Qualitätssicherung von Software

Prof. Dr. Holger Schlingloff

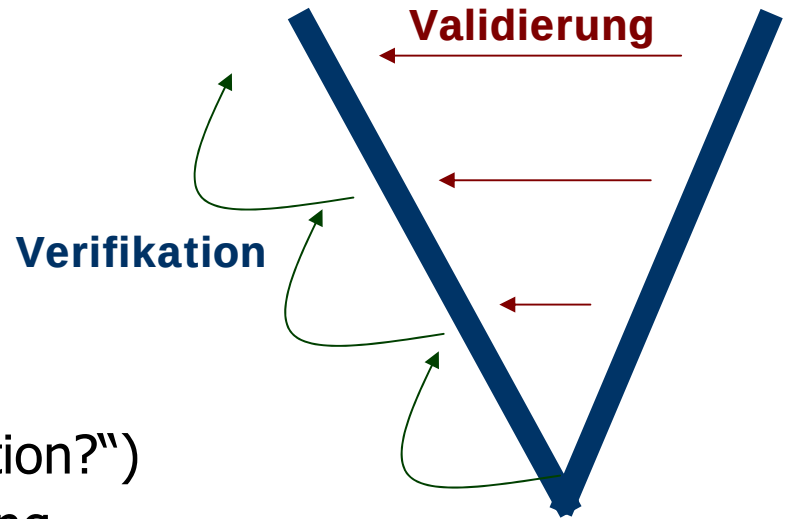
Humboldt-Universität zu Berlin
und
Fraunhofer FIRST

Inhaltsplanung (20.10.)

1. Einleitung, Begriffe, Software-Qualitätskriterien
2. manuelle und automatisierte Testverfahren
3. Verifikation und Validierung, Modellprüfung
4. statische und dynamische Analysetechniken
5. Softwarebewertung, Softwaremetriken
6. Codereview- und andere Inspektionsverfahren
7. Zuverlässigkeitstheorie, Fehlerbaumanalyse
8. Qualitätsstandards, Qualitätsmanagement, organisatorische Maßnahmen

Verifikation und Validierung

- Sprachgebrauch im V-Modell
  - **Verifikation:** Vergleich des Ergebnisses eines Entwicklungsschritts mit dem vorherigen
  - **Validierung:** Vergleich des analytischen Artefaktes mit dem entsprechenden synthetischen Artefakt



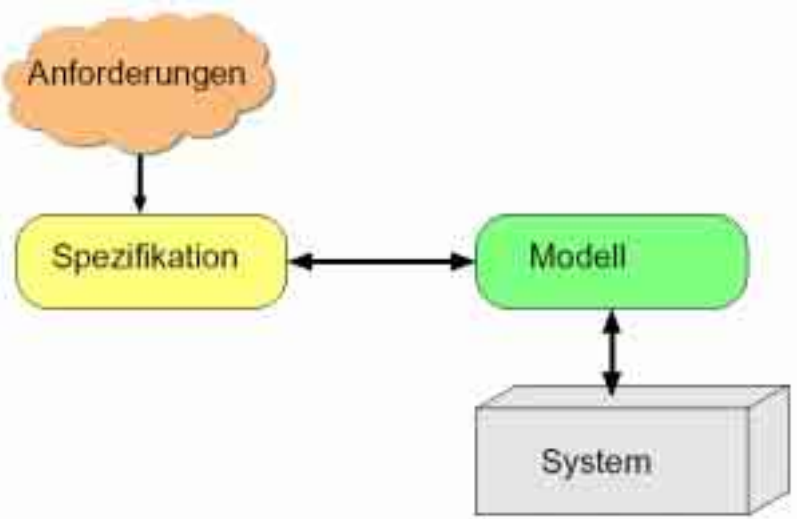
- alternativer Sprachgebrauch
  - **Verifikation** = formaler Beweis
    („entspricht der Code der Spezifikation?")
  - **Validierung** = informelle Überprüfung
    („entspricht die Spezifikation den Anforderungen?")

Techniken für V&V

- **Validierung**
  - Requirements Engineering, strukturierte Analyse
  - Konsistenz- und Vollständigkeitsprüfung
  - Schablonentechnik, systematische Fragebögen
  - ...
- **Verifikation**
  - Hoare-Kalkül, mathematische Programmbeweise
  - Modellprüfung, Erreichbarkeitsgraphen
  - interaktive Abstraktions/Verfeinerungsbeweise
  - ...

Modellprüfung

- Überprüfung eines formalen Modells des Systems gegenüber einer formalen Spezifikation der Anforderungen
- vollautomatisch für zustandsendliche Systeme

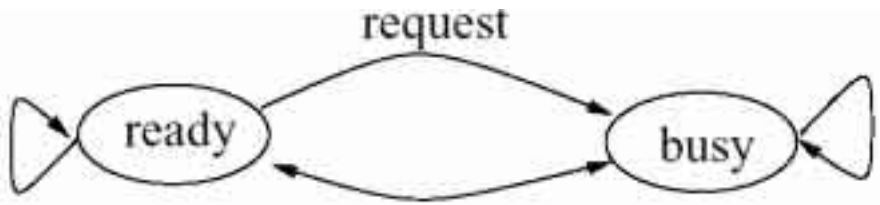


- Modellierung: Automaten, Petrinetze, StateCharts, SDL, …
- Spezifikation: temporale Logik, Erreichbarkeit, Verklemmungsfreiheit, …

Beispiel (SMV)

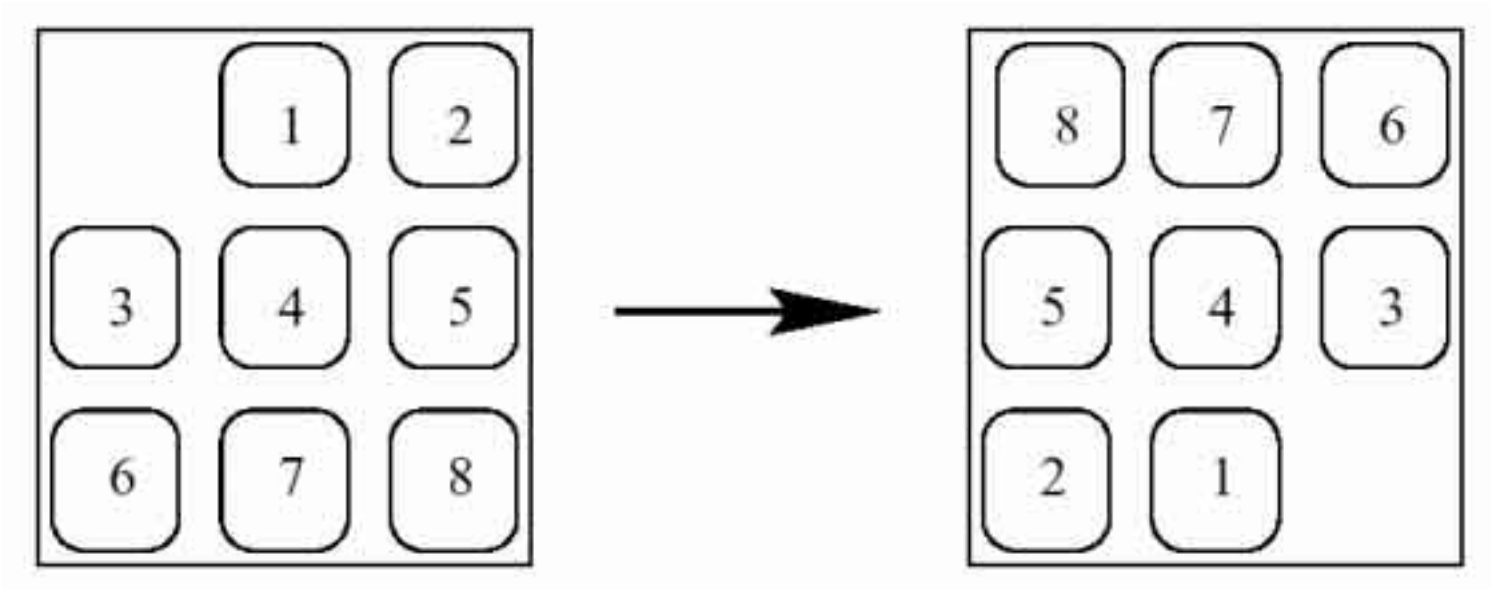
```
MODULE main
VAR request:  boolean;
    state:  {ready, busy};
ASSIGN
  init(state) := ready;
  next(state) := case
    state=ready & request: busy;
    1:  {ready, busy}; esac;

DEFINE is_busy := state = busy;
SPEC AG(request -> AF is_busy)
SPEC AG(request -> AG is_busy)
```




```
-- specification AG (request -> AF is_busy) is true
-- specification AG (is_busy -> AG is_busy) is false
-- as demonstrated by the following execution
-- loop starts here --
state 1.1: is_busy = 0
           request = 0
           state = ready
state 1.2: is_busy = 1
           state = busy
state 1.3: is_busy = 0
           state = ready
resources used:
BDD nodes allocated: 216; Bytes allocated: 917504
BDD nodes representing transition relation: 5 + 1
```

Ein Spielbeispiel



- Wie viele erreichbare Zustände gibt es?
- Wie kann man feststellen, ob ein Zustand erreichbar ist?

Coding in SMV

```
MODULE main
DEFINE h := 3; v := 3;
VAR move: u,d,l,r;
    hpos: array 0..(h*v-1) of 1..h;
    vpos: array 0..(h*v-1) of 1..v;
ASSIGN
next(vpos[0]) := case
  (move=l) & !(vpos[0]=1) : vpos[0] - 1;
  (move=r) & !(vpos[0]=v) : vpos[0] + 1;
  1: vpos[0]; esac;
next(hpos[0]) := case
  (move=u) & !(hpos[0]=1) : hpos[0] - 1;
  (move=d) & !(hpos[0]=h) : hpos[0] + 1;
  1: hpos[0]; esac;
```

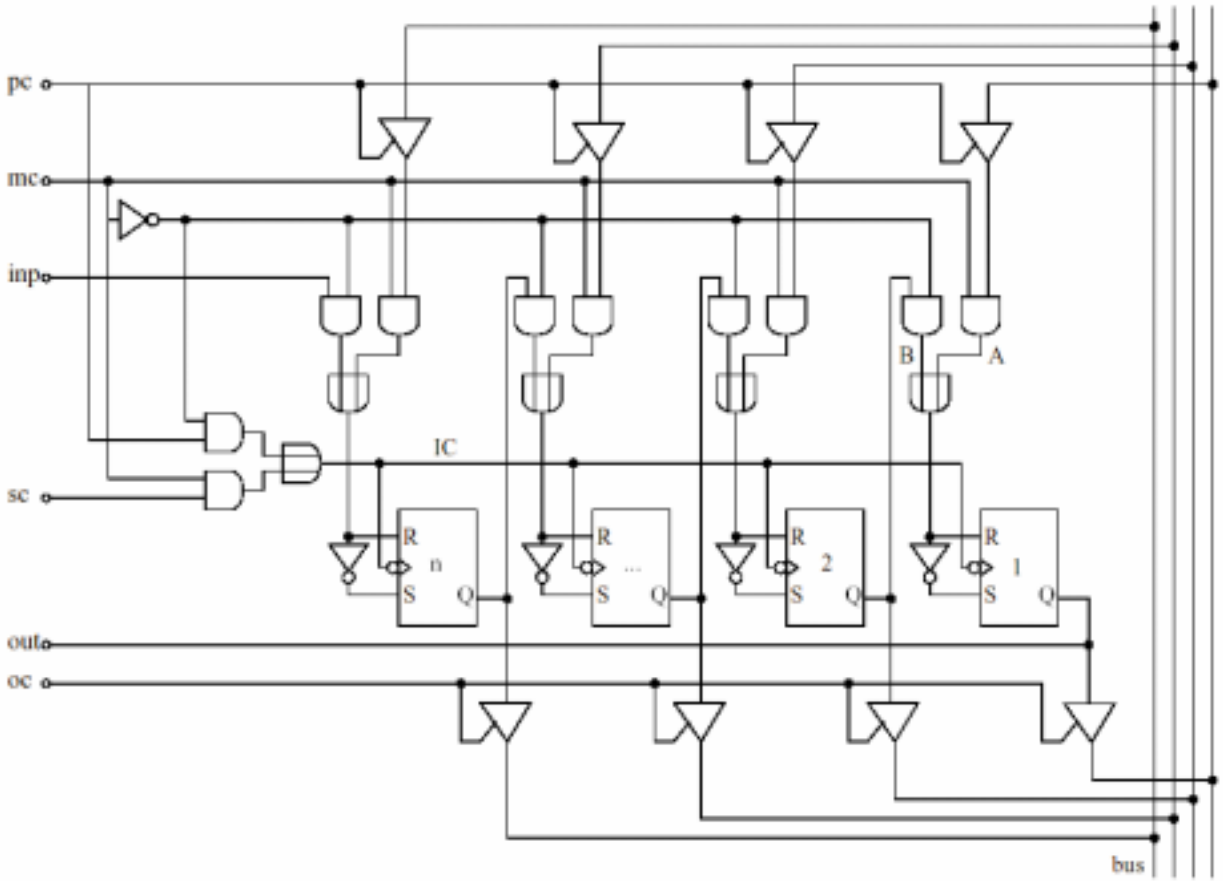
Coding in SMV (cont.)

```
for all i:
next(vpos[i]) := case
  (move=l) & !(vpos[0]=1) & hpos[i]=hpos[0] & vpos[i]=vpos[0]+1 |
  (move=r) & !(vpos[0]=v) & hpos[i]=hpos[0] & vpos[i]=vpos[0]-1 : vpos[0];
  1: vpos[i]; esac;
next(hpos[i]) := case
  (move=u) & !(hpos[0]=1) & vpos[i]=vpos[0] & hpos[i]=hpos[0]-1 |
  (move=d) & !(hpos[0]=h) & vpos[i]=vpos[0] & hpos[i]=hpos[0]+1 : hpos[0];
  1: hpos[i]; esac;
init(hpos[i]) := i div v; init(vpos[i]) := i mod v;
DEFINE goal := (hpos[i] = 3 - (i div v) & vpos[i] = 3 - (i mod v))
SPEC !EF goal
```

- SMV findet sofort eine Lösung (rrddlluurrddlluurrddlluurrdd)
- Lola (Humboldt Univ.) ist sogar noch schneller
- für 3*4 gut machbar, 4*4 machbar, 5*5 nicht machbar (10^{25})

Ein Hardware-Beispiel

A shift register for data bus interfacing (74x95 TTL family)



- mc: mode control
- pc: parallel clock
- sc: serial clock
- oc: output clock
- inp: serial input
- out: serial output

SR-Latches:

S	R	Q'
0	0	Q
1	0	1
0	1	0
1	1	-

Verifikationsmodell des Schieberegisters

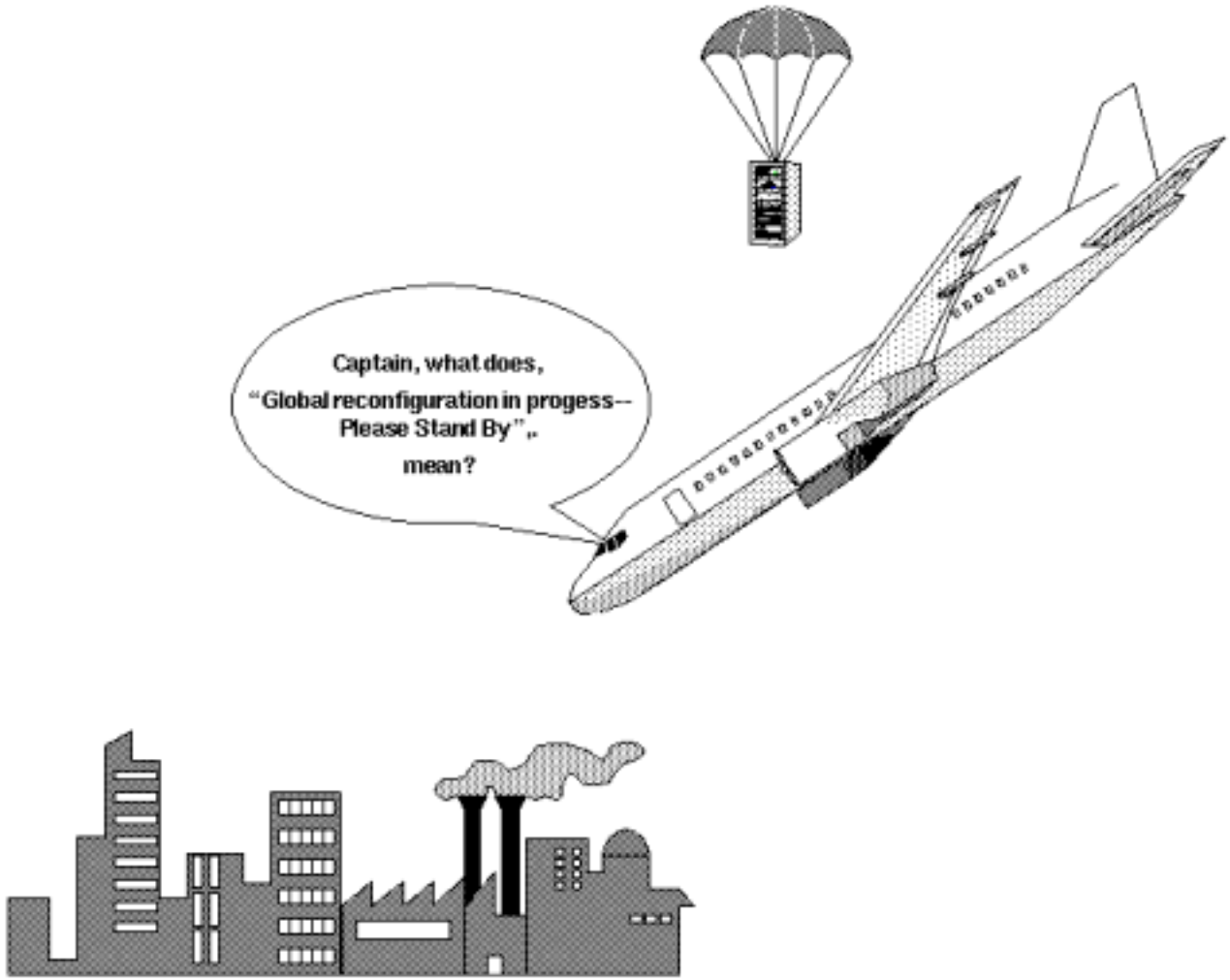
```
MODULE main
VAR Q, bus:  array 1..n of boolean;  -- n SR-latches, n databits
    inp, mc, pc, sc, oc:  boolean;   -- input lines
DEFINE out := Q[1]; ic := ((mc & pc)  |  (!mc & sc));
       A[i] := mc & pc & bus[i]; B[i] := !mc & Q[i + 1];
       R[i] := !(A[i] | B[i]); S[i] := !R[i];
ASSIGN next(Q[i]) := case ic:  case
                                !S[i] & !R[i]:  Q[i];          --hold
                                 S[i] & !R[i]:  1;             --set
                                !S[i] &  R[i]:  0;             --reset
                                 S[i] &  R[i]:  {0,1}; esac; --undef
                          !ic:  Q[i]; esac; -- unchanged if no input
       next(bus[i]) := case oc:  Q[i];  !oc:  {0, 1}; esac;
FAIRNESS ic FAIRNESS oc
```

Reale Beispiele

- Hardware-Verifikation: Stand der Technik
  - ALUs, PLAs, Memory Controller, Chip Design, ...
- Software-Verifikation: Stand der Forschung
  - Luftfahrtcomputer, Zugsteuerungen, Automobil-Steuergeräte, nichttriviale Suchprobleme, ...

http://www-2.cs.cmu.edu/~modelcheck/tour.htm

Pause!


Captain, what does,
"Global reconfiguration in progess--
Please Stand By",.
mean?

Propositional verification

- In order to validate a system, we need to represent the transition relation.
- The representation is of utmost importance for the success of a verification attempt
- Algorithms and data structures are heavily dependent onto each other
- Needed: representation of sets, relations, boolean formulas, ...

Binary Encoding of Domains

- Any variable on a finite domain D can be replaced by log(D) binary variables
  - similar to encoding of data types by compilers
  - e.g. var v: {0..15} can be replaced by
    var v1,v2,v3,v4: boolean
    (0=0000, 1= 0001, 2=0010, 3=0011, ..., 15=1111)
- State space
  - still in the order of original domain!
  - e.g. three int8-variables can have $2^{24}=10^8$ states
  - e.g. buffer of length 10 with 10-bit values ➔ 10^{30} states
- Representation of large sets of states?

Representation of Sets

Example: Consider the domain $D \triangleq \{0..15\}$

symbolic: $S \triangleq \{x \mid x \bmod 2 = 0 \vee x > 12\}$

enumeration: $S = \{0, 2, 4, 6, 8, 10, 12, 13, 14, 15\}$

bitstring: $S = (1010101010101111)$

binary: $S = \{0000, 0010, 0100, 0110, 1000, 1010, 1100, 1101, 1110, 1111\}$

propositional formula: $S = \{\vec{v} \mid v_4 = 0 \vee v_1 = 1 \wedge v_2 = 1\}$

logical spectrum: $S = \mathsf{Ite}(v_1, \mathsf{Ite}(v_2, \top, \mathsf{Ite}(v_4, \bot, \top)), \mathsf{Ite}(v_4, \bot, \top))$

Ordered Tree Form

- Normal form for propositional formulas
- Uses only the connective Ite

$$\mathsf{Ite}(\varphi, \psi_1, \psi_2) \triangleq ((\varphi \rightarrow \psi_1) \wedge (\neg\varphi \rightarrow \psi_2))$$

- Linear ordering on the set of propositions
  - e.g., most significant bit first
- Shannon expansion

$$\varphi \leftrightarrow \mathsf{Ite}(v, \varphi\{v := \top\}, \varphi\{v := \bot\})$$

Truth table and tree form formula

v_1	v_2	v_3	v_4	S
1	1	1	1	1
1	1	1	0	1
1	1	0	1	1
1	1	0	0	1
1	0	1	1	0
1	0	1	0	1
1	0	0	1	0
1	0	0	0	1
0	1	1	1	0
0	1	1	0	1
0	1	0	1	0
0	1	0	0	1
0	0	1	1	0
0	0	1	0	1
0	0	0	1	0
0	0	0	0	1

$$
\begin{aligned}
S = \mathsf{Ite}(v_1,\ & \\
& \mathsf{Ite}(v_2,\ \\
& \quad \mathsf{Ite}(v_3,\ \\
& \quad\quad \mathsf{Ite}(v_4, \top, \top),\ \\
& \quad\quad \mathsf{Ite}(v_4, \top, \top)),\ \\
& \quad \mathsf{Ite}(v_3,\ \\
& \quad\quad \mathsf{Ite}(v_4, \bot, \top),\ \\
& \quad\quad \mathsf{Ite}(v_4, \bot, \top))),\ \\
& \mathsf{Ite}(v_2,\ \\
& \quad \mathsf{Ite}(v_3,\ \\
& \quad\quad \mathsf{Ite}(v_4, \bot, \top),\ \\
& \quad\quad \mathsf{Ite}(v_4, \bot, \top)),\ \\
& \quad \mathsf{Ite}(v_3,\ \\
& \quad\quad \mathsf{Ite}(v_4, \bot, \top),\ \\
& \quad\quad \mathsf{Ite}(v_4, \bot, \top))))
\end{aligned}
$$

Reduction: Replace Ite (v,ψ,ψ) by ψ

Abbreviations

$$S = \mathsf{Ite}(v_1, \mathsf{Ite}(v_2, \top, \mathsf{Ite}(v_4, \bot, \top)), \mathsf{Ite}(v_4, \bot, \top))$$

- Introduce abbreviations

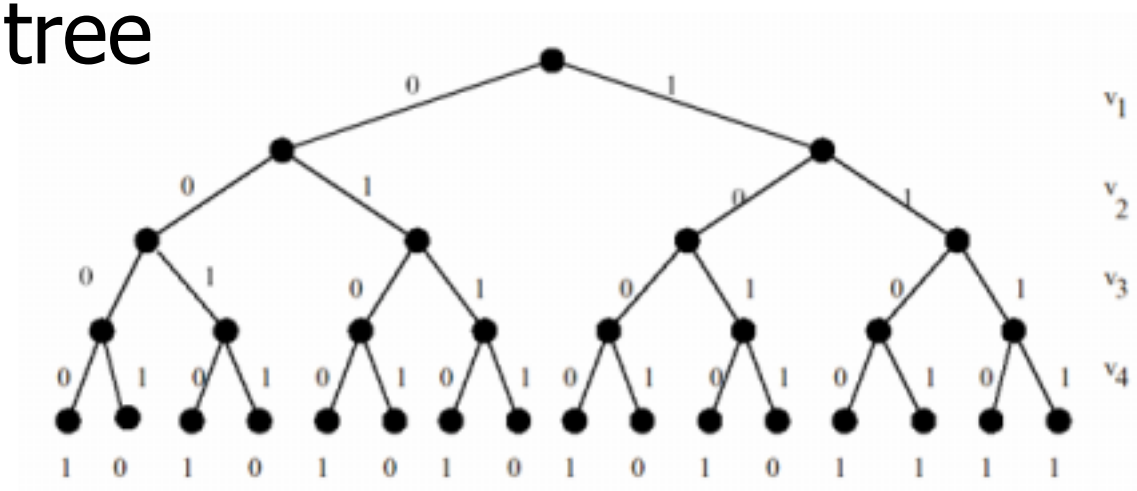
$$S = \mathsf{Ite}(v_1, \mathsf{Ite}(v_2, \top, \delta), \delta), \text{ where } \delta \triangleq \mathsf{Ite}(v_4, \bot, \top)$$

- maximally abbreviated

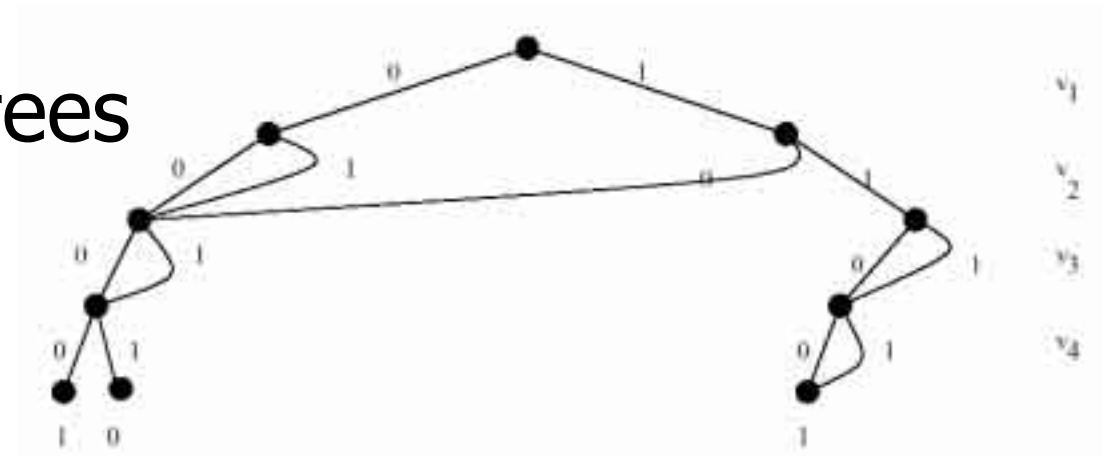
$$S = \mathsf{Ite}(v_1, \delta_1, \delta_2), \text{ where } \delta_1 \triangleq \mathsf{Ite}(v_2, \top, \delta_2) \text{ and } \delta_2 \triangleq \mathsf{Ite}(v_4, \bot, \top)$$

Binary Decision Trees (BDTs)

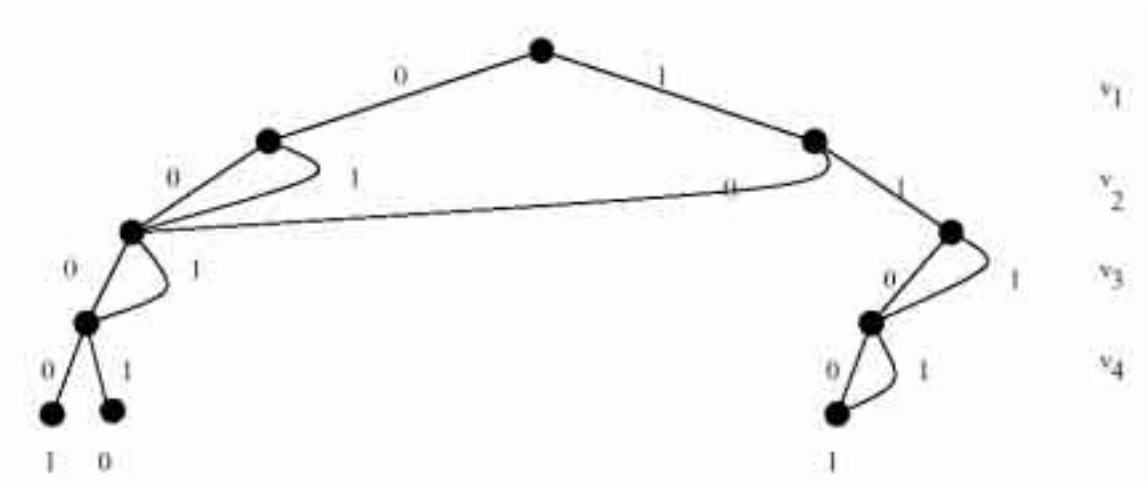
- Binary decision tree



- Elimination of isomorphic subtrees (abbreviations)



Binary Decision Diagrams (BDDs)



- Elimination of redundant nodes (redundant subformulas)
  Ite (v,ψ,ψ) by ψ