

# Verlässliche Echtzeitsysteme

Übungen zur Vorlesung

Abstrakte Interpretation

Phillip Raffeck, Florian Schmaus, Simon Schuster

Friedrich-Alexander-Universität Erlangen-Nürnberg  
Lehrstuhl Informatik 4 (Verteilte Systeme und Betriebssysteme)  
<https://www4.cs.fau.de>

Sommersemester 2020

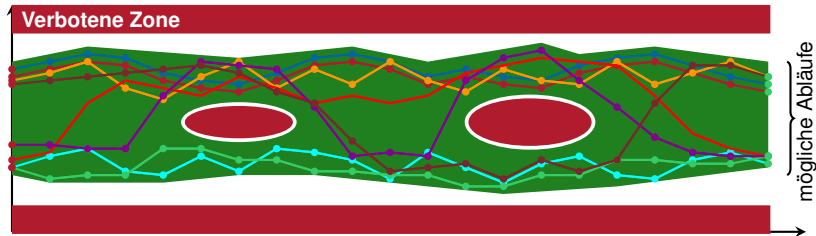


- Ziel: *bewiesenermaßen* korrektes Verhalten von Programmen
  - bisher: nichtfunktionale Eigenschaften
  - jetzt: semantische, *funktionale* Eigenschaften
- Werkzeuge: Astrée & Frama-C
- Grundvoraussetzung: Regeln zum logischen Schließen
  - In unserer Konfiguration (an sich mächtiger):
  - Astrée: Abstrakte Interpretation über der Intervallsemantik
  - Frama-C: wp-Kalkül
- Vorgehen: grobe *Beweisidee* zum Erreichen des Ziels
  - Astrée: Kurvendiskussion, Funktionalanalysis
  - Frama-C: Induktion über Datenstrukturen



- Wunsch: komplett automatisiertes Beweisen
  - *Gödelsches Unvollständigkeitstheorem* (1931)  
*„Jedes hinreichend mächtige formale System ist entweder widersprüchlich oder unvollständig.“*
  - *Satz von Rice* (1953)  
*Es ist nicht möglich, eine beliebige nicht-triviale Eigenschaft der erzeugten Funktion einer Turing-Maschine algorithmisch zu entscheiden.*
    - siehe einführende Erklärung zur Unentscheidbarkeit des Halteproblems [2]
  - Teilweise enormer menschlicher Aufwand nötig
    - Werkzeugunterstützung trotzdem sinnvoll
- ~> Astrée & Frama-C





- **Abstrakte Interpretation** (engl. *abstract interpretation*)
  - Betrachtet eine **abstrakte Semantik** (engl. *abstract semantics*)
    - Sie umfasst **alle Fälle der konkreten Programmsemantik**
  - Ist die abstrakte Semantik sicher  $\Rightarrow$  konkrete Semantik ist sicher

- Ziel: Nachweis der Abwesenheit von Laufzeitfehlern
- Findet *alle* potentiellen Laufzeitfehler
- Leider auch *falsch-positiv*  
→ **Gödelsches Unvollständigkeitstheorem**

## Programm ist korrekt, wenn

- Astrée keine Alarme meldet
- Oder für alle Alarme nachgewiesen, dass falsch-positiv



- Überschreitung von Array-Grenzen
- Ganzzahldivision durch Null
- Ungültige Dereferenzierung, arithmetische Überläufe
- Ungültige Gleitkommaoperationen
- Unerreichbarer Code
- Lesezugriff auf nicht initialisierte Variablen
- Verletzung benutzerdefinierter Zusicherungen  
~> `assert()`



- Rekursion prinzipiell erlaubt, wird aber nicht analysiert
  - ↪ für Rekursionsergebnis werden keine Einschränkungen ermittelt
- Auswertungsreihenfolge in C nicht vollständig spezifiziert
  - ↪ *eine* bestimmte Ordnung wird angenommen
    - stimmt nicht notwendigerweise mit Compiler überein
    - optionale Warnung durch Astrée
- Funktionen der Standard-C-Bibliothek werden nicht erkannt
  - ↪ mitgelieferte Stubs nutzen
- Dynamischer Speicher nicht erlaubt
  - ↪ kein `malloc()`
    - keine Einschränkung im sicherheitskritischen Echtzeitbereich



Astrée nimmt an, dass folgende semantische Regeln gelten:

1. Der C99-Standard
2. Implementierungsabhängiges Verhalten
  - Größe von Datentypen
  - Gleitkommastandard
  - ...
3. Benutzerdefinierte Einschränkungen
  - z. B. ob statische Variablen mit 0 initialisiert werden
4. Außerdem *benutzerspezifizierte Zusicherungen*  
↪ und da wird es interessant 😊





# Benutzerdefinierte Zusicherungen

## `__ASTREE_known_fact((B))`

- Analyser nimmt an, dass B gegeben ist
- Analyser warnt, falls B *nie wahr werden kann*
- `__ASTREE_known_range((V; [a, b]))`  $\rightsquigarrow$  Wertebereich

## `assert((B))/__ASTREE_assert((B))`

- Analyser erzeugt Alarm, falls B *nicht immer wahr ist*
- Analyser nimmt danach an, dass B gilt
- B kann nicht von der Form `e1 ? e2 : e3` sein
- `__ASTREE_global_assert(( ))`  $\rightsquigarrow$  gesamtes Programm
- `__ASTREE_check((B))`  $\rightsquigarrow$  keine Annahme über B danach
  
- B muss seiteneffektfrei sein
- *Doppelte Klammerung ist wichtig!*



# Beispiel

```
1 #include <astree.h> // Astree-Makros ggf. abschalten
2
3 float filter(Alpha_State *s, float val) {
4     __ASTREE_known_fact((val == val)); // known_fact(!isnan(val))
5     __ASTREE_known_fact((-10.0f < val && val < 10.0f));
6     __ASTREE_known_fact((s->val == s->val));
7     __ASTREE_known_fact((FLT_MIN < s->val
8                          && s->val < FLT_MAX));
9     __ASTREE_assert((0.0f < s->alpha));
10    __ASTREE_assert((s->alpha < 1.0f));
11
12    float residual = val - s->val;
13    s->val = s->val + s->alpha * residual;
14
15    __ASTREE_assert((s->val == s->val));
16    // ...
17    return s->val;
18 }
```



```
__ASTREE_modify((V1, ..., Vn[;effect]))
```

- Modelliert Veränderung der Variablen V1 bis Vn

→ Braucht man um Stubs zu bauen

- Beispiele
  - Emulation von Sensoren
  - Beschreibung des Verhaltens von Bibliotheksfunktionen
- Kein *effect* → kompletter Wertebereich (inklusive NaN, +/-Inf)

## Beispiel

```
1 #ifdef __ASTREE__  
2 __ASTREE_modify((x; full_range)); // alles außer NaN, +/-Inf  
3 __ASTREE_modify((x; [10, 20])) // Einschränkung auf Intervall  
4 #else  
5 // ... Implementierung  
6 #endif
```



# Schleifen ausrollen

- Astrée beschreibt abstrakte Semantik
- Frage: Wie viele Schleifendurchgänge betrachten?
- ↳ Astrée versucht Aufwand zu vermeiden
- ↳ Schleifen werden (standardmäßig) einmal ausgerollt
- Konsequenz:

## Beispiel

```
1 unsigned int i = 0;  
2 unsigned int j = 20;  
3 while (j > 0) { --j; ++i; }
```

- Astrée kann nicht zeigen, daß die Schleife terminiert (☞ Satz von Rice)
- ↳ Annahme für weitere Analyse: i läuft über

## Lösung:

```
1 __ASTREE_unroll((30))  
2 while (j > 0) { --j; ++i; }
```

# Verzweigungen

- Dito bei Verzweigungen
- Astrée betrachtet normalerweise nur den schlimmsten Fall aller Zweige
- ↪ Pessimistisches Ergebnis
- Falls Betrachtung der unterschiedlichen Pfade erforderlich:
- Lösung: Analyse vorübergehend aufspalten:

## Verzweigungsanalyse

```
1 __ASTREE_partition_control
2 if (...) { ... }
3 else { ... }
4 ...
5 __ASTREE_partition_merge_last();
```

- Auch für Schleifen, `switch` und Funktionsaufrufe
- `__ASTREE_partition_merge` verschmilzt *alle* Partitionen
- ↪ Blick ins Handbuch: es gibt noch weitere Tricks



## `__ASTREE_volatile_input((V))`

- Zeigt an, dass V sich jederzeit ändern kann

↪ Modelliert Eingabe von außen

## `__ASTREE_volatile_input((Vp, r))`

- p ist Pfad in der Variablen,  
z. B. `V.a[3-4].b` ↪ Variable V, Arrayelemente `a[3]` und `a[4]`,  
Struct-Element b
  - `[i]` ↪ Element i
  - `[i-j]` ↪ Elemente i bis j
  - `[]` ↪ alle Elemente
- r schränkt Wertebereich ein `[i, j]` ↪ von i bis j



- Viele Echtzeitsysteme endlosschleifenbasiert
- Allerdings durch andere Umstände begrenzt
- 1kHz Ausführungsfrequenz, Neustart nach 7 Tagen  $\leadsto$  604,800,000 Ticks
- `__ASTREE_max_clock((N))` legt Obergrenze für (virtuelle) Clock
- `__ASTREE_wait_for_clock(( ))` wartet auf nächsten Clock-Tick

```
1 __ASTREE_max_clock((10)); // maximale Anzahl Clock-Ticks
2 void main(void) {
3     int state_log = 0;
4     while (1) {
5         state_log = increment_state(state_log);
6         __ASTREE_wait_for_clock(( )); // auf naechsten Clock-Tick warten
7     }
8 }
```

⇒ Wert von `state_log` begrenzt



- Astrée modelliert auch asynchrone Ausführung von Aufgaben
- Keine Annahmen über Scheduler oder Prioritäten
- `automatic-shared-variables` muss auf `yes` stehen

```
1 int x, y;
2 volatile int s;
3
4 void t1(void) {
5     x = 1; s = 1; x = 0;
6 }
7
8 void t2(void) {
9     if (s > 0) {
10        y = -1;
11    } else {
12        y = 1;
13    }
14 }
15
16 void main(void) {
17     x = y; // synchroner Teil
18     __ASTREE_asynchronous_loop((t1(), t2()));
19 }
```





```
__ASTREE_analysis_log(()
```

- Gibt Zustand der Analyse an dieser Stelle aus

```
__ASTREE_log_vars((V1, ..., Vn))
```

- Zeigt Zustand der Analyse in Bezug auf einzelne Variablen an

```
__ASTREE_print("text")
```

- Gibt Text aus



# Analyse untersuchen

The screenshot displays the Astrée IDE interface. The top menu bar includes Project, Analysis, Editors, Edit, Tools, and Help. The toolbar contains icons for file operations and execution. The left sidebar shows a project tree with folders for Configuration (Preprocessor, Parser, Analyzer, Annotations), Results (Overview, Call graph, Reports), and Files (Preprocessed, Original). The main window is split into three panes:

- Top Left:** Analyzed file: `...p/07_Astree/src/main.c *`. It shows the source code for `main(void)` with line numbers 346 to 358. The code includes `ASTREE_modify`, `ASTREE_log_vars`, and `ASTREE_assert` calls.
- Top Right:** Original source: `...07_Astree/src/main.c`. It shows the original source code with line numbers 1 to 12, including `#include` directives for `ab_filter.h`, `sensor.h`, `stdio.h`, `assert.h`, `astree.h`, and `bndbuf.h`.
- Bottom:** A detailed analysis report for line 357, column 1. It includes warnings such as `ASTREE_alarm((raise_at_caller;check_stdlib_limits));` and `ASTREE_assert((i > 10 && i < 100));`. It also shows domain and guard information, such as `Domains: Pointers, and Guard domain, and Packed (Octagons), and High_passband_domain(10), and Sec`. The report concludes with `Executing <main> /*` and provides a call stack entry: `[ call#main@348 at main.c:348.0-396.1`.

At the bottom left, the 'Errors' section shows 0 errors.

```
__ASTREE_analysis_log(()
```

- Gibt Zustand der Analyse an dieser Stelle aus

```
__ASTREE_log_vars((V1, ..., Vn))
```

- Zeigt Zustand der Analyse in Bezug auf einzelne Variablen an

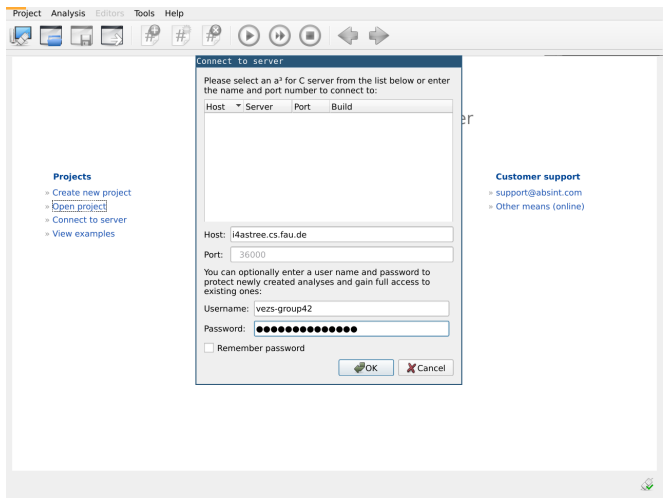
```
__ASTREE_print("text")
```

- Gibt Text aus



- Astrée im CIP:  
% /proj/i4ezs/tools/astree\_c/bin/a3c
- Anmeldung mit Benutzername und Passwort  
↪ Passwort wird bei der ersten Anmeldung festgelegt
- Dokumentation
  - PDF: /proj/i4ezs/tools/astree\_c/share/a3\_c/help/a3c.pdf





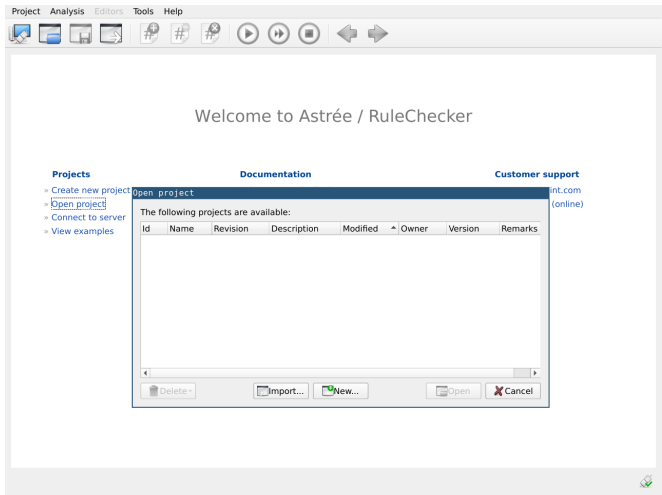
Host i4astree.cs.fau.de

Benutzer vezs-groupXX

Port 36000

Passwort Beliebig wählbar

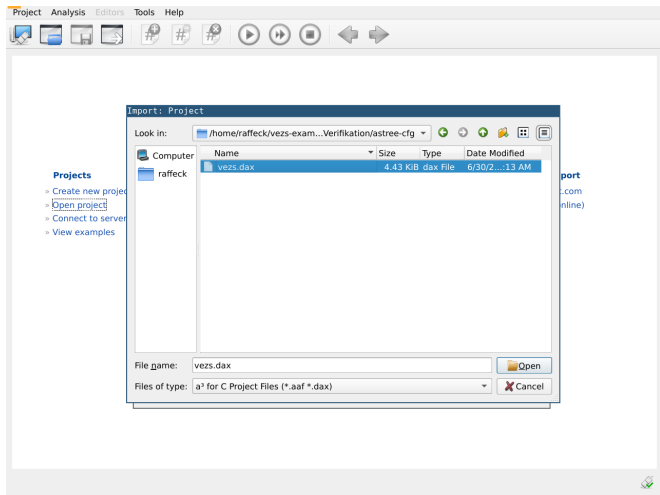
# Projekt anlegen



- „Import...“
- Projekt aus Vorgabedatei `astree-cfg/vezs.dax` importieren



# Projekt anlegen



■ „Import...“

■ Projekt aus Vorgabedatei `astree-cfg/vezs.dax` importieren



# Quelldateien konfigurieren

Project Analysis Editors Tools Help

Preprocessor

Use the built-in preprocessor

Operating system: None

Sources

Insert files for preprocessing:

Default configuration

Base directory: /home/raffack/vezs-example

Language: C

Include paths:

add files

Macro definitions:

\_\_ASTREE\_\_

Automatic includes:

Preprocess Find suitable includes

Project Summary Resource Monitor

Errors: 0

Code locations with alarms:

Run-time errors: 0

Flow anomalies: 0

Rule violations: 0

Memory locations with alarms:

Data races: 0

Reached code: n/a

Duration: n/a

Output Findings Not reached Data flow Watch Search

## Quelltextdateien und Includepfade definieren





# Quelldateien konfigurieren

The screenshot displays the Astrée IDE interface. A dialog box titled "Select files to add..." is open, showing a file selection process. The "Look in:" field is set to "/home/raffack/vezs-example/07\_Verifikation/src". A table lists the files in the directory:

Name	Size	Type	Date Modified
bndbuf.c	2.53 KiB	c File	6/30/2...13 AM
gaussian.c	7.75 KiB	c File	6/30/2...13 AM

The "File name:" field contains "gaussian.c", and the "Files of type:" dropdown is set to "C Source Files (\*.c)". The "Open" button is highlighted. In the background, the IDE's configuration panel is visible, with the "Preprocessor" section selected. The "Files" section shows "Preprocessed" and "Orig" tabs. The "Errors:" section shows 0 errors. The "Code locations with alarms:" section shows 0 run-time errors, 0 flow anomalies, and 0 rule violations. The "Memory locations with alarms:" section shows 0 data races. The "Reached code:" is n/a, and the "Duration:" is n/a. The "Automatic includes:" section is empty. The "Preprocess" button and "Find suitable includes" button are visible at the bottom right of the configuration panel.

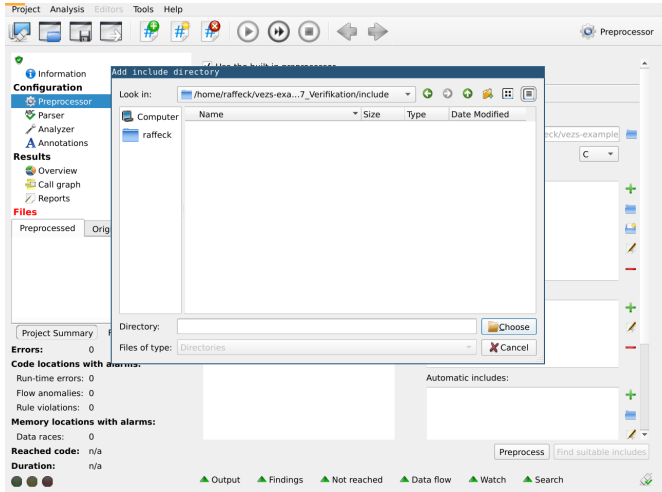
## ■ Quelltextdateien und Includepfade definieren

# Quelldateien konfigurieren

The screenshot displays the 'Preprocessor' configuration dialog. The 'Configuration' section is active, showing the 'Preprocessor' sub-section. The 'Use the built-in preprocessor' checkbox is checked. The 'Operating system' is set to 'None'. Under 'Sources', 'Default configuration' and 'gaussian.c' are listed. The 'Base directory' is set to '/home/raffeck/vezs-example' and the 'Language' is 'C'. The 'Include paths' section is empty, with a 'browse for include' button highlighted. The 'Macro definitions' section contains the text '\_\_\_ASTREE\_\_'. The 'Automatic includes' section is empty. The 'Preprocess' and 'Find suitable includes' buttons are at the bottom right. The status bar at the bottom shows icons for Output, Findings, Not reached, Data flow, Watch, and Search.

## ■ Quelltextdateien und Includepfade definieren

# Quelldateien konfigurieren



## ■ Quelltextdateien und Includepfade definieren



# Quelldateien konfigurieren

Project Analysis Editors Tools Help

Preprocessor

Use the built-in preprocessor

Operating system: None

Sources

Insert files for preprocessing:

- Default configuration
- gaussian.c

Base directory: /home/raffeck/vezs-example

Language: C

Include paths:

- /raffeck/vezs-example/07\_Verifikation/include

Macro definitions:

- \_ASTREE\_

Automatic includes:

Preprocess Find suitable includes

Project Summary Resource Monitor

Errors: 0

Code locations with alarms:

- Run-time errors: 0
- Flow anomalies: 0
- Rule violations: 0

Memory locations with alarms:

- Data races: 0

Reached code: n/a

Duration: n/a

Output Findings Not reached Data flow Watch Search

## Quelltextdateien und Includepfade definieren



# Analyse starten

Start analysis

Start analysis & Preprocess

The screenshot shows the Astrée IDE interface. The top toolbar contains several icons, with two red boxes highlighting the 'Start analysis' (play button) and 'Start analysis & Preprocess' (play button with a refresh symbol) icons. Below the toolbar, the 'Findings/C' window displays a table with 21 alarms and a pie chart titled 'Alarms (21 findings)'. The 'Output' window at the bottom shows the following text:

```
*** Starting postprocessing...
***
*** Postprocessing project with id 17 revision 1 terminated successfully on 2020/06/30 a
***
*** Analysis used not more than 380 MB (RSS 195 MB) of memory (total analysis time 0:28 m
***
/* Result summary */
Errors: 0
Code locations with alarms 19
Run-time errors: 19
Flow anomalies: 0
Rule violations: 1
Memory locations with alarms 0
Data races: 0
Reached code: 5 % (98/1944 statements reached, 84/98 (85 %) statements p
Duration: 28 s (28s)
```





AbsInt Angewandte Informatik GmbH.  
*The Static Analyzer Astrée*, April 2012.



Rolf Wanka.  
Sachen gibt's, die gibt's gar nicht.

