

Verlässliche Echtzeitsysteme

Übungen zur Vorlesung

Florian Franzmann, Tobias Klaus

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

2. Juni 2015



Überblick

- 1 C-Quiz Teil III
- 2 Abstrakte Interpretation mit Astrée
- 3 Aufgabenstellung
- 4 Hinweise zur Implementierung



Annahmen

- C99
- x86 bzw. x86-64, d. h.
 - vorzeichenbehaftete Integer als Zweierkomplement implementiert
 - char hat 8 Bit
 - short hat 16 Bit
 - int hat 32 Bit
 - long hat 32 Bit auf x86 und 64 Bit auf x86-64



Frage 7

Zu was wird `INT_MAX + 1` ausgewertet?

1. 0
2. 1
3. `INT_MAX`
4. `UINT_MAX`
5. nicht definiert

Erklärung

`signed int`-Überlauf ist nicht definiert.



Frage 8

Zu was wird `-INT_MIN` ausgewertet?

1. 0
2. 1
3. `INT_MAX`
4. `UINT_MAX`
5. `INT_MIN`
6. nicht definiert

Erklärung

Es gibt keine Zweierkomplementdarstellung für `-INT_MIN`



Frage 9

Angenommen `x` hat Typ `int`. Ist `x << 0 ...`

1. definiert für alle Werte
 2. definiert für manche Werte
 3. definiert für keinen Wert
- von `x`?

Erklärung

- negative Werte können nicht nach links verschoben werden
- noch nicht einmal um 0 Bit



Astrée [1]

- Ziel: Nachweis der Abwesenheit von Laufzeitfehlern
- findet *alle* potentiellen Laufzeitfehler
- leider auch *falsch-positiv*
~> 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



Astrée weist nach

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



Einschränkungen

- Rekursionen prinzipiell erlaubt, werden 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
↪ Stubs anlegen
- dynamischer Speicher nicht erlaubt
↪ kein malloc()
 - keine Einschränkung im sicherheitskritischen Echtzeitbereich



Semantik

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 warnt, falls B *nie wahr werden kann*

`assert((B))`

- alternativ auch `__ASTREE_assert((B))`
- Analyser erzeugt Alarm, falls B *nicht immer wahr ist*
- B kann nicht von der Form `e1 ? e2 : e3` sein

- Analyser nimmt danach an, dass B wahr ist
- B muss seiteneffektfrei sein
- *die doppelten Klammern sind 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));
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 }
```



Stubs

```
__ASTREE_modify((V1, ..., Vn))
```

- zeigt an, dass Variablen V1 bis Vn unbekanntes Wert haben
- ↳ braucht man um Stubs zu bauen
- z. B. um Sensoren zu emulieren

Beispiel

```
1 #ifdef __ASTREE__
2 __ASTREE_modify((x));
3 __ASTREE_known_fact((x >= 0));
4 #else
5 // ... Implementierung
6 #endif
```



Synchrone Programme

- 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 }
```



Asynchrone Programme

- 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 }
```



volatile

```
__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



Analyse untersuchen

`__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 verwenden

- Astrée im CIP:

% /proj/i4ezs/tools/astree_c-b240070/bin/astreec

- Anmeldung mit Benutzername und Passwort

↪ Passwort wird bei der ersten Anmeldung festgelegt

- Wir wissen nicht, wie man es nachträglich ändert

↪ gut merken ☹

- Dokumentation unter

/proj/i4ezs/tools/astree_c-b240070/share/astree_c/help



Anmelden

Host `fau48f.informatik.uni-erlangen.de`

Port `36000`

Username nach Belieben

Passwort bei der ersten Anmeldung festlegen und *gut merken*

Please select an Astrée server from the list below or enter the name and port number to connect to:

Host	Server	Port
------	--------	------

Host:

Port:

You can optionally enter a user name and password to protect newly created analyses and gain full access to existing ones:

Username: @fau48r0

Password:



Projekt anlegen

- Quelldateien zum Projekt hinzufügen

General
Specify the project name, the entry point of the analysis and the sources that you want to analyze.

Project name:

Analysis start:

Files to be used for the analysis:

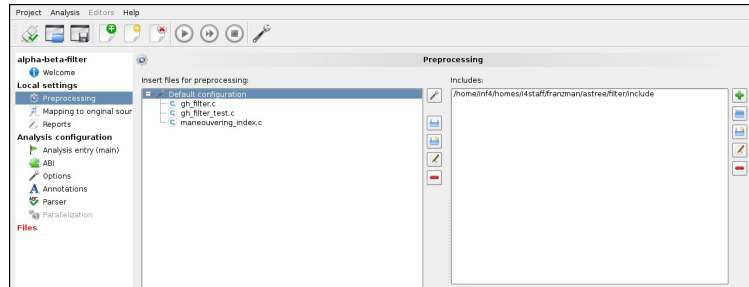
Add already preprocessed files

Add the source files and preprocess them with the built-in preprocessor



Präprozessoreinstellungen

■ Include-Pfade festlegen



ABI

■ ABI festlegen

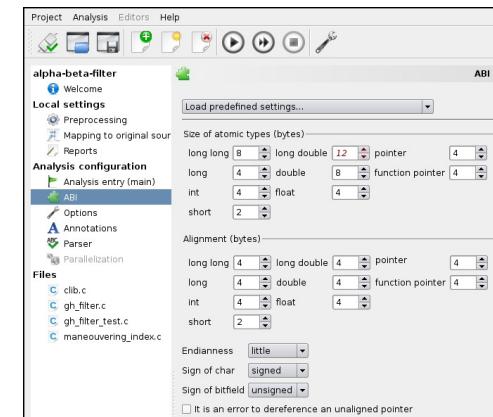


Table of Contents

- 1 C-Quiz Teil III
- 2 Abstrakte Interpretation mit Astrée
- 3 Aufgabenstellung
- 4 Hinweise zur Implementierung

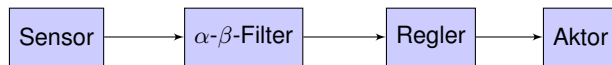


Aufgabenstellung

- Erweiterung eurer Warteschlange für Jobverwaltung
- Ersetzen nicht verifizierbarer Funktionen
 - Astrée kann kein malloc
- Sensorstubs implementieren
- einfaches Filter implementieren
- System erstellen:
 - Sensoren und Filter initialisieren
 - Arbeitsaufträge erstellen
 - Arbeitsaufträge in Warteschlange einfügen
 - Arbeitsaufträge nach Priorität ausführen
- Korrektheit der Implementierung nachweisen
 - Astrée



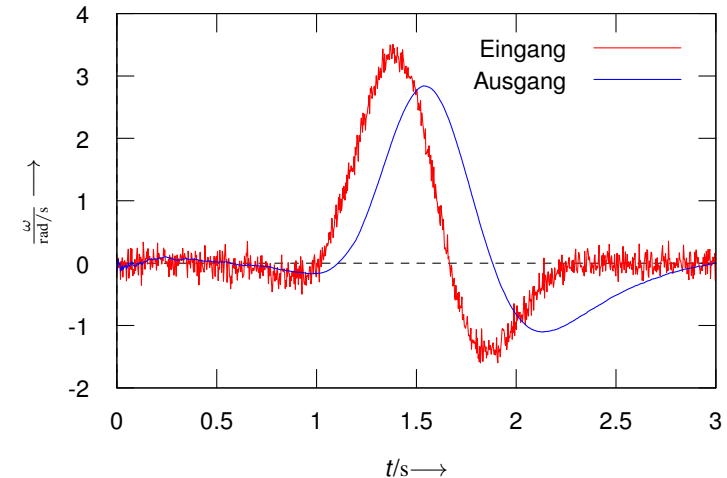
α - β -Filter [3]



- 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
 - ~ α - β -Filter für *Ratenwandlung* verwendet
 - ~ nutzt gewonnene Information vollständig



Beispiel α - β -Filterung



Filteralgorithmus

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

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

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

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

- sinnvolle Werte für α und β ?
 - Literatur beschreibt viele Verfahren ~ hier beispielhaft nur eines [5, 4]



Filterparameter

- T Abtastintervall
 - ~ in welchem Zeitabstand gemessen wird
- σ_w^2 Prozessvarianz
 - ~ wie lebhaft der gemessene Prozess ist
- σ_v^2 Rauschvarianz
 - ~ wie verrauscht das Signal ist

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

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

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

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



Initialisierungsphase

- 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



Korrektheitsbedingung

- Filter ist nur dann korrekt, wenn es auch *stabil* ist
- ↪ für wertbegrenzte Eingabe erfolgt wertbegrenzte Ausgabe [6]
- ↪ für Eingabe 0 geht der Filterausgang asymptotisch gegen 0

- α - β -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!



Sinnvolle Wertebereiche

- 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!



Literatur I

- [1] AbsInt Angewandte Informatik GmbH.
The Static Analyzer Astrée, April 2012.
- [2] C. Frank Asquith.
Weight selection in first-order linear filters.
Technical report, Army Inertial Guidance and Control Laboratory Center, Redstone Arsenal, Alabama, 1969.
- [3] Eli Brookner.
Tracking and Kalman Filtering Made Easy.
Wiley-Interscience, 1st edition, 4 1998.
- [4] 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.
- [5] Paul R. Kalata.
The tracking index: A generalized parameter for α - β and α - β - γ target trackers.
IEEE Transactions on Aerospace and Electronic Systems, AES-20(2):174–181, mar 1984.



- [6] Richard G. Lyons.
Understanding Digital Signal Processing.
Prentice Hall, 3rd edition, 11 2010.



- 1 C-Quiz Teil III
- 2 Abstrakte Interpretation mit Astrée
- 3 Aufgabenstellung
- 4 Hinweise zur Implementierung



Funktionszeiger

- Prioritätswarteschlange soll nun Jobs verwalten
- Nicht mehr Ganzzahlen, sondern Arbeitsaufträge
- C erlaubt es, Zeiger auf Funktionen anzulegen

```
1 typedef void(*Job)(void);
```

- Zeiger auf eine Funktion, die
 - keinen Rückgabewert hat
 - keine Parameter übernimmt

Verwendung

```
1 void job_function(void) { ...; }
2 void f_pointer_test(Job job) { job(); }
3
4 int main(void) {
5     f_pointer_test(job_function);
6     return 0;
7 }
```



Bitoperationen

- Einfacher Speicherallocator soll implementiert werden
- Freier Speicher soll durch eine Bitmaske verwaltet werden

~> notwendige Operationen

```
1 void set_bit(unsigned int *x, unsigned int bit_number) {
2     *x = *x | (1U << bit_number);
3 }
4
5 void reset_bit(unsigned int *x, unsigned int bit_number) {
6     *x = *x & ~(1U << bit_number);
7 }
8
9 bool is_set(unsigned int x, unsigned int bit_number) {
10     return 1U == ((x >> bit_number) & 1U);
11 }
```



Fragen?

