

# Verlässliche Echtzeitsysteme

## Übungen zur Vorlesung

Phillip Raffeck, Florian Schmaus, Simon Schuster

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

Sommersemester 2019





- TANs werden in der Übung verteilt
- $\geq 70\%$  Rückläuferquote  $\leadsto$  Belohnung: **Semesterabschlussgrillen**
- Termin der Evaluation ?



- 1 Abstrakte Interpretation mit Astrée
- 2 Formale Verifikation mit Frama-C
- 3 Aufgabenstellung

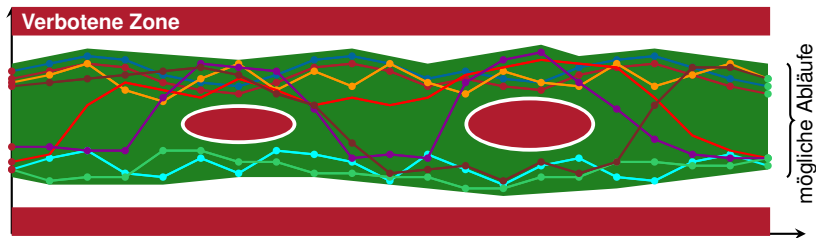


**1** Abstrakte Interpretation mit Astrée

2 Formale Verifikation mit Frama-C

3 Aufgabenstellung





- **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-positive*  
→ wegen **Gödelschem Unvollständigkeitstheorem** (1931)  
*„Jedes hinreichend mächtige formale System ist entweder widersprüchlich oder unvollständig.“*

## 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]))`  $\leadsto$  Wertebereich

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

- Analyser erzeugt Alarm, falls B *nicht immer wahr ist*
- B kann nicht von der Form `e1 ? e2 : e3` sein
- `__ASTREE_global_assert(( ))`  $\leadsto$  gesamtes Programm

- B muss seiteneffektfrei sein
- *Doppelte Klammerung ist wichtig!*



```
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))
```

- Zeigt an, dass Variablen V1 bis Vn unbekannten Wert haben

~> Braucht man um Stubs zu bauen

- Beispiel: Emulation von Sensoren

## Beispiel

```
1 #ifdef __ASTREE__
2 __ASTREE_modify((x; full_range));
3 __ASTREE_modify((x; [10, 20]))
4 __ASTREE_known_fact((x >= 0));
5 #else
6 // ... Implementierung
7 #endif
```



# Schleifen ausrollen

- Astrée beschreibt abstrakte Semantik
- Frage: Wie viele Schleifendurchgänge betrachten?
- ~> Astrée versucht Aufwand zu vermeiden
- ~> Schleifen/Verzweigungen werden nicht 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, 1953)
- ~> Annahme für weitere Analyse: i läuft über

## Lösung:

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



- 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();
```

- Auch für Schleifen, `switch` und Funktionsaufrufe
- Blick ins Handbuch: es gibt noch weitere Tricks



- Viele Echtzeitsysteme endlosschleifenbasiert ☹

→ Astrée modelliert dies

```
__ASTREE_max_clock((N))
```

beschränkt Schleifendurchläufe

```
__ASTREE_wait_for_clock(() )
```

wartet auf nächsten Tick

```
1 __ASTREE_max_clock((10)); // sonst evt. keine Schleifeninvariante
2 void main(void) {
3     while (1) {
4         // 1. 'volatile'-Werte lesen
5         // 2. Zustand und Ausgabe berechnen
6         // 3. Ausgabe schreiben
7         __ASTREE_wait_for_clock(() ); // auf naechsten Clock-Tick warten
8     }
9 }
```



- 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_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



```
__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 with the following components:

- Project Analysis Editors Edit Tools Help** menu bar.
- Left Panel:**
  - Welcome** section.
  - Configuration** section with icons for Preprocessor, Parser, Analyzer, and Annotations.
  - Results** section with icons for Overview, Call graph, and Reports.
  - Files** section with tabs for **Preprocessed** and **Original**. The **Preprocessed** tab is active, showing a file tree with `vezs19.cfg`, `ab_filter.c`, `bndbuf.c`, `clib.c`, `main.c *` (selected), `main`, and `sensor.c`.
- Main Editor:**
  - Top Bar:** **Analyzed file: /...p/07\_Astree/src/main.c \*** and **Original source: ...07\_Astree/src/main.c**.
  - Code Editor:** Displays the C source code for `main.c`. The **Original source** tab is active, showing lines 1 through 12. The **Analyzed file** tab shows lines 346 through 358. The code includes headers like `ab_filter.h`, `sensor.h`, `stdio.h`, `assert.h`, and `astree.h`. It defines `EXTENDED` and includes `bndbuf.h`. The `main` function is defined, starting with `int i = 0;` and a loop `while (i < 100) { ... }`. The `ASTREE_assert` macro is used to check `i > 10 && i < 100`.
  - Bottom Panel:** Shows the analysis results for `main.c`. It includes a summary of the analysis, such as `ASTREE_alarm((raise_at caller;check_stdlib_limits));` and `ASTREE_alarm((raise_at caller;check_stdlib_limits));`. It also displays the `Domains: Pointers, and Guard domain, and Packed (Octagons), and High_passband_domain(10), and Sec` and the `Executing <main> /*` section. The `Errors` section shows 0 errors.

- Astrée im CIP:

- % /proj/i4ezs/tools/astree\_c/bin/astreec

- Anmeldung mit Benutzername und Passwort

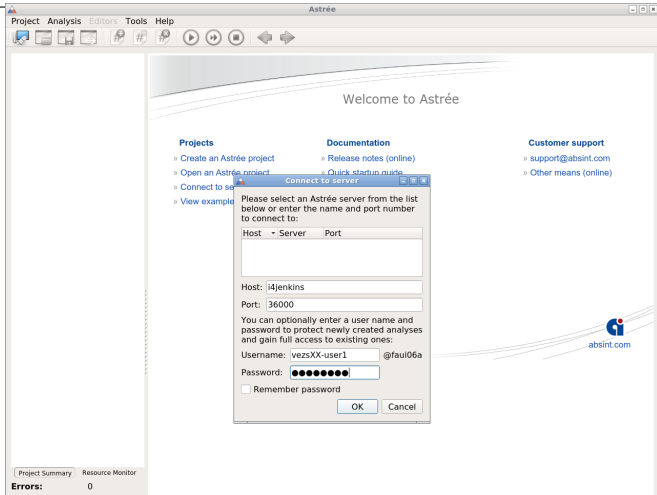
- ~> Passwort wird bei der ersten Anmeldung festgelegt

- Dokumentation

- HTML: /proj/i4ezs/tools/astree\_c/share/astree\_c/share/html/index.html

- PDF: /proj/i4ezs/tools/astree\_c/help/astreec\_QuickStart.pdf





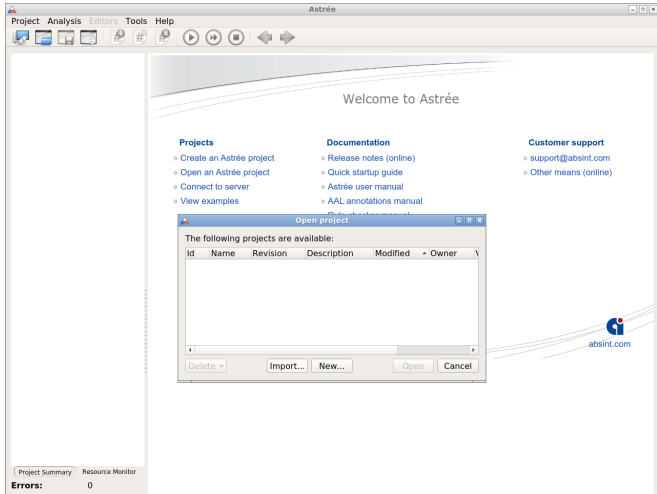
Host i4jenkins.cs.fau.de

Port 36000

Beliebigen Benutzernamen/Passwort wählen.



# Projekt anlegen

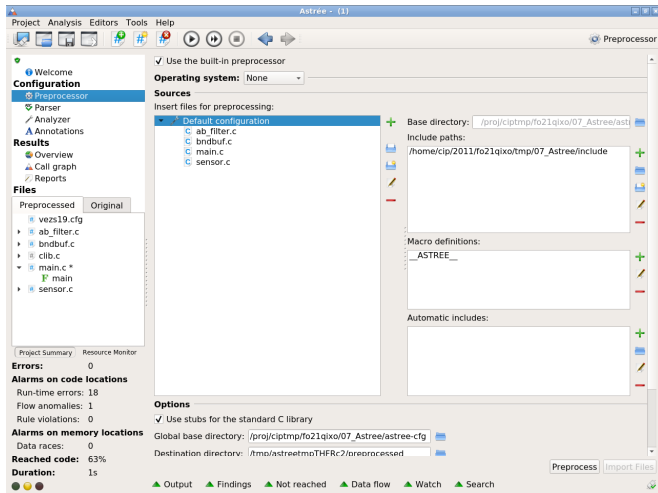


„Import...“

Projekt aus Vorgabedatei `astree-cfg/vezs19.dax` importieren



# Quelldateien konfigurieren



■ Quelltextdateien und Includepfade definieren

■ Haken bei „Empty destination directory before preprocessing“ setzen

Start analysis

Start analysis & Preprocess

The screenshot shows the Astrée IDE interface with the following components:

- Project Analysis Editors Tools Help** menu bar.
- Project Analysis Editors Tools Help** toolbar.
- Project Analysis Editors Tools Help** sidebar with tabs: Overview, Configuration, Parser, Analyzer, Annotations, Results, Files.
- Configuration** tab: Overview, Configuration, Parser, Analyzer, Annotations, Results, Files.
- Results** tab: Overview, Configuration, Parser, Analyzer, Annotations, Results, Files.
- Files** tab: Overview, Configuration, Parser, Analyzer, Annotations, Results, Files.
- Findings** tab: Overview, Configuration, Parser, Analyzer, Annotations, Results, Files.
- Findings** table:

| Count | Name                   |
|-------|------------------------|
| 12    | Alarms (12 findings)   |
| 10    | Overflow in conversion |
| 10    | Overflow in arithmetic |

- Alarms (12 findings)** pie chart showing 100%.
- Errors, alarms, notes, and info** table:

| File  | Line | Column | Severity | Message                   |
|-------|------|--------|----------|---------------------------|
| atanh | 4    | 0      | A        | % at % Filter.c:129:6-12  |
| atanh | 4    | 0      | A        | % at % Filter.c:135:7-12  |
| atanh | 4    | 0      | A        | % at % Filter.c:185:12-18 |
| atanh | 4    | 0      | A        | % at % Filter.c:187:12-17 |
| atanh | 4    | 0      | A        | % at % Filter.c:118:12-18 |
| atanh | 4    | 0      | A        | % at % Filter.c:117:6-12  |
| atanh | 4    | 0      | A        | % at % Filter.c:116:7-12  |
| atanh | 4    | 0      | A        | % at % Filter.c:184:12-17 |
| atanh | 4    | 0      | A        | % at % Filter.c:115:12-18 |

- Result summary** section:

  - Errors: 0
  - Alarms on code locations: 12
  - Run-time errors: 0
  - Flow anomalies: 0
  - Rule violations: 0
  - Alarms on memory locations: 0
  - Reached code: 10 % (123/1184 statements reached, 116/123 (95 %) statements proven)
  - Duration: 1.21 s (0 h 0 m 1 s)

- Summary** section:

  - Errors: 0
  - Alarms on code locations: 12
  - Run-time errors: 0
  - Flow anomalies: 0
  - Rule violations: 0
  - Alarms on memory locations: 0
  - Data races: 0
  - Reached code: 98%
  - Duration: 1s

- Output** section:

  - Analysis of project vezs-gruppe-42 (s/a) with id 1578966432 revision 1 terminated successfully on 2018/06/28 at 17:12:41
  - Starting postprocessing...
  - Postprocessing project vezs-gruppe-42 (s/a) with id 1578966432 revision 1 terminated successfully on 2018/06/28 at 17:12:42
  - Analysis used not more than 101 MB (RSS 65 MB) of memory (total analysis time 0:01 minutes)



1 Abstrakte Interpretation mit Astrée

**2 Formale Verifikation mit Frama-C**

3 Aufgabenstellung





- ## Formale Verifikation mit Frama-C



- Bestimmt die **schwächste notwendige Vorbedingung**  $wp(S, Q)$ 
  - Für ein gegebenes **imperatives Programmsegment**  $S$
  - Um die ebenfalls gegebene Nachbedingung  $Q$  sicherzustellen
  - Dieser Sachverhalt wird beschrieben durch:  $P \Rightarrow wp(S, Q)$ 
    - Lässt sich die schwächste notwendige Vorbedingung  $wp(S, Q)$  aus der gegebenen Vorbedingung  $P$  folgern?
- Das WP-Kalkül ist eine **Rückwärtsanalyse**
  - Sie beginnt mit der Nachbedingung und durchläuft das Programmsegment in umgekehrter Reihenfolge
  - „Sozusagen“ umgekehrter Einsatz der Regeln des Hoare-Kalküls
- Jeder Anweisung wird eine **Prädikattransformation** zugewiesen
  - Abbildung: Nachbedingung  $\mapsto$  notwendige schwächste Vorbedingung
  - Eine rückwärtige **symbolische Ausführung** des Programmsegments
- Frama-C setzt WP-Kalkül ein: Nachweis der Schritte erfolgt über verschiedene Theorembeweiser und Löser: Alt-Ergo, Coq, Why3, Z3, ...



# Beispiel: WP-Kalkül Maximumsfunktion

WP:

?  $(a > b \Rightarrow a \geq b \wedge a \geq a) \wedge (\neg(a > b) \Rightarrow b \geq b \wedge b \geq a)$

$\Leftrightarrow (a > b \Rightarrow a \geq b) \wedge (a \leq b \Rightarrow b \geq a)$

$\Leftrightarrow \top$  // Gilt ohne Vorbedingung

if (a > b)

$a > b \Rightarrow a \geq b \wedge a \geq a$

result = a;

$a > b \Rightarrow \text{result} \geq b \wedge \text{result} \geq a$

else

$\neg(a > b) \Rightarrow b \geq b \wedge b \geq a$

result = b;

$\neg(a > b) \Rightarrow \text{result} \geq b \wedge \text{result} \geq a$

$\text{result} \geq b \wedge \text{result} \geq a$

return result;

$\backslash \text{res} \geq b \wedge \backslash \text{res} \geq a$

Q:

$wp(\text{if } b \text{ then } S1 \text{ else } S2, Q) \equiv (b \Rightarrow wp(S1, Q)) \wedge (\neg b \Rightarrow wp(S2, Q))$

$wp(x = y, Q) \equiv Q[x/y]$

$wp(x = y, Q) \equiv Q[x/y]$

$wp(\text{return } x, Q) \equiv Q[\backslash \text{res}/x]$

Auswertungsreihenfolge



- erlaubt Nachweise funktionaler Eigenschaften mittels wp-Kalkül
  - Prädiaktenlogik erster Stufe
    - $\forall$ : `\forallall`
    - $\exists$ : `\existsexists`
    - $\Rightarrow$ : Implikation, `==>`
    - Die aus C bekannten aussagenlogischen Verknüpfungen: `!`, `&&`, `|`, `==`, `!=`, ...
- Vor-/Nachbedingungen als Kommentare direkt im C-Code vor der betreffenden Funktion
  - `//@` oder `/*@ */`
- Vorbedingung: `requires`
- Nachbedingung: `ensures`
- Variablen, die durch die Funktion verändert werden: `assigns`
  - inklusive Arraybereiche: `a[start..end]`
  - Spezialfall `\nothing`
- Ergebnis des Funktionsaufrufs: `\result`



# Beispiel: Maximumsfunktion

$P$ : wahr

```
S: int maximum(int a, int b) {  
    int result = INT_MIN;  
  
    if(a > b)  
        result = a;  
    else  
        result = b;  
  
    return INT_MAX;  
}
```

$Q$ :  $\text{result} \geq a \wedge \text{result} \geq b \wedge$   
 $(\text{result} == a \vee \text{result} == b)$

```
/*@  
    assigns \nothing;  
  
    ensures \result >= a;  
    ensures \result >= b;  
    ensures \result == a || \result == b;  
*/  
int maximum(int a, int b) {  
    int result = INT_MIN;  
    if (a > b) {  
        result = a;  
    } else {  
        result = b;  
    }  
    return result;  
}
```



```
// Swap array[0] with pointer other, do not modify the rest of array
/*@ requires \valid(array) && \valid(other);
    requires \base_addr(array) != \base_addr(other);

    assigns array[0], *other;

    ensures array[1 .. (length - 1)] == \old(array[1 .. (length - 1)]);
    ensures *other == \old(array[0]);
    ensures array[0] == \old(*other); */
void swap_first_element(int *array, size_t length, int *other) {
    int tmp = array[0];
    array[0] = *other;
    *other = tmp;
}
```

- Übergebene Zeiger sind gültig: `\valid(ptr)`
- Speicherobjekte hinter Zeigern überlappen sich nicht:  
`\base_addr(ptr1) != \base_addr(ptr2)`
- `assigns`: Nur bestimmte (außen sichtbare) Variablen werden verändert
- `ensures`: Zusammenhänge zwischen Werten vor (`\old(var)`) und nach (`var`) der Ausführung anfordern



Gesucht:  $wp(\text{while } (B) \ S; , Q)$

- Wann ist die Berechnung korrekt?

- Stellt  $Q$  her  $\leadsto$  Schleifen**invariante**
- Terminiert  $\leadsto$  Schleifen**variante**

- Schleifenvariante: Streng monoton fallender, *nichtnegativer* Ausdruck

In Frama-C: `//@ loop variant length - i;`

- Schleifeninvariante: Ausdruck, der vor der Schleife als nach jedem Durchlauf des Schleifenkörpers  $B$  gilt

In Frama-C: `//@ loop invariant i % 2 == 0;`

- Ferner: Schleife überschreibt nur bestimmte Werte

In Frama-C: `//@ loop assign i, j, *ptr;`

- Beispiel:

```
int i = 0;
```

```
while(i < 30) { i++; }
```

■ Schleifenvariante:  $30 - i \geq 0$

■ Schleifeninvariante:  $0 \leq i \leq 30$

Die Schleifenvariante gilt vor, während und nach der Schleife

Nach der Schleife ist die Schleifenbedingung falsch

$\leadsto \neg(i < 30) \wedge (0 \leq i \leq 30) \leadsto i == 30$





# Beispiel: Maximum in Liste finden

$P: a \neq \text{NULL} \wedge \text{length} > 0$

```
S: int findMax(int *a, size_t length) {  
    int max = a[0];  
    for (size_t i = 0; i < length; i++) {  
        if (a[i] > max) {  
            max = a[i];  
        }  
    }  
}
```

$Q: \forall k. k \geq 0 \wedge k < \text{length}$   
 $\Rightarrow a[k] \leq \text{result}$   
 $\exists k. k \geq 0 \wedge k < \text{length}$   
 $\Rightarrow a[k] == \text{result}$

```
/*@ requires length > 0;  
    requires \valid(a + (0..length-1));  
    assigns \nothing;  
    ensures \forall int k;  
        0 <= k < length ==> \result >= a[k];  
    ensures \exists int k;  
        0 <= k < length && \result == a[k];  
*/  
int findMax(int *a, int length) {  
    int max = a[0];  
    int max_idx = 0;  
    /*@ loop invariant \forall int k;  
        0 <= k < i ==> max >= a[k];  
        loop invariant a[max_idx] == max;  
        loop invariant 0 <= i <= length;  
        loop invariant 0 <= max_idx < length;  
        loop assigns i, max_idx, max;  
*/  
    for (int i = 0; i < length; i++) {  
        if (a[i] > max) {  
            max = a[i];  
            max_idx = i;  
        }  
    }  
    return max;  
}
```



- „Wiederverwendbare“ aussagenlogische Formulierung

→ logische Prädikate: predicate

- „Funktionen“ auf ACSL-Ebene, erleichtern Formulierung von Bedingungen

- Beispiel: Maximum einer Sequenz

```
//@ predicate is_max_of_seq(int max, int *seq, int start, int end) =  
    \forall int i; start <= i < end ==> max >= seq[i];  
/*@ ...  
    ensures is_max_of_seq(\result, a, (int)0, length);  
    ...  
*/  
int findMax(int *a, int length) {  
    ...  
    /*@ loop invariant is_max_of_seq(max, a, (int)0, i);  
        ...  
    */  
    for (int i = 0; i < length; i++) {  
        ...  
    }  
    return max;  
}
```



- Invarianten über der Datenstruktur definieren

- Jede Methode:

- Setzt Gültigkeit der Invarianten voraus
- Erhält Sie nach Ausführung

→ Bei Kapselung: Korrektheit der Datenstruktur sichergestellt

- Beispiel: Konto

```
struct account {  
    int balance;  
    int credit_limit;  
};  
  
/*@ requires \valid(a);  
    ensures valid_account(a); */  
void init(struct account *a) {  
    a->balance = 0; a->credit_limit = 0;  
}
```

- Invariante: Kreditrahmen wird nie verletzt:

```
/*@ predicate valid_account(struct account *a) =  
    \valid(a) // Gültiger Zeiger  
    && a->balance >= a->credit_limit; // Kreditrahmen nicht verletzt  
*/
```

- Alle Methoden erfordern und erhalten die Invariante:

```
/*@ requires valid_account(a);  
    requires amount > 0;  
    ensures valid_account(a); */  
bool withdraw(struct account *a, int amount) {  
    /* ... */  
    return true;  
}  
  
/*@ requires valid_account(a);  
    requires amount > 0;  
    ensures valid_account(a); */  
void deposit(struct account *a, int amount) {  
    /* ... */  
}
```

→ Solange das Konto nur per `withdraw` und `deposit` modifiziert wird, kann der Kreditrahmen nie verletzt werden



- Oft gelten Nachbedingungen nur für bestimmte Eingabewerte

```
/*@ requires i != INT_MIN;  
    ensures i < 0 ==> \result == -i;  
    ensures i >= 0 ==> \result == i; */  
int abs(int i) {  
    if (i >= 0) return i;  
    else      return -i;  
}
```

- Behaviors beschreiben Verhalten in bestimmten Kontexten:

- assumes: Voraussetzungen, die das Verhalten aktiviert
- ensures: Nachbedingung, die die Funktion dann einhält

*Eingabe ist negativ*

behavior negative:

```
assumes i < 0;  
ensures \result == -i;
```

*Eingabe ist positiv*

behavior positive:

```
assumes i >= 0;  
ensures \result == i;
```

- complete behaviors: Die Beschreibung ist vollständig  
Behaviors beschreiben das Verhalten für alle Aufrufkontexte
- disjoint behaviors: Die beschriebenen Verhalten überlappen sich nicht



```
#include <limits.h>

/*@ requires i != INT_MIN;
    behavior negative:
        assumes i < 0;
        ensures \result == -i;
    behavior positive:
        assumes i >= 0;
        ensures \result == i;
    complete behaviors;
    disjoint behaviors; */
int abs(int i) {
    if (i >= 0) return i;
    else      return -i;
}
```



- Die Frama-C-Gui bietet keinen Editor an!
- Reihenfolge der Annotationen z.T. relevant. Empfehlung:

## *Funktionen*

- requires
- assigns
- ensures

## *Schleifen*

- loop invariants
- loop assigns
- loop variants

- `assert()` wird als nicht terminierend angenommen

~> Frama-C assert verwenden: `//@ assert x == 1;`

- Mehrere Annotationen immer in einen gemeinsamen Block `/*@ */`
- Weitere Informationen:

- Fraunhofer Fokus: ACSL-By-Example:

<https://github.com/fraunhoferfokus/acsl-by-example>

- A. Blanchard: Introduction to C program proof with Frama-C and its WP plugin:

<https://allan-blanchard.fr/publis/frama-c-wp-tutorial-en.pdf>



1 Abstrakte Interpretation mit Astrée

2 Formale Verifikation mit Frama-C

**3 Aufgabenstellung**



- Sensorstubs implementieren
- Implementierung eines Ringpuffers zur Verwaltung der Sensorwerte
- System erstellen:
  - Sensoren und Filter initialisieren
  - Sensorwerte abrufen
  - Sensorwerte filtern
- Abwesenheit von Laufzeitfehlern nachweisen
  - Astrée
- Numerische Stabilität des Filters nachweisen
  - Astrée
- Funktionale Korrektheit des Ringpuffers nachweisen
  - Frama-C



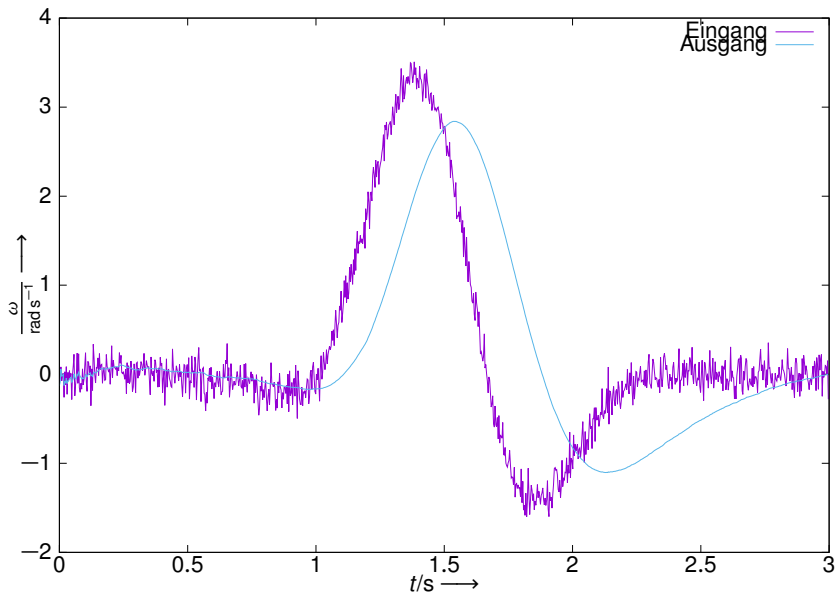




- Rauschunterdrückungsfilter
- Geeignet zur Schätzung von physikalischen Größen
  - mit Ableitung  $\neq 0$
  - z. B. Position eines Flugzeugs, Lagewinkel ...
  - im I4Copter
    - liefern Sensoren häufiger Werte als diese später verarbeitet werden
  - ~>  $\alpha$ - $\beta$ -Filter für *Ratenwandlung* verwendet
  - ~> nutzt gewonnene Information vollständig



# Beispiel $\alpha$ - $\beta$ -Filterung



- Wird für jeden Messwert ausgeführt
- $y[\kappa]$ : Eingabewert für Abtastschritt  $\kappa$
- $\hat{x}[\kappa]$ : Schätzung der Messgröße zum Abtastschritt  $\kappa$
- $T$  Abtastintervall,  $\alpha$ ,  $\beta$  Filterparameter
- Initialisierung: z. B.  $\hat{x}[0] = \dot{\hat{x}}[0] = 0$

$$r[\kappa] = y[\kappa] - \hat{x}[\kappa - 1] \quad \leadsto \text{Schätzfehler} \quad (1)$$

$$\dot{\hat{x}}[\kappa] = \dot{\hat{x}}[\kappa - 1] + \frac{\beta}{T} \cdot r[\kappa] \quad \leadsto \text{1. Ableitung} \quad (2)$$

$$\hat{x}[\kappa] = \hat{x}[\kappa - 1] + T \cdot \dot{\hat{x}}[\kappa] + \alpha \cdot r[\kappa] \quad \leadsto \text{Schätzwert} \quad (3)$$

- Sinnvolle Werte für  $\alpha$  und  $\beta$ ?
  - Literatur beschreibt viele Verfahren  $\leadsto$  hier beispielhaft nur eines [6, 5]



$T$  Abtastintervall

$\leadsto$  in welchem Zeitabstand gemessen wird

$\sigma_w^2$  Prozessvarianz

$\leadsto$  wie lebhaft der gemessene Prozess ist

$\sigma_v^2$  Rauschvarianz

$\leadsto$  wie verrauscht das Signal ist

$$\lambda = \frac{\sigma_w T^2}{\sigma_v} \quad \leadsto \text{Tracking Index} \quad (4)$$

$$\theta = \frac{4 + \lambda - \sqrt{8\lambda + \lambda^2}}{4} \quad \leadsto \text{Dämpfungsparameter} \quad (5)$$

$$\alpha = 1 - \theta^2 \quad \leadsto \text{Gewicht für Wert} \quad (6)$$

$$\beta = 2(2 - \alpha) - 4\sqrt{1 - \alpha} \quad \leadsto \text{Gewicht für Ableitung} \quad (7)$$



- Zu Beginn hat das Filter keinen gültigen Zustand
- Einschwingphase, in der das Filter „lernt“
- $n$  startet bei 1 und wächst in jeder Runde um 1

$$\alpha_n = \frac{2(2n+1)}{(n+2)(n+1)} \quad (8)$$

$$\beta_n = \frac{6}{(n+2)(n+1)} \quad (9)$$

- Einschwingphase endet, wenn  $\alpha_n < \alpha$
- Wird der Filterzustand im Betrieb ungültig, wird eine neue Einschwingphase eingeleitet



- Filter ist nur dann korrekt, wenn es auch *stabil* ist
  - ↪ für wertbegrenzte Eingabe erfolgt wertbegrenzte Ausgabe [7]
  - ↪ für Eingabe 0 geht der Filterausgang asymptotisch gegen 0
- $\alpha$ - $\beta$ -Filter stabil, wenn gilt

$$0 < \alpha \leq 1 \quad (10)$$

$$0 < \beta \leq 2 \quad (11)$$

$$0 < 4 - 2\alpha - \beta \quad (12)$$

- Außerdem: laut [2] Rauschunterdrückung nur dann, wenn

$$0 < \beta < 1 \quad (13)$$

- Stabilität muss auch in der Initialisierungsphase gegeben sein!



- Erfahrungen mit dem I4Copter haben gezeigt, dass sich die Parameter in folgenden Bereichen bewegen:

Abtastintervall  $T \in (0 \dots 1]$

Prozessvarianz  $\sigma_w^2 \in [0.5 \dots 30.0]$

Rauschvarianz  $\sigma_v^2 \in [10^{-3} \dots 10^{-1}]$

Wert  $y[k] \in [-10 \dots 10]$

→ Korrektheit mindestens für diese Wertebereiche zeigen!



## ■ Beispiel für dynamisch angelegten Ringpuffer

```
1 struct BndBuf{  
2     int* buf; // array of <size>  
3     int size;  
4     int read_pos;  
5     int write_pos;  
6 };
```

## ■ „Füllstandsanzeige”: Speichern von

- Belegten Plätzen
- Verfügbaren Plätzen

## ■ Mögliche Signalisierung falls

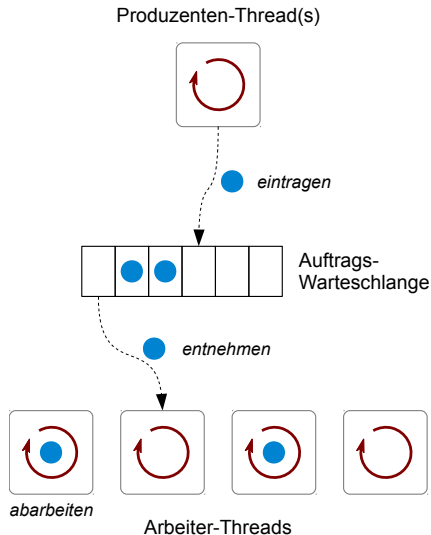
- Plätze frei wurden
- Plätze belegt wurde





# Ringpuffer – Anwendungsszenario

- Verwaltung einer begrenzten Anzahl an Ressourcen
- Beispiel: Thread-Pool
- Feste Menge von Arbeiter-Threads:
  - laufen endlos
  - erhalten Aufträge zur Abarbeitung
- Verteilen der Aufträge mittels zentraler, synchronisierter Warteschlange (z. B. Ringpuffer)
- Vorteil: kein ständiges Erzeugen + Zerstören von Threads für Aufträge





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



C. Frank Asquith.  
Weight selection in first-order linear filters.  
Technical report, Army Intertial Guidance and Control Laboratory Center, Redstone Arsenal, Alabama, 1969.



Eli Brookner.  
*Tracking and Kalman Filtering Made Easy*.  
Wiley-Interscience, 1st edition, 4 1998.



Edsger W. Dijkstra.  
Guarded commands, nondeterminacy and formal derivation of programs.  
*Communications of the ACM*, 18(8):453–457, August 1975.



E. Gray, J. and W. Murray.  
A derivation of an analytic expression for the tracking index for the alpha-beta-gamma filter.  
*IEEE Trans. on Aerospace and Electronic Systems*, 29:1064–1065, 1993.



Paul R. Kalata.  
The tracking index: A generalized parameter for  $\alpha$ - $\beta$  and  $\alpha$ - $\beta$ - $\gamma$  target trackers.  
*IEEE Transactions on Aerospace and Electronic Systems*, AES-20(2):174–181, mar 1984.





Richard G. Lyons.

*Understanding Digital Signal Processing.*

Prentice Hall, 3rd edition, 11 2010.

