Proseminar *Programmanalyse und Formale Methoden*

Einführung

Wintersemester 2025/26, 22. Oktober 2025

László Antal, Jozsef Kovacs, Jan-Gustav Michnia, Thomas Noll
Software Modeling and Verification Group
RWTH Aachen University

`https://moves.rwth-aachen.de/teaching/ws-2025-26/ipa/`

Chair of Software Modeling and Verification (Computer Science 2)
RWTH AACHEN UNIVERSITY

Übersicht

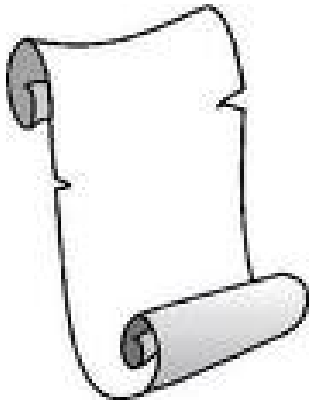
Traum der Programmanalyse

Programm

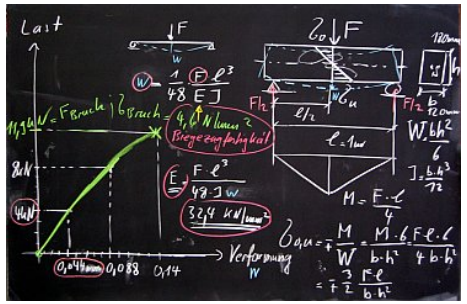
```
                print "ncfiles: Socket error (%s) for host %s (%s)" % (errno,
for h3 in page.findAll("h3"):
    value = (h3.contents[0])
    if value != "Afdeling":
        print >> txt, value
        import codecs
        f = codecs.open("alle.txt", "r", encoding="utf-8")
        text = f.read()
        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



Proseminar *Programmanalyse und Formale Methoden*
László Antal, Jozsef Kovacs, Jan-Gustav Michnia, Thomas Noll
Wintersemester 2025/26

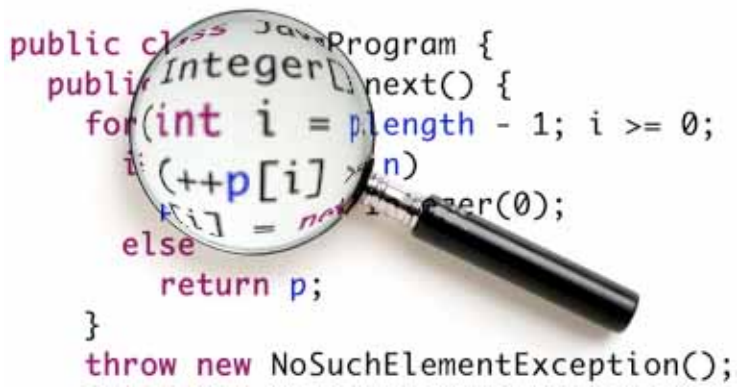
Chair of Software Modeling and Verification (Computer Science 2)
RWTH AACHEN UNIVERSITY

Seminarthema

Thema des Proseminars

Einführung in Grundkonzepte der (automatisierten) Programmanalyse und Formaler Methoden

- 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
- Formale Methoden für komplexe Systeme:
  - Neuronale Netzwerke
  - Hybride Systeme



Zielsetzung

Ziele des Proseminars

- Selbstständiges Einarbeiten in ein neues Thema
- Literaturrecherche
  - Einstiegsreferenz auf Webseite
  - Weitere Literatur online und in Informatikbibliothek
- Verständliches Präsentieren
- Kurzdarstellung des Inhalts in einer wissenschaftlich orientierten Ausarbeitung

Anforderungen Vortrag

Vortrag

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

Anforderungen Ausarbeitung

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
- LaTeX-Vorlage wird zur Verfügung gestellt
- Sprache Deutsch oder Englisch
- Korrekte Sprache wird vorausgesetzt

Übersicht

Themenauswahl

Verfahren

- Themenliste wurde/wird ausgehändigt
- Priorisierte Auswahl
- Abgabe hier oder bis (spätestens) Montag, 27. Oktober 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

Bibliothekseinführung

Einführung in die Literaturrecherche

- Einweisung in themenspezifische Literaturrecherche
- Dauer: ca. 90 min
- Teilnahme für BSc-Studierende verpflichtend
- Termin: Dienstag, 28.10., 15:30 Uhr
- Teilnahme bitte auf Themenliste vermerken

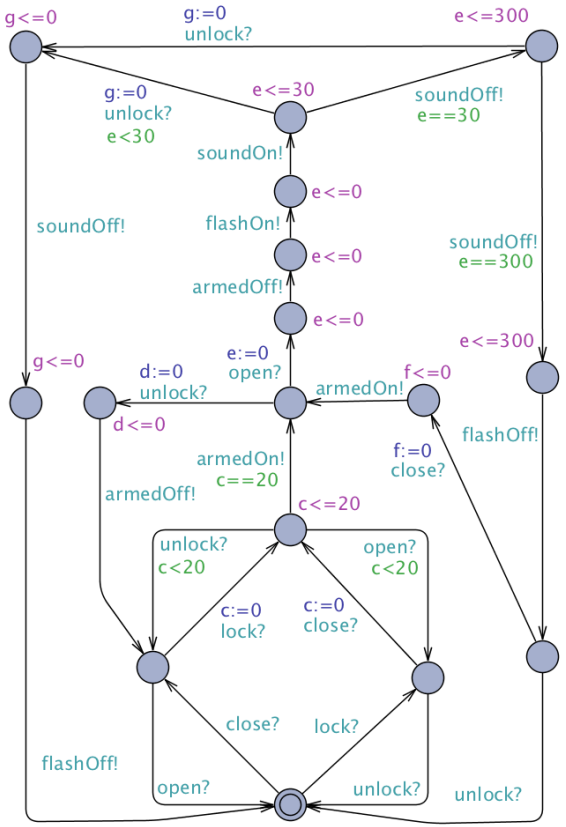
Deadlines

Folgende Termine sind verpflichtend:

- 27. Oktober: Frist für Themenauswahl
- 3. November: letzte Rücktrittsmöglichkeit
- 17. November: Vorlage des detaillierten Vortragsstruktur
  - nicht: „1. Einleitung/2. Hauptteil/3. Schluss“
  - sondern: detaillierte Strukturübersicht (Einteilung in Abschnitte, wesentliche Definitionen/Aussagen/Beispiele, ...)
- 8. Dezember: vollständige Fassung der Vortragsfolien
- 4. Februar (?): Blockseminar
- 18. Februar: Einreichung der Ausarbeitung

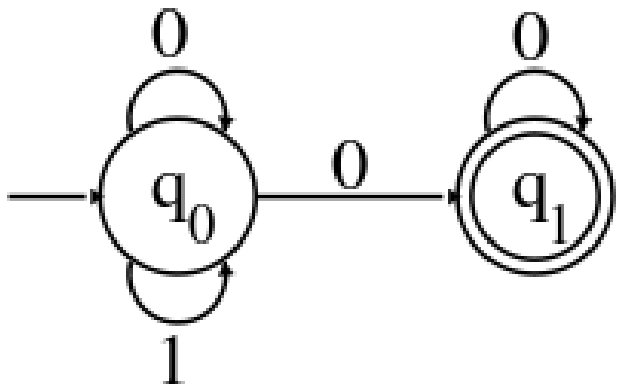
Übersicht

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

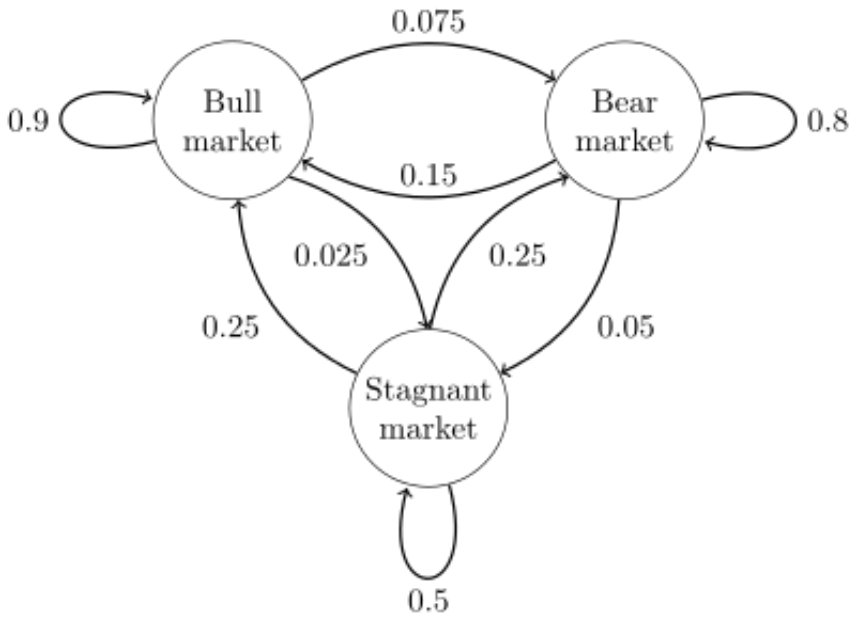
2. Büchi Automata



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

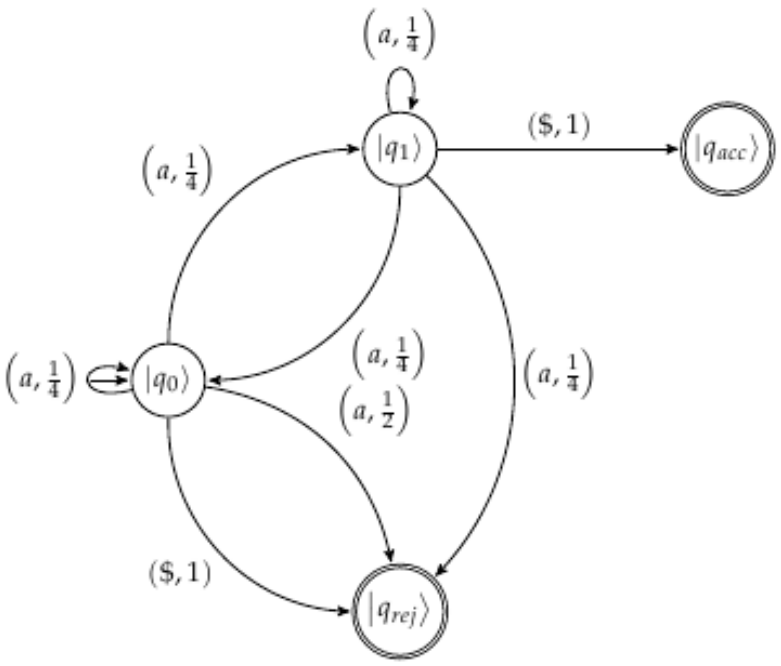
- Syntax identical to DFA/NFA
- But define languages of infinite words
- Accepts an infinite input sequence if there exists a run that visits one of the final states infinitely often
- Enables analysis of non-terminating system behaviour
- Non-deterministic variant more expressive

3. Markov Chains



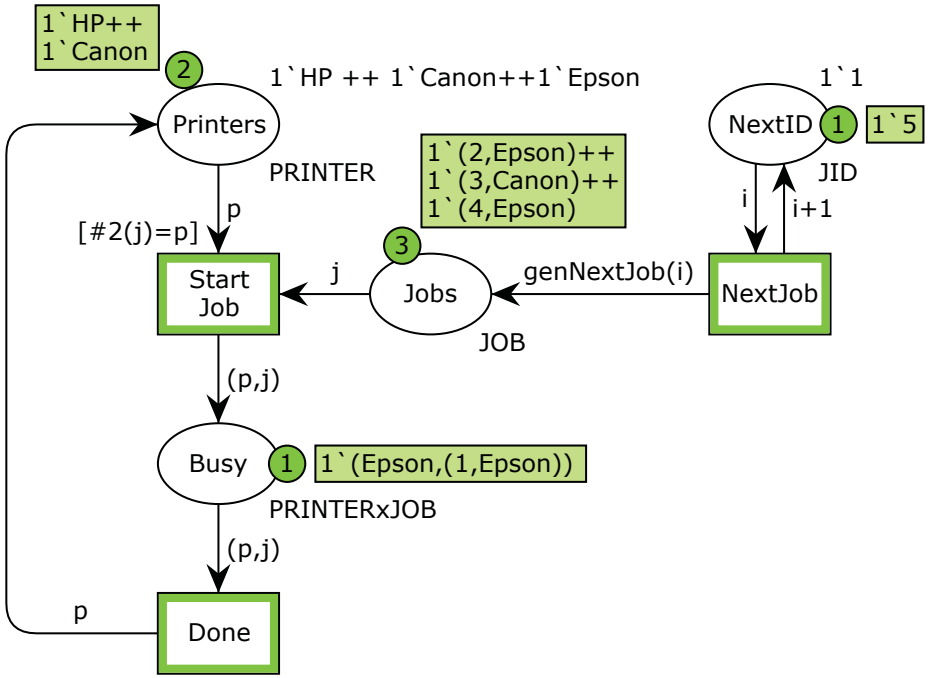
- States + probabilistic transitions
- Models „memoryless“ stochastic behaviour
- Two variants:
  - discrete time (one step per time unit, outgoing probabilities sum up to 1)
  - continuous time (outgoing transitions labelled with exit rates)
- Numerous applications

4. Probabilistic Automata



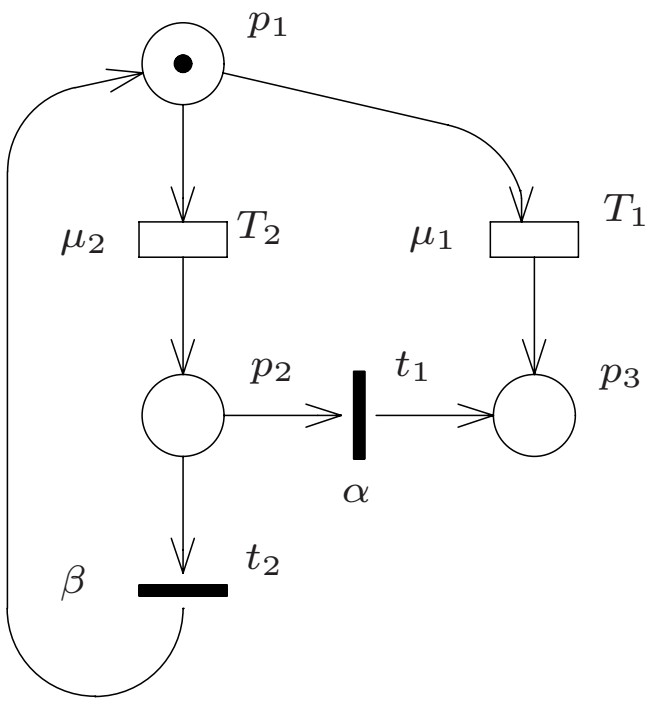
- Generalisation of
  - finite automata: + transition probabilities
  - Markov chain: + transition labels
- Define stochastic languages as sets of words that are recognised with a certain (minimal) probability
  - regular languages are proper subclass

5. Colored Petri Nets



- Extension of Petri Nets
  (places, transitions, token, firing, ...)
- Additionally: colors assigned to tokens
- Enables modelling of different kinds of jobs etc.

6. Generalized Stochastic Petri Nets



- Another extension of Petri Nets adding timing
- Two classes of transitions:
  - (exponentially distributed) timed transitions: model random delays associated with the execution of activities
  - immediate transitions: represent logical actions that do not consume time
- Many applications:
  - performance analysis of communication/production/... systems
  - reliability analysis of HW/SW systems (fault trees, ...)

Übersicht

$$\dfrac{\dfrac{\{A \wedge b\}\, c\, \{B\} \qquad \dfrac{(A \wedge \neg b) \Rightarrow B}{\{(A \wedge \neg b)\}\ \texttt{skip}\ \{B\}}}{\{A\}\ \texttt{if}\ b\ \texttt{then}\ c\ \texttt{else skip}\ \{B\}}}{\{A\}\ \texttt{if}\ b\ \texttt{then}\ c\ \{B\}}$$

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

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

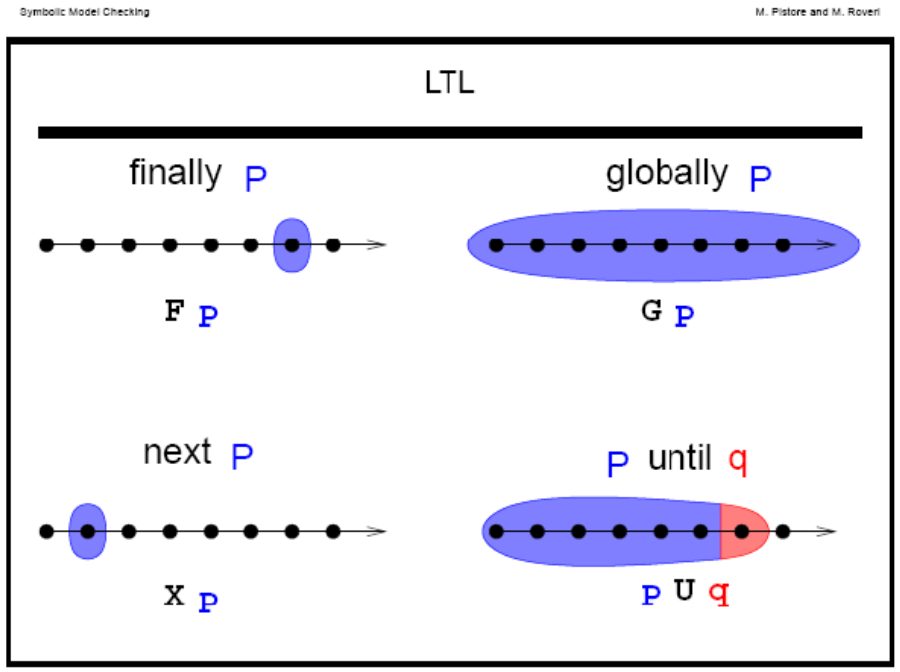
  with assertions A, B and program c
- Systematic construction of proof using logical rules of the form

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

- Critical part: deriving loop invariants

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

8. Linear-Time Temporal Logic (LTL)



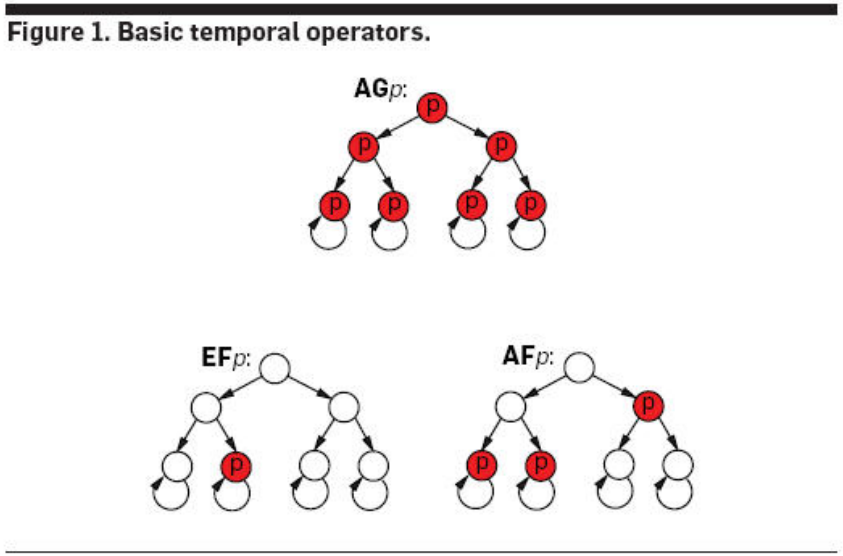
- Modal temporal logic
- Formulae specify properties of infinite system traces
- Safety: something bad never happens

$$G(\neg Bad)$$

- Liveness: something good will happen

$$G(Request \implies F\ Response)$$

9. Computation Tree Logic (CTL)



- 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

10. Separation Logic

Frame rule of SL:

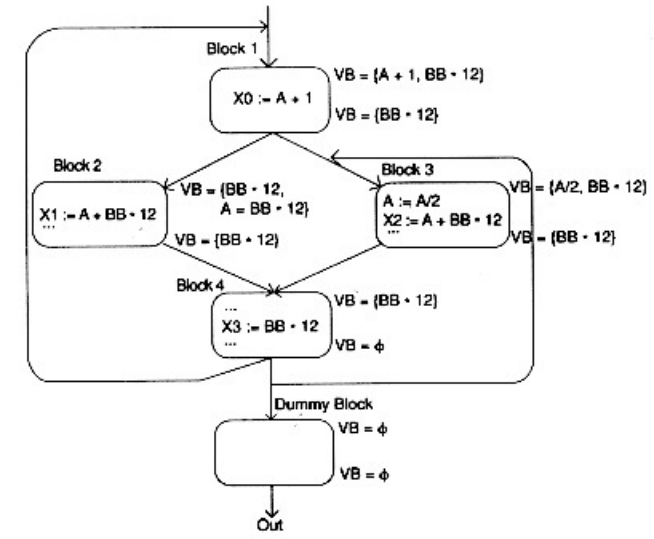
$$\text{(frame)}\ \frac{\{P\}\, c\, \{Q\}}{\{P * R\}\, c\, \{Q * R\}}$$

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

- 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

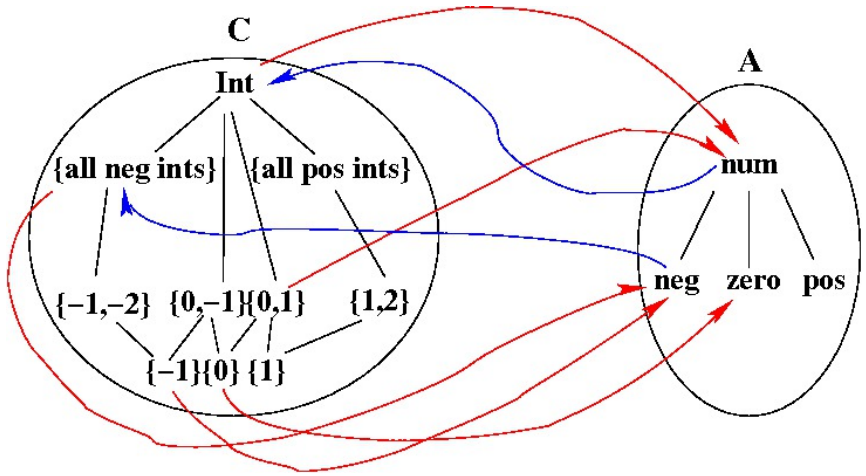
Übersicht

11. Data-Flow Analysis

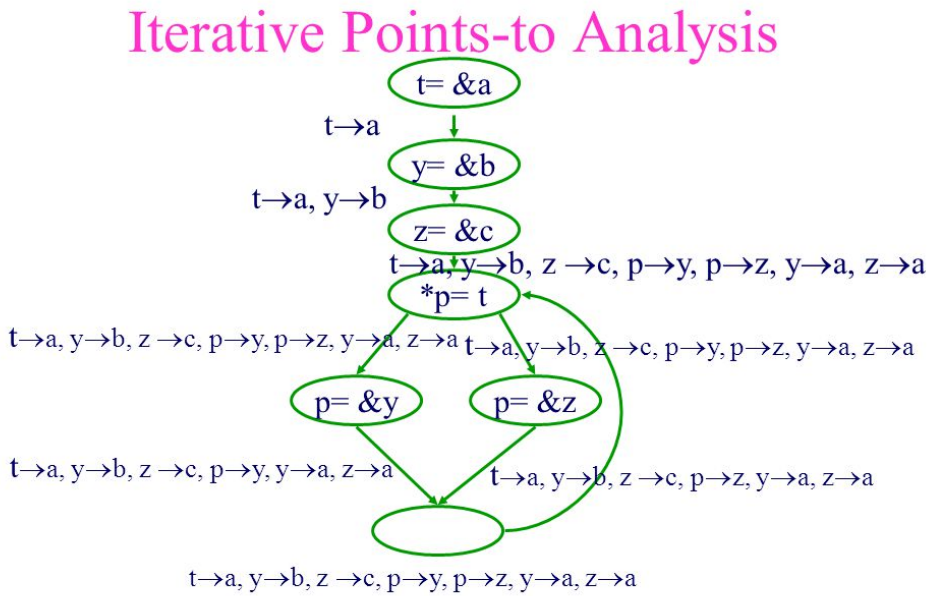


- Technique for associating information with control points of computer program
  - values of variables
  - available expressions
  - live variables, ...
- Distinctions:
  - dependence on statement order:
    flow-sensitive vs. flow-insensitive analyses
  - direction of flow:
    forward vs. backward analyses
  - quantification over paths:
    may (union) vs. must (intersection) analyses
  - handling of 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: every concrete computation captured by some abstract execution of program



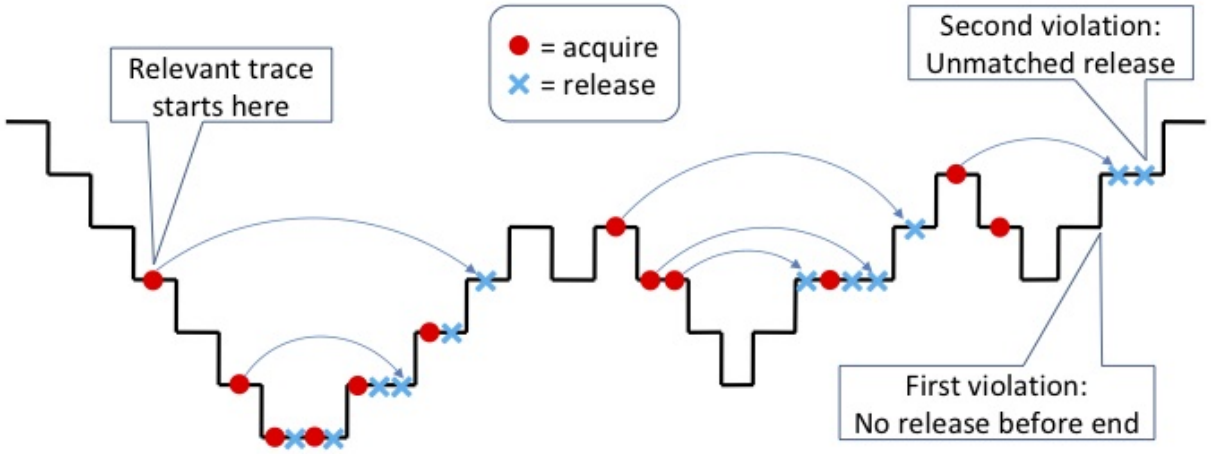
- Static code analysis technique that establishes which pointers (heap references) can point to which variables (storage locations)
- Often a basic component of more complex analyses such as escape analysis
- Example:

```
int x;
int y;
int* p = unknown() ? &x : &y;
```

yields $\{\texttt{x}, \texttt{y}\}$ as points-to set of `p`

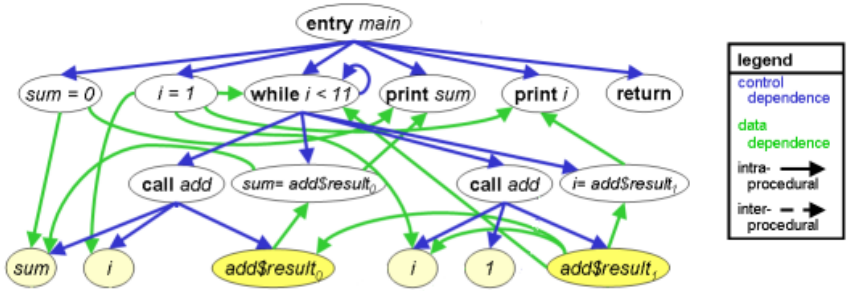
Übersicht

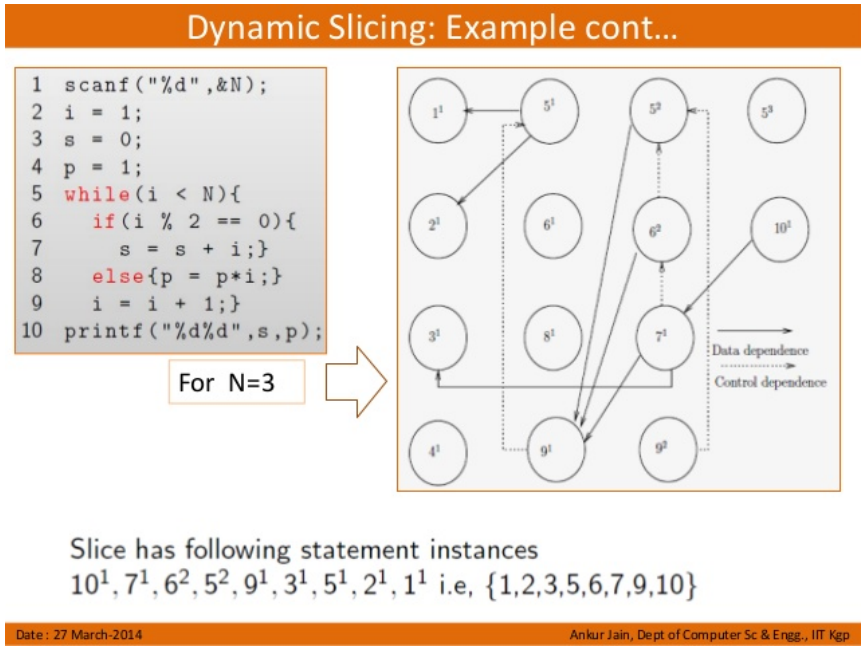
14. Runtime Verification



- 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, ...)
  - ...

15. Program Slicing

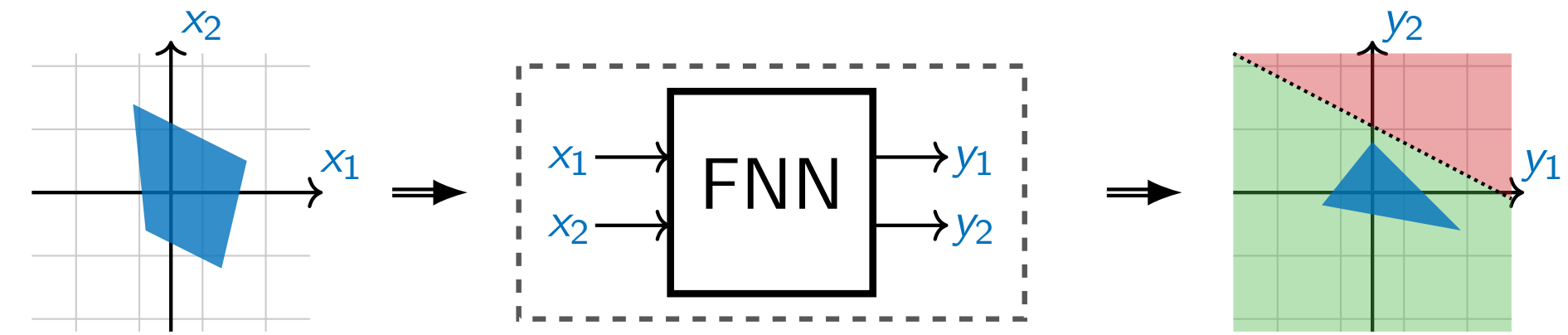




- Computation of the part of program („program slice“) that may affect the values at some point of interest („slicing criterion“)
- Static slicing:
  - no assumptions regarding input
  - based on program dependence graph
  - applications: software maintenance (regression testing), information flow control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

Übersicht

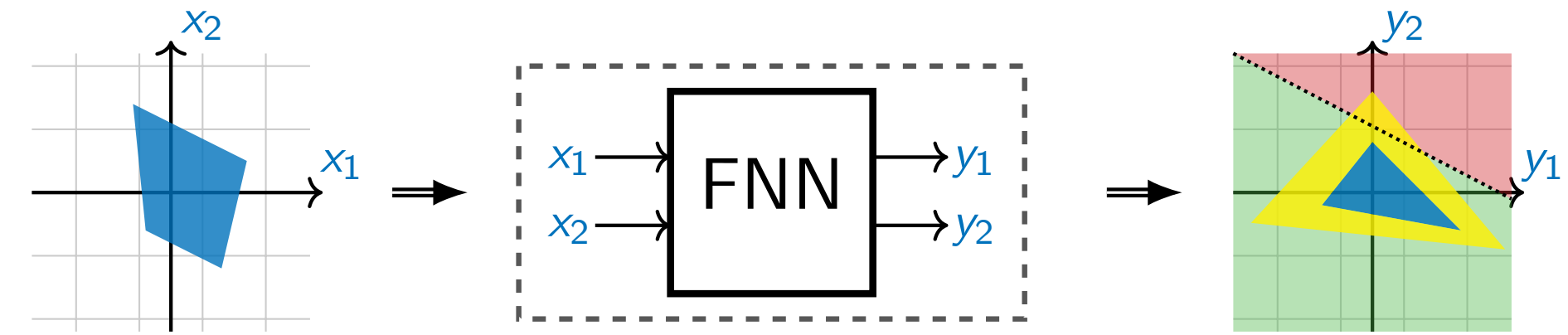
A Neural Network Verification



Reachability Analysis based Verification

Exact
Complete
Sound
Slow

A Neural Network Verification

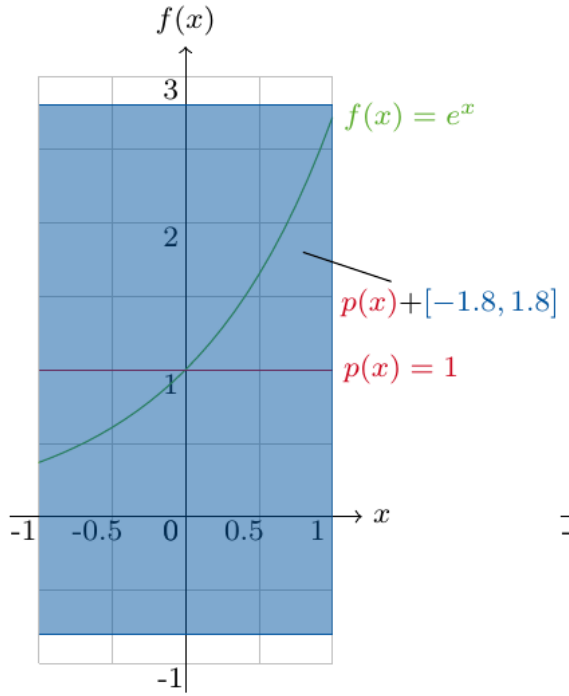


Reachability Analysis based Verification

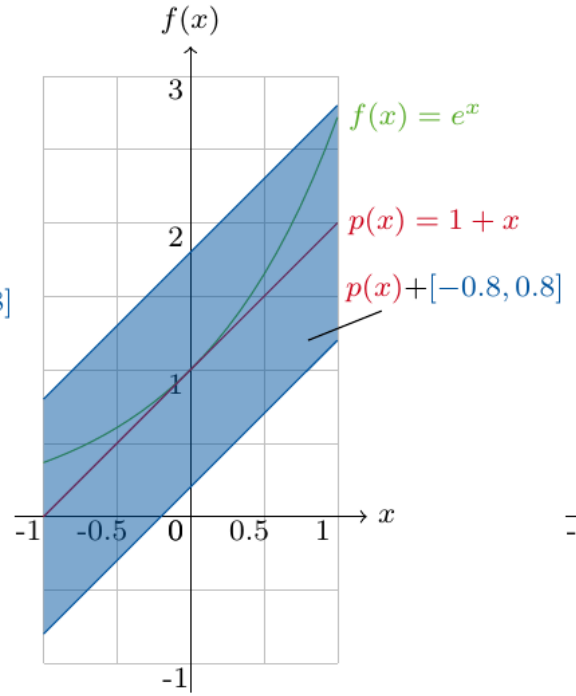
Exact	Over-Approximate
Complete	Incomplete
Sound	Sound
Slow	Fast

B Taylor models

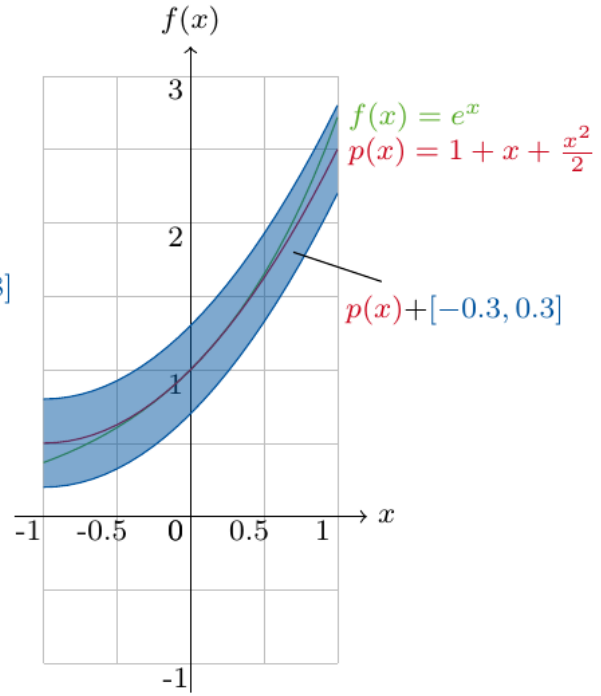
$$f(a) + \frac{f'(a)}{1!}(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \cdots = \sum_{n=0}^{\infty} \frac{f^{(n)}(a)}{n!}(x-a)^n.$$



0-order over-approximation



1-order over-approximation



2-order over-approximation