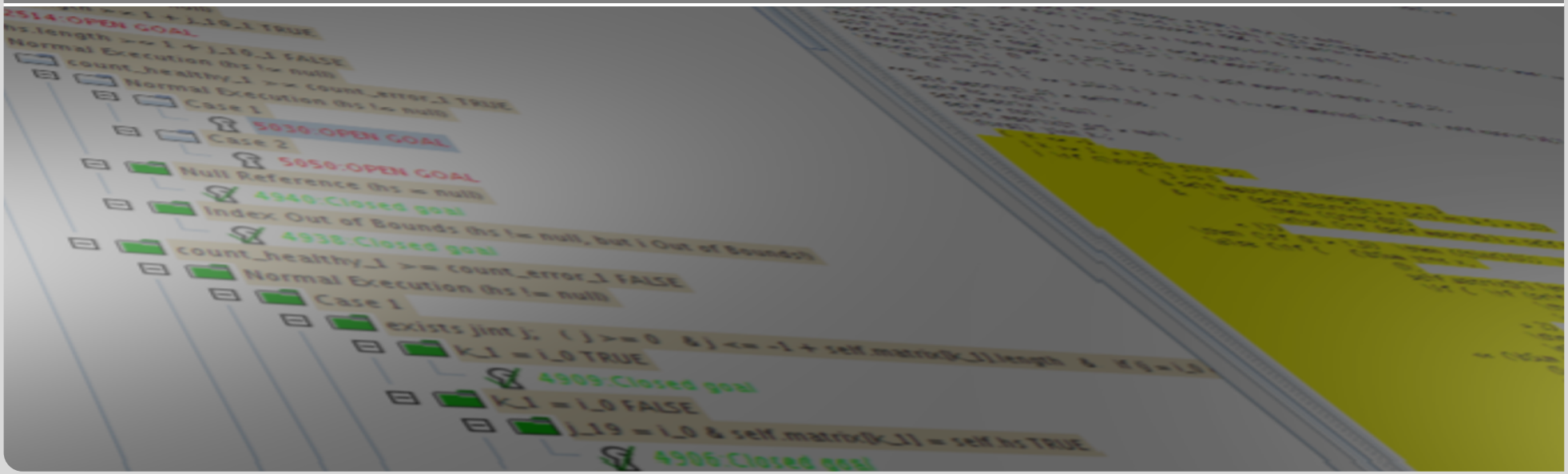
Karlsruher Institut für Technologie

Automatisches Prüfen von Programmeigenschaften

Prof. Bernhard Beckert, Thorsten Bormer, Daniel Bruns | 24. Oktober 2012

Institut für Theoretische Informatik – Anwendungsorientierte formale Verifikation



Organisatorisches

PSE am ITI Beckert

Prof. Dr.
B. Beckert

Dipl.-Inform.
Thorsten
Bormer

Dipl.-Inform.
Daniel
Bruns

- **Ziel**: Ein Werkzeug zur Analyse und Korrektheitsprüfung von Programmen bzgl. ihrer Informationsflüsse.
- **Kommunikation**
  - **Website**: `http://formal.iti.kit.edu`
  - **wöchentliche Treffen** ($\rightarrow$ Doodle)
  - **eMail**: `{bormer,bruns}@kit.edu`

Zeitplan

Phase	von – bis	Dauer
Erstes Gruppentreffen	24.10.	
Pflichtenheft	29.10. – 16.11.	3 Wochen
Entwurf	19.11. – 14.12.	4 Wochen
Implementierung	17.12. – 25.01.	4 Wochen
Qualitätssicherung	28.01. – 08.02.	2 Wochen
Klausurpause	11.02. – 22.02.	
Qualitätssicherung	25.02. – 01.03.	1 Wochen
interne Abnahme	04.03. – 08.03.	
Abschlusspräsentation	11.03. – 15.03.	

Nachweis von Informationsflusseigenschaften

Motivation

Unerwünschte Informationsflüsse – Beispiele aus der Praxis[1]:

CVE-1999-1231 “ssh [. . .] allows valid user names to attempt to enter the correct password multiple times, but only prompts an invalid user name for a password once [. . .]”

[1] Quelle: `http://cve.mitre.org`

Motivation

Unerwünschte Informationsflüsse – Beispiele aus der Praxis[1]:

CVE-1999-1231 “ssh [. . .] allows valid user names to attempt to enter the correct password multiple times, but only prompts an invalid user name for a password once [. . .]”

CVE-2008-2049 “[. . .] POP3 server [. . .] displays the password in a POP3 error message.”

[1] Quelle: `http://cve.mitre.org`

Motivation

Unerwünschte Informationsflüsse – Beispiele aus der Praxis[1]:

CVE-1999-1231 “ssh [. . .] allows valid user names to attempt to enter the correct password multiple times, but only prompts an invalid user name for a password once [. . .]”

CVE-2008-2049 “[. . .] POP3 server [. . .] displays the password in a POP3 error message.”

CVE-2012-4429 “Vino [. . .] allows remote attackers to read clipboard activity by listening on TCP port 5900.”

[1] Quelle: `http://cve.mitre.org`

Informationsfluss – Beispiel I

```
for (int i=0; i<secretValues.length; i++) {
    int val = publicValue / secretValues[i];
    doSomething(val);
}
return publicValue;
```

Informationsfluss:

Informationsfluss – Beispiel I

```
for (int i=0; i<secretValues.length; i++) {
    int val = publicValue / secretValues[i];
    doSomething(val);
}
return publicValue;
```

Informationsfluss: Array `secretValues` enthält eine `0`.

Informationsfluss – Beispiel II

```
for (int i=0; i<secretValues.length; i++) {
    secretValues[i] = 0;
}
return secretValues;
```

Informationsfluss:

Informationsfluss – Beispiel II

```
for (int i=0; i<secretValues.length; i++) {
    secretValues[i] = 0;
}
return secretValues;
```

Informationsfluss: Länge des Arrays `secretValues`.

Informationsfluss – Beispiel IV

```
for (int i=0; i<secretValues.length; i++) {
    secretValues[i] = 0;
}
return publicValue;
```

Informationsfluss:

Informationsfluss – Beispiel IV

```
for (int i=0; i<secretValues.length; i++) {
    secretValues[i] = 0;
}
return publicValue;
```

Informationsfluss: Länge des Arrays `secretValues` (Seitenkanal durch unterschiedlichen Zeitbedarf der Schleife).

Informationsfluss – Beispiel IV

```
for (int i=0; i<secretValues.length; i++) {
    secretValues[i] = 0;
}
return publicValue;
```

Informationsfluss: Länge des Arrays `secretValues` (Seitenkanal durch unterschiedlichen Zeitbedarf der Schleife).

Diese Art von Informationsfluss wird im PSE nicht betrachtet!

Informationsfluss – Beispiel V

```
class PasswordFile {
  private int[] names, passwords;

  public boolean check(int user, int password) {
    for (int i = 0; i < names.length; i++) {
      if (names[i] == user &&
        passwords[i] == password) {
          return true;
      }
    }
    return false;
  }
}
```

Informationsfluss:

Informationsfluss – Beispiel V

```
class PasswordFile {
  private int[] names, passwords;

  public boolean check(int user, int password) {
    for (int i = 0; i < names.length; i++) {
      if (names[i] == user &&
        passwords[i] == password) {
          return true;
      }
    }
    return false;
  }
}
```

Informationsfluss: Es gibt ein Vorkommen der Name/Passwort-Kombination in den Arrays `names`, bzw. `passwords` (an gleichem Index).

Sicherstellen von Informationsflusseigenschaften

Sicherstellen der gewünschten Informationsflusseigenschaften in Programmen durch. . .

Sicherstellen von Informationsflusseigenschaften

Sicherstellen der gewünschten Informationsflusseigenschaften in Programmen durch. . .

- guten Entwurf (Trennung der Belange, Datenkapselung, . . .)

Sicherstellen von Informationsflusseigenschaften

Sicherstellen der gewünschten Informationsflusseigenschaften in Programmen durch. . .

- guten Entwurf (Trennung der Belange, Datenkapselung, . . .)
- Einhaltung der ’best practices’ in der Programmierung

Sicherstellen von Informationsflusseigenschaften

Sicherstellen der gewünschten Informationsflusseigenschaften in Programmen durch. . .

- guten Entwurf (Trennung der Belange, Datenkapselung, . . .)
- Einhaltung der ’best practices’ in der Programmierung
- **Nachweis** im Quelltext des Programms

Sicherstellen von Informationsflusseigenschaften

Sicherstellen der gewünschten Informationsflusseigenschaften in Programmen durch. . .

- guten Entwurf (Trennung der Belange, Datenkapselung, . . .)
- Einhaltung der ’best practices’ in der Programmierung
- **Nachweis** im Quelltext des Programms
  - Code reviews

Sicherstellen von Informationsflusseigenschaften

Sicherstellen der gewünschten Informationsflusseigenschaften in Programmen durch. . .

- guten Entwurf (Trennung der Belange, Datenkapselung, . . .)
- Einhaltung der ’best practices’ in der Programmierung
- **Nachweis** im Quelltext des Programms
  - Code reviews
  - Programmabhängigkeitsgraphen, Typsysteme

Sicherstellen von Informationsflusseigenschaften

Sicherstellen der gewünschten Informationsflusseigenschaften in Programmen durch. . .

- guten Entwurf (Trennung der Belange, Datenkapselung, . . .)
- Einhaltung der ’best practices’ in der Programmierung
- **Nachweis** im Quelltext des Programms
  - Code reviews
  - Programmabhängigkeitsgraphen, Typsysteme
  - **Programmverifikation**

Non-interference

Durch den Benutzer vorgegeben: Einteilung der Variablen des Programms in:

- Variablen mit schützenswerten, geheimen Daten (*high*-Variablen)
- öffentlich einsehbare Variablen (*low*-Variablen)

Non-interference

Durch den Benutzer vorgegeben: Einteilung der Variablen des Programms in:

- Variablen mit schützenswerten, geheimen Daten (*high*-Variablen)
- öffentlich einsehbare Variablen (*low*-Variablen)

Informationsflusseigenschaft: Non-interference

Die Belegung der high-Variablen beeinflusst nicht die Belegung der low-Variablen nach Ablauf des Programms. Kenntnis der Werte der low-Variablen im Endzustand führt nicht zu Erkenntnisgewinn bzgl. der Werte der high-Variablen.

Non-interference

Durch den Benutzer vorgegeben: Einteilung der Variablen des Programms in:

- Variablen mit schützenswerten, geheimen Daten (*high*-Variablen)
- öffentlich einsehbare Variablen (*low*-Variablen)

Informationsflusseigenschaft: Non-interference

Die Belegung der high-Variablen beeinflusst nicht die Belegung der low-Variablen nach Ablauf des Programms. Kenntnis der Werte der low-Variablen im Endzustand führt nicht zu Erkenntnisgewinn bzgl. der Werte der high-Variablen.

D.h., zwei Läufe des Programms gestartet mit identischen Werten der low-Variablen führt zu identischen Werten der low-Variablen im Endzustand – *unabhängig von der Belegung der high-Variablen*.

Beispiel – Nachweis von Non-interference

Beispielprogramm

```
int m(int secret, int pub) {
  if (high > 0) {return pub;} else {return 0;}
}
```

Beispiel – Nachweis von Non-interference

Karlsruher Institut für Technologie

Beispielprogramm

```
int m(int _(high) secret, int pub) {
  if (high > 0) {return pub;} else {return 0;}
}
```

Beispiel – Nachweis von Non-interference

Beispielprogramm

```
int m(int _(high) secret, int pub) {
  if (high > 0) {return pub;} else {return 0;}
}
```

Nachweis durch *self composition*

```
void checkFlow() {
  int low = rand();
  int high1 = rand();
  int high2 = rand();

  int res1 = m(high1, low);
  int res2 = m(high2, low);
  //hier muss res1 == res2 gelten
}
```

Beispiel – Nachweis von Non-interference

Nachweis durch *self composition*

```
void checkFlow() {
  int low = rand();
  int high1 = rand();
  int high2 = rand();

  int res1 = m(high1, low);
  int res2 = m(high2, low);
  //hier muss res1 == res2 gelten
}
```

Beispiel – Nachweis von Non-interference

Nachweis durch *self composition*

```
void checkFlow() {
  int low = rand();
  int high1 = rand();
  int high2 = rand();

  int res1 = m(high1, low);
  int res2 = m(high2, low);
  _(assert res1 == res2)
}
```

Beispiel – Nachweis von Non-interference

Nachweis durch *self composition*

```
void checkFlow() {
  int low = rand();
  int high1 = rand();
  int high2 = rand();

  int res1 = m(high1, low);
  int res2 = m(high2, low);
  _(assert res1 == res2)
}
```

Nachweis der Korrektheit für *alle* Belegungen von `low`, `high1` und `high2`:

Beispiel – Nachweis von Non-interference

Nachweis durch *self composition*

```
void checkFlow() {
  int low = rand();
  int high1 = rand();
  int high2 = rand();

  int res1 = m(high1, low);
  int res2 = m(high2, low);
  _(assert res1 == res2)
}
```

Nachweis der Korrektheit für *alle* Belegungen von `low`, `high1` und `high2`:

- Umwandlung des Programms + Eigenschaften in logische Formel

Beispiel – Nachweis von Non-interference

Nachweis durch *self composition*

```
void checkFlow() {
  int low = rand();
  int high1 = rand();
  int high2 = rand();

  int res1 = m(high1, low);
  int res2 = m(high2, low);
  _(assert res1 == res2)
}
```

Nachweis der Korrektheit für *alle* Belegungen von `low`, `high1` und `high2`:

- Umwandlung des Programms + Eigenschaften in logische Formel
- Beweis dieser Formel mit existierendem Beweiswerkzeug (Z3)

Deklassifikation – “Password Checker”

```
int[] names, passwords;

boolean check(int user, int password)
{ ... }
```

Deklassifikation – “Password Checker”

```
_(high) int[] names, passwords;

boolean check(int user, int password)
{ ... }
```

Deklassifikation – “Password Checker”

```
_(high) int[] names, passwords;

boolean check(int user, int password)
{ ... }
```

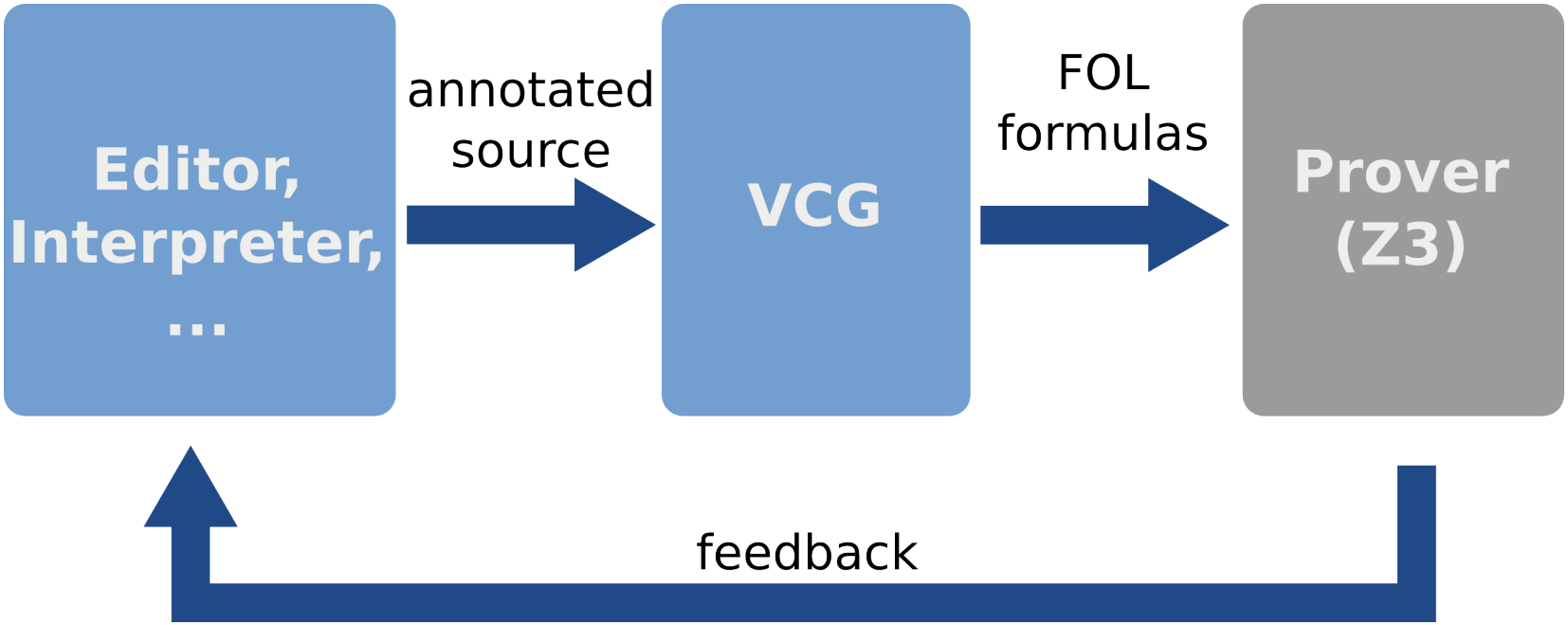
Informationsfluss: Es gibt ein Vorkommen der Name/Passwort-Kombination in den Arrays `names`, bzw. `passwords` (an gleichem Index).

Deklassifikation – “Password Checker”

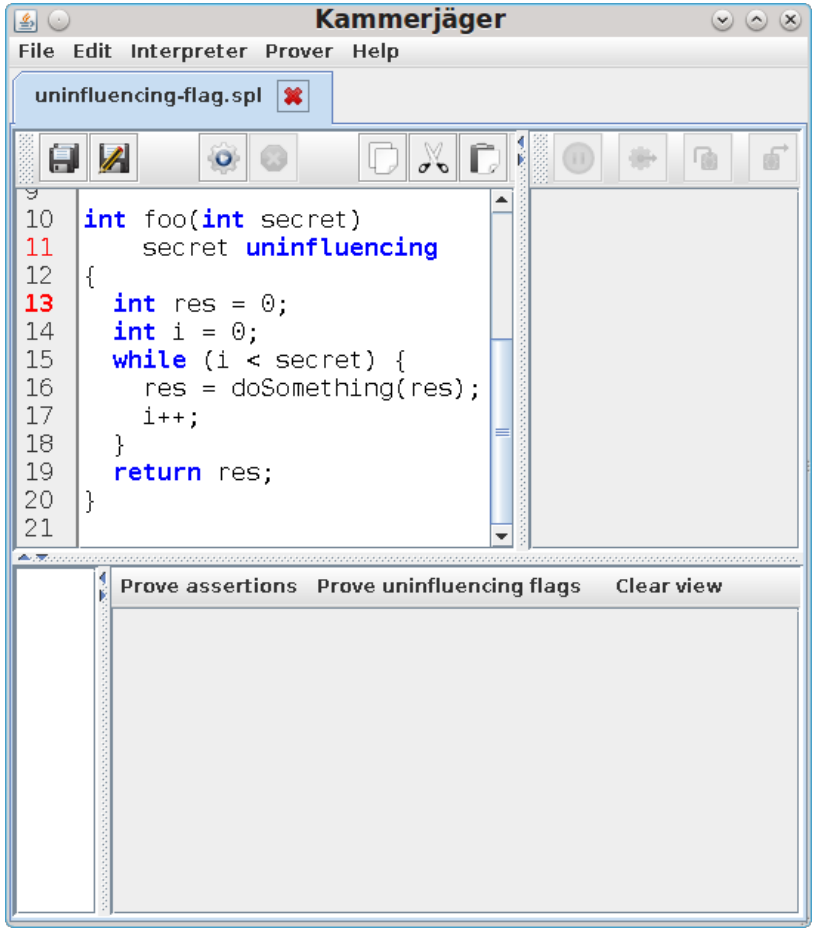
```
_(high) int[] names, passwords;

boolean check(int user, int password)
    _(declassify \exists int i; i >= 0
      && i < names.length &&
      names[i] == user && passwords[i] == password)
{ ... }
```

Informationsfluss: Es gibt ein Vorkommen der Name/Passwort-Kombination in den Arrays `names`, bzw. `passwords` (an gleichem Index).

Komponenten des Werkzeugs



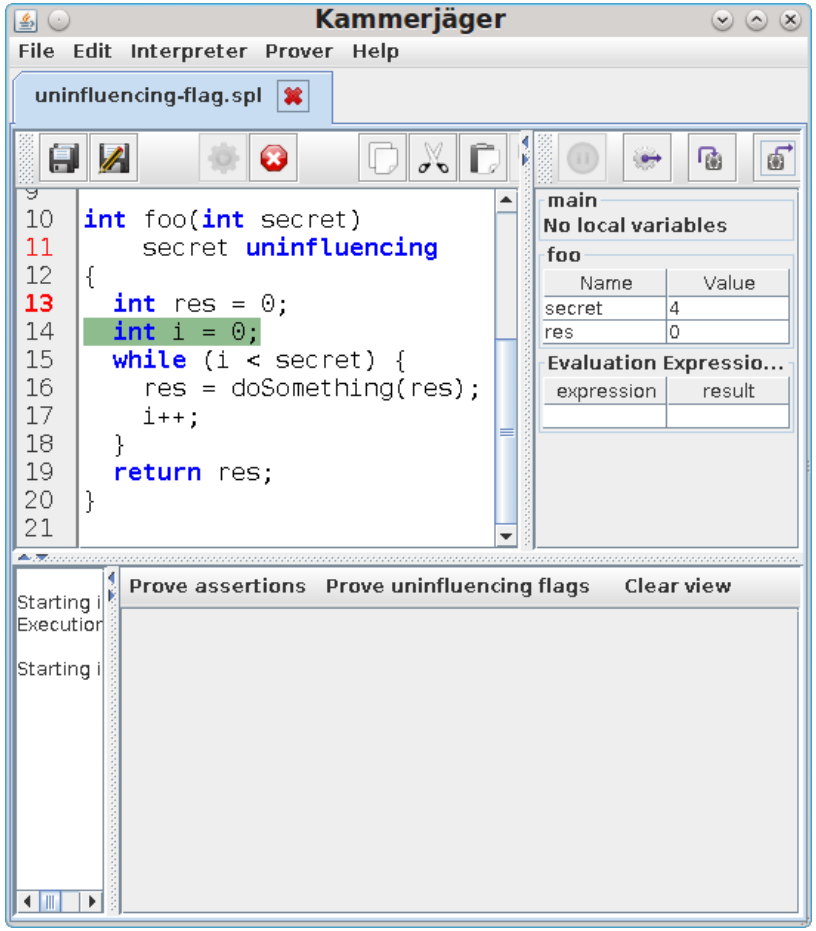
Ergebnis des PSE: Das Analysewerkzeug



Bestandteile der GUI

- Editor

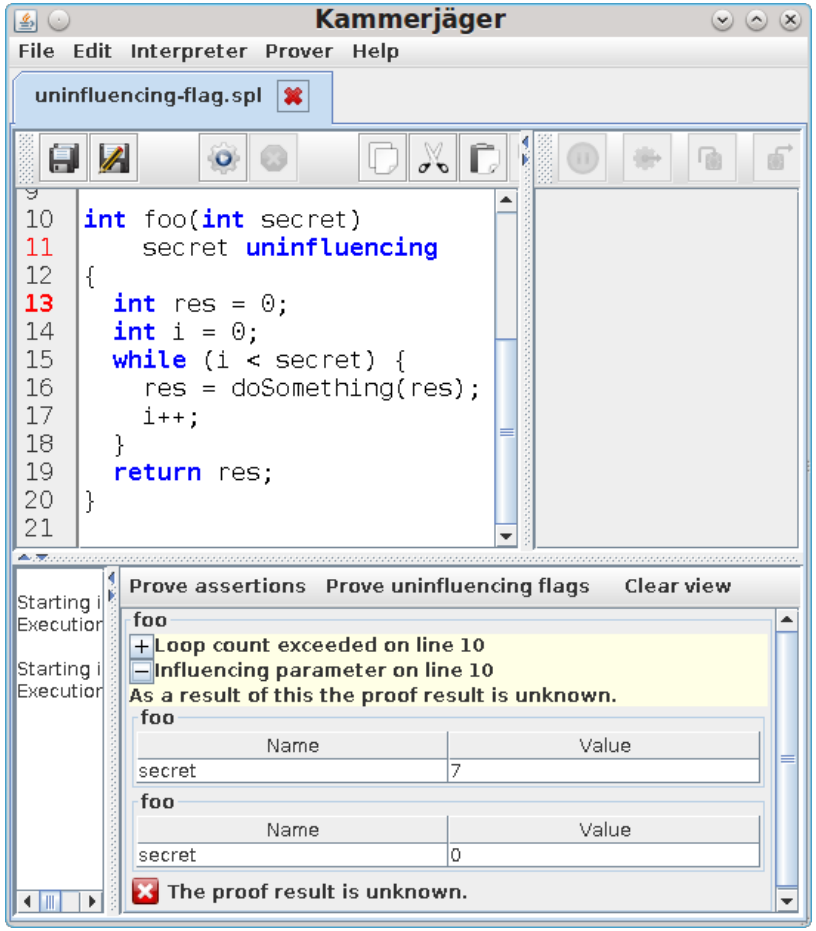
Ergebnis des PSE: Das Analysewerkzeug



Bestandteile der GUI

- Editor
- Interpreter

Ergebnis des PSE: Das Analysewerkzeug



Bestandteile der GUI

- Editor
- Interpreter
- Programmverifikation

Weiterführende Materialien

- Goos, Zimmermann: Vorlesungen über Informatik
  (Band 1: Kapitel 4.2, Prädikatenlogik
  Band 2: Kapitel 8.2, Zusicherungskalkül)
- Web-interface zu Verifikationswerkzeugen: `http://rise4fun.com`