Formale Grundlagen der Informatik 3

Bounded Model Checking, CBMC

Prof. Stefan Katzenbeisser
Security Engineering Group
Technische Universität Darmstadt

skatzenbeisser@acm.org
http://www.seceng.informatik.tu-darmstadt.de

Wiederholung: Model Checking

E. Clarke, A. Emerson, J. Sifakis

„P und Q sind nie gleichzeitig in einem kritischen Abschnitt"

Eigenschaft / Spezifikation

```
byte n = 0;
active proctype P() {
  n = 1;
}
active proctype Q() {
  n = 2;
}
```

Programm / „Modell"

Model Checker

✔

Gegenbeispiel

Bounded Model Checking
Ziele

TECHNISCHE UNIVERSITÄT DARMSTADT

Generelle Idee:

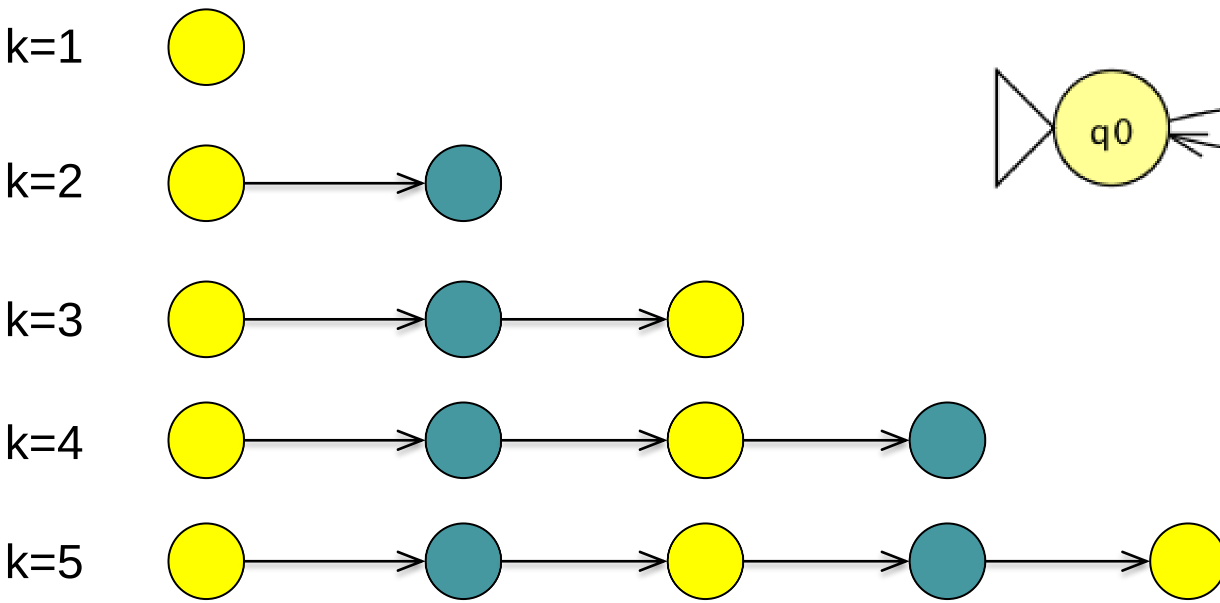
- Suche nach Gegenbeispielen mit einer maximalen Länge („bounded“)
- Generierung einer **aussagenlogischen** Formel, die genau dann erfüllt ist wenn ein Gegenbeispiel existiert
- Suchen nach einer Lösung für diese Formel mittels „SAT-solver“

Vorteile:

- Suche nach Gegenbeispiel ist relativ schnell
- Das gefundene Gegenbeispiel hat die kürzeste Länge
- Ideal zur Fehlersuche in größeren Programmen

Bounded Model Checking
Beschränkte Pfade

- Betrachte nur endliche „Anfangsstücke“ der Pfade
- Maximale Länge der Pfade wird iterativ erhöht
- Auch ein endliches „Anfangsstück“ kann einen unendlichen Pfad repräsentieren wenn es eine Schleife zu einem früheren Zustand enthält



Bounded Model Checking
Prinzipielle Idee

$\mathcal{M} = (M, R, q, E, \Sigma)$

$\downarrow$

Übersetzen aller Pfade der Länge *k* in aussagenlogische Formel

$[\![\mathcal{M}]\!]_k$

φ

$\downarrow$

Übersetzen in aussagenlogische Formel

$[\![\varphi]\!]_k$

$\downarrow$

Test ob erhaltene Formel erfüllbar ist (SAT Solver)

$[\![\mathcal{M}, \varphi]\!]_k$

$\downarrow$

Gegenbeispiel oder
Nachweis der Erfüllbarkeit

Umwandlung einer Kripke-Struktur (1)
Charakteristische Funktion einer Menge

TECHNISCHE UNIVERSITÄT DARMSTADT

Die charakteristische Funktion einer Menge $T \subseteq X$ ist die Funktion

$C_T(t) : X \to \{0, 1\}$ mit

$$C_T(t) = \begin{cases} 1 & \text{falls } t \in T \\ 0 & \text{sonst} \end{cases}$$

Beispiel: Menge $T = \{1, 3, 5\} \subseteq \mathbb{N}$

Funktion: $C_T(1) = C_T(3) = C_T(5) = 1$

$$C_T(0) = C_T(2) = C_T(4) = C_T(6) = \ldots = 0$$

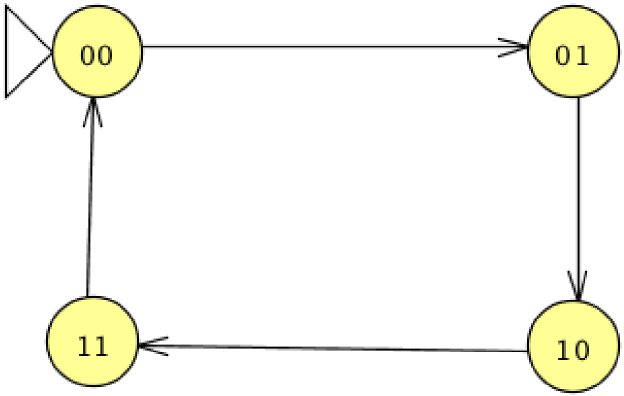
- Wir betrachten nun nur mehr den Fall dass *T* binär codiert ist
- Die charakteristische Funktion kann dann einfach als **aussagenlogische Formel** geschrieben werden!

Umwandlung einer Kripke-Struktur (2)
Charakteristische Funktion einer Kripke-Struktur

- Ausgangspunkt: Kripke-Struktur $\mathcal{M} = (M, R, q, E, \Sigma)$
- Wir nehmen an dass die Elemente in *M* binär codiert sind: jeder Zustand besteht aus *k* binären Variablen
- Ein Zustand wird damit durch $m_0 m_1 \ldots m_k$ repräsentiert

Beispiel:

- Zustände 00,01,10,11
- Übergänge 00->01, 01->10, 10->11, 11->00

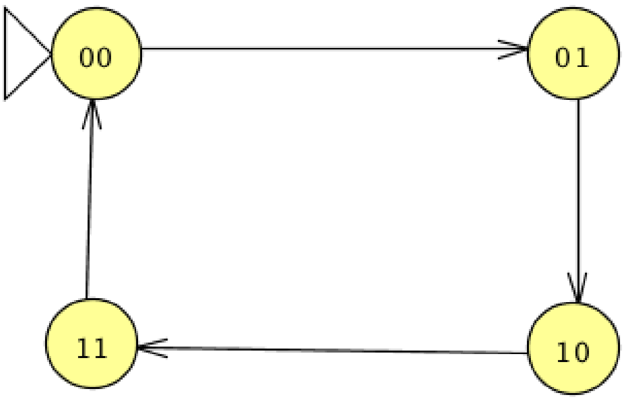


Umwandlung einer Kripke-Struktur (3)
Charakteristische Funktion einer Kripke-Struktur

TECHNISCHE UNIVERSITÄT DARMSTADT

Beispiel:

- Zustände 00,01,10,11
- Übergänge 00->01, 01->10, 10->11, 11->00



Charakteristische Funktion der Übergangsrelation:

$$C_R(m,n) = C_R(m_0, m_1, n_0, n_1)$$

Für die Kripke-Struktur oben gilt daher:

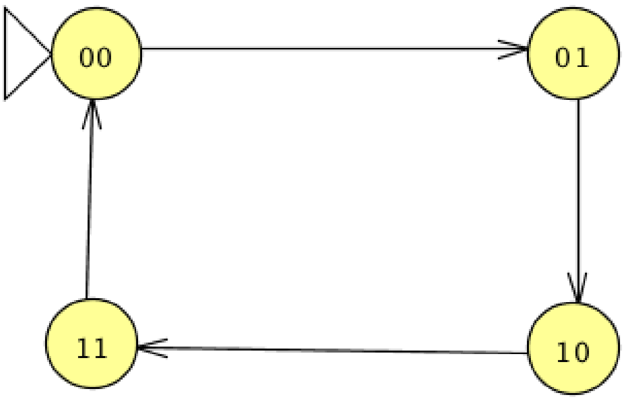
$$C_R(0,0,0,1) = C_R(0,1,1,0) = C_R(1,0,1,1) = C_R(1,1,0,0) = 1$$

$$C_R(0,0,0,0) = C_R(0,0,1,0) = C_R(0,0,1,1) = \ldots = 0$$

Umwandlung einer Kripke-Struktur (4)
Charakteristische Funktion einer Kripke-Struktur

Charakteristische Funktion der Übergangsrelation:

m0	m1	n0	n1	
0	0	0	0	0
0	0	0	1	1
0	0	1	0	0
…				
0	1	0	0	0
0	1	0	1	0
0	1	1	0	1
…				

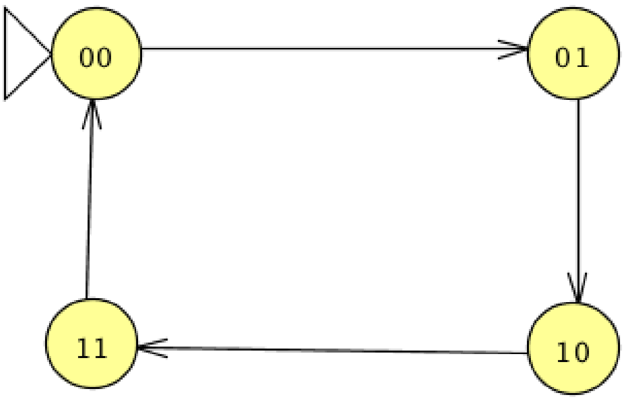


➔ ist eine Boolesche Funktion,
kann daher als aussagenlogische Formel geschrieben werden!

Umwandlung einer Kripke-Struktur (5)
Charakteristische Funktion einer Kripke-Struktur

Charakteristische Funktion der Übergangsrelation:

$$C_R(m_0, m_1, n_0, n_1) =$$
$$((n_1 \leftrightarrow \neg m_1) \land (n_0 \leftrightarrow (m_0 \otimes m_1)))$$



Charakteristische Funktion des Startzustandes:

$$C_q(m_0, m_1) = \neg m_0 \land \neg m_1$$

Umwandlung einer Kripke-Struktur (6)
Spezifikation eines Pfades

TECHNISCHE UNIVERSITÄT DARMSTADT

Gegeben: Kripke-Struktur $\mathcal{M} = (M, R, q, E, \Sigma)$

Charakteristische Funktion der Übergangsrelation C_R

Charakteristische Funktion des Startzustandes C_q

Ein gültiger Pfad $\pi_k = m_0, m_1, m_2, \ldots, m_k$ der Länge k durch die Kripke-Struktur kann durch die folgende aussagenlogische Formel spezifiziert werden:

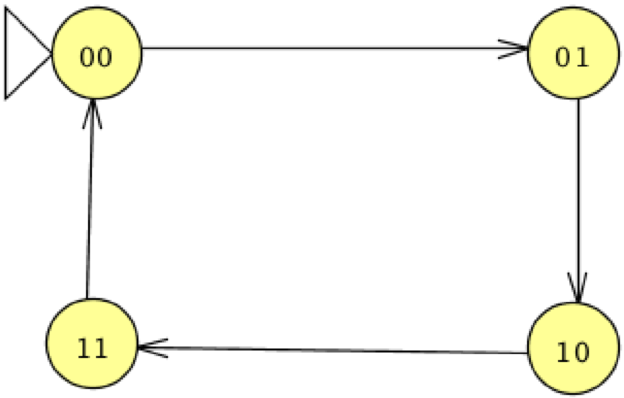
$$C_q(m_0) \wedge \bigwedge_{i=0}^{k-1} C_R(m_i, m_{i+1})$$

SecEng Security Engineering

Umwandlung einer Formel (1)
Safety-Eigenschaften

Spezifikation von Eigenschaften $\mathbf{AG}p = \mathbf{EF}\neg p$

- Betrachte wieder Beispiel des 2-Bit Zählers von vorne

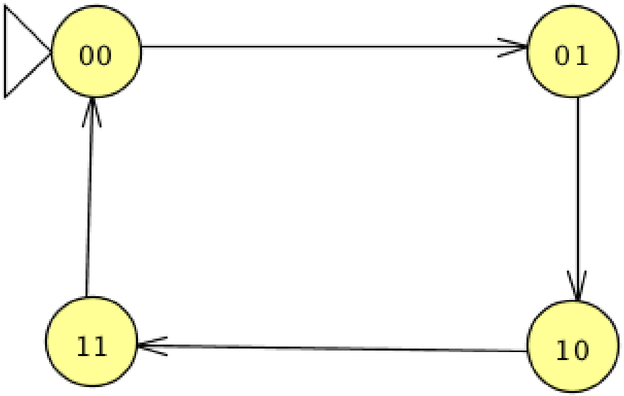


- Nutze Prädikat für Test ob p in einem Zustand gilt

- Prädikat darf in keinem der Zustände erfüllt sein, z.B. für Pfad mit 3 Zuständen:

$$\neg p(m_0) \vee \neg p(m_1) \vee \neg p(m_2)$$

Umwandlung einer Formel (2)
Safety-Eigenschaften, Beispiel



- Länge des Pfades *k=3*
  Spezifikation jedes Zustandes durch zwei Variablen: $(m_0, m_1, n_0, n_1, r_0, r_1)$

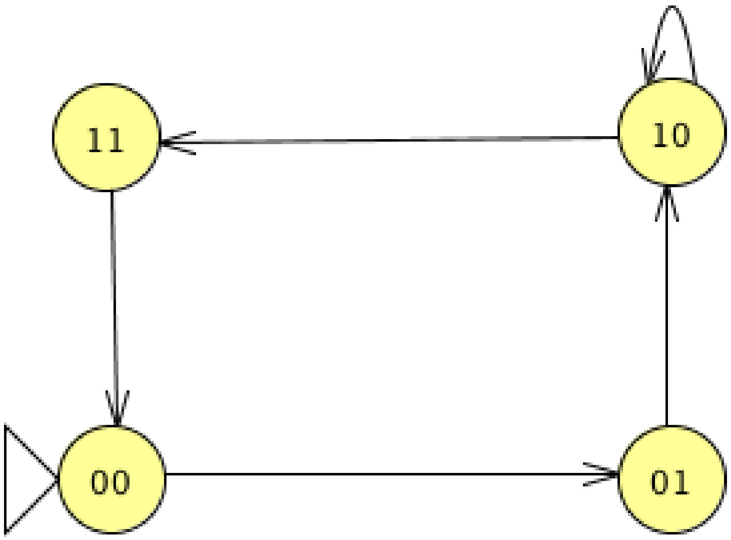
- **Beispielspezifikation:**
  In maximal 2 Schritten ist Zustand 11 erreichbar

$$
\begin{array}{ll}
C_q(m_0, m_1): & (\neg m_0 \wedge \neg m_1) \wedge \\
C_T(m_0, m_1, n_0, n_1): & ((n_1 \leftrightarrow \neg m_1) \wedge (n_0 \leftrightarrow (m_0 \otimes m_1))) \wedge \\
C_T(n_0, n_1, r_0, r_1): & ((r_1 \leftrightarrow \neg n_1) \wedge (r_0 \leftrightarrow (n_0 \otimes n_1))) \wedge \\
 & ((m_0 \wedge m_1) \vee (n_0 \wedge n_1) \vee (r_0 \wedge r_1))
\end{array}
$$

- Die Formel ist unerfüllbar, daher die Spezifikation falsch.

Umwandlung einer Formel (3)
Liveness-Eigenschaften, Beispiel

Spezifikation von Eigenschaften $\mathbf{AF}\varphi = \mathbf{EG}\neg\varphi$



- Beispiel: Zustand 11 ist immer erreichbar
- Länge des Pfades *k*=4, Teil einer Schleife

  $(m_0, m_1, n_0, n_1, r_0, r_1, s_0, s_1)$

- **Charakteristische Funktionen der Kripke-Struktur:**

$$C_T(m_0, m_1, n_0, n_1) = ((n_1 \leftrightarrow \neg m_1) \wedge (n_0 \leftrightarrow (m_0 \otimes m_1))) \vee (m_0 \wedge \neg m_1 \wedge n_0 \wedge \neg n_1)$$

$$C_q(m_0, m_1) = (\neg m_0 \wedge \neg m_1)$$

$$(s_0 \leftrightarrow m_0 \wedge s_1 \leftrightarrow m_1) \vee (s_0 \leftrightarrow n_0 \wedge s_1 \leftrightarrow n_1) \vee (s_0 \leftrightarrow r_0 \wedge s_1 \leftrightarrow r_1)$$

Umwandlung einer Formel (4)
Liveness-Eigenschaften, Beispiel

$$C_T(m_0,m_1,n_0,n_1) \quad = \quad ((n_1 \leftrightarrow \neg m_1) \wedge (n_0 \leftrightarrow (m_0 \otimes m_1)))\vee$$
$$(m_0 \wedge \neg m_1 \wedge n_0 \wedge \neg n_1)$$

$$C_q(m_0,m_1) = (\neg m_0 \wedge \neg m_1)$$

- **Spezifikation:**

Übergangsrelation

$$C_q(m_0,m_1) \wedge C_T(m_0,m_1,n_0,n_1) \wedge C_T(n_0,n_1,r_0,r_1) C_T(r_0,r_1,s_0,s_1)\wedge$$

$$(s_0 \leftrightarrow m_0 \wedge s_1 \leftrightarrow m_1) \vee (s_0 \leftrightarrow n_0 \wedge s_1 \leftrightarrow n_1) \vee (s_0 \leftrightarrow r_0 \wedge s_1 \leftrightarrow r_1)\wedge$$

$$(\neg m_0 \vee \neg m_1) \wedge (\neg n_0 \vee \neg n_1) \wedge (\neg r_0 \vee \neg r_1)$$

Schleife

zu prüfende Eigenschaft:
Zustand 11 nicht erreichbar

- Erfüllbare Belegung der Variablen liefert Gegenbeispiel

SAT (1)

TECHNISCHE UNIVERSITÄT DARMSTADT

SAT = Satisfiability of Boolean Formulas

- Test ob eine Boolesche Formel in konjunktiver Normalform erfüllbar ist, d.h. ob eine erfüllende Belegung der Variablen existiert.

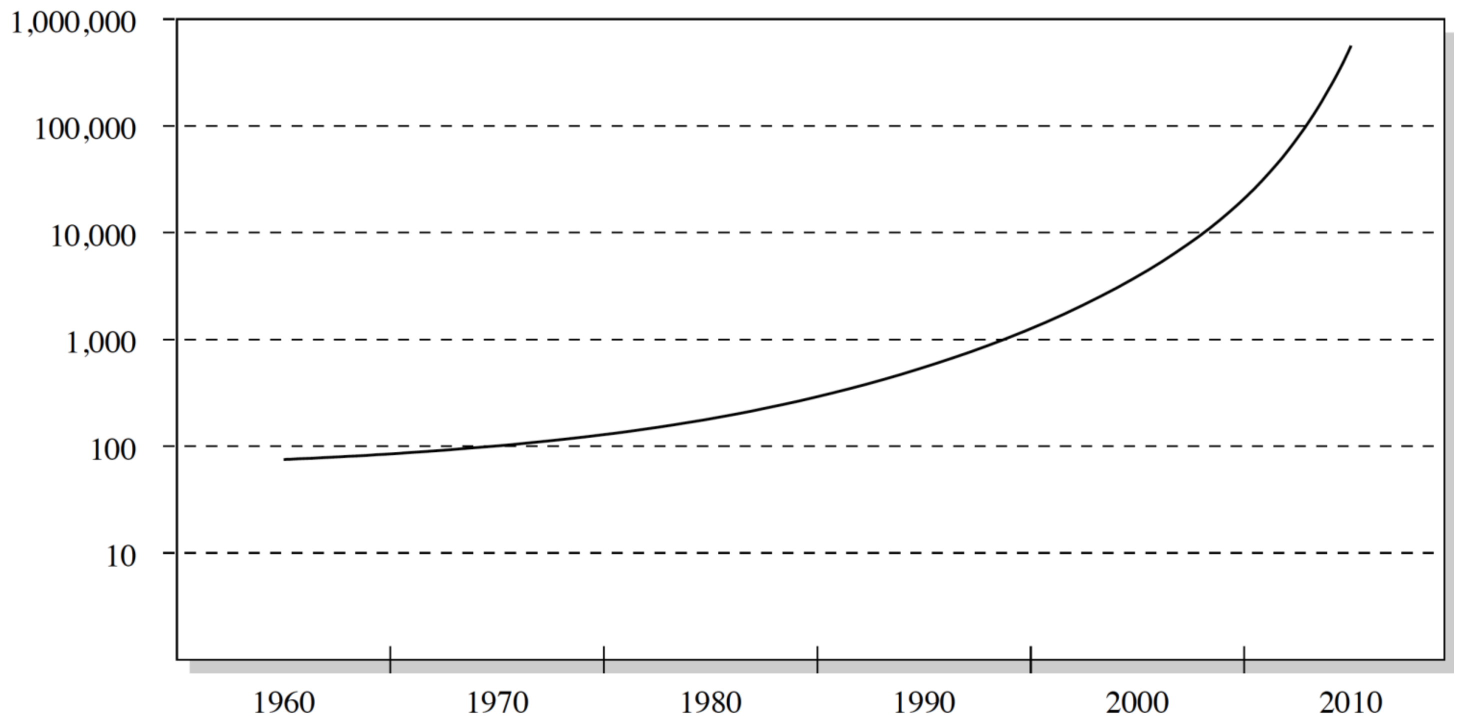
$$\bigwedge_i \bigvee_j p_{i,j}, \quad p_{i,j} = x_{i,j} \text{ oder } p_{i,j} = \neg x_{i,j}$$

- Problem benötigt exponentielle Rechenzeit, ist **NP-vollständig**
  ➔ Details siehe nächste Vorlesung

- Gut untersuchte Problemklasse, gute Algorithmen für kleinere Probleme existieren!

SAT (2)

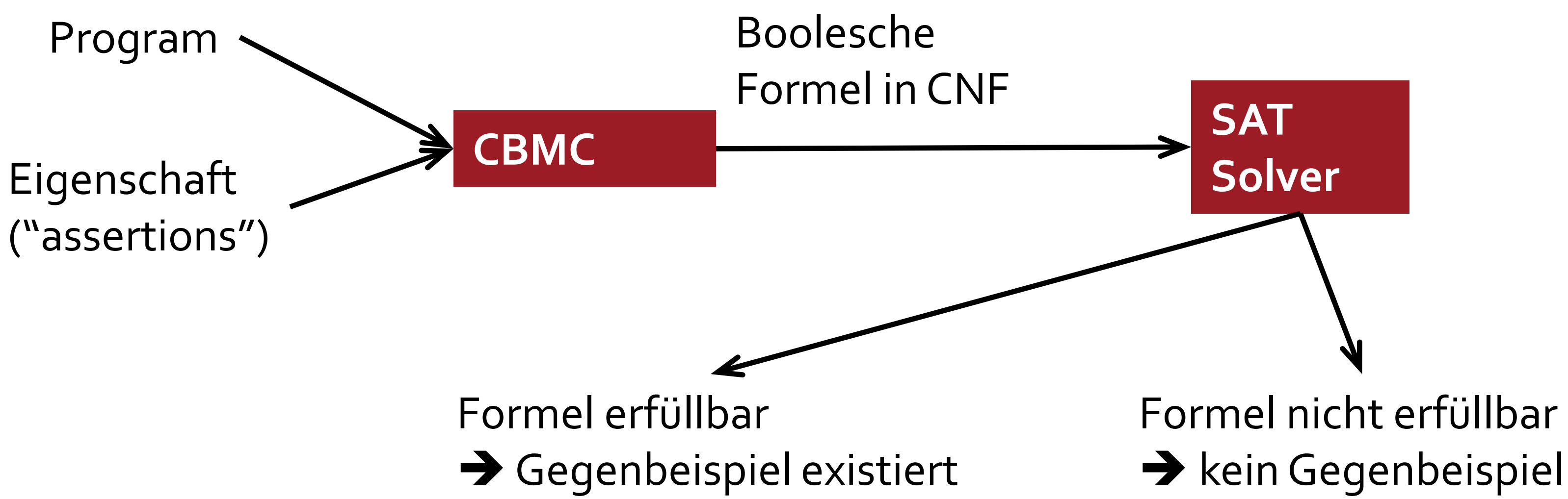
Größe der Probleme (Zahl der Variablen), die im entsprechenden Jahrzehnt durch SAT-Solver lösbar war



Quelle: D. Kroening, http://www.cprover.org/cbmc/doc/cbmc-slides.pdf

Bounded Model Checking in der Praxis: CBMC (1)

- CBMC sucht in einem Programm nach Abläufen die gewisse Eigenschaften verletzen
- Eingabe ist Programm in ANSI-C inklusive „assertions“
- CBMC testet Gültigkeit der assertions.



Bounded Model Checking in der Praxis: CBMC (2)

- Entwickelt von Daniel Kroening
- Verfügbar unter http://www.cprover.org
- Verschiedene Plattformen (Windows, OSX, Unix)

- Skaliert bis zu Programmen mit ca. 30 Tausend Codezeilen
- Wurde mehrfach erfolgreich in der Suche nach Fehlern in Gerätetreibern genutzt

- Eigenschaften können spezifiziert werden als C-assertions
- Implizite Eigenschaften: array bounds, Division durch Null, Dereferenzierung von NULL-Zeigern, Arithmetische Overflows
- Hier: Fokus auf assertions der Form $\mathbf{AG}p$

Bounded Model Checking in der Praxis: CBMC (3)

Grobe Vorhehensweise:

1. Vereinfachung des Koltrollflusses
   (wie Ersatz von `j=i++` durch `j=i; i=i+1`)
2. Ersatz von Anweisungen wie `continue` und `break` durch `goto`
3. Ersatz von allen Schleifen durch `while`
4. Loop unrolling
   - Entfernen aller Schleifen
   - Ersatz durch Kopien des Schleifenkörpers
   - Benutzung von statischer Analyse um Obergrenze für Schleifenvariable zu bestimmen, sonst Nutzung einer globalen oberen Schranke
5. Single Static Assignment: Allen Variablen wird nur ein Mal ein Wert zugewiesen; falls dies nicht der Fall ist wird Kopie der Variable erzeugt
6. Generierung einer Formel aus dem erhaltenen Programm

Bounded Model Checking in der Praxis: CBMC, Loop unrolling (4)

```
void f(...) {
  ...
  while(cond) {
    Body;
  }
  Remainder;
}
```

Bounded Model Checking in der Praxis: CBMC, Loop unrolling (4)

```
void f(...) {

}
```

```
void f(...) {
  ...
  if(cond) {
    Body;
    while(cond) {
      Body;
    }
  }
  Remainder;
}
```

Bounded Model Checking in der Praxis: CBMC, Loop unrolling (4)

TECHNISCHE UNIVERSITÄT DARMSTADT

```
void f(...) {

}
```

```
void f(...) {
  ⋮
  if
  
  }
  Re
}
```

```
void f(...) {
  ⋮
  if(cond) {
    Body;
    if(cond) {
      Body;
      while(cond) {
        Body;
      }
    }
  }
  Remainder;
}
```

Bounded Model Checking in der Praxis: CBMC, Loop unrolling (4)

```
void f(...) {
  ...
  if(cond) {
    Body;
    if(cond) {
      Body;
      if(cond) {
        Body;
        assert(!cond);
      }
    }
  }
  Remainder;
}
```

- While-Schleifen werden iterativ entfernt
- Ersatz durch if und Kopie des Schleifenkörpers
- Assertion am Ende die zusichert dass genügend viele Kopien erstellt wurden

Bounded Model Checking in der Praxis: CBMC, Single Static Assignment (5)

```
...
x = x + y;
x = x * 2;
a[x] = y;
...
```

→

```
...
x1 = x0 + y0;
x2 = x1 * 2;
a0[x2] = y0;
...
```

- Jeder Variable wird nur ein Mal ein Wert zugewiesen
- Falls dies nicht der Fall ist: Kopie der Variable erzeugen

Bounded Model Checking in der Praxis: CBMC (6)

Generierung einer Formel aus dem erhaltenen Programm:

- Bit-präzise Modellierung eines Programms, jedes Bit des „Speichers" (der Variablen im Programm) wird eine Boolesche Variable
- Übersetzung des Kontrollflusses in aussagenlogische Formel
- Übersetzung der arithmetischen Operationen durch entsprechende Schaltkreise („circuits")
- Jede Assertion wird wie $\mathbf{AG}p$ behandelt: Übersetzung in $\bigvee_i \neg p(s_i)$