Karlsruhe Institute of Technology

Entscheidungsverfahren mit Anwendungen in der Softwareverifikation

I: Einführung

Dr. Stephan Falke
Dr. Carsten Sinz

Institut für Theoretische Informatik

15.04.2013

Ist mein Programm korrekt?

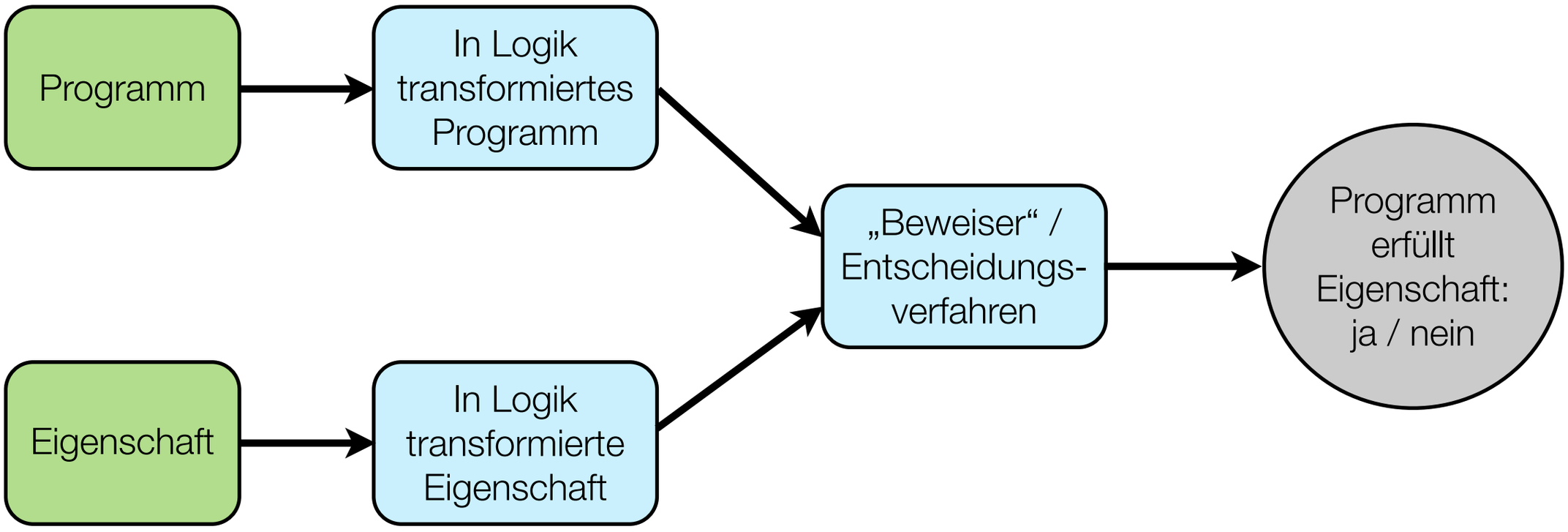
- **Beispiel:** Bestimme Anzahl der 1-Bits in einem Wort (*population-count*)

```
uint32_t reference_popcount(uint32_t x)
{
    int i, s = 0;
    for(i = 0; i < 32; ++i)
        if(x & (1 << i))
            s++;
    return s;
}
```

```
uint32_t optimized_popcount(uint32_t x)
{
    x = x - ((x >> 1 ) & 0x55555555);
    x = (x & 0x33333333) + ((x >> 2) & 0x33333333);
    x = (x + (x >> 4)) & 0x0F0F0F0F;
    x += (x >> 8);
    x += (x >> 16);
    return x & 0x0000003F;
}
```

- **Frage:** Liefert die optimierte Version immer das gleiche Ergebnis wie die Referenzimplementierung?

Der Logik-Ansatz



Zu klären

- **Welche Programme wollen wir prüfen?**
  - ➜ C, C++, Java, ... (Imperative Programmiersprachen)
- **Welche Eigenschaften wollen wir prüfen?**
  - ➜ Keine Programmabstürze
  - ➜ Keine „falschen“ Rechenergebnisse
  - ➜ Keine Sicherheitslücken
- **Welche Logik ist dafür geeignet?**
  - Modellierung von primitiven Datentypen (`int`, `float`, ...) und Operationen darauf
  - Modellierung von komplexen Datentypen
  - Modellierung von Programmfluss
  - Modellierung von Speicher

Welche Logik?

- **Erwünschte Eigenschaften:**
  - Möglichst ausdrucksstark
  - Möglichst entscheidbar
  - Eingebaute Theorien (z.B. Arithmetik)
- **Kandidaten:**
  - Aussagenlogik
  - Prädikatenlogik erster Stufe
  - CTL / LTL
  - Spezielle Logiken

Definition Entscheidbarkeit

- **Definition (Erfüllbarkeit, *Gültigkeit*):**
  Eine Formel ist *erfüllbar*, wenn es *eine* Variablenbelegung und eine Interpretation der (nicht-logischen) Symbole *gibt*, so dass die Formel zu **wahr** evaluiert. Eine Formel ist gültig, wenn sie für *alle* Variablenbelegungen und *alle* Interpretationen der (nicht-logischen) Symbole zu **wahr** evaluiert.
- **Definition (*Entscheidungsproblem*):**
  Das *Entscheidungsproblem* für eine gegebene Formel (in einer Logik L) ϕ ist zu bestimmen, ob ϕ gültig ist.
- **Definition (*Korrektheit*):**
  Ein Algorithmus für das Entscheidungsproblem ist *korrekt*, falls immer wenn er „gültig“ als Ergebnis liefert die Eingabeformel wirklich gültig ist.
- **Definition (*Vollständigkeit*):**
  Ein Algorithmus für das Entscheidungsproblem ist *vollständig*, wenn er (a) immer terminiert und (b) „gültig“ als Ergebnis liefert, sofern die Eingabeformel gültig ist.
- **Definition (*Entscheidungsverfahren*):**
  Ein Algorithmus A heißt *Entscheidungsverfahren* für eine Logik L, wenn A korrekt und vollständig ist für jede Formel aus L.
- **Definition (*Entscheidbarkeit einer Logik*):**
  Eine Logik L heißt entscheidbar, wenn es ein Entscheidungsverfahren für L gibt.

Entscheidbarkeit für verschiedene Logiken

Logik	Entscheidbarkeit
Prädikatenlogik (erster Stufe)	**unentscheidbar**
Monadische Prädikatenlogik	**entscheidbar**
Polynomgleichungen über den ganzen Zahlen	**unentscheidbar**
Presburger-Arithmetik	**entscheidbar**
Quantorenfreie Logik der Bitvektoren	**entscheidbar**

Vorlesungsüberblick / Themen

Grundlagen	15.04.	Sinz / Falke	Einführung
	22.04.	Sinz	SAT-Solving: Grundlegende Algorithmen (DPLL)
	29.04.	Sinz	SAT-Solving: Fortgeschrittene Algorithmen
	06.05.	Falke	DPLL(T): DPLL mit Theorie-Atomen
	13.05.	Falke	Uninterpretierte Funktionen mit Gleichheit (EUF)
	27.05.	Sinz	Lineare Arithmetik der ganzen Zahlen (Fourier-Motzkin, Omega-Test)
	03.06.	Falke	Logik der Bitvektoren
	10.06.	Sinz	Array-Logik
	17.06.	Falke	Kombination von Theorien
Anwendungen	24.06.	Falke	LLBMC: Software Bounded Model Checking
	01.07.	Falke	KLEE: Symbolic Execution
	08.07.	Sinz	SATABS: Predicate Abstraction
	15.07.	Merz	Lab: LLBMC, KLEE, SATABS

Enthält dieser C-Code einen Fehler?

```
int SATD (void)
{
  int d[16];
   ⋮

  int satd = 0, dd, k;
  for (dd=d[k=0]; k<16; dd=d[++k]) {
    satd += (dd < 0 ? -dd : dd);
  }
  return satd;
}
```

[Code from SPEC CPU 2006 Benchmark `464.h264ref`; via John Regehr's blog *Embedded in Academia*; March 22, 2013]

Enthält dieser C-Code einen Fehler?

```
int SATD (void)
{
  int d[16];
   ⋮

  int satd = 0, dd, k;
  for (dd=d[k=0]; k<16; dd=d[++k]) {
    satd += (dd < 0 ? -dd : dd);
  }
  return satd;
}
```

array index out of bounds in iteration 16

[Code from SPEC CPU 2006 Benchmark `464.h264ref`; via John Regehr's blog *Embedded in Academia*; March 22, 2013]

Enthält dieser C-Code einen Fehler?

- **Array-index-out-of-bounds error**
  - What are the consequences?

    None?!
  - How can we detect this error?
    By testing? **No!**

```
int SATD (void)
{
  int d[16];
   ⋮
  int satd = 0, dd, k;
  for (dd=d[k=0]; k<16; dd=d[++k]) {
    satd += (dd < 0 ? -dd : dd);
  }
  return satd;
}
```

Enthält dieser C-Code einen Fehler?

- **Array-index-out-of-bounds error**
  - What are the consequences?

    None?!
  - How can we detect this error?
    By testing? **No!**

```
int SATD (void)
{
  int d[16];
   ⋮
  int satd = 0, dd, k;
  for (dd=d[k=0]; k<16; dd=d[++k]) {
    satd += (dd < 0 ? -dd : dd);
  }
  return satd;
}
```

GCC pre-4.8 generates the following code:

```
SATD:
     ...
.L2:
     jmp .L2
```

Enthält dieser C-Code einen Fehler?

- **Array-index-out-of-bounds error**
  - What are the consequences?

    Unexpected behavior!
  - How can we detect this error?
    By testing? **Perhaps (depending on the compiler)**

```
int SATD (void)
{
  int d[16];
   ⋮

  int satd = 0, dd, k;
  for (dd=d[k=0]; k<16; dd=d[++k]) {
    satd += (dd < 0 ? -dd : dd);
  }
  return satd;
}
```

GCC pre-4.8 generates the following code:

```
SATD:
     ...
.L2:
     jmp .L2
```

Literatur

- Daniel Kröning / Ofer Strichman:
  Decision Procedures – An Algorithmic Point of View
  (Springer, 2008)
  [Innerhalb des Uni-Netzes ist eine Online-Version im Volltext verfügbar.]

- Aaron R. Bradley / Zohar Manna:
  The Calculus of Computation: Decision Procedures with Applications to Verification
  (Springer, 2007)