



Proseminar *Einführung in die Programmanalyse*

Einführung

Sommersemester 2022; April 2022

Thomas Noll et al.

Software Modeling and Verification Group

RWTH Aachen University

<https://moves.rwth-aachen.de/teaching/ss-22/ipa/>

Übersicht

Einführung

Termine

Modelle

Logiken

Statische Analyseverfahren

Dynamische und Hybride Analyseverfahren

Traum der Programmanalyse

Programm

```
socket.error: [Errno 111] Connection refused
print "ncfiles: Socket error (%s) for host %s (%s)" % (errno, strerror)

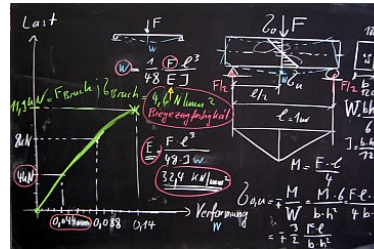
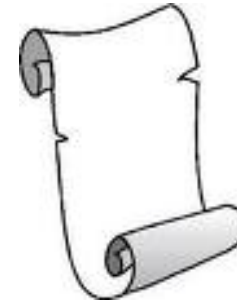
for h3 in page.findAll("h3"):
    value = (h3.contents[0])
    if value != "Afdeling":
        print >> txt, value
        import codecs
        f = codecs.open("alle.txt", "r", encoding="utf-8")
        text = f.read()
        f.close()
        # open the file again for writing
        f = codecs.open("alle.txt", "w", encoding="utf-8")
        f.write(value+"\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: (++p[i] > n)  
            p[i] = nextInteger(0);  
        else  
            return p;  
        }  
        throw new NoSuchElementException();  
    }
```

Zielsetzung

Ziele des Proseminars

- Selbstständiges Einarbeiten in ein neues Thema
- Literaturrecherche (Referenz auf Webseite)
- Darstellen des Inhalts in einer **wissenschaftlichen Ausarbeitung**
- **Verständliches Präsentieren**

Bearbeitung in Zweiergruppen

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

Ausarbeitung

- Selbstständiges Verfassen einer Ausarbeitung von ca. **10–12 Seiten**
- **Vollständiges** Literaturverzeichnis
- Korrektes **Zitieren**
- **Plagiarismus:**
Die nicht gekennzeichnete Übernahme fremder Inhalte führt zum **sofortigen Ausschluss**.
- 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:
 ≥ 10 Fehler pro Seite $\implies$ Abbruch der Korrektur

Vortrag

- 20-minütiger Vortrag (zwei pro Thema)
- 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

Übersicht

Einführung

Termine

Modelle

Logiken

Statische Analyseverfahren

Dynamische und Hybride Analyseverfahren

Verfahren

- **Foodle-Umfrage** unter <https://terminplaner.dfn.de/yvTfXBi5e6SowYqj>
- Bitte mindestens drei Ja-Stimmen ✓
- Möglichst weitere Vielleicht-Stimmen (✓)
- Als **Kommentar** angeben:
 - Themenpriorität (falls gewünscht)
 - Wunschpartner(in) (falls vorhanden)
- **Ausfüllen bis (spätestens) Montag, 11. April**
- Wir bemühen uns (ohne Garantie) um ein „optimales“ Matching
- Zuordnung der Themen und Betreuer bis ca. 13. April online

Rücktritt vom Proseminar

- Bis zu **drei Wochen** nach Themenvergabe: ohne Folgen
- Danach: Fehlversuch

Einführung in die Literaturrecherche

- Einweisung in themenspezifische Literaturrecherche
- [Video hier](#) und auf Seminarseite verlinkt

Deadlines

Folgende Termine sind **verpflichtend**:

- 11. April: Frist für Themenauswahl
- 6. Mai: letzte Rücktrittsmöglichkeit
- 9. Mai: Vorlage der detaillierten Inhaltsübersicht
 - nicht: „1. Einleitung/2. Hauptteil/3. Schluss“
 - sondern: Strukturübersicht (Abschnittsüberschriften, wesentliche Beweise/Theoreme, ...) und Anfang des Hauptteils (eine Seite)
- 7. Juni: vollständige Fassung der Ausarbeitung
- 27. Juni: vollständige Fassung der Folien
- 11./12./13. Juli (?): Blockseminar

Übersicht

Einführung

Termine

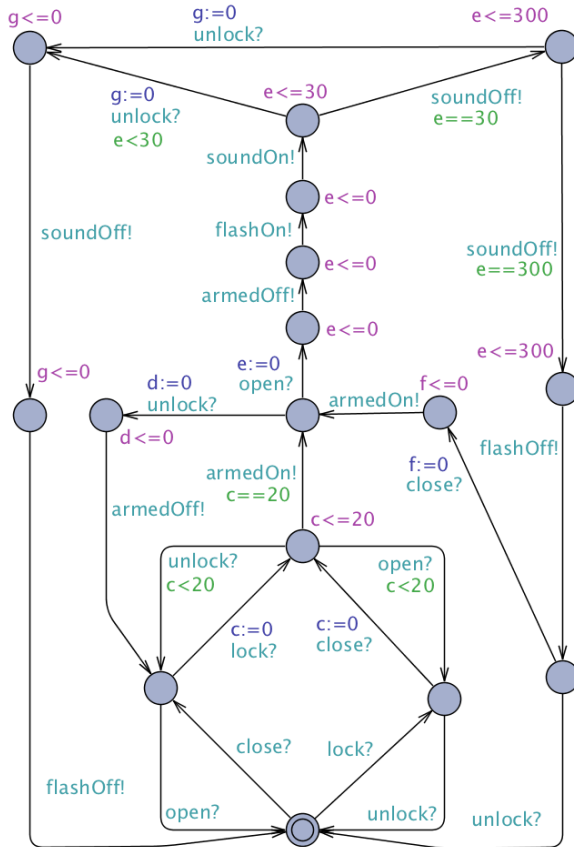
Modelle

Logiken

Statische Analyseverfahren

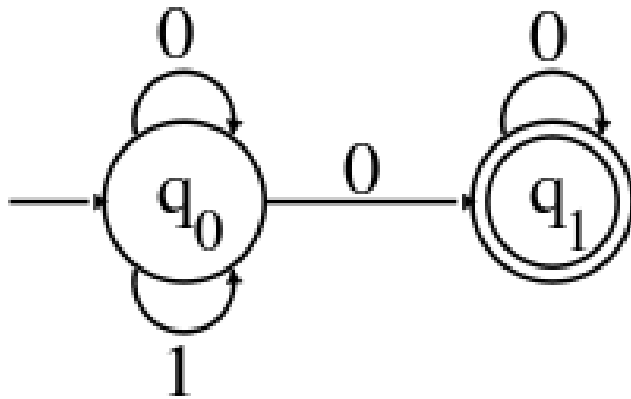
Dynamische und Hybride Analyseverfahren

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

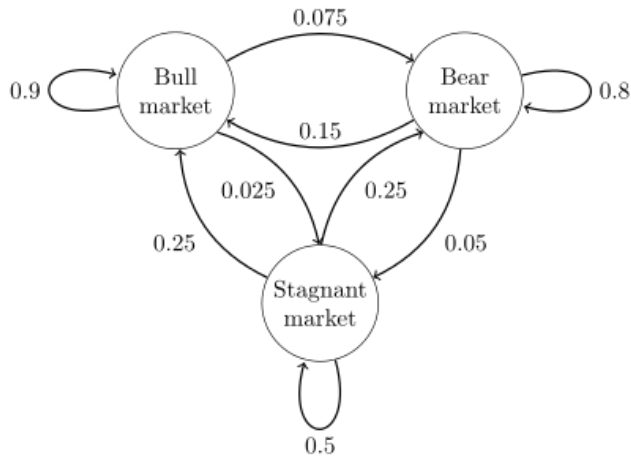
2. Büchi Automata



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

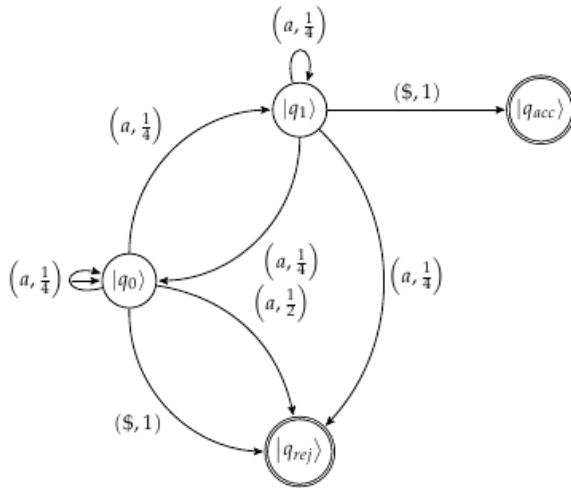
- Syntax identical to DFA/NFA
- But define languages of infinite words
- Accepts an infinite input sequence if there exists a run that visits one of the final states infinitely often
- Enables analysis of non-terminating system behaviour
- Non-deterministic variant more expressive

3. Markov Chains



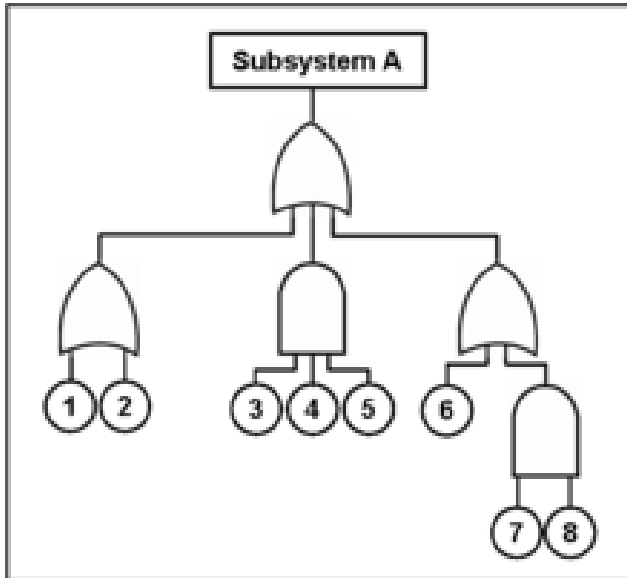
- States + probabilistic transitions
- Models „memoryless“ stochastic behaviour
- Two variants:
 - discrete time (one step per time unit, outgoing probabilities sum up to 1)
 - continuous time (outgoing transitions labelled with exit rates)
- Numerous applications

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

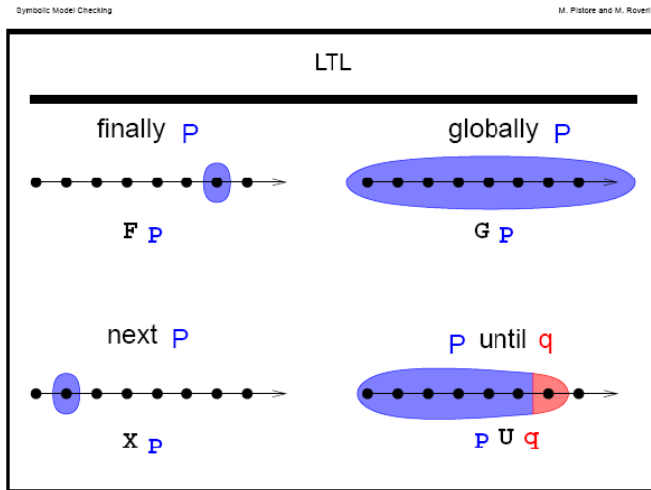
- Systematic construction of proof using logical rules of the form

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

- Critical part: deriving loop invariants

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

7. Linear-Time Temporal Logic (LTL)



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

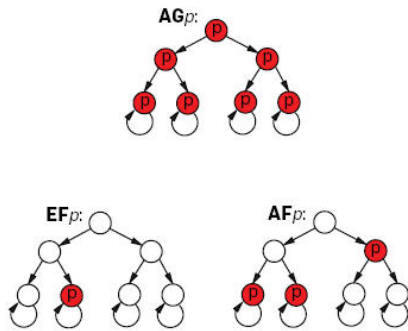
$$G(\neg \text{Bad})$$

- 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 arguments respectively hold

10. Hennessy–Milner Logic

- Modal logic used to specify properties of labelled transition systems
- Constructs:
 - tt satisfied by all states
 - ff satisfied by no state
 - $\varphi \wedge \psi$ satisfied by all states that satisfy both φ and ψ
 - $\varphi \vee \psi$ satisfied by all states that satisfy either φ or ψ or both
 - $\langle \alpha \rangle \varphi$ satisfied by all states that afford an α -labelled transition to a state satisfying φ (possibility)
 - $[\alpha] \varphi$ satisfied by all states such that all their α -labelled transitions lead to a state satisfying φ (necessity)
- Example: responsiveness

$[\text{request}] \langle \text{reply} \rangle \text{tt}$
- Extension by recursion (least and greatest fixed points) to support reasoning about arbitrarily long computations (e.g., „no deadlock state reachable“)

Übersicht

Einführung

Termine

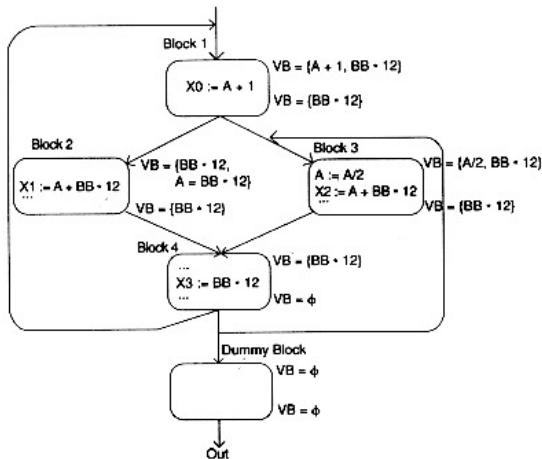
Modelle

Logiken

Statische Analyseverfahren

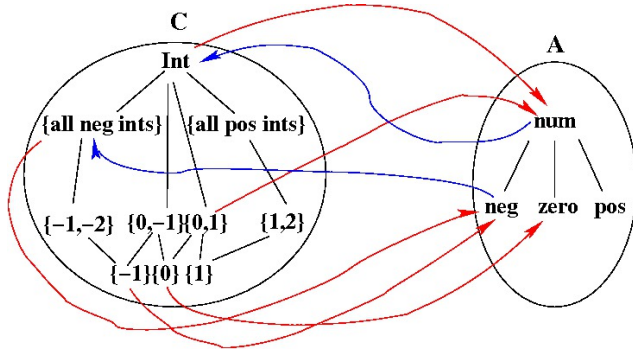
Dynamische und Hybride Analyseverfahren

11. 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
 - procedures:
 - interprocedural vs. intraprocedural analyses
- Approach: solution of data-flow equation system by fixpoint iteration

12. 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: all concrete computations captured by abstraction

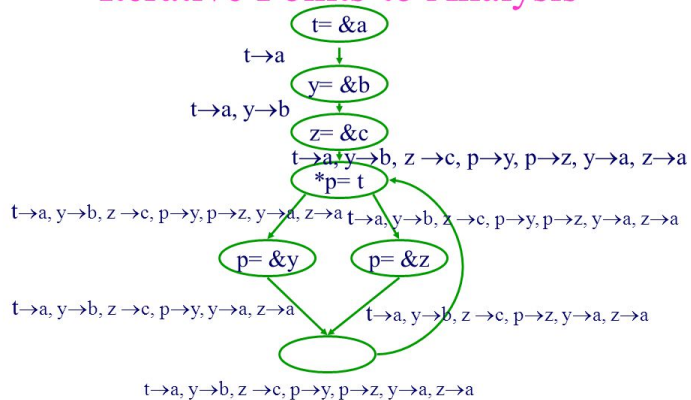
13. Type Systems

$$\text{(add)} \frac{e_1 : \text{int} \quad e_2 : \text{int}}{e_1 + e_2 : \text{int}}$$

- Type system: collection of rules that assign a type property to constructs of computer program
 - variables, expressions, functions, modules, ...
- Main purpose: reduce possibilities for bugs in computer programs
 - definition of interfaces between program parts
 - checking that parts have been connected in a consistent way
- Static vs. dynamic
- „Strong“ vs. „weak“

14. 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 (topic 16)
- Example:

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

15. Escape Analysis

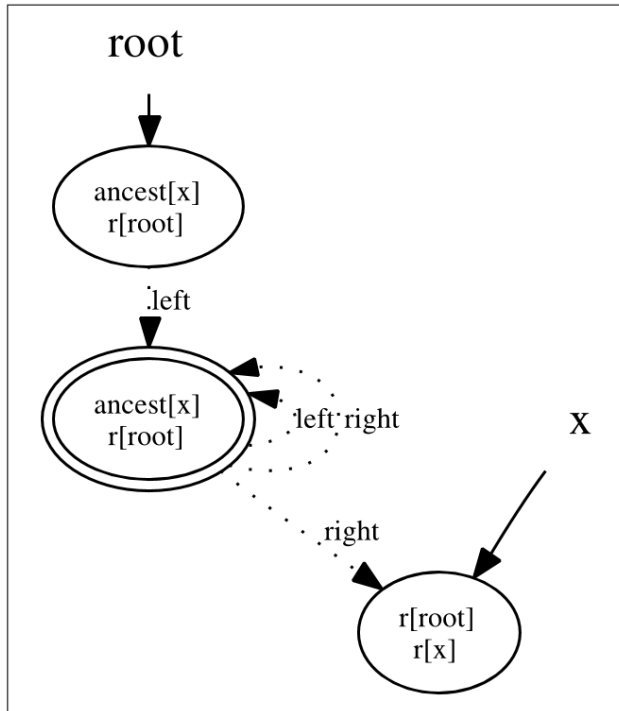
Escape analysis example

```
// Sum returns the sum of the numbers 1 to 100.
func Sum() int {
    numbers := make([]int, 100)
    for i := range numbers {
        numbers[i] = i + 1
    }
    var sum int
    for _, i := range numbers {
        sum += i
    }
    return sum
}
```

numbers never escapes Sum()

- Method for determining the dynamic scope of pointers
- When an object is allocated in a subroutine, a pointer to that object can escape to other contexts
- Non-escaping results can be used as basis for (compiler) optimisation:
 - converting heap allocations to stack allocations (avoids garbage collection)
 - removal of redundant synchronisation operations (if object accessible from one thread only)

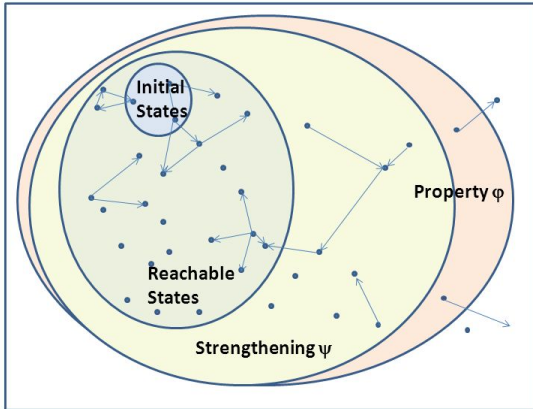
16. Shape Analysis



- Technique that discovers and verifies properties of linked, dynamically allocated data structures
- Used at compile time to find elementary bugs or to verify high-level correctness properties of programs
 - absence of memory leaks
 - non-destructive traversal
 - correctness of list-sorting method
- Finite-state abstraction of state space through summary nodes

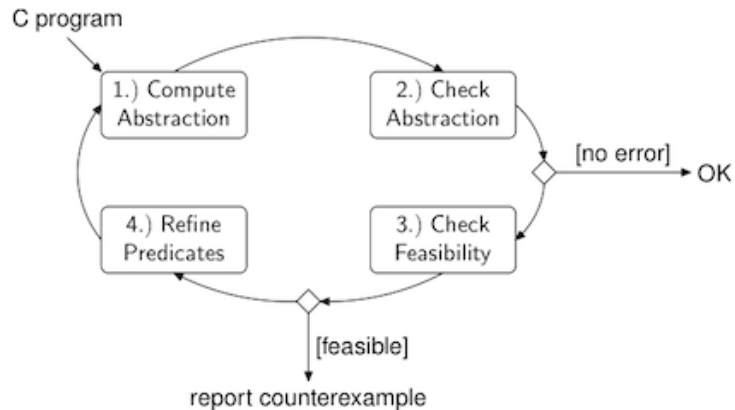
17. Inductive Invariants

Inductive Invariants



- Technique to establish invariant safety properties: „it is never the case that ...“
 - the value of variable x becomes zero
 - more than one process is in the critical section
 - ...
- Inductive:
 - satisfied by initial state(s) of program
 - closed under execution steps

18. Counterexample-Guided Abstraction Refinement (CEGAR)



- Iterative procedure for checking safety properties
- To cope with state explosion problem in state-space exploration
- Start with simple abstraction of system with only few states
- In each iteration, check whether abstract system satisfies property
 - if yes, system is safe
 - if no, check feasibility of counterexample
 - if feasible, system is unsafe
 - otherwise, refine abstraction

Übersicht

Einführung

Termine

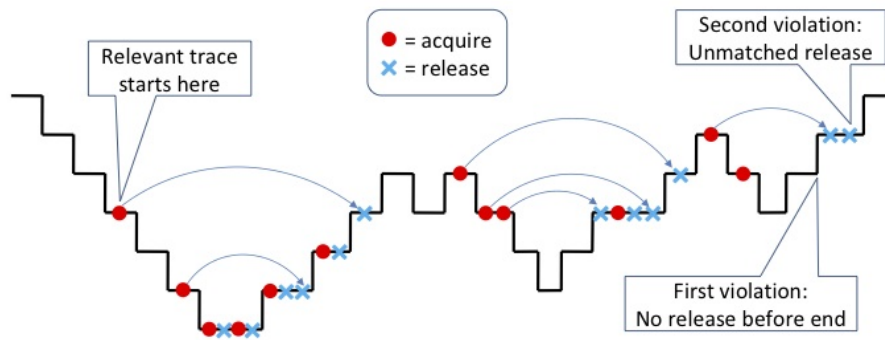
Modelle

Logiken

Statische Analyseverfahren

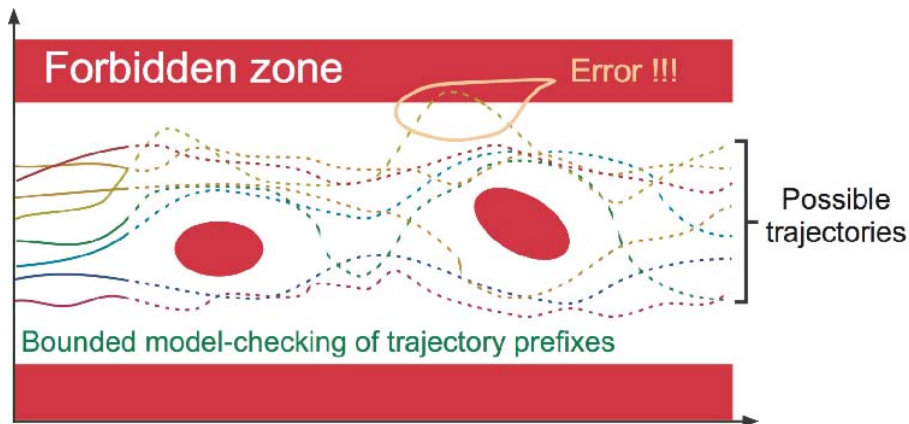
Dynamische und Hybride Analyseverfahren

19. Runtime Verification



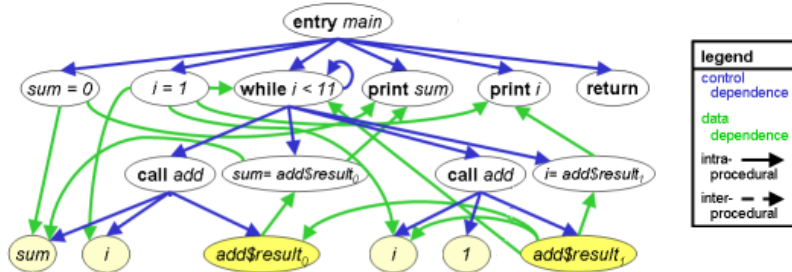
- Observation of running system to detect (and possibly react to) behaviours violating certain properties
 - deadlocks
 - data races
 - improper use of synchronisation mechanisms
 - ...
- Desired properties specified by predicates over execution traces
 - finite automata/regular expressions
 - context-free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

20. Bounded Model Checking



- Model checking: exploration of state space of a system to match against specification
- Usually exhaustive
⇒ state-space explosion problem
- Bounded model checking: fast exploration of bounded fragment of state space

21. Program Slicing



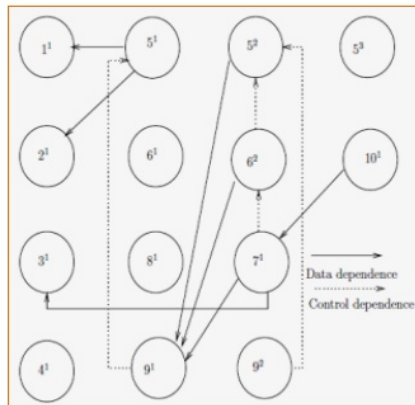
Dynamic Slicing: Example cont...

```

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
 - applications: debugging