j++)  
            if (a[j - 1] > a[j])  
                SWAP(a[j-1],a[j]);  
}  
void main() {int a[20]; bubble(a, 20);}
```

# FShell: Test Case Generation Workflow



```
> ADD SOURCECODE "bubble.c"
```

bubble.c

CBMC C parser

AST  
(bubble.c)

```
> q := PATH PROC main TO FILE  
"bubble.c" LINE 3 COVERAGE  
PREDICATE TO PROC main  
EXITPOINTS
```

Path  
query  $q$

```
void bubble(int a[], int N) {  
    int i, j, t;  
    for (i = N - 1; i >= 0; i--)  
        for (j = 1; j <= i; j++)  
            if (a[j - 1] > a[j])  
                SWAP(a[j-1],a[j]);  
}  
void main() {int a[20]; bubble(a, 20);}
```

# FShell: Test Case Generation Workflow

> ADD SOURCECODE "bubble.c"

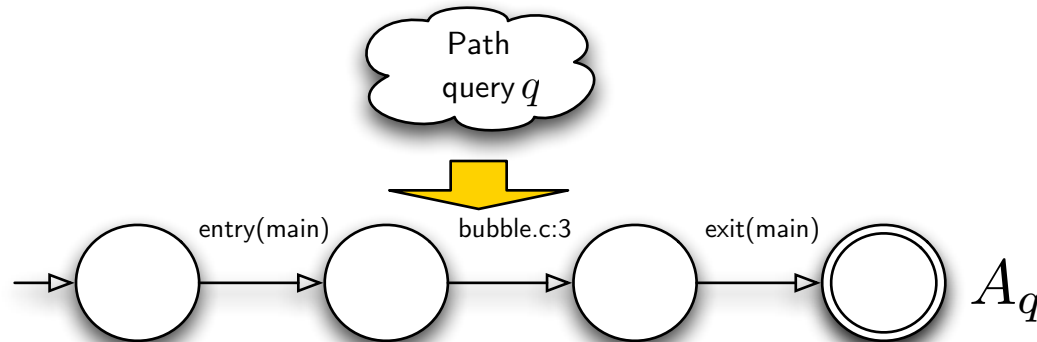
bubble.c

CBMC C parser

AST  
(bubble.c)

> q := PATH PROC main TO FILE  
"bubble.c" LINE 3 COVERAGE  
PREDICATE TO PROC main  
EXITPOINTS

```
void bubble(int a[], int N) {  
    int i, j, t;  
    for (i = N - 1; i >= 0; i--)  
        for (j = 1; j <= i; j++)  
            if (a[j - 1] > a[j])  
                SWAP(a[j-1],a[j]);  
}  
void main() {int a[20]; bubble(a, 20);}
```



# FShell: Test Case Generation Workflow

> ADD SOURCECODE "bubble.c"

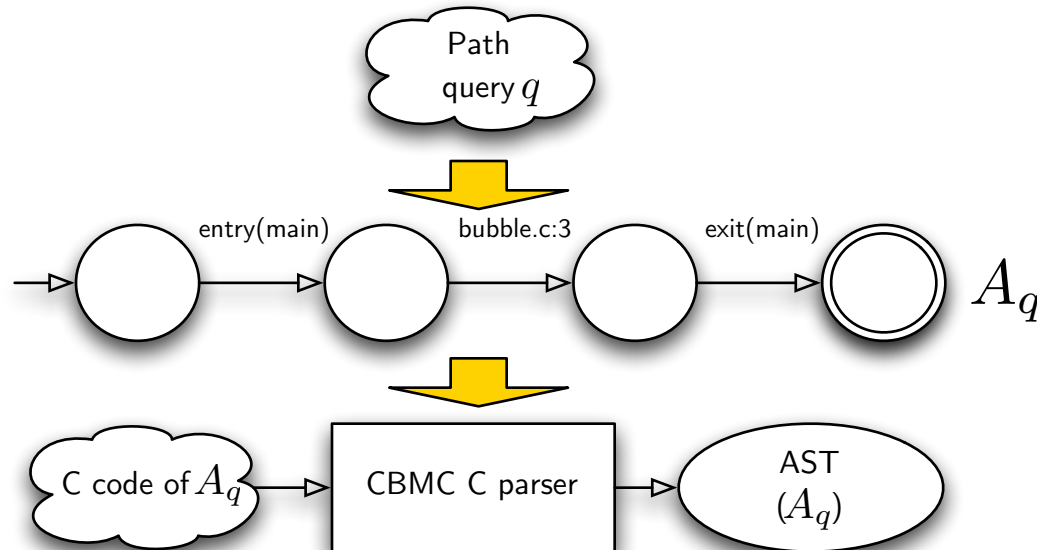
bubble.c

CBMC C parser

AST  
(bubble.c)

> q := PATH PROC main TO FILE  
"bubble.c" LINE 3 COVERAGE  
PREDICATE TO PROC main  
EXITPOINTS

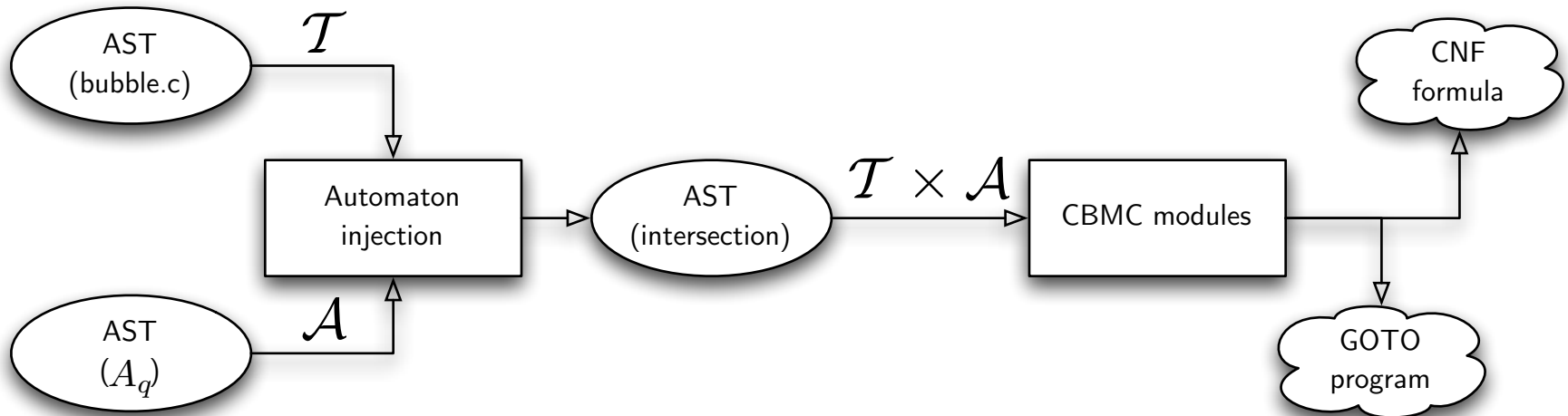
```
void bubble(int a[], int N) {  
    int i, j, t;  
    for (i = N - 1; i >= 0; i--)  
        for (j = 1; j <= i; j++)  
            if (a[j - 1] > a[j])  
                SWAP(a[j-1],a[j]);  
}  
void main() {int a[20]; bubble(a, 20);}
```



# FShell: Test Case Generation Workflow

- Use CBMC to obtain CNF for finite paths in  $\mathcal{T} \times \mathcal{A}$
- Path feasibility is checked using SAT solver

```
void bubble(int a[], int N) {  
    int i, j, t;  
    for (i = N - 1; i >= 0; i--)  
        for (j = 1; j <= i; j++)  
            if (a[j - 1] > a[j])  
                SWAP(a[j-1],a[j]);  
}  
void main() {int a[20]; bubble(a, 20);}
```



# FShell: Test Case Generation Workflow



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

```
void bubble(int a[], int N) {
    int i, j, t;
    for (i = N - 1; i >= 0; i--)
        for (j = 1; j <= i; j++)
            if (a[j - 1] > a[j])
                SWAP(a[j-1],a[j]);
}
void main() {int a[20]; bubble(a, 20);}
```

# FShell: Test Case Generation Workflow



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

```
> TEST q METHOD BMC
```

```
IN: a={0,112826,...,1465438334}
```

```
IN: a={0,-21909,...,1709483866}
```

```
void bubble(int a[], int N) {
    int i, j, t;
    for (i = N - 1; i >= 0; i--)
        for (j = 1; j <= i; j++)
            if (a[j - 1] > a[j])
                SWAP(a[j-1],a[j]);
}
void main() {int a[20]; bubble(a, 20);}
```

# FShell: Test Case Generation Workflow

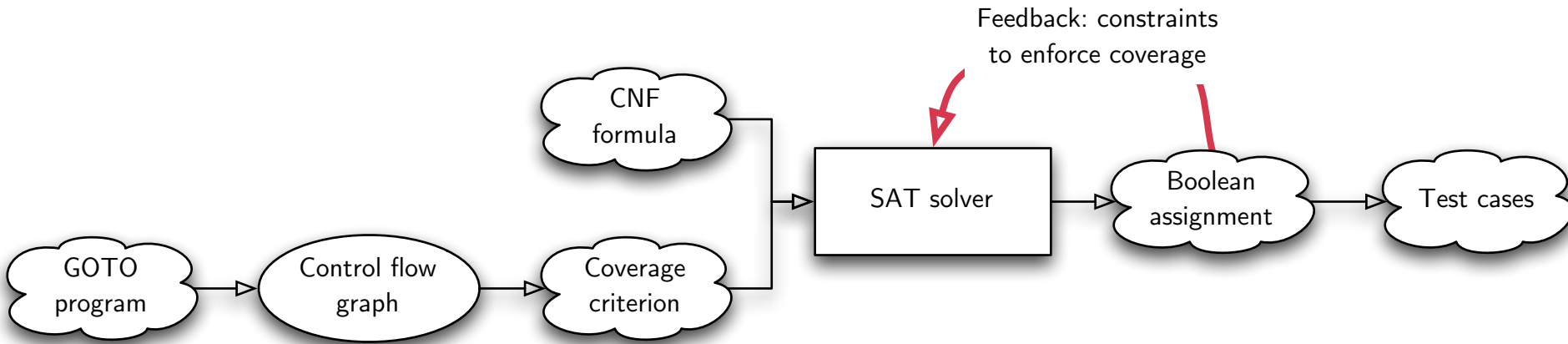


> TEST q METHOD BMC

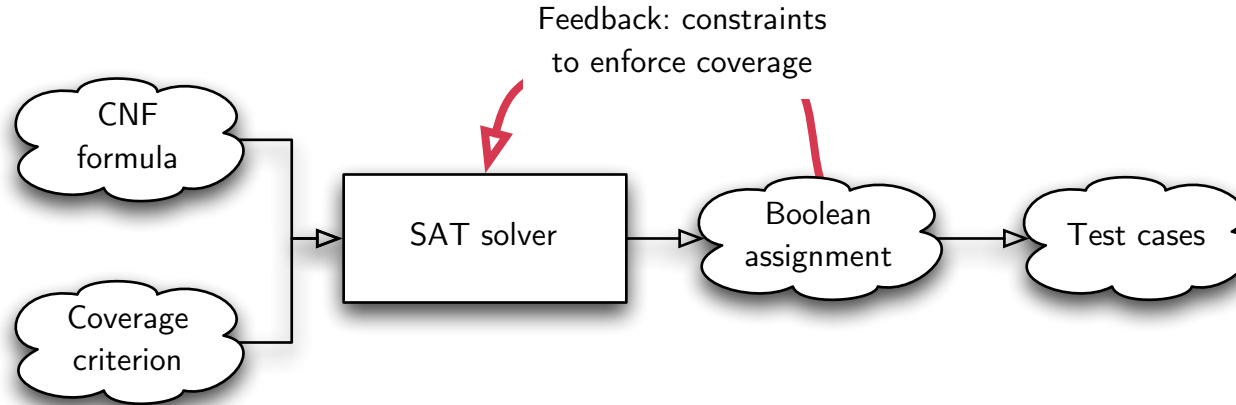
IN: a={0,112826,...,1465438334}

IN: a={0,-21909,...,1709483866}

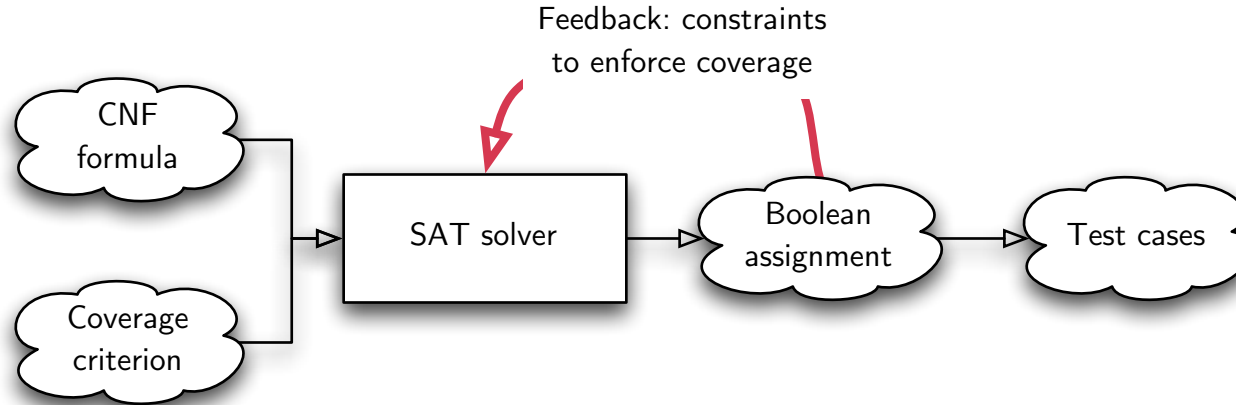
```
void bubble(int a[], int N) {
    int i, j, t;
    for (i = N - 1; i >= 0; i--)
        for (j = 1; j <= i; j++)
            if (a[j - 1] > a[j])
                SWAP(a[j-1],a[j]);
}
void main() {int a[20]; bubble(a, 20);}
```



# Guided SAT Enumeration for Fast Test Case Generation

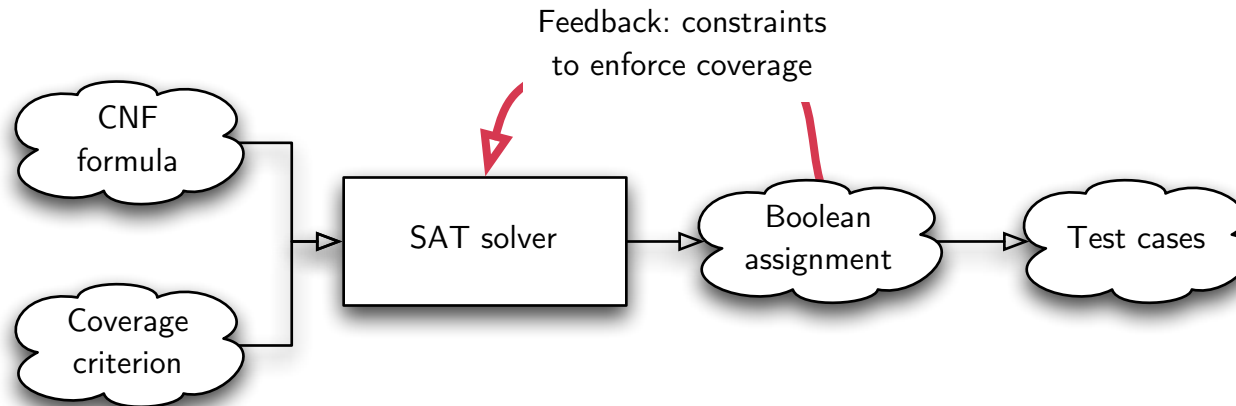


# Guided SAT Enumeration for Fast Test Case Generation



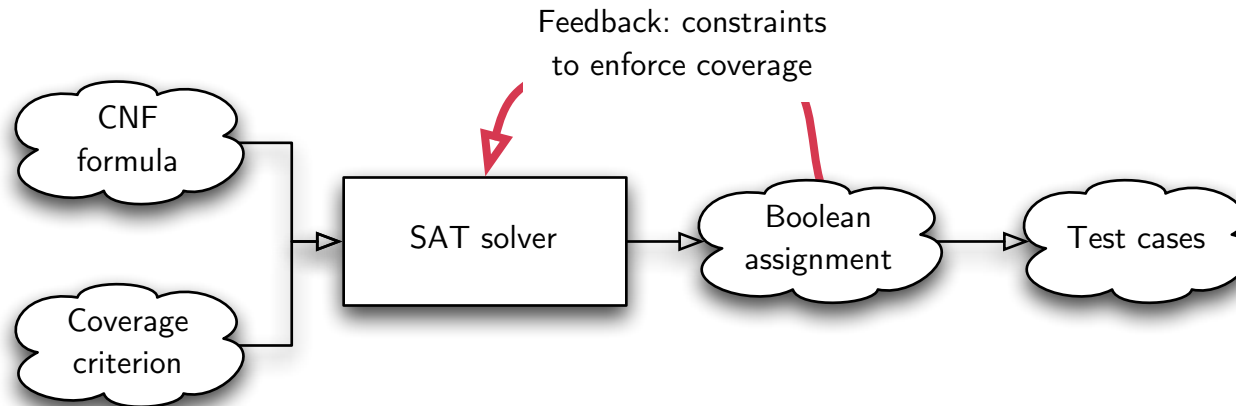
- Incremental SAT solving
  - Conflict database is kept valid
  - Additional clauses added in feedback loop

# Guided SAT Enumeration for Fast Test Case Generation



- Incremental SAT solving
  - Conflict database is kept valid
  - Additional clauses added in feedback loop
- Instance becomes unsatisfiable iff no more test cases exist

# Guided SAT Enumeration for Fast Test Case Generation



- Incremental SAT solving
  - Conflict database is kept valid
  - Additional clauses added in feedback loop
- Instance becomes unsatisfiable iff no more test cases exist
- Approach is scalable:
  - 1 test case: 0.3s
  - 1000000 test cases: 670s

---

# Example: Condition Coverage

---



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT



# Example: Condition Coverage

```
if (x < 5) {  
    if (0 == x % 2) { ... }  
    else { ... }  
} else {  
    if (x > 7) { ... }  
    else { ... }  
}
```



# Example: Condition Coverage

```
if (x < 5) {  
    if (0 == x % 2) { ... }  
    else { ... }  
} else {  
    if (x > 7) { ... }  
    else { ... }  
}
```



# Example: Condition Coverage

```
if (x < 5) {  
    if (0 == x % 2) { ... }  
    else { ... }  
} else {  
    if (x > 7) { ... }  
    else { ... }  
}
```



# Example: Condition Coverage

```
if (x < 5) {  
    if (0 == x % 2) { ... }  
    else { ... }  
} else {  
    if (x > 7) { ... }  
    else { ... }  
}
```

# Example: Condition Coverage

```
if (x < 5) {  
  if (0 == x % 2) { ... }  
  else { ... }  
} else {  
  if (x > 7) { ... }  
  else { ... }  
}
```

■  $e_i$ ... condition  $i$  is enabled (reached)

## Example: Condition Coverage

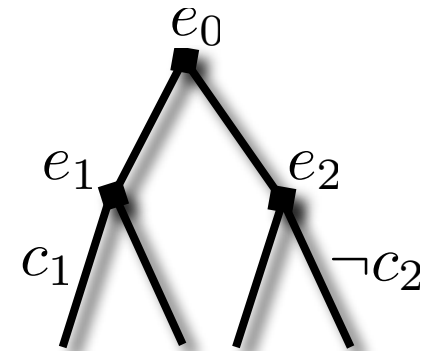
```
if (x < 5) {  
    if (0 == x % 2) { ... }  
    else { ... }  
} else {  
    if (x > 7) { ... }  
    else { ... }  
}
```

- $e_i \dots$  condition  $i$  is enabled (reached)
- $C_i \dots$  condition  $i$  evaluates to true

# Example: Condition Coverage

```
if (x < 5) {  
  if (0 == x % 2) { ... }  
  else { ... }  
} else {  
  if (x > 7) { ... }  
  else { ... }  
}
```

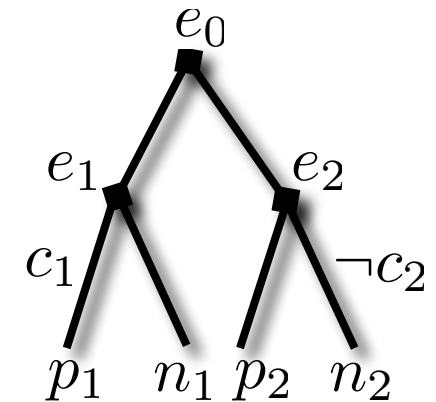
- $e_i$  ... condition  $i$  is enabled (reached)
- $c_i$  ... condition  $i$  evaluates to true



# Example: Condition Coverage

```
if (x < 5) {  
  if (0 == x % 2) { ... }  
  else { ... }  
} else {  
  if (x > 7) { ... }  
  else { ... }  
}
```

- $e_i \dots$  condition  $i$  is enabled (reached)
- $c_i \dots$  condition  $i$  evaluates to true



# Example: Condition Coverage

```

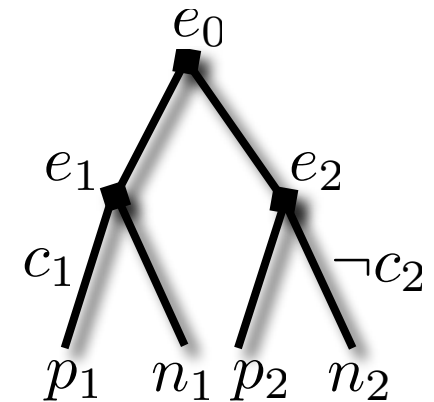
if (x < 5) {
  if (0 == x % 2) { ... }
  else { ... }
} else {
  if (x > 7) { ... }
  else { ... }
}

```

- $e_i$  ... condition  $i$  is enabled (reached)
- $c_i$  ... condition  $i$  evaluates to true

- Initial coverage criterion

$$\beta_0 = \bigwedge_i p_i \Leftrightarrow e_i \wedge c_i \\
 \bigwedge_i n_i \Leftrightarrow e_i \wedge \neg c_i \\
 \bigwedge_i p_i \vee n_i$$

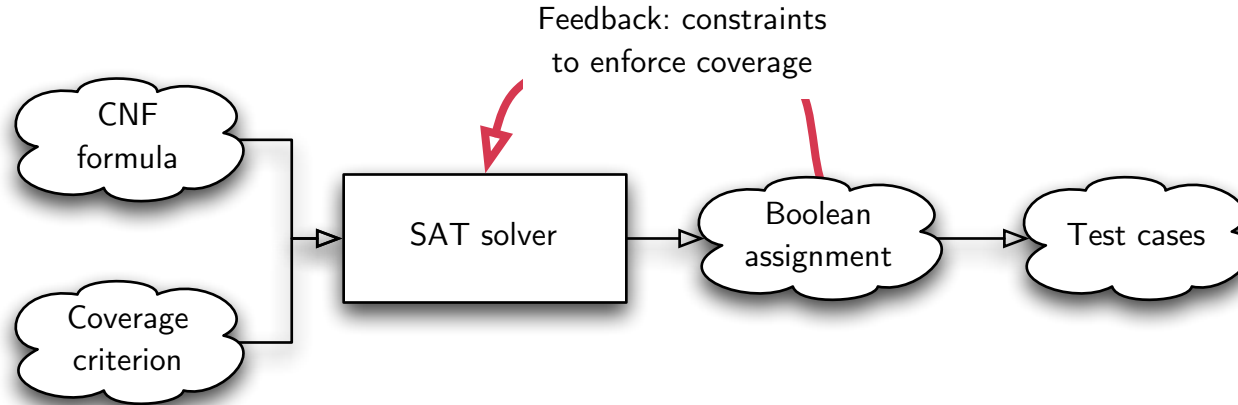


# Example: Condition Coverage

- Initial coverage criterion

$$\beta_0 = \bigwedge_i p_i \Leftrightarrow e_i \wedge c_i$$
$$\bigwedge_i n_i \Leftrightarrow e_i \wedge \neg c_i$$
$$\bigwedge_i p_i \vee n_i$$

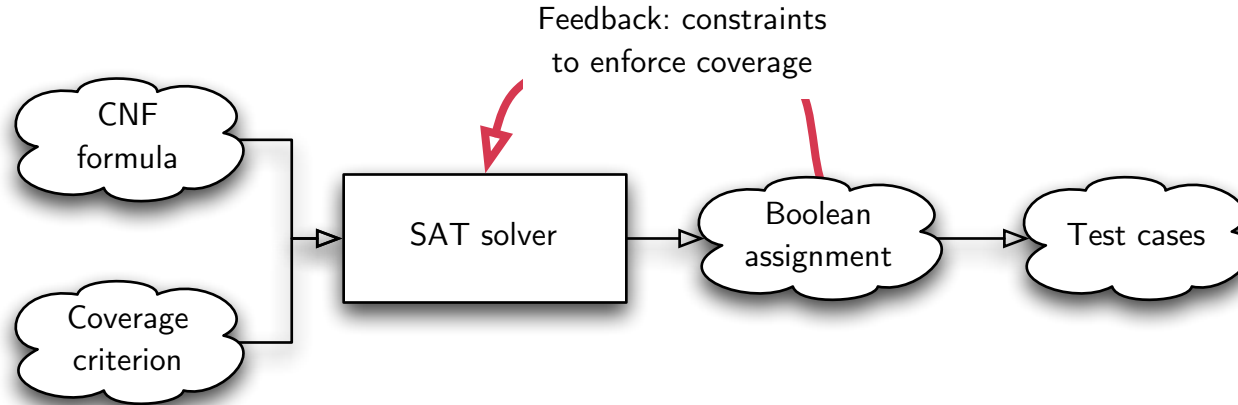
# Example: Condition Coverage



## ▪ Initial coverage criterion

$$\beta_0 = \bigwedge_i p_i \Leftrightarrow e_i \wedge c_i$$
$$\bigwedge_i \bigwedge_n n_i \Leftrightarrow e_i \wedge \neg c_i$$
$$\bigwedge_i \bigvee p_i \vee n_i$$

# Example: Condition Coverage



## ▪ Initial coverage criterion

$$\beta_0 = \bigwedge_i p_i \Leftrightarrow e_i \wedge c_i \\ \bigwedge_i \bigwedge_n n_i \Leftrightarrow e_i \wedge \neg c_i \\ \bigwedge_i \bigvee p_i \vee n_i$$

## ▪ Feedback: new clause $\beta_j$ containing all $p_i, n_i$ not covered yet

# Results

## Industrial Engine Controller

- Code autogenerated from MATLAB/Simulink model
- C source code of 2033 LLOC
- No abstraction applied
- Basic block coverage: 5 test cases
- Computation took 18 seconds



# Results

## Comparison with BLAST

Source file	LLOC	#cases	Time[s]	FShell #cases	FShell Time[s]	Speed- up
kbfiltr.i	4879	39	300	66	17	17.9
floppy.i	6435	111	1500	288	1305	1.1
cdaudio.i	8022	85	1500	159	748	2.0
parport.i	20698	213	5460	312	1999	2.7
parclass.i	45283	219	2520	716	1511	1.7

```
FShell> ADD SOURCECODE "driver.i"
```

```
FShell> path_basic_block_coverage := PATH PROC main COVERAGE  
BASICBLOCK TO PROC main EXITPOINTS
```

```
FShell> TEST path_basic_block_coverage METHOD BMC LIMIT COUNT  
1000
```

```
FShell> QUIT
```

# Summary

---



- Testing remains an integral part of the software development process
- FShell enables automated test case generation for large high quality test suites for real-world C code
- Test specification language is part of FShell
- Guided SAT enumeration to allow fast generation of large test suites