



Proseminar *Einführung in die Programmanalyse*

Einführung

Sommersemester 2025, 16. April 2025

Thomas Noll et al.

Software Modeling and Verification Group

RWTH Aachen University

<https://moves.rwth-aachen.de/teaching/ss-25/ipa/>

Übersicht

Einführung

Termine

Modelle

Logiken

Statische Analyseverfahren

Dynamische und Hybride Analyseverfahren

Traum der Programmanalyse

Programm

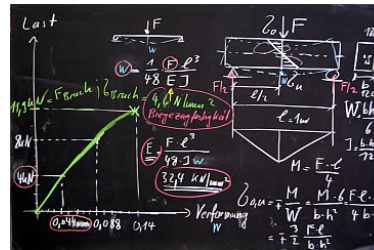
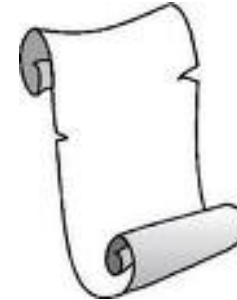
```
...socket.error: [Errno 111] Connection refused\n...print "ncfiles: Socket error (%s) for host %s (%s)" % (errno, strerror)\n\nfor h3 in page.findAll("h3"):\n    value = (h3.contents[0]):\n    if value != "Afdeling":\n        print >> txt, value\n        import codecs\n        f = codecs.open("alle.txt", "r", encoding="utf-8")\n        text = f.read()\n        f.close()\n        # open the file again for writing\n        f = codecs.open("alle.txt", "w", encoding="utf-8")\n        f.write(value+"\n")\n        # write the original contents
```



Analyse



Ergebnis

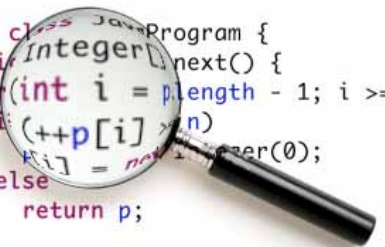


Spezifikation

Thema des Proseminars

Einführung in Grundkonzepte der (automatisierten) Programmanalyse

- Modelle:
 - (abstrakte) Darstellung des Systemverhaltens
 - Spezifikation des erwarteten Verhaltens
- Logiken:
 - formale Beschreibung von (Korrektheits-)Eigenschaften
- Statische Verfahren:
 - basierend auf Quellcode der Software
 - üblicherweise vollständige Abdeckung
- Dynamische/hybride Verfahren:
 - basierend auf beobachtetem Ausführungsverhalten
 - üblicherweise unvollständige Abdeckung



```
public class JavaProgram {  
    public Integer next() {  
        for (int i = p.length - 1; i >= 0; i--)  
            if (++p[i] > n)  
                p[i] = nextElement(0);  
        else  
            return p;  
    }  
    throw new NoSuchElementException();  
}
```

Zielsetzung

Ziele des Proseminars

- Selbstständiges Einarbeiten in ein neues Thema
- Literaturrecherche
 - Einstiegsreferenz auf Webseite
 - Schulung in Informatikbibliothek
- **Verständliches Präsentieren**
- Kurzdarstellung des Inhalts in einer **wissenschaftlich orientierten Ausarbeitung**

Bearbeitung in Zweiergruppen

- Zwei separate Vorträge (mit nicht notwendigerweise verschiedenen Foliensätzen)
- Gemeinsame Anfertigung der Ausarbeitung

Vortrag

- 20-minütiger Vortrag
- Zielgruppengerechte Präsentation der Inhalte
- übersichtliche Folien:
 - ≤ 15 Textzeilen
 - sinnvoller Einsatz von Farben
- $\text{\LaTeX}$ /beamer-Vorlage wird zur Verfügung gestellt
- Vortrag in Deutsch oder Englisch

Ausarbeitung

- Selbstständiges Verfassen einer Zusammenfassung von gut **5 Seiten**
- **Vollständiges** Literaturverzeichnis
- Korrektes **Zitieren**
- **Plagiarismus:**
Die nicht gekennzeichnete Übernahme fremder Inhalte (Literatur, Web, KI-Tools) führt zum **sofortigen Ausschluss** – dies gilt auch für LLMs!
- Schriftgröße **12pt**, übliche Seitenränder
- **Titelseite** mit Thema, Titel Proseminar, Semester, Name, Datum
- **L^AT_EX-Vorlage** wird zur Verfügung gestellt
- **Sprache** Deutsch oder Englisch
- **Korrekte Sprache** wird vorausgesetzt

Übersicht

Einführung

Termine

Modelle

Logiken

Statische Analyseverfahren

Dynamische und Hybride Analyseverfahren

Verfahren

- Themenliste wurde/wird ausgehändigt
- Priorisierte Auswahl
- ggf. Angabe Wunschpartner(in)
- Abgabe hier oder bis (spätestens) Dienstag, 22. April per Mail (noll@cs.rwth-aachen.de)
- Wir bemühen uns (ohne Garantie) um ein „optimales“ Matching
- Zuordnung der Themen und Betreuer:innen bis Mitte kommender Woche online

Rücktritt vom Proseminar

- Bis zu **einer Woche (!)** nach Themenvergabe: ohne Folgen
- Danach: Fehlversuch

Deadlines

Folgende Termine sind **verpflichtend**:

- 22. April: Frist für Themenauswahl
- 30. April: letzte Rücktrittsmöglichkeit
- 19. Mai: Vorlage des detaillierten Vortragsstruktur
 - nicht: „1. Einleitung/2. Hauptteil/3. Schluss“
 - sondern: detaillierte Strukturübersicht (Einteilung in Abschnitte, wesentliche Definitionen/Aussagen/Beispiele, ...)
- 16. Juni: vollständige Fassung der Vortragsfolien
- 15. Juli (?): Blockseminar
- 28. Juli: Einreichung der Ausarbeitung

Übersicht

Einführung

Termine

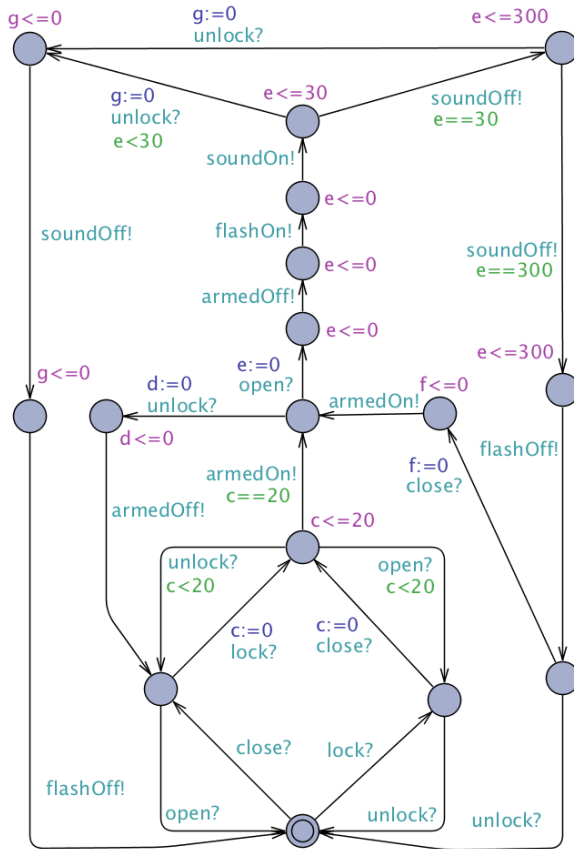
Modelle

Logiken

Statische Analyseverfahren

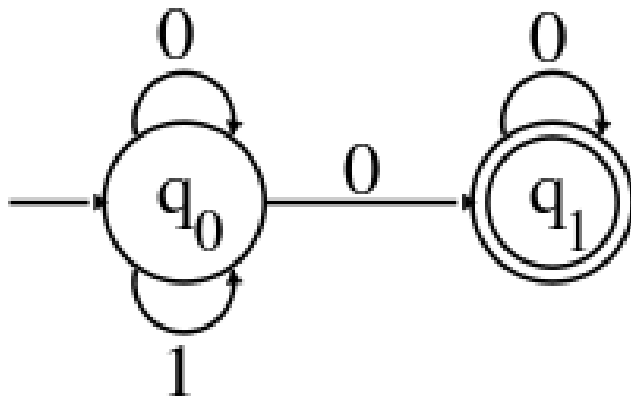
Dynamische und Hybride Analyseverfahren

1. Timed Automata



- Timed automaton = finite automaton + real-valued clocks
- Clock values increase at same speed
- Can be reset in transitions
- Can be tested in states (invariants) and transitions (guards)
- Enables modelling and analysis of time-dependent system behaviour

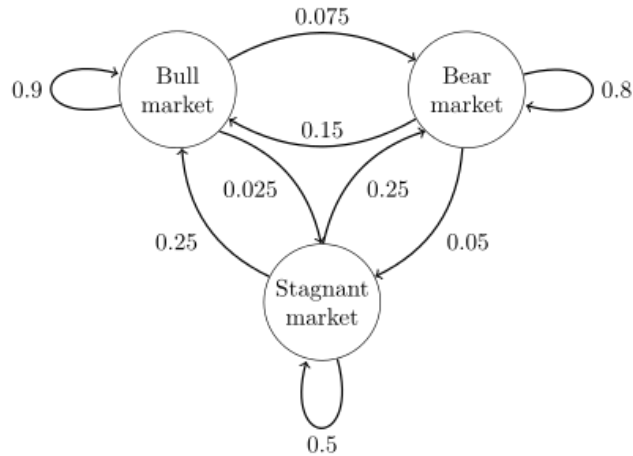
2. Büchi Automata



BA for $(0|1)^*0^\omega$

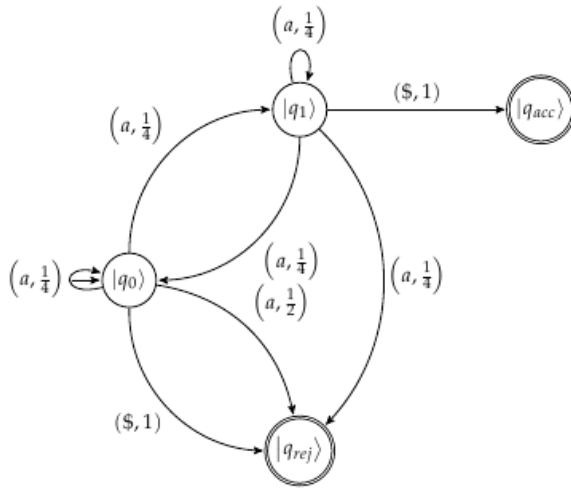
- Syntax identical to DFA/NFA
- But define languages of infinite words
- Accepts an infinite input sequence if there exists a run that visits one of the final states infinitely often
- Enables analysis of non-terminating system behaviour
- Non-deterministic variant more expressive

3. Markov Chains



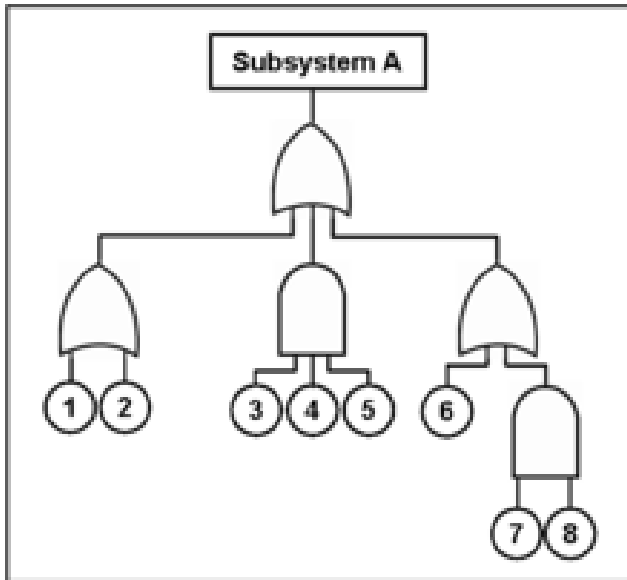
- States + probabilistic transitions
- Models „memoryless“ stochastic behaviour
- Two variants:
 - discrete time (one step per time unit, outgoing probabilities sum up to 1)
 - continuous time (outgoing transitions labelled with exit rates)
- Numerous applications

4. Probabilistic Automata



- Generalisation of
 - finite automata: + transition probabilities
 - Markov chain: + transition labels
- Define stochastic languages as sets of words that are recognised with a certain (minimal) probability
 - regular languages are proper subclass

5. (Dynamic) Fault Trees



- Visualise possible failure behaviours of HW/SW system
- Represented by tree structure with gates (OR, AND, Priority-AND, ...)
- Applications in safety/reliability engineering

Übersicht

Einführung

Termine

Modelle

Logiken

Statische Analyseverfahren

Dynamische und Hybride Analyseverfahren

6. Hoare Logic

$$\frac{\{A \wedge b\} c \{B\} \quad \frac{(A \wedge \neg b) \Rightarrow B}{\{(A \wedge \neg b)\} \text{skip} \{B\}}}{\{A\} \text{if } b \text{ then } c \text{ else skip } \{B\}} \quad \{A\} \text{if } b \text{ then } c \{B\}$$

- Formal system for reasoning about correctness of computer programs
- Goal: establish partial (or total) correctness properties of the form

$$\{A\} c \{B\}$$

with assertions A, B and program c

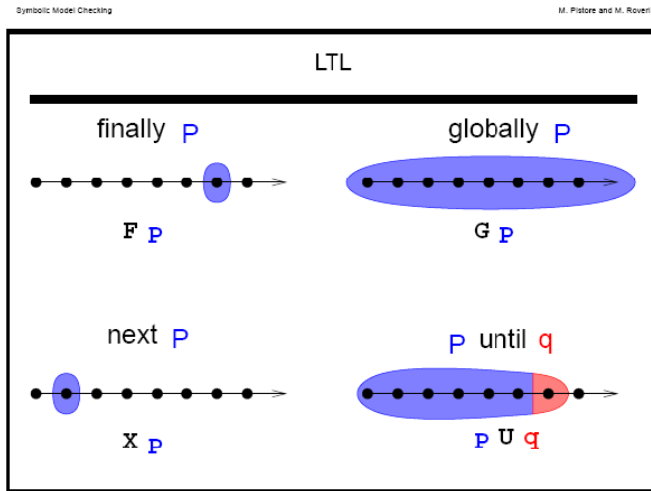
- Systematic construction of proof using logical rules of the form

$$\frac{\text{Premise(s)}}{\text{(Name) Conclusion}}$$

- Critical part: deriving loop invariants

$$\frac{\{A \wedge b\} c \{A\}}{\text{(while) } \{A\} \text{ while } b \text{ do } c \text{ end } \{A \wedge \neg b\}}$$

7. Linear-Time Temporal Logic (LTL)



- Modal temporal logic
- Formulae specify properties of infinite system traces
- Safety: something bad never happens

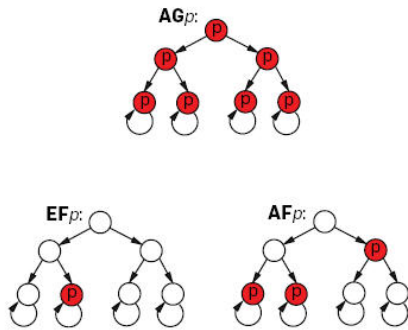
$$G(\neg \text{Bad})$$

- Liveness: something good will happen

$$G(\text{Request} \implies F \text{Response})$$

8. Computation Tree Logic (CTL)

Figure 1. Basic temporal operators.



- Similar to LTL but supports branching time
- Time is tree structured with several possible futures
- Path operators:
 - $A\varphi$ (all): φ has to hold on all paths starting from current state
 - $E\varphi$ (exists): there exists at least one path starting from current state where φ holds
- State operators:
 - $X\varphi$ (next): φ has to hold at next state
 - $G\varphi$ (globally): φ has to hold on entire subsequent path
 - $F\varphi$ (finally): φ has to hold somewhere on subsequent path
 - $\varphi U \psi$ (until): φ has to hold until ψ holds

9. Separation Logic

Frame rule of SL:

$$\text{(frame)} \frac{\{P\} c \{Q\}}{\{P * R\} c \{Q * R\}}$$

if $\text{mod}(c) \cap \text{free}(R) = \emptyset$

- Extension of Hoare logic to reason about programs that manipulate pointer data structures
 - dereferencing invalid pointers
 - creation of memory leaks
 - invalidation of data structures
- Additional operator $*$ (separating conjunction): expresses that heap can be split into two disjoint parts where its two subformulae respectively hold

Übersicht

Einführung

Termine

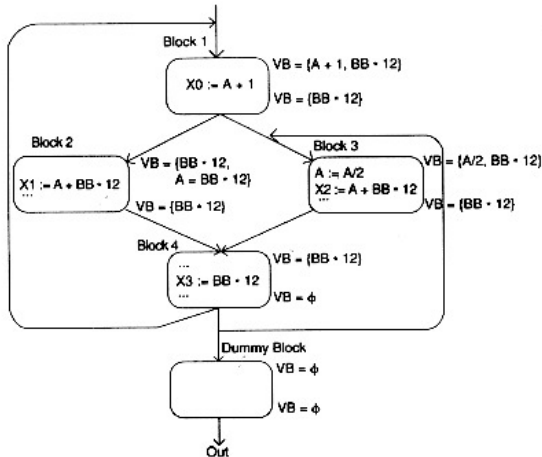
Modelle

Logiken

Statische Analyseverfahren

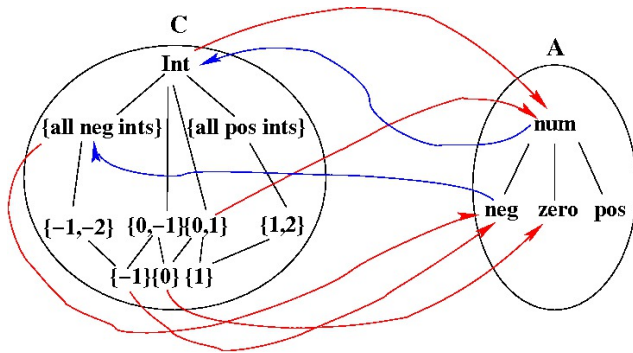
Dynamische und Hybride Analyseverfahren

10. Data-Flow Analysis



- Technique for associating information with control points of computer program
 - values of variables
 - available expressions
 - live variables, ...
- Distinctions:
 - dependence on statement order:
flow-sensitive vs. flow-insensitive analyses
 - direction of flow:
forward vs. backward analyses
 - quantification over paths:
may (union) vs. must (intersection) analyses
 - handling of procedures:
interprocedural vs. intraprocedural analyses
- Approach: solution of data-flow equation system by fixpoint iteration

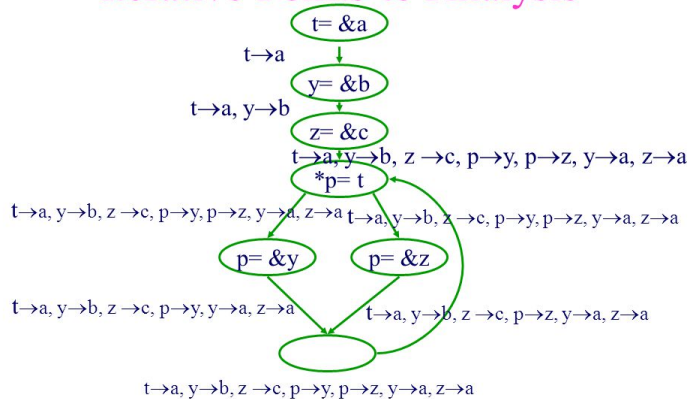
11. Abstract Interpretation



- Theory of (sound) approximation of the semantics of computer programs
 - integer values $\rightsquigarrow$ signs
 - integer values $\rightsquigarrow$ value intervals (array bounds checking)
 - concrete values $\rightsquigarrow$ types (JVM byte code verifier)
- Formalisation by abstraction and concretisation mappings that form a Galois connection
- Soundness: every concrete computation captured by some abstract execution of program

12. Pointer Analysis

Iterative Points-to Analysis



- Static code analysis technique that establishes which pointers (heap references) can point to which variables (storage locations)
- Often a basic component of more complex analyses such as escape analysis
- Example:

```
int x;  
int y;  
int* p = unknown() ? &x : &y;  
yields {x, y} as points-to set of p
```

Übersicht

Einführung

Termine

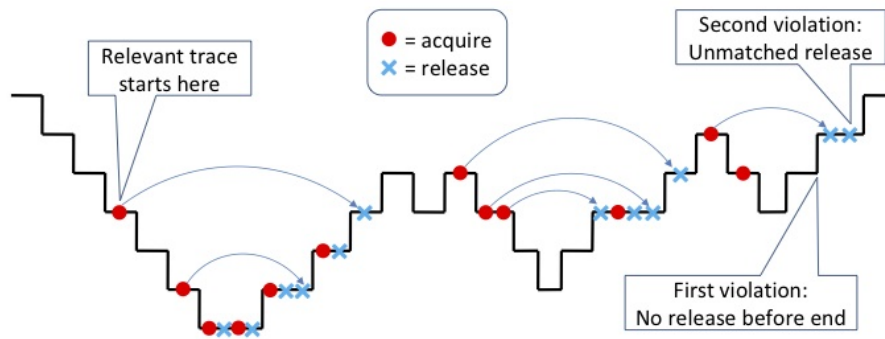
Modelle

Logiken

Statische Analyseverfahren

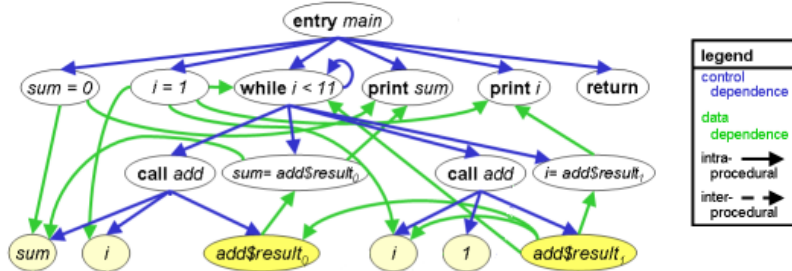
Dynamische und Hybride Analyseverfahren

13. Runtime Verification



- Observation of running system to detect (and possibly react to) behaviours violating certain properties
 - deadlocks
 - data races
 - improper use of synchronisation mechanisms
 - ...
- Desired properties specified by predicates over execution traces
 - finite automata/regular expressions
 - context-free grammars
 - linear temporal logics (LTL, ...)
 - ...

14. Program Slicing



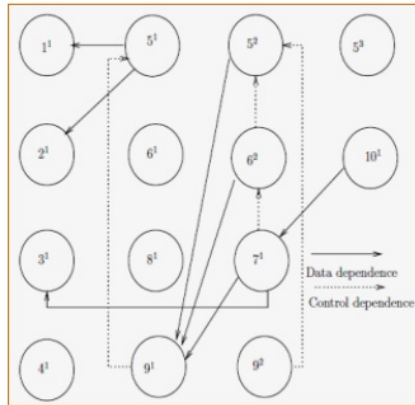
Dynamic Slicing: Example cont...

```

1 scanf("%d",&N);
2 i = 1;
3 s = 0;
4 p = 1;
5 while(i < N){
6   if(i % 2 == 0){
7     s = s + i;
8   } else {p = p * i;}
9   i = i + 1;
10  printf("%d%d",s,p);

```

For N=3



Slice has following statement instances

10¹, 7¹, 6², 5², 9¹, 3¹, 5¹, 2¹, 1¹ i.e. {1,2,3,5,6,7,9,10}

Date: 27 March-2014

Ankur Jain, Dept of Computer Sc & Engg., IIT Kgp

- Computation of the part of program („program slice“) that may affect the values at some point of interest („slicing criterion“)
- Static slicing:
 - no assumptions regarding input
 - based on program dependence graph
 - applications: software maintenance (regression testing), information flow control
- Dynamic slicing:
 - assumes fixed input for program
 - based on execution trace
 - application: debugging