Zeitlogik und Verifikation

Helmut Veith, veith@dbai.tuwien.ac.at
58801-18431

Abteilung f. Datenbanken und AI

Zeitlogik

- Klassische Logik ist ungeeignet, die Dynamik veränderlicher Systeme zu beschreiben.
- Zeitlogik enthält spezielle Operatoren, um Zeit auszudrücken.
- Beispiel:
  F(ampel1_rot & ampel2_gruen)

"In der Zukunft (**F**uture) wird Ampel 1 rot und Ampel 2 grün."

2

Hardware Verifikation

- Chipdesigns sind sehr komplex und enthalten komplizierte Kontrollmechanismen.
  $> 10^8$ Gatter $> 2^{10^8}$ Zustände
- Designs werden automatisch aus Programmen in zB VHDL oder VERILOG generiert.
- Bugs in späten Designphasen (Pentium ...) sind sehr teuer und aufwendig.
- 30% to 70% der Entwicklungsingenieure sind mit Verifikation im weitesten Sinn beschäftigt: Simulation, Testen, Äquivalenzprüfung.

3

Verifikation und Zeitlogik

Verifikation von Hardware und Software:

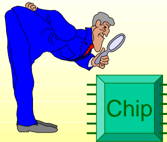
Zeitlogik wird verwendet, um Spezifikationen auszudrücken, deren Korrektheit vollautomatisch durch einen effizienten Suchalgorithmus verifiziert wird.

Diese Methode heisst Model Checking, und wird in der Praxis industriell eingesetzt.

Intel, IBM, Microsoft, Fujitsu, Siemens, Synopsis, ...

4

Was ist Model Checking ?



Chip |= φ

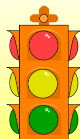
Transitionssystem
= Kripkestruktur
= Endlicher Automat

Specifikation
= Formel in Zeitlogik

5

Eine einfache Kripke Struktur

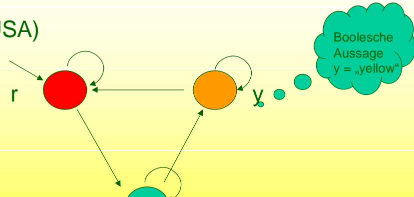
Ampel (USA)



6

Eine einfache Kripke Struktur

Ampel (USA)



Spur:
<r><g><y><r><g><y>...

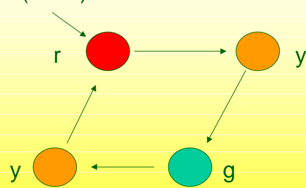
Spuren:
<r><r><g><y><y>...

Wie unterscheiden sich Ampeln in Wien ?

7

Wiener Kripke Struktur

Ampel (Wien)



Spur: <r><y><g><y><r><y>...

8

K-Strukturen und “echte” Systeme

- Betrachte Hardwaredesigns als Kripke Strukturen. (Mealy-Maschinen !)
- Hardwaresprachen wie SMV, VERILOG werden verwendet, um die Funktion des Designs zu beschreiben.
- Ein Compiler übersetzt das Programm in eine K-Struktur.
- Die K-Struktur ist i.A. nicht deterministisch.

9

SMV: Programmbeispiel

```
MODULE main
VAR
 request: boolean;
 state: {ready, busy};
ASSIGN
 init(state) := ready;
 next(state) :=
   case
    state = ready & request : busy;
    1: {ready, busy};
   esac;
```

10

Logik & Zeit: zwei Philosophien

Lineare Zeit:

Betrachte Zeitverhalten auf allen möglichen Spuren einzeln.

Verzweigte Zeit:

Betrachte alle möglichen Verzweigungen der K-Struktur.

Im Prinzip ausdrucksstärker als lineare Zeit.

Lineare Zeit = Spezialfall

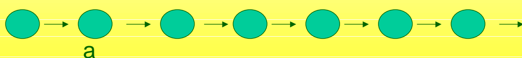
11

LTL - Linear Time Logic

- auf Spuren definiert, d.h. lineare Zeit.
- basiert auf Aussagenlogik
- zwei Zeitoperatoren:

X a — *"a gilt im nächsten (neXt) Zustand."*

a U b — *"a gilt und bleibt wahr bis (Until) b wahr wird."*



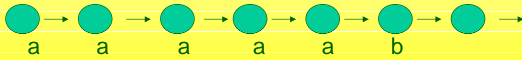
12

LTL - Linear Time Logic

- auf Spuren definiert, d.h. lineare Zeit.
- basiert auf Aussagenlogik
- zwei Zeitoperatoren:

X a — *"a gilt im nächsten (neXt) Zustand."*

a U b — *"a gilt und bleibt wahr bis (Until) b wahr wird."*

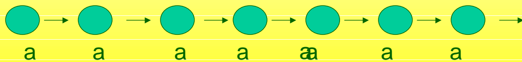


Nochmals LTL

Weitere Operatoren:

Fa := true U a — "a wird einmal wahr sein"

Ga := ¬ F ¬ a — "a wird immer wahr sein"



LTL auf Kripke Strukturen

Eine Formel ist auf einer K-Struktur wahr, wenn sie auf allen Spuren wahr ist.

Beispielformel G(request -> F ack)

"Every request will be eventually acknowledged."

Logik & Zeit: zwei Philosophien

Lineare Zeit:

Betrachte Zeitverhalten auf allen möglichen Spuren einzeln.

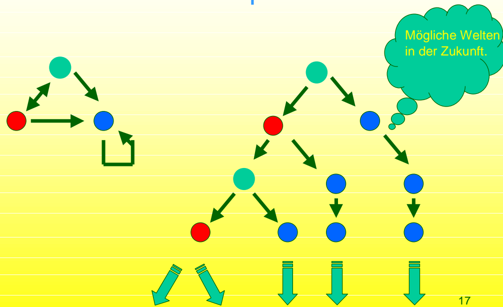
Verzweigte Zeit:

Betrachte alle möglichen Verzweigungen der K-Struktur.

Im Prinzip stärker als lineare Zeit.
Lineare Zeit = Spezialfall

16

Verzweigte Zeit: Abwicklen von Kripke Strukturen



17

CTL*: Extended Computation Tree Logic

- Definiert auf abgewickelten K-Strukturen (= unendlichen Bäumen)
- CTL* = LTL + Quantoren:

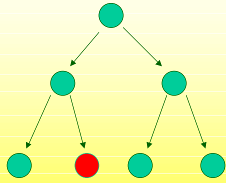
  A φ: " φ gilt auf allen Spuren"
  E φ: " φ gilt auf manchen Spuren"

- CTL = wie CTL, aber nur spezielle Zeitoperatoren sind erlaubt: AX, EX, AF, EF, AG, EF, AU, EU.

Vorteil von CTL: Modell Checking in Zeit O(n).

18

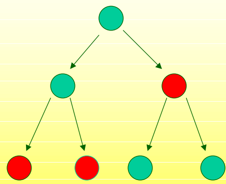
CTL: Beispiel



EF g "g wird potentiell wahr"

19

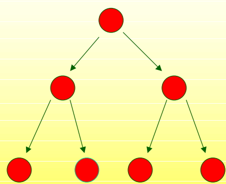
CTL: Beispiel



AF g "g wird sicher wahr"

20

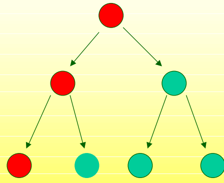
CTL: Beispiel



AG g "g ist eine Invariante"

21

CTL: Beispiel

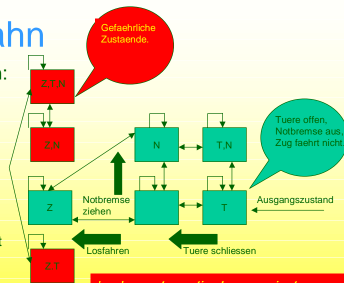


EG g "g ist eine potentielle Invariante"

22

U-Bahn

Boolesche Aussagen:
T... Tuere offen
Z... Zug Faehrt
N... Notbremse



CTL Spezifikationen:

AG (T→ ¬Z)
„Bei geoeffneter Tuere faehrt der Zug niemals."

A(¬Z U (¬T & ¬N))
„Der Zug bleibt solange stehen, bis die Tuer geschlossen und die Notbremse ausgeschalten ist."

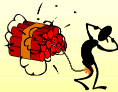
In der automatisch generierten K-Struktur gibt es auch gefaehrliche Zustaende !

23

Grosse Kripke Strukturen

Hauptproblem im Model Checking:

State Explosion Problem



Die Zahl der möglichen Zustände kann extrem gross werden. Typische Hardwaredesigns haben $2^{1000000}$ oder mehr Zustände.

Solche Designs können mit den dzt. Methoden nicht vollständig verifiziert werden.

Linearzeitalgorithmen sind nur ein erster Schritt.

24

State Explosion

Gründe für State Explosion:

- Die K-Struktur ist oft exponentiell grösser als das Programm:
  - Parallele synchrone Prozesse
  - Instantiierung der Variablen
  - Verschachtelte Prozeduren
- Der Raum aller möglichen Zustände wird durchsucht.

25

State Explosion: 2 Ansätze

Konservativ:

Darstellung der K-Strukur durch eine spezielle Datenstruktur, d.h. Information komprimieren.

zB symbolisches Model Checking.

Aggressiv:

Verwende Wissen über die K-Struktur, um von irrelevanten Details zu abstrahieren. d.h. Information filtern.

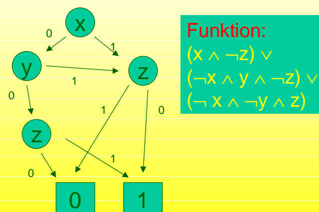
zB existentielle Abstraktion.

26

Ordered Binary Decision Diagrams

OBDDs (Bryant):

graph-basierte Datenstruktur für Boolesche Funktionen.



Funktion:
$(x \wedge \neg z) \vee$
$(\neg x \wedge y \wedge \neg z) \vee$
$(\neg x \wedge \neg y \wedge z)$

Variablenordnung:
$x < y < z$

27

Symbolic Model Checking

1991: McMillan und Clarke verwenden OBDDs, um K-Strukturen komprimiert darzustellen.

Das System SMV führt zu einem Paradigmenwechsel in der Verifikation, und erzeugt grosses industrielles Interesse.

Basiert auf “Symbolischen Algorithmen”, d.h., Algorithmen, die effektiv mit OBDDs anstatt von expliziten Strukturen operieren.

28

Abstraktion

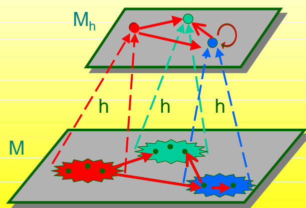
- Viele Spezifikationen beziehen sich nur auf einen Teil des Designs

  - Manuelle Abstraktion ist sehr aufwendig
  - Automatische Abstraktion ist schwierig

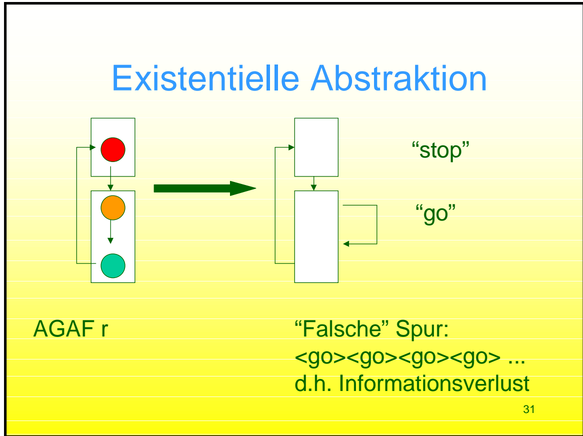
29

Existentielle Abstraktion

Eine Funktion h bildet konkrete Zustände auf abstrakte Zustände ab.



30



Verifikation von Software

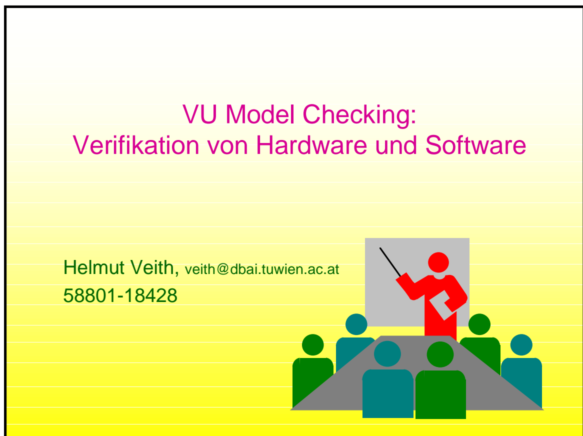
Spezielle Probleme:

- Dynamische Datenstrukturen
- Dynamische Threads (Java!)
- Rekursive Prozeduren
- Sicherheit von Protokollen

Neue Anwendungen:

- Realtime Software
- Security Protokolle
- UML/Statecharts

32



Ablauf der VU

- Vorlesungsblock (17.-18. November)
- Übungen mit Softwarepaketen zur Verifikation; Ziel der Übungen: Präsentation der Software (Funktionalität, Aufbau, Erweiterbarkeit ...)

34

VIELEN DANK FÜR IHRE AUFMERKSAMKEIT

35