1.9 Das DPLL-Verfahren

Das Davis-Putnam-Logemann-Loveland Verfahren

Gegeben: Aussagenformel in KNF
(oder, alternativ, Menge N von Klauseln)

Ziel: Entscheide Erfüllbarkeit.

Erfüllbarkeit von Klauselmengen

$\mathcal{A} \models N$ gdw. $\mathcal{A} \models C$ für alle Klauseln C in N

$\mathcal{A} \models C$ gdw. $\mathcal{A} \models L$ für ein Literal L in N

Partielle Wertbelegungen

Partielle Abbildungen $\mathcal{A} : \Pi \rightarrow \{0, 1\}$

Wir beginnen mit der leeren Wertbelegung, und versuchen die Belegung zu allen Aussagenvariablen in Π zu erweitern.

Falls $\mathcal{A}$ partielle Wertbelegung, so sind Literalen und Klauseln in $\mathcal{A}$ wahr, falsch, oder nicht definiert.

Eine Klausel ist:

- wahr in $\mathcal{A}$ falls ein von seinen Literalen wahr ist;
- falsch ("widersprüchlich") in $\mathcal{A}$ falls alle seine Literale falsch sind;
- nicht definiert ("ungelöst") sonst.

Unit Klauseln

Bemerkung 1:

Sei $\mathcal{A}$ eine partielle Wertbelegung. Falls N eine Klausel C enthält so dass nur ein Literal L in C wahr oder nicht definiert ist, und alle anderen Literale falsch sind, so sind die folgenden Eigenschaften äquivalent:

(1) Es gibt eine Wertbelegung, die Modell für N ist, und $\mathcal{A}$ erweitert

(2) Es gibt eine Wertbelegung, die Modell für N ist, $\mathcal{A}$ erweitert und L wahr macht.

C heißt "unit Klausel"; L heißt "unit Literal".

Pure Literale

Bemerkung 2:

Sei $\mathcal{A}$ eine partielle Wertbelegung, und P eine Variable, die in $\mathcal{A}$ nicht definiert ist. Falls P nur positiv (bzw. negativ) in allen nicht gelösten Klauseln in N auftritt, so sind die folgenden Eigenschaften äquivalent:

(1) Es gibt eine Wertbelegung, die Modell für N ist, und $\mathcal{A}$ erweitert

(2) Es gibt eine Wertbelegung, die Modell für N ist, $\mathcal{A}$ erweitert und P wahr (bzw. falsch) macht.

P heißt "pures Literal".

DPLL-Verfahren

```
boolean DPLL (Klauselmenge N, partielle Wertbelegung 𝒜) {
  if (alle Klauseln in N wahr in 𝒜) return: wahr
  elsif (eine Klausel in N falsch in 𝒜) return: falsch
  elsif (N enthält Unitklausel P) return DPLL(N, 𝒜∪{P↦1})
  elsif (N enthält Unitklausel ¬P) return DPLL(N, 𝒜∪{P↦0})
  elsif (N enthält pures Literal P) return DPLL(N, 𝒜∪{P↦1})
  elsif (N enthält pures Literal ¬P) return DPLL(N, 𝒜∪{P↦0})
  else {
      sei P eine nicht definierte Variable in N:
      if (DPLL(N, 𝒜∪{P↦0}) return: wahr
      else return DPLL(N, 𝒜∪{P↦1});
   }
}
```

DPLL-Verfahren

Am Anfang: DPLL($N, \mathcal{A}$), wobei $\mathcal{A}$ die leere Bewertung.

DPLL-Verfahren

In Implementierungen, gibt es einige Änderungen der Prozedur:

- das “pure Literal”-Test wird oft nicht gemacht (zu teuer)
- die Variablen im letzten Schritt nicht random gewählt
- iterative Implementierung;
  “backtrack stack” explizit repräsentiert und verwendet

(manchmal nützlich mehrere Schritte auf einmal zurückzukehren)

Iteratives DPLL Verfahren

Eine iterative (und verallgemeinerte) Version

```
status = preprocess();
if (status != UNKNOWN) return status;
while(1) {
    decide-next-branch();
    while(1) {
        status = deduce();
        if (status == CONFLICT) {
            blevel = analyze-conflict();
            if (blevel == 0) return UNSATISFIABLE
            else backtrack(blevel); }
        else if (status == SATISFIABLE) return SATISFIABLE
        else break;
}   }
```

Iterative DPLL Verfahren

preprocess()

Vorverarbeitung des Inputs

(so weit wie möglich ohne Verzweigung)

return: CONFLICT oder UNSATISFIABLE oder UNKNOWN

decide-next-branch()

Wähle die nächste Variable für Verzweigung

Entscheide, ob diese den Wert 0 oder 1 bekommt

Steige das Backtrack Level

Iteratives DPLL Verfahren

deduce()

weitere Wertbelegungen für die Variablen machen
(z.B. mit Hilfe der Unitklausel Regel)
bis eine erfullende Wertbelegung gefunden ist, oder
bis Verzweigung notwendig
return: CONFLICT oder UNSATISFIABLE oder UNKNOWN

Iterative DPLL Verfahren

analyze-conflict()

teste wenn zu backtracken

backtrack(blevel)

backtrack bis zu blevel

- wechsele die Verzweigungsvariable bis zu diesem Niveau

- vergesse alle Variablenbelegungen die dazwischen gemacht worden sind.

Verzweigungsheuristiken

Die Wahl der richtigen nichtdefinierten Variable für Verzweigung ist wichtig für Effizienz, aber die Verzweigungsheuristiken können teuer (zeitaufwendig) sein.

State of the art: benutze Verzweigungsheuristiken die nicht zu oft gewechselt werden müssen.

Im Allgemeinen: wähle Variablen, die oft vorkommen.

Der deduce Algorithmus

Um die Unitklausel-Regel anzuwenden, müssen wir die Anzahl der Literale in einer Klausel, die nicht falsch sind, wissen.

Aufrechterhaltung dieser Zahl ist zeitaufwendig.

Der deduce Algorithmus

Bessere Methode: Zwei beobachtete Literale

In jede Klausel selektiere zwei (nicht definierte) “beobachtete” Literale. Für jede Variable P, erhalte:

- eine Liste mit alle Klauseln, in die P beobachtet wird, und
- eine Liste aller Klauseln, in die $\neg P$ beobachtet wird.

Falls eine nicht definierte Variable den Wert 0 (oder 1) bekommt:

- verifiziere alle Klauseln in die P (oder $\neg P$) beobachtet ist, und
- beobachte in diese Klauseln ein anderes (wahres oder nicht definiertes) Literal, wenn möglich.

Information zu beobachteten Literalen muss nicht wiederhergestellt werden nach backtracking

Konfliktanalyse und Lernen

Ziel: Wiederverwendung der Information, die in einem Zweig erhalten wurde

Methode: Lernen

> Wenn eine Konfliktklausel gefunden ist, verwende die Resolutionsregel, um eine neue Klausel zu erhalten, und füge diese Klausel der aktuellen Klauselmenge hinzu.

Problem: Sehr viele Klausel können so erzeugt werden, und es kann notwendig sein, einige Klauseln zu löschen, um Speicher zu sparen.

Backtracking

Verwandtes Verfahren:

nicht chronologisches Backtracking (backjumping)

> Falls ein Konflikt unabhängig ist von einer früheren Verzweigung, versuche direkt bis vor diesem Punkt zu backtracken

Restart

Die Laufzeit von DPLL-Stil Prozeduren ist von der Wahl von Verzweigungsvariablen sehr abhängig.

Falls keine Lösung in einer gewissen Zeitspanne gefunden worden ist, kann es hilfreich sein, die Prozedur wieder zu starten, mit einer anderen Wahl von Verzweigungsvariablen.

Gelernte Klauseln können erhalten werden.

Weitere Informationen

Die vorher beschriebenen Ideen sind in dem SAT-Checker Chaff implementiert worden.

Weitere Informationen:

Lintao Zhang und Sharad Malik:

The Quest for Efficient Boolean Satisfiability Solvers

Proc. CADE-18, LNAI 2392, pp.295-312, Springer, 2002.

Anwendungsgebiete

- Verifikation boole'scher Schaltkreisen
- Model checking

Folien: "Angewandte Logik" (Modellüberprüfung)

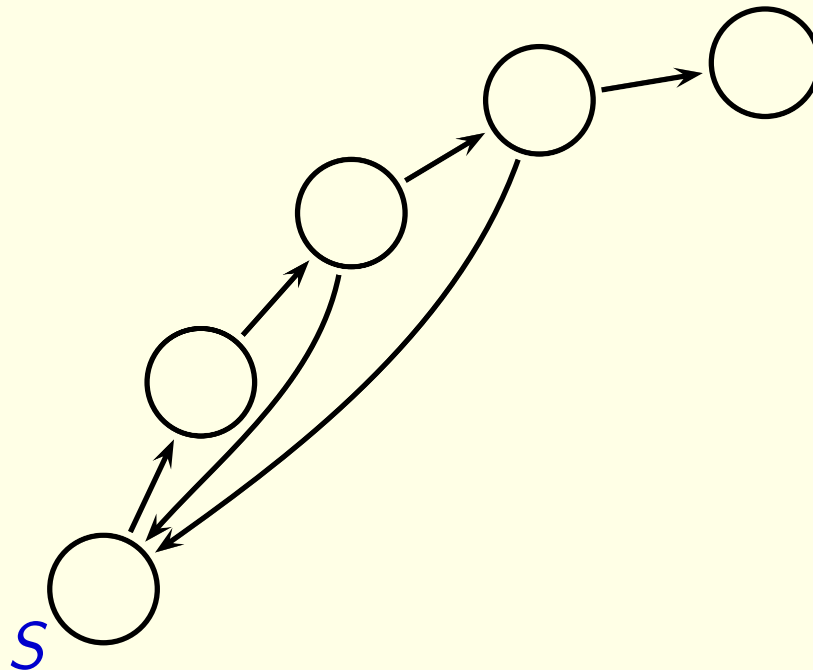
Viorica Sofronie Stokkermans, Uwe Waldmann

`http://www.mpi-sb.mpg.de/~sofronie/teaching/slides.ps.gz`

Beispiel: Handy

Wir stellen die Zustände durch Kreise dar.

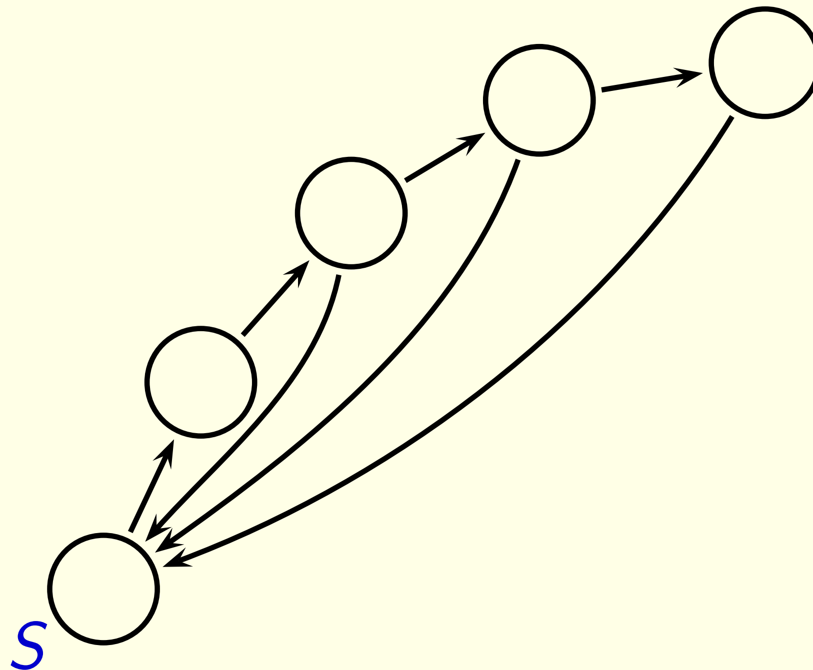
Wenn es möglich ist, von einem Zustand in einen anderen zu wechseln, zeichnen wir einen Pfeil dazwischen:



Beispiel: Handy

Wir stellen die Zustände durch Kreise dar.

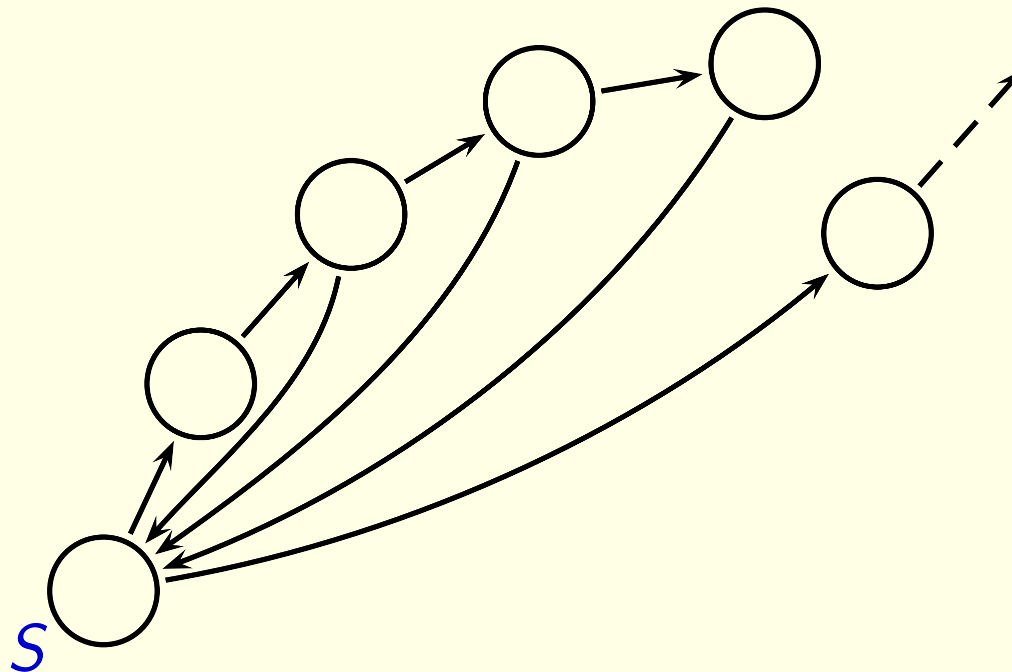
Wenn es möglich ist, von einem Zustand in einen anderen zu wechseln, zeichnen wir einen Pfeil dazwischen:



Beispiel: Handy

Wir stellen die Zustände durch Kreise dar.

Wenn es möglich ist, von einem Zustand in einen anderen zu wechseln, zeichnen wir einen Pfeil dazwischen:



Beispiel: Handy

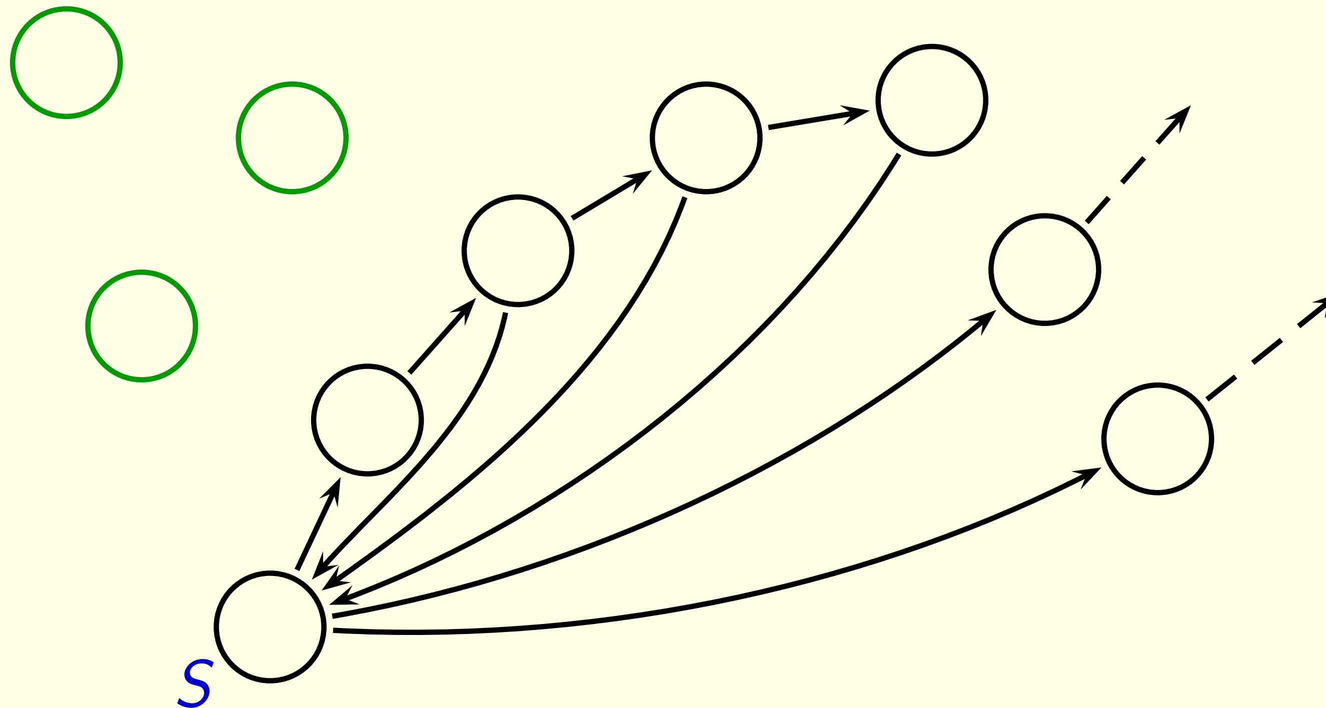
Wir stellen die Zustände durch Kreise dar.

Wenn es möglich ist, von einem Zustand in einen anderen zu wechseln, zeichnen wir einen Pfeil dazwischen:

S

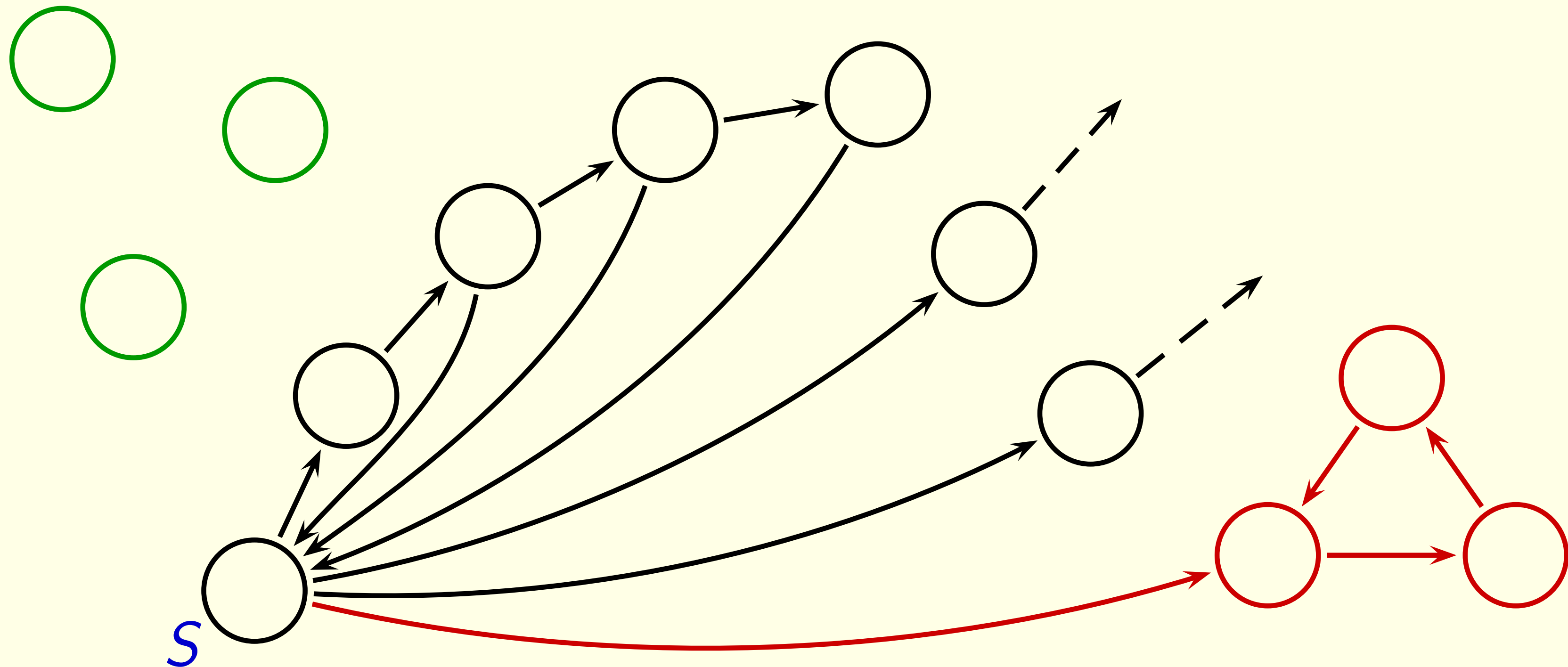
Beispiel: Handy

Es kann auch „unmögliche“ Zustände geben.
Aber diese dürfen dann natürlich nicht vom Startzustand aus erreichbar sein.



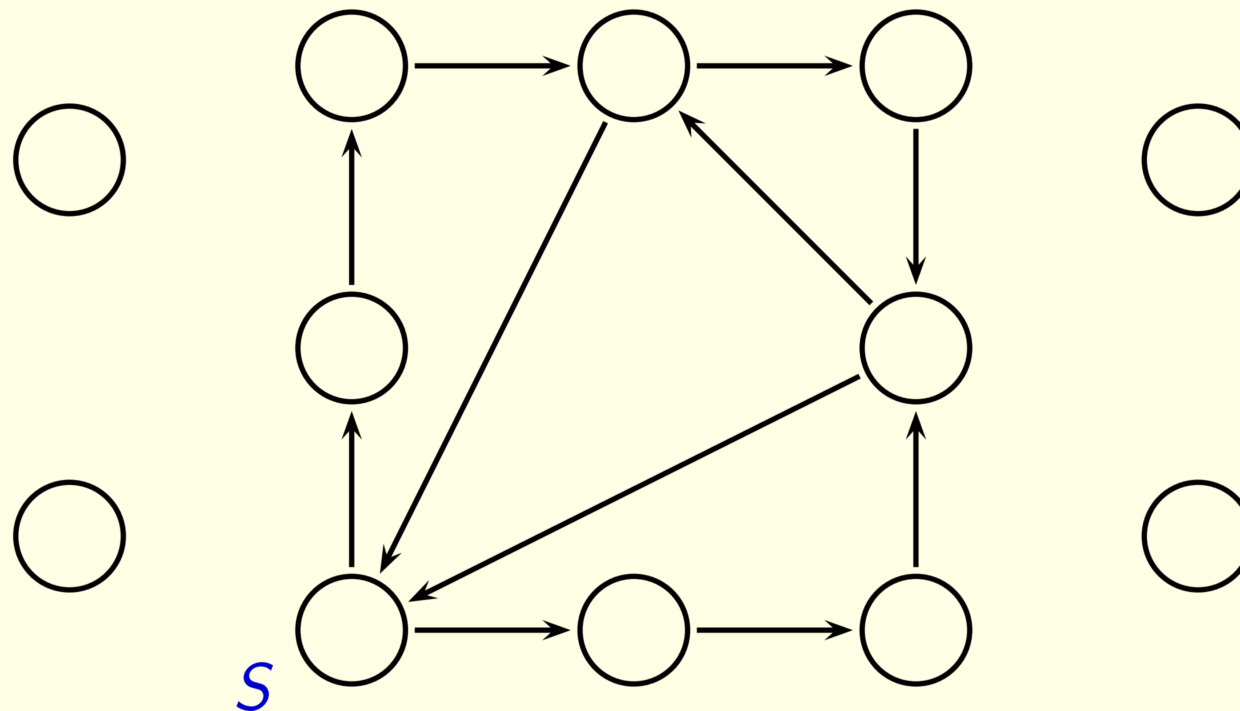
Beispiel: Handy

Was es aber auf keinen Fall geben darf, das sind Sackgassen.



Beispiel: Handy

Frage: Wie kann man automatisch feststellen, ob irgendein Zustandsdiagramm eine Sackgasse besitzt?

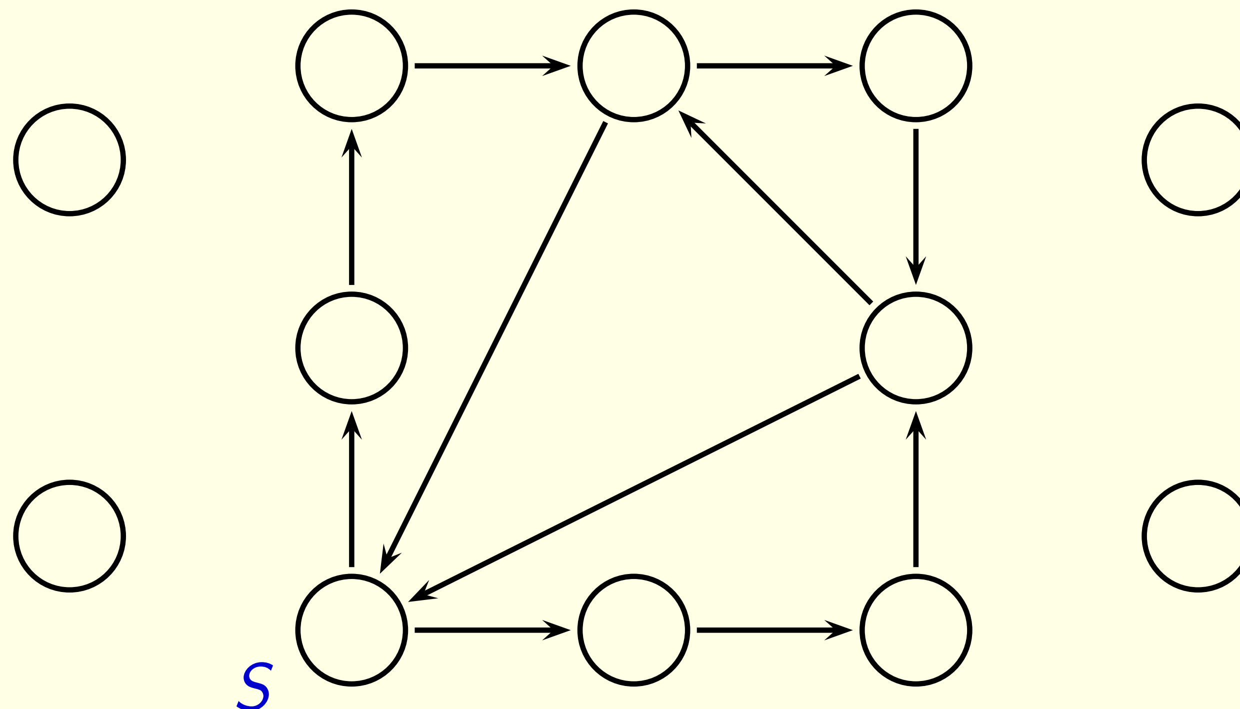


Beispiel: Handy

Als Formel ausgedrückt: Hat S die Eigenschaft $\mathsf{EF}\,\neg\ \mathsf{EF}\,S$?

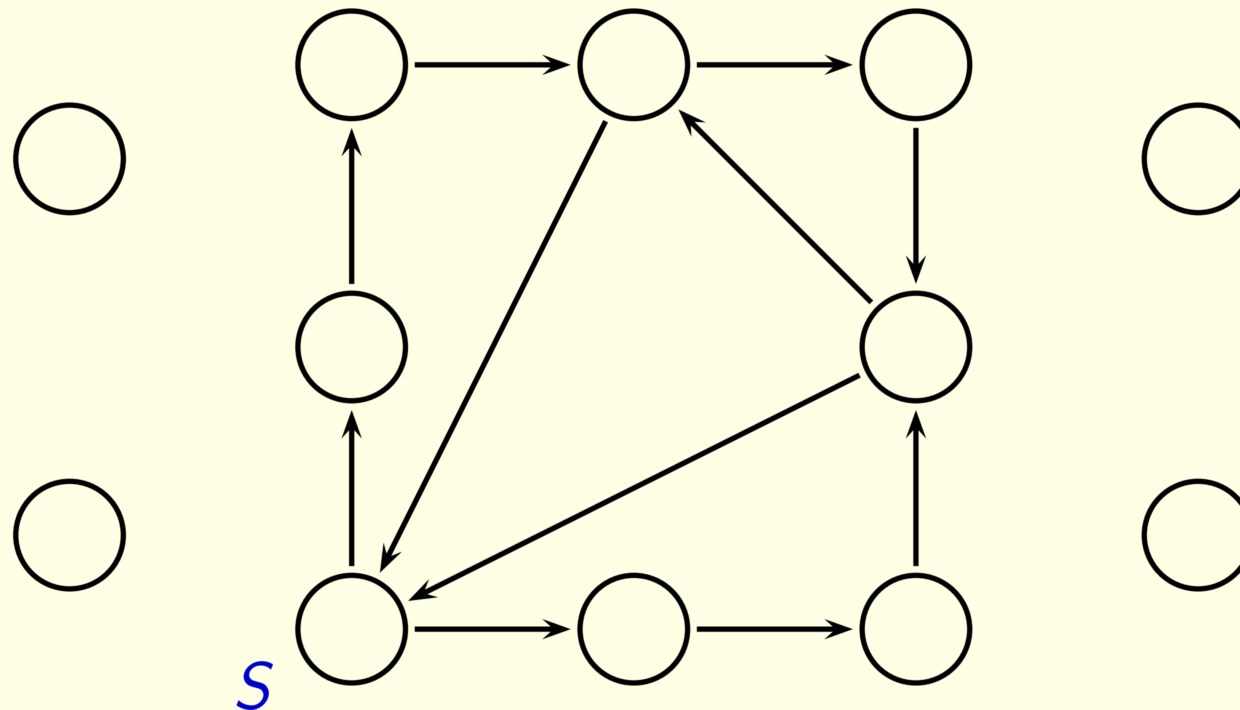
Dabei bedeutet EF:

„man kann einen Zustand erreichen mit der Eigenschaft . . . “



Beispiel: Handy

Zuerst ermitteln wir alle Zustände mit der Eigenschaft EF S,
das heißt, alle Zustände, von denen aus man S erreichen kann.



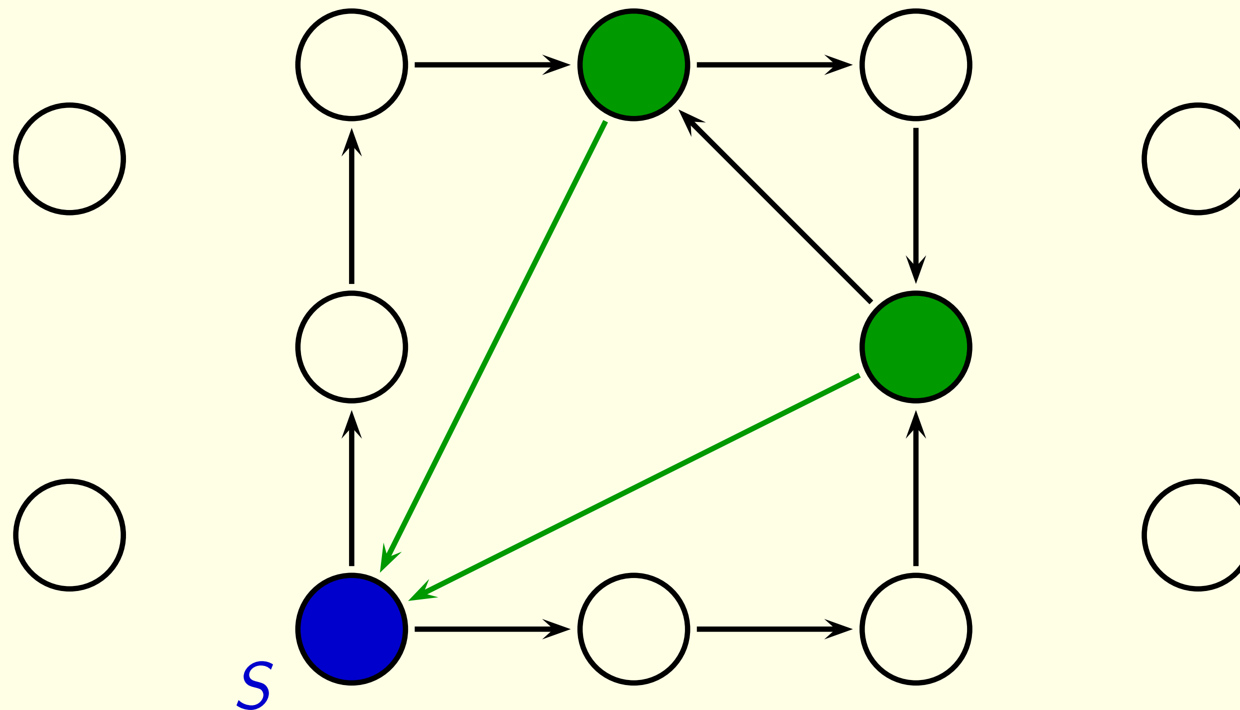
Beispiel: Handy

Zuerst ermitteln wir alle Zustände mit der Eigenschaft EF S,
das heißt, alle Zustände, von denen aus man S erreichen kann.

S

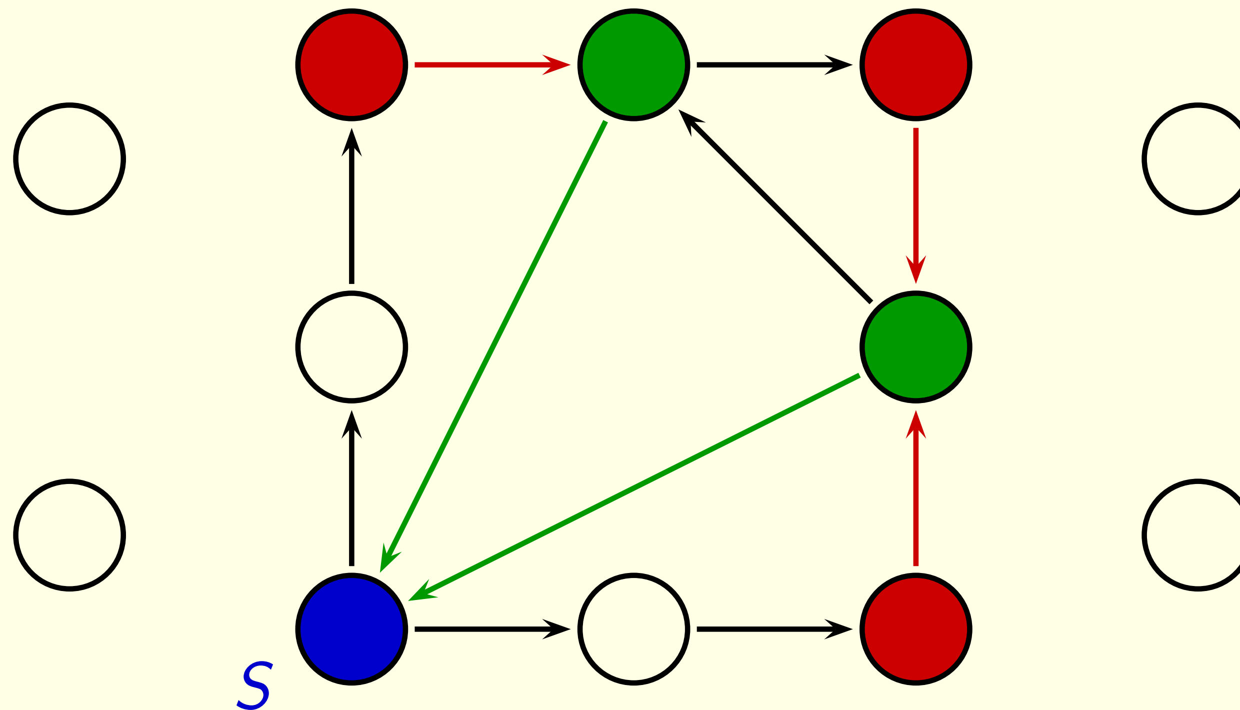
Beispiel: Handy

Zuerst ermitteln wir alle Zustände mit der Eigenschaft EF S,
das heißt, alle Zustände, von denen aus man S erreichen kann.



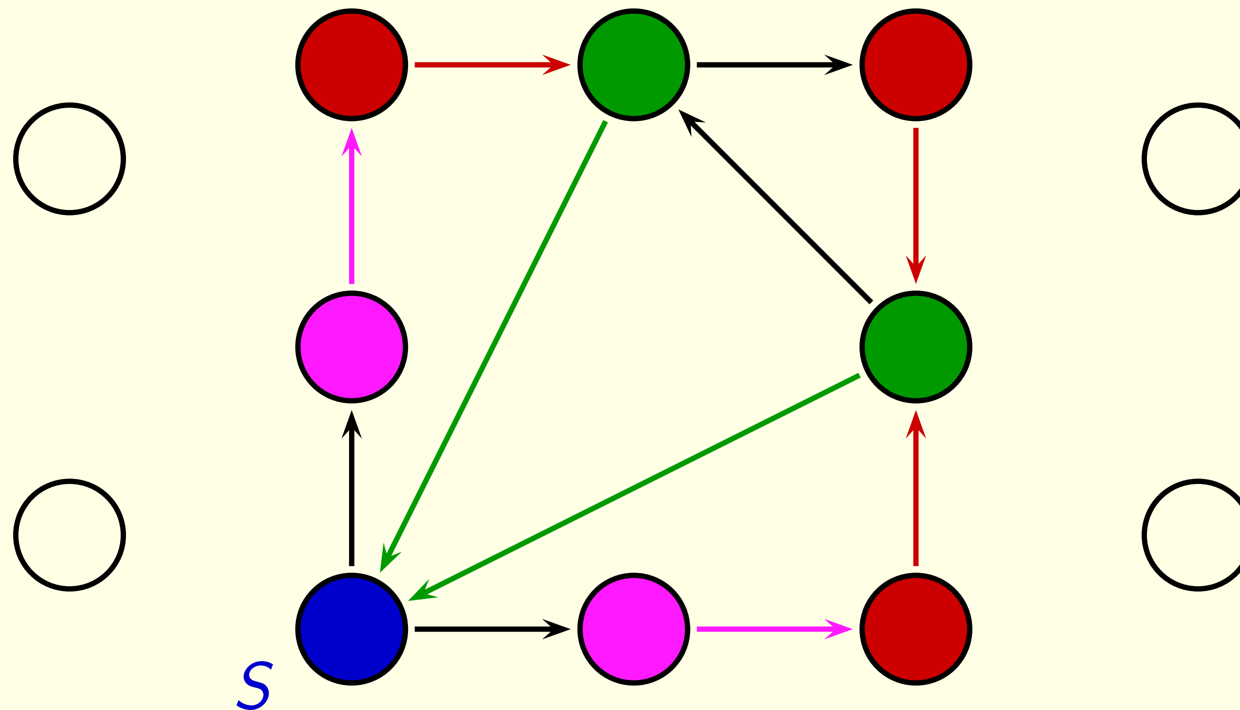
Beispiel: Handy

Zuerst ermitteln wir alle Zustände mit der Eigenschaft EF S,
das heißt, alle Zustände, von denen aus man S erreichen kann.



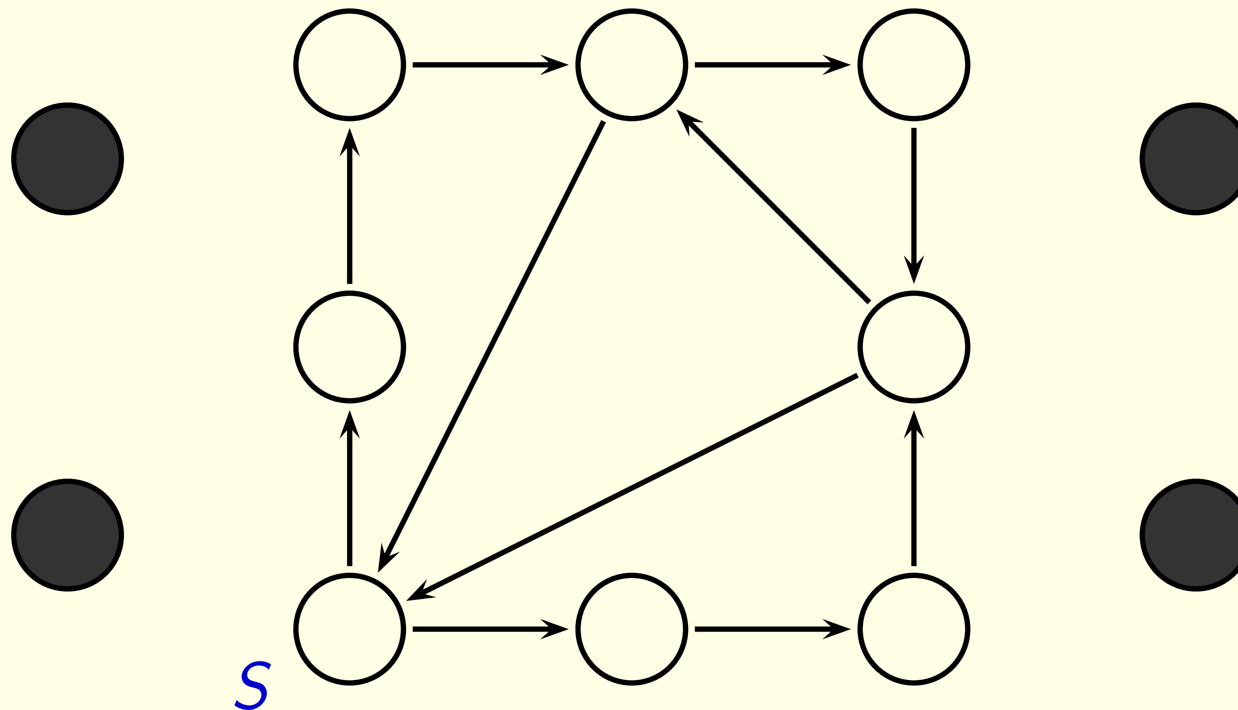
Beispiel: Handy

Zuerst ermitteln wir alle Zustände mit der Eigenschaft EF S,
das heißt, alle Zustände, von denen aus man S erreichen kann.



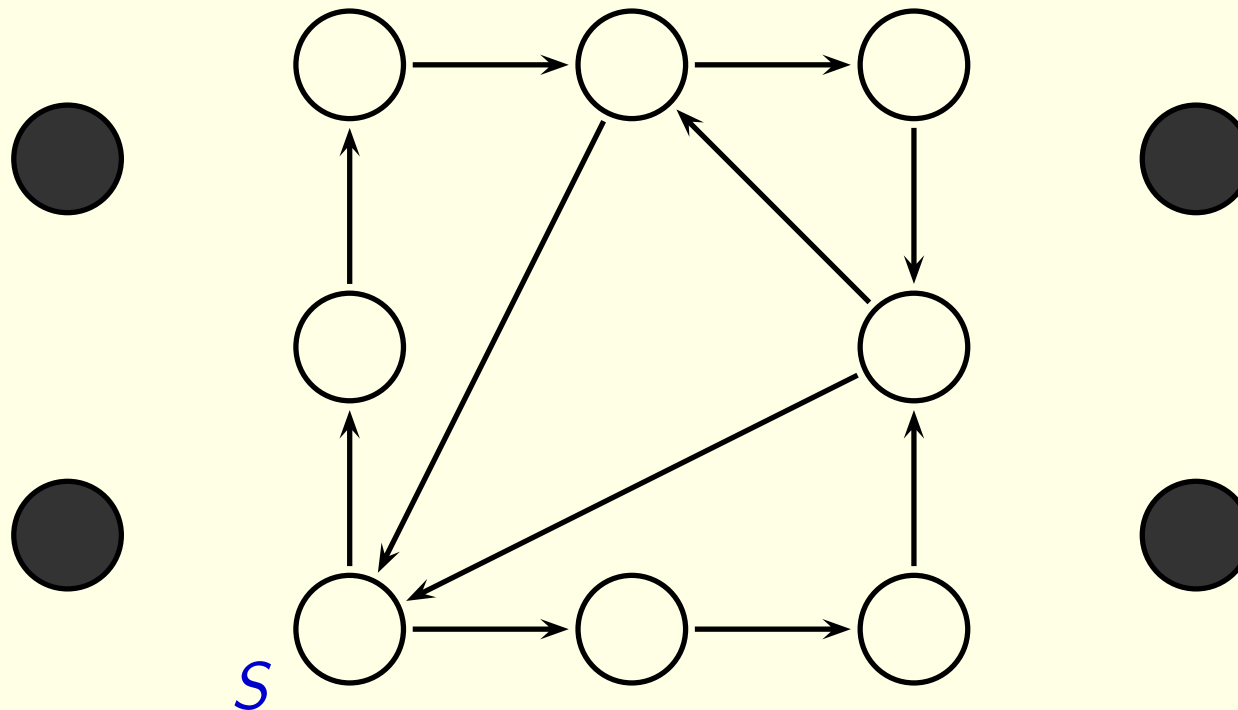
Beispiel: Handy

Dann ermitteln wir alle Zustände mit der Eigenschaft $\neg$ EF S,
das heißt, alle Zustände, die **nicht** die Eigenschaft EF S haben.



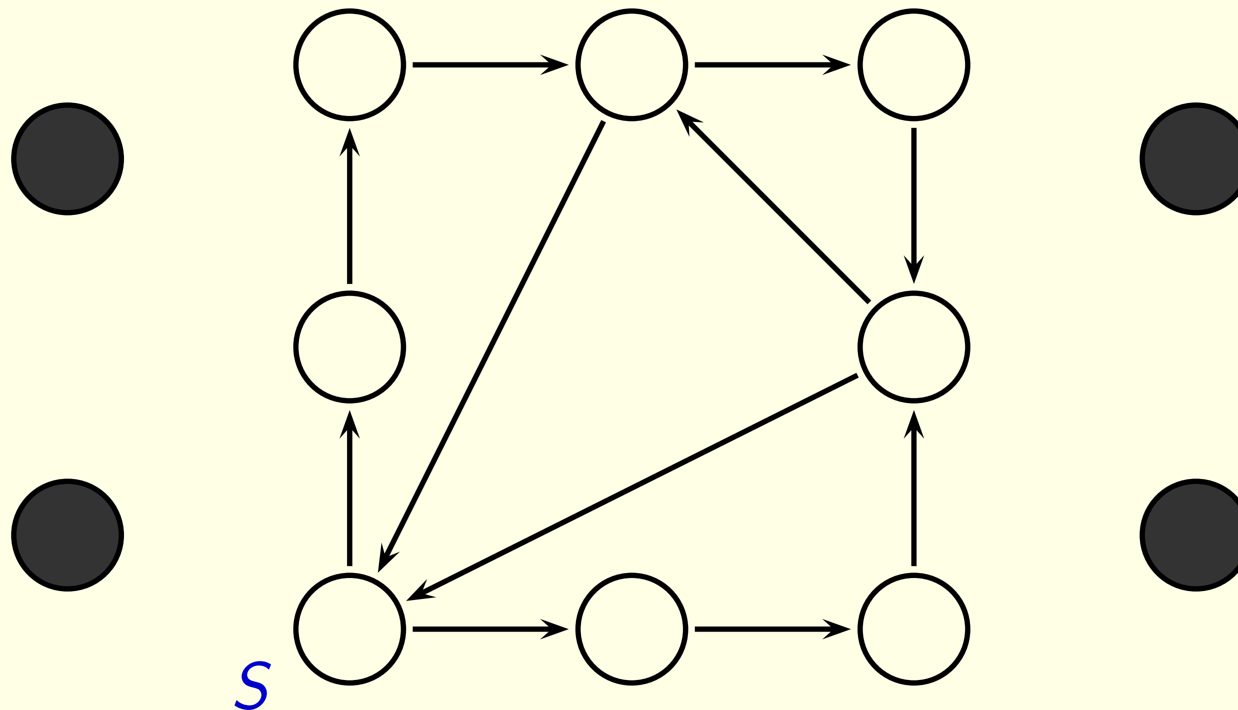
Beispiel: Handy

Zum Schluß ermitteln wir alle Zustände mit der Eigenschaft $\mathsf{EF}\,\neg\,\mathsf{EF}\,S$, also diejenigen, von denen aus man einen grauen Zustand erreicht. (Das sind aber gerade die grauen Zustände selbst.)



Beispiel: Handy

Da der Zustand S **nicht** zu diesen Zuständen gehört, wissen wir nun, daß wir von S aus keine Sackgasse erreichen.



Modellüberprüfung

Gegeben: - Endliches Modell

(Zustände, Markierung, Anfangszustände, Transitionen)

- Formel in eine gewissen Logik (CTL)

Ziel: Beweise, dass die Formel wahr in dem Modell ist

Modellüberprüfung

1. Explizite Zustandsraum Darstellung

Annahme: Eindeutige Markierung mit Aussagenvariablen

$$\text{Label}(s) = \text{Label}(s') \Rightarrow s = s'$$

n Zustände

Markierung mit Aussagenvar: boole'sche Matrix $(n \times |\Pi|)$

Anfangszustände: boole'sches Vektor (n)

Transitionen: boole'sche Matrix $(n \times n)$

Zustand: Vektor von Länge $|\Pi|$: $[s] = (b_1, \ldots, b_k)$

Modellüberprüfung

Die Hauptschwierigkeit ist die Größe der Probleme:

10 bits $\hat{=}$ $1\,024$ Zustände

20 bits $\hat{=}$ $1\,048\,576$ Zustände

30 bits $\hat{=}$ $1\,073\,741\,824$ Zustände

40 bits $\hat{=}$ $1\,099\,511\,627\,776$ Zustände

Ausweg:

Zustände und Übergange nicht explizit darstellen, sondern Zustandsmengen durch (hoffentlich kleine) Formeln kodieren.

Modellüberprüfung

2. Symbolische Zustandsraum Darstellung

Annahme: Eindeutige Markierung mit Aussagenvariablen

$\text{Label}(s) = \text{Label}(s') \Rightarrow s = s'$

Zustand: Bitvektor von Länge $|\Pi|$: $[s] = (b_1, \ldots, b_k)$

$F_s = P_1^{b_1} \wedge \cdots \wedge P_n^{b_n}$ wobei $P^0 = \neg P, P^1 = P$

Anfangszustände: $I = \{s_1, \ldots, s_k\}$

$$F_I = F_{s_1} \vee \cdots \vee F_{s_k}$$

Transitionsrelation: $R = \{(s_1, s'_1), (s_2, s'_2), \ldots\}$

$$F_R = (F_{s_1} \wedge F_{s'_1}) \vee (F_{s_2} \wedge F_{s'_2}) \vee \ldots$$

Modellüberprüfung

Bemerkung: Modellüberprüfungsmethoden operieren mit Mengen von Zuständen

$\mathsf{Sat}(\top) = Z$ die Menge aller Zustände

$\mathsf{Sat}(\bot) = \emptyset$

$\mathsf{Sat}(p) = \{s \in Z \mid p \in \mathsf{Label}(s)\}$

$\mathsf{Sat}(\neg F) = Z \backslash \mathsf{Sat}(F)$

$\mathsf{Sat}(F_1 \vee F_2) = \mathsf{Sat}(F_1) \cup \mathsf{Sat}(F_2)$

$\mathsf{Sat}(F_1 \wedge F_2) = \mathsf{Sat}(F_1) \cap \mathsf{Sat}(F_2)$

$\mathsf{Sat}(EX(F)) = \{s \in Z \mid \exists s'(R(s, s') \text{ and } s' \in \mathsf{Sat}(F))\}$

...

Modellüberprüfung

Methoden operieren mit Mengen von Zuständen

$\mapsto$ Darstellung mittels OBDDs

Operationen für OBDDs

“Symbolic Model Checking”

$\mapsto$ Reduktion zu einem Gültigkeits- oder Erfüllbarkeitstest

OBDDs, DPLL

“Deductive Model Checking”

Modellüberprüfung

Weitere Anwendungsbereiche:

- Software-Verifikation
- Hybride Systeme (d. h., Systeme, in denen Zustände auch durch reelle Zahlen beschrieben werden)

In beiden Fällen ist die Anzahl der Zustände unendlich.

Ausweg:

Zustände zu endlich vielen Intervallen oder Klassen zusammenfassen (funktioniert aber leider nicht immer).

Modellüberprüfung

Weitere Anwendungsbereiche:

- Software-Verifikation
- Hybride Systeme (d. h., Systeme, in denen Zustände auch durch reelle Zahlen beschrieben werden)

In beiden Fällen ist die Anzahl der Zustände unendlich.

Andere Auswege:

- Berechnungen “modulo” eine Theorie (e.g. **DPLL(T)**)
- Prädikantenlogik erster Stufe