

# Verlässliche Echtzeitsysteme - Übungen

Abstrakte Interpretation & Verifikation

---

Wintersemester 2024

Eva Dengler, Peter Wägemann

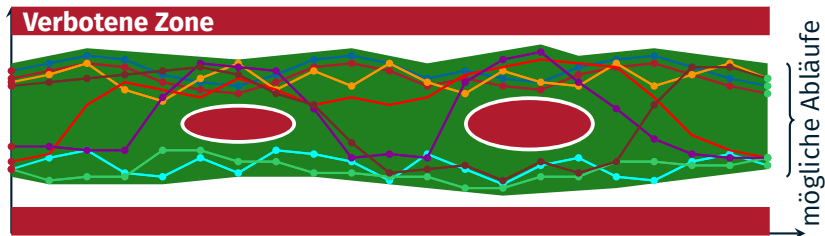
Friedrich-Alexander-Universität Erlangen-Nürnberg

Lehrstuhl Informatik 4 (Systemsoftware)

<https://sys.cs.fau.de>

- 1 Abstrakte Interpretation mit Astrée
- 2 Formale Verifikation mit Frama-C
- 3 Aufgabenstellung
- 4  $\alpha$ - $\beta$ -Filter

# Abstrakte Interpretation



- *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*  
    ↪ **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  
     $\rightsquigarrow$  `assert()`

# Einschränkungen

- 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

```
#include <astree.h> // Astree-Makros ggf. abschalten

float filter(Alpha_State *s, float val) {
    __ASTREE_known_fact((val == val)); // known_fact(!isnan(val))
    __ASTREE_known_fact((-10.0f < val && val < 10.0f));
    __ASTREE_known_fact((s->val == s->val));
    __ASTREE_known_fact((FLT_MIN < s->val
                        && s->val < FLT_MAX));
    __ASTREE_assert((0.0f < s->alpha));
    __ASTREE_assert((s->alpha < 1.0f));

    float residual = val - s->val;
    s->val = s->val + s->alpha * residual;

    __ASTREE_assert((s->val == s->val));
    // ...
    return s->val;
}
```

## \_\_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

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

# Schleifen ausrollen

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

## Beispiel

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

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

## Lösung:

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

# Verzweigungen

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

## Verzweigungsanalyse

```
__ASTREE_partition_control  
if (...) { ... }  
else { ... }  
...  
__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

# Exkurs: Beschränkte Laufzeit

- 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

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

⇒ Wert von `state_log` begrenzt

# Exkurs: Asynchrone Programme

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

```
int x, y;
volatile int s;

void t1(void) {
    x = 1; s = 1; x = 0;
}

void t2(void) {
    if (s > 0) {
        y = -1;
    } else {
        y = 1;
    }
}

void main(void) {
    x = y; // synchroner Teil
    __ASTREE_asynchronous_loop((t1(), t2()));
}
```

```
__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

\_\_ASTREE\_

■ Gibt Zu

\_\_ASTREE\_

■ Zeigt Zu

\_\_ASTREE\_

■ Gibt Te

The screenshot shows the Astree IDE interface. The main editor displays the original source code of `main.c` and the analyzed version. The analyzed code includes annotations from the ASTREE analyzer, such as `<init: i : initialized >`, `<integers (intv+cong+bitfield+set): i in [1, 20] /\ != 0 >`, and `Equivalence Class:i:{i};`. The bottom status bar indicates 0 errors and 1 alarm.

```
346
347
348 int main(void) {
349     int i = 0;
350     __ASTREE_modify((i: [1,20]));
351     __ASTREE_log_vars((i));
352     i += 20;
353     __ASTREE_log_vars((i));
354     __ASTREE_assert((i > 10 && i < 100));
355
356
357
358 /**
```

```
1 #include "ab_filter.h"
2 #include "sensor.h"
3 /**
4  * Sonstige benoetigten Header-Dateien
5  **/
6 #include <stdio.h>
7 #include <assert.h>
8 #include "astree.h"
9
10 // #define EXTENDED
11 #include "bndbuf.h"
12
13
```

Line 357, column 1

Line 1, column 1

Errors: 0

- Astrée im CIP:

  - % /proj/i4ezs/tools/astree\_c/bin/a3c

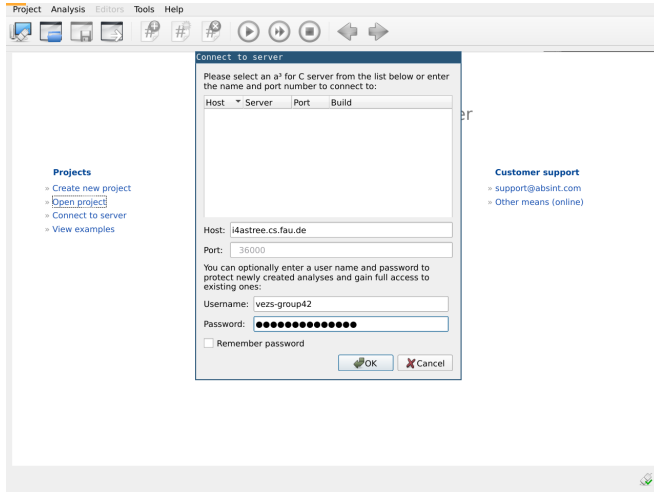
- Anmeldung mit Benutzername und Passwort

  - ↪ Passwort wird bei der ersten Anmeldung festgelegt

- Dokumentation

  - PDF: /proj/i4ezs/tools/astree\_c/help/a3c.pdf

# Anmelden



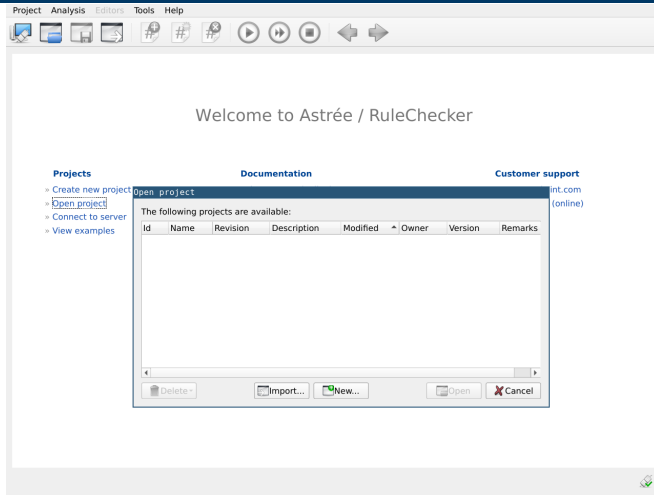
**Host** i4astree.cs.fau.de

**Port** 36000

**Benutzer** vezs-groupXX

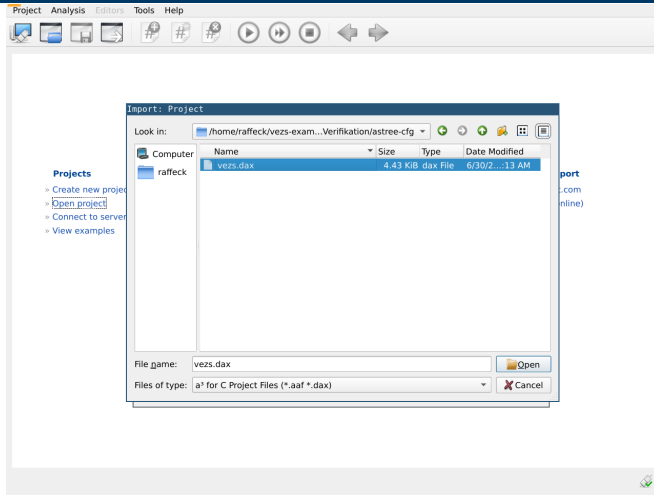
**Passwort** Beliebig wählbar

# 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 Preprocessor Tools Help

Information  
Preprocessor  
Patterns to ignore  
External declarations  
Options  
ABI  
Rules  
Reached code  
Annotations  
Comment Patterns  
Diff Comments  
Contributors  
Support

Preprocess base directory: /home/cip/2016/na79zegy/vezs/07\_Verifikation/astree-cfg

Operating system: None

Sources

Insert files for preprocessing: + [ ] [ ] [ ] -

Default configuration [add files]

Use the built-in preprocessor

Config base directory: /home/cip/2016/na79zegy/vezs/07\_Verifikation/a...

Language: C

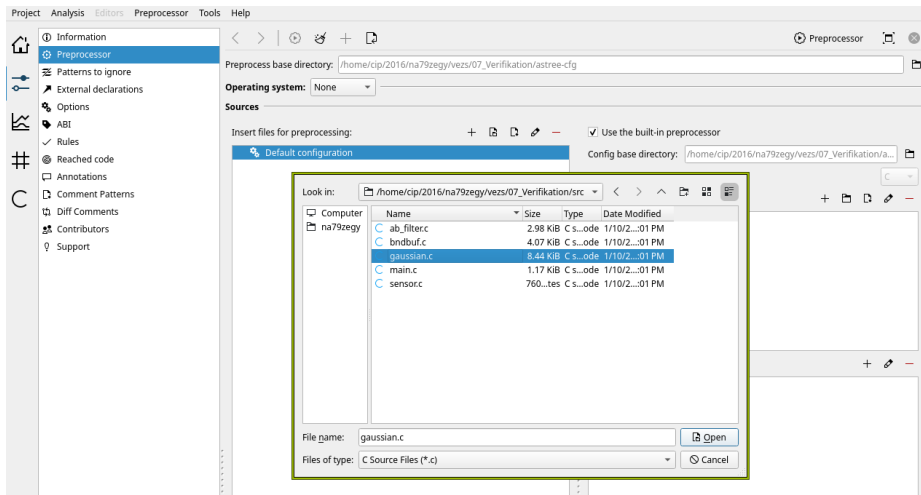
Include paths: + [ ] [ ] [ ] [ ] -

Macro definitions: + [ ] [ ] -

\_ASTREE\_

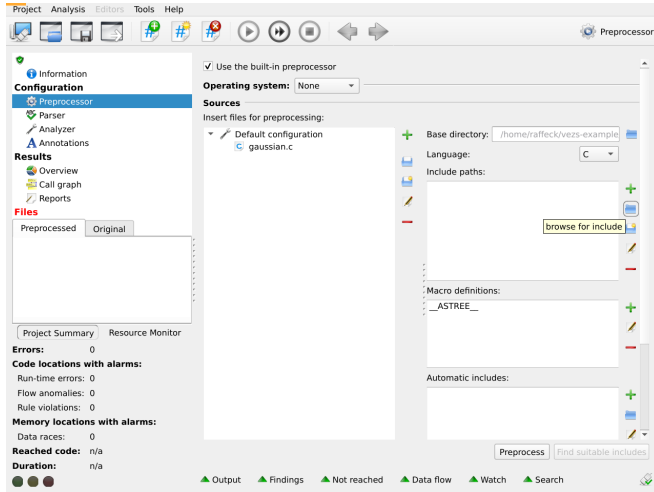
- Quelltextdateien und Includepfade definieren

# Quelldateien konfigurieren



- Quelltextdateien und Includepfade definieren

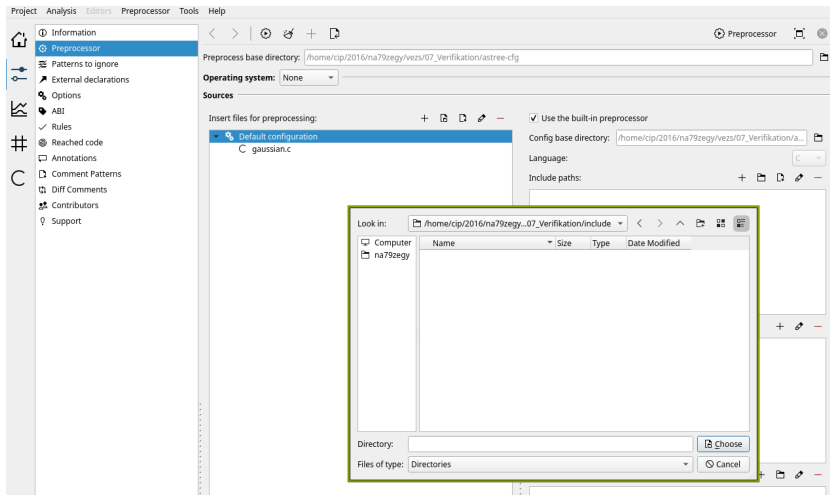
# Quelldateien konfigurieren



- Quelltextdateien und Includepfade definieren



# Quelldateien konfigurieren



- Quelltextdateien und Includepfade definieren

# Analyse starten

The screenshot shows the 'Preprocessor' configuration window in a code analysis tool. The window is titled 'Preprocessor' and has a menu bar with 'Project', 'Analysis', 'Editors', 'Preprocessor', 'Tools', and 'Help'. On the left, there is a sidebar with a tree view containing the following items: Information, Preprocessor (selected), Patterns to ignore, External declarations, Options, ABI, Rules, Reached code, Annotations, Comment Patterns, Diff Comments, Contributors, and Support. Below the sidebar is a 'Filter...' dropdown and a summary of error counts: 'Errors: 0', 'Code locations with alarms', 'Run-time errors: 0', 'Flow anomalies: 0', 'Preprocess and start analysis' (highlighted with a yellow box), 'Memory locations with alarms', 'Data races: 0', 'Reached code: n/a', and 'Duration: n/a'. The main area of the window is divided into several sections: 'Preprocess base directory' with a text field containing '/home/cip/2016/na79zegy/vezs/07\_Verifikation/astree-cfg'; 'Operating system' with a dropdown set to 'None'; 'Sources' section containing 'Insert files for preprocessing:' with a list of files under 'Default configuration' including 'gaussian.c'; 'Use the built-in preprocessor' checked; 'Config base directory' with a text field; 'Language' with a dropdown set to 'C'; 'Include paths' with a text field containing '/home/cip/2016/na79zegy/vezs/07\_Verifikation/include'; 'Macro definitions' with a text field containing '\_\_\_ASTREE\_\_'; and 'Automatic includes' with an empty text field. At the bottom, there is an 'Options' section with 'Use stubs for the standard libraries' and 'Keep comments' checked. The bottom status bar shows icons for 'Output', 'Findings', 'Locations', 'Search', and 'Project monitor'.

Project Analysis Editors Preprocessor Tools Help

Preprocessor

Preprocess base directory: /home/cip/2016/na79zegy/vezs/07\_Verifikation/astree-cfg

Operating system: None

Sources

Insert files for preprocessing:

- Default configuration
  - gaussian.c

Use the built-in preprocessor

Config base directory: /home/cip/2016/na79zegy/vezs/07\_Verifikation/...

Language: C

Include paths: /home/cip/2016/na79zegy/vezs/07\_Verifikation/include

Macro definitions: \_\_\_ASTREE\_\_

Automatic includes:

Filter:

Options

- Use stubs for the standard libraries
- Keep comments

Output Findings Locations Search Project monitor

Errors: 0

Code locations with alarms

Run-time errors: 0

Flow anomalies: 0

Preprocess and start analysis

Memory locations with alarms

Data races: 0

Reached code: n/a

Duration: n/a

# Programme zur Softwareverifikation

- Astrée
- Frama-C
- VeriFast
- Dafny
- SeaHorn
- Coq
- Isabelle/HOL
- Agda
- Idris
- ...



```
method fact(n: nat) : nat {
  require n > 0
  ensure n! == n * fact(n-1)
  ensure n! > 0
}

let fact = fun n =>
  if n == 0 then 1
  else n * fact(n-1)
end

let fact2 = fun n =>
  if n == 0 then 1
  else n * fact2(n-1)
end

let fact3 = fun n =>
  if n == 0 then 1
  else n * fact3(n-1)
end
```



SeaHorn



# Programme zur Softwareverifikation

- Astrée
- **Frama-C**
- VeriFast
- Dafny
- SeaHorn
- Coq
- Isabelle/HOL
- Agda
- Idris
- ...



- 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?

- 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

- 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 *Prädikatransformation* zugewiesen
  - Abbildung: Nachbedingung  $\mapsto$  notwendige schwächste Vorbedingung
  - Eine rückwärtige *symbolische Ausführung* des Programmsegments

- 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: ?

```
if (a > b)
```

```
    result = a;
```

```
else
```

```
    result = b;
```

```
return result;
```

# Beispiel: WP-Kalkül Maximumsfunktion

WP: ?

Auswertungsreihenfolge

```
if (a > b)
```

```
  result = a;
```

```
else
```

```
  result = b;
```

```
return result;
```

```
\result ≥ b ∧ \result ≥ a
```

Q:

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

# Beispiel: WP-Kalkül Maximumsfunktion

WP: ?

Auswertungsreihenfolge

```
if (a > b)
```

```
  result = a;
```

```
else
```

```
  result = b;
```

```
result ≥ b ∧ result ≥ a
```

```
return result;
```

```
\result ≥ b ∧ \result ≥ a
```

Q:

```
wp(return x, Q) ≡ Q[\result/x]
```

# Beispiel: WP-Kalkül Maximumsfunktion

WP: ?

Auswertungsreihenfolge

if (a > b)

result = a;

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

else

result = b;

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

$result \geq b \wedge result \geq a$

return result;

$\backslash result \geq b \wedge \backslash result \geq a$

Q:

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

# Beispiel: WP-Kalkül Maximumsfunktion

WP: ?

Auswertungsreihenfolge

if (a > b)

result = a;

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

else

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

result = b;

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

$result \geq b \wedge result \geq a$

return result;

$\backslash result \geq b \wedge \backslash result \geq a$

Q:

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

# Beispiel: WP-Kalkül Maximumsfunktion

WP: ?

Auswertungsreihenfolge

if ( a > b )

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

result = a;

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

else

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

result = b;

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

$result \geq b \wedge result \geq a$

return result ;

$\backslash result \geq b \wedge \backslash result \geq a$

Q:

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

# Beispiel: WP-Kalkül Maximumsfunktion

WP:

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

if ( a > b )

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

result = a;

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

else

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

result = b;

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

$$result \geq b \wedge result \geq a$$

return result ;

$$\backslash result \geq b \wedge \backslash result \geq a$$

Q:

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

Auswertungsreihenfolge

# Beispiel: WP-Kalkül Maximumsfunktion

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

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

if ( a > b )

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

result = a;

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

else

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

result = b;

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

$result \geq b \wedge result \geq a$

return result ;

$\backslash result \geq b \wedge \backslash result \geq a$

WP:

Auswertungsreihenfolge

Q:



# 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;

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

else

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

    result = b;

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

result  $\geq b \wedge result \geq a$

return result ;

Q:  $\backslash result \geq b \wedge \backslash result \geq a$

Auswertungsreihenfolge

- erlaubt Nachweise funktionaler Eigenschaften mittels wp-Kalkül
  - Prädiaktenlogik erster Stufe
    - $\forall$ : `\forallall`
    - $\exists$ : `\existsexists`
    - $\Rightarrow$ : Implikation, `==>`
    - aussagenlogische 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

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

# 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;  
}
```

# 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;  
}
```

$Q$ :  $\backslash \text{result} \geq a \wedge \backslash \text{result} \geq b$

# Beispiel: Maximumsfunktion

$P$ : wahr

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

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

# Beispiel: Maximumsfunktion

$P$ : wahr

```
S: int maximum(int a, int b) {
    int result = INT_MIN;

    if(a > b)
        result = a;
    else
        result = b;

    return result;
}
```

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

```
/*@
    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;
}
```

## Frama-C (II): Speicherspezifikation

```
// Swap array[0] with pointer other, do not modify the rest of array
```

```
void swap_first_element(int *array, size_t length, int *other) {  
    int tmp = array[0];  
    array[0] = *other;  
    *other = tmp;  
}
```



## Frama-C (II): Speicherspezifikation

```
// 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 sichtb.) 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;`

# Schleifen (Fortsetzung)

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

```
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];  
        }  
    }  
    return max;  
}
```

# 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];  
        }  
    }  
    return max;  
}
```

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

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

# Frama-C: Datenstrukturen beschreiben

- 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 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**: Beschreibung ist vollständig  
Behaviors beschreiben das Verhalten für alle Aufrufkontexte
- **disjoint behaviors**: beschriebene Verhalten überlappen sich nicht



# Frama-C: Beispiel Behaviors

```
#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;
}
```

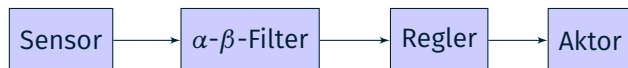
- Die Frama-C-Gui bietet keinen Editor an!
- Reihenfolge der Annotationen z.T. relevant. Empfehlung:
  - Funktionen*
    - requires
    - assigns
    - ensures
  - Schleifen*
    - loop invariants
    - loop assigns
    - loop variants
- `assert()` wird als nicht terminierend angenommen  
↪ Frama-C `assert` 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**
- 4  $\alpha$ - $\beta$ -Filter

# Aufgabenstellung

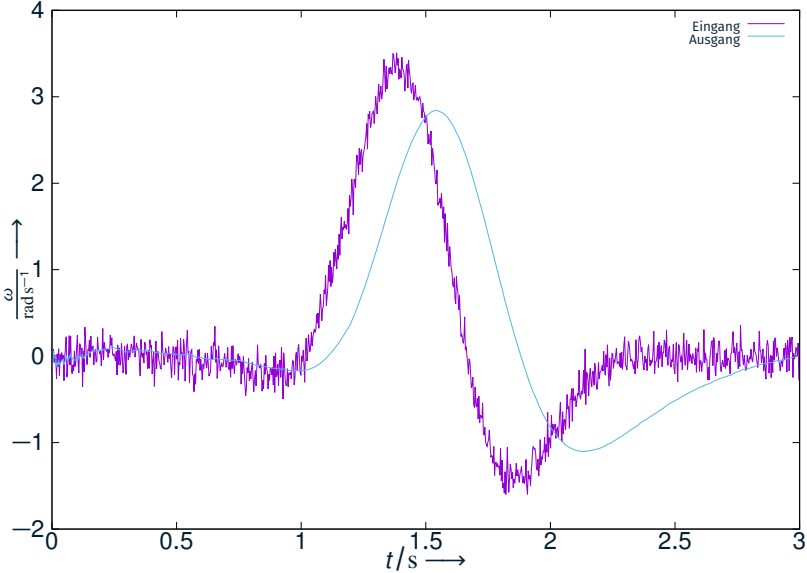
- Sensorstubs implementieren
- Implementierung eines Ringpuffers zur Verwaltung der Sensorwerte
- System erstellen:
  - Sensoren und Filter initialisieren
  - Sensorwerte abrufen
  - Sensorwerte filtern
- Abwesenheit von Laufzeitfehlern nachweisen
  - ↳ Astrée
- Numerische Stabilität des Filters nachweisen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

- 1 Abstrakte Interpretation mit Astrée
- 2 Formale Verifikation mit Frama-C
- 3 Aufgabenstellung
- 4  $\alpha$ - $\beta$ -Filter**



- 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 $\alpha$ - $\beta$ -Filter



# Filteralgorithmus

- 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.  $\hat{x}[0] = \hat{\dot{x}}[0] = 0$

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

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

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

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



# Filterparameter

$T$  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

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

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

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

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

# Initialisierungsphase

- Zu Beginn hat der Filter keinen gültigen Zustand
- ↪ Einschwingphase, in der der 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 er 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

```
struct BndBuf{  
    int* buf; // array of <size>  
    int size;  
    int read_pos;  
    int write_pos;  
};
```

## ■ „Füllstandsanzeige“: Speichern von

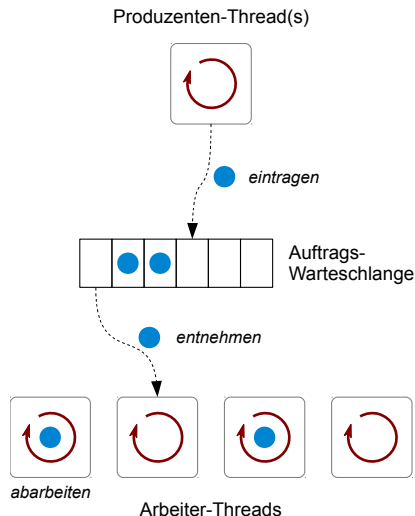
- Belegten Plätzen
- Verfügbaren Plätzen

## ■ 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





42

-  AbsInt Angewandte Informatik GmbH.  
***The Static Analyzer Astrée, April 2012.***
-  C. F. Asquith.  
***Weight selection in first-order linear filters.***  
Technical report, Army Inertial Guidance and Control Laboratory  
Center, Redstone Arsenal, Alabama, 1969.
-  E. Brookner.  
***Tracking and Kalman Filtering Made Easy.***  
Wiley-Interscience, 1st edition, 4 1998.



-  E. W. Dijkstra.  
**Guarded commands, nondeterminacy and formal derivation of programs.**  
*Commun. ACM*, 18(8):453–457, Aug. 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.

-  P. 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.
-  R. G. Lyons.  
***Understanding Digital Signal Processing.***  
Prentice Hall, 3rd edition, 11 2010.