Verlässliche Echtzeitsysteme - Übungen

Git

Wintersemester 2024

Eva Dengler, Peter Wägemann

Friedrich-Alexander-Universität Erlangen-Nürnberg
Lehrstuhl Informatik 4 (Systemsoftware)
https://sys.cs.fau.de

Überblick

Anforderungen

Typische Aufgaben eines Versionsverwaltungssystems sind:

- *Sichern* alter Zustände
- *Zusammenführung* paralleler Entwicklung
- *Transportmedium*

Idealerweise zusätzlich:

- *Unabhängige Entwicklung* ohne zentrale Infrastruktur

git

- in VEZS: `git`
- 2005 von Linus Torvalds für den Linux-Kernel geschrieben
- Konsequenz der Erfahrungen mit *bitkeeper*
- Eigenschaften:
  - dezentrale, parallele Entwicklung
  - Koordinierung hunderter Entwickler
  - Visualisierung von Entwicklungszweigen

`git`-Commits

- Was speichert ein Commit?
  - Wer? ⇝ Autor
  - Warum? ⇝ Commit-Nachricht
  - Was? ⇝ Vorher/Nachher *Zustände*
  - Vorgänger Commits, auch *mehrere!*
  - *Keine* Nachfolger

⇝ Commit-Id: SHA-1 Hash über Inhalt

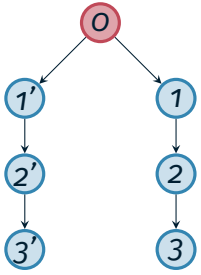
⇝ Gerichteter Azyklischer Graph (DAG)

- ⇝ Sprünge zurück *möglich*
- ⇝ Sprünge vorwärts *nicht möglich*

- Woher “obere“ Commits?

⇝ Symbolische Namen (Zeiger)

- HEAD: Aktueller Commit
- Branch: Zeiger auf Commit

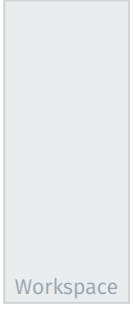


Git-Repository einrichten

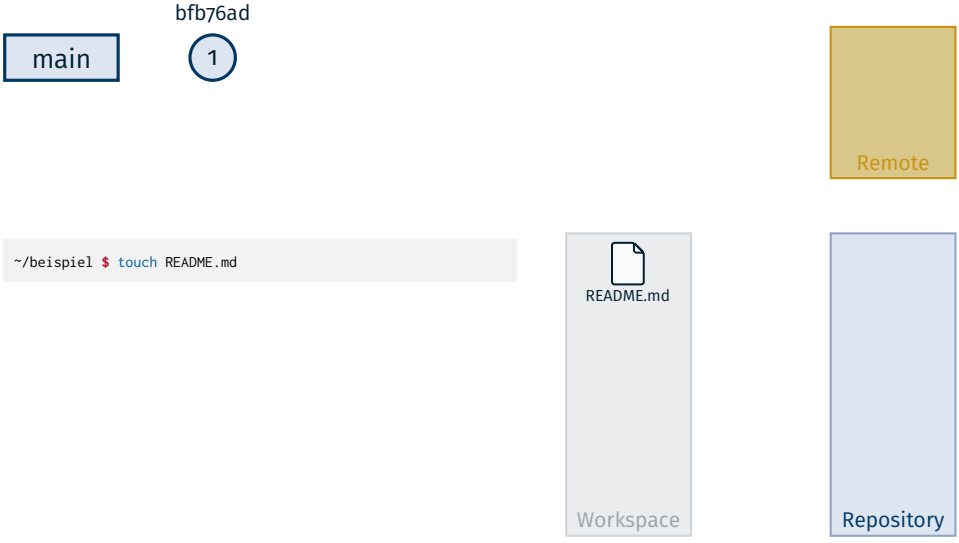

```
~$ git clone gitos.rrze.fau.de:vezs-example/test.git beispiel
Cloning into 'beispiel'...
```

Git-Repository einrichten

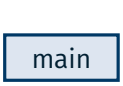

```
~$ git clone gitos.rrze.fau.de:vezs-example/test.git beispiel
Cloning into 'beispiel'...
~$ cd beispiel
```



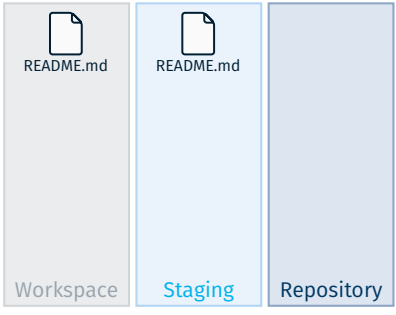
Dateien mit GIT verwalten



Dateien mit GIT verwalten

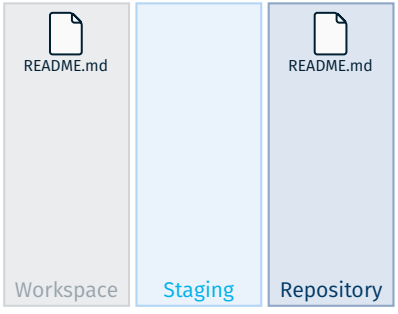



```
~/beispiel $ touch README.md
~/beispiel $ git add README.md
```



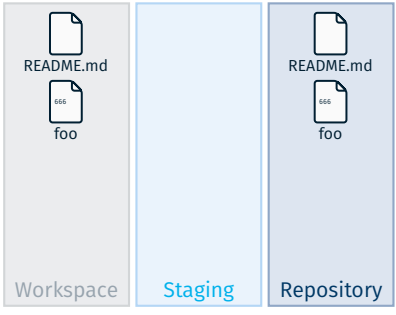
Dateien mit GIT verwalten


```
~/beispiel $ touch README.md
~/beispiel $ git add README.md
~/beispiel $ git commit -m "Liesmich hinzugefügt"
[main bd2de5c] Liesmich hinzugefügt
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 README.md
```



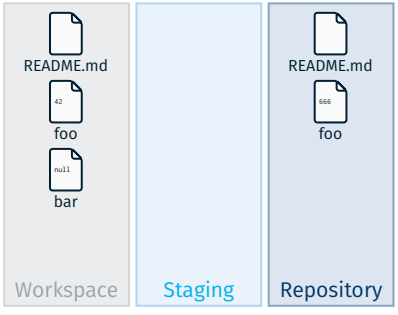
Dateien mit GIT verwalten


```
~/beispiel $ echo "666" > foo
~/beispiel $ git add foo
~/beispiel $ git commit -m "Datei foo erstellt"
[main df7aa5a] Datei foo erstellt
 1 file changed, 1 insertion(+)
 create mode 100644 foo
```



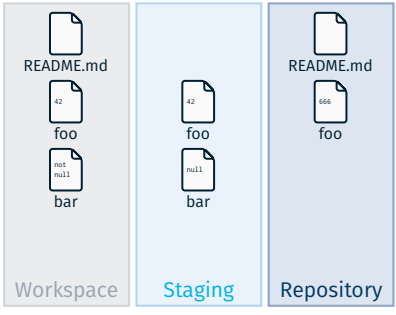
Dateien mit GIT verwalten


```
~/beispiel $ echo "42" > foo
~/beispiel $ echo "null" > bar
~/beispiel $ git status
On branch main
Your branch is up to date with 'origin/main'.

Changes not staged for commit:
     modified: foo

Untracked files:
     bar

no changes added to commit
```



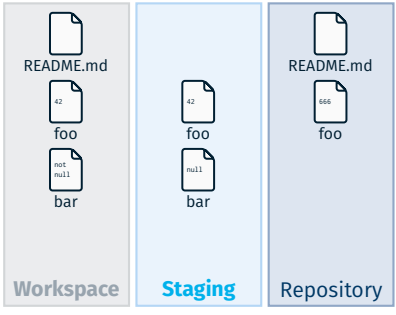
Dateien mit GIT verwalten


```
~/beispiel $ git add foo bar
~/beispiel $ echo "not null" > bar
~/beispiel $ git status
On branch main
Your branch is up to date with 'origin/main'.

Changes to be committed:
    new file: bar
    modified: foo

Changes not staged for commit:
    modified: bar
```



Dateien mit GIT verwalten

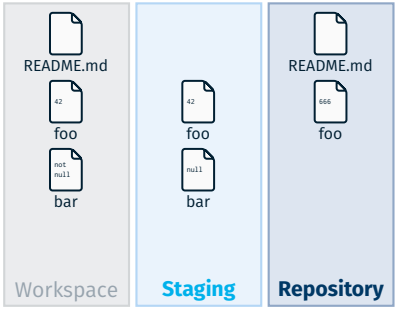

```
~/beispiel $ git diff
diff --git a/bar b/bar
index 19765bd..b263a85 100644
--- a/bar
+++ b/bar
@@ -1 +1 @@
-null
+not null
```



Dateien mit GIT verwalten

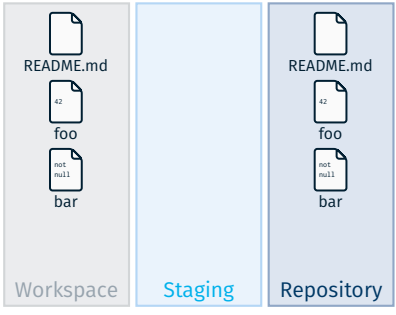
Remote

```
~/beispiel $ git diff --staged
diff --git a/bar b/bar
new file mode 100644
index 0000000..19765bd
— a/bar
+++ b/bar
@@ -0,0 +1 @@
+null
diff --git a/foo b/foo
index 7cc86ad..d81cc07 100644
[...]
```



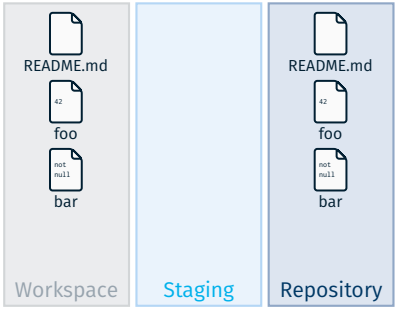
Dateien mit GIT verwalten


```
~/beispiel $ git add bar
~/beispiel $ git commit -m \
          "Foo korrigiert und Bar erstellt"
[main 90f7cfe] Foo korrigiert und Bar erstellt
 2 files changed, 2 insertions(+), 1 deletion(-)
 create mode 100644 bar
```



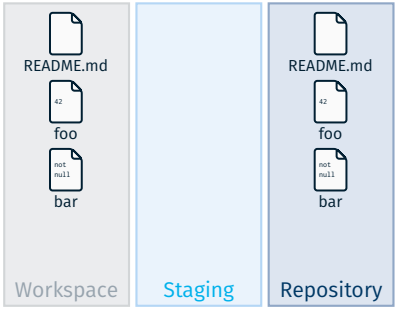
Dateien mit GIT verwalten


```
~/beispiel $ git shortlog
Nutzer (3):
     Liesmich hinzugefügt
     Datei foo erstellt
     Foo korrigiert und Bar erstellt
```



Dateien mit GIT verwalten


```
~/beispiel $ git push
Enumerating objects: ...
    ...
To gitos.rrze.fau.de:vezs-example/test.git
   bfb76ad..90f7cfe main -> main
```



git push [<remote> [<branch>]]

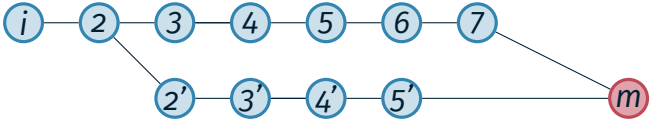
- schiebt Commits nach `<remote>` in den ausgewählten `<branch>`
- dies geht nur, wenn lokales Repo auf dem aktuellen Stand ist!
- sonst beschwert sich `git`:

```
% git push origin main
To /tmp/test.git
 ! [rejected]        main -> main (non-fast-forward)
error: failed to push some refs to '/tmp/test.git'
To prevent you from losing history, non-fast-forward updates were rejected
Merge the remote changes (e.g. 'git pull') before pushing again.  See the
'Note about fast-forwards' section of 'git push --help' for details.
```

⇝ wir müssen das Repository erst auf den aktuellen Stand bringen

git pull [<remote> [<branch>]]

- holt Änderungen aus `remote` in den aktuellen Branch
- verschmilzt aktuellen Branch mit geholten Änderungen



% git pull origin

```
remote: Counting objects: 5, done.
remote: Total 3 (delta 0), reused 0 (delta 0)
Unpacking objects: 100% (3/3), done.
From /tmp/test
   38b95cb..8ec6e93  main     -> origin/main
Auto-merging test.txt
CONFLICT (content): Merge conflict in test.txt
Automatic merge failed; fix conflicts and then commit the result.
```

- Änderungen an gleicher Stelle in der Zwischenzeit

  ⇝ Konflikte müssen von Hand behoben werden

Konflikt beheben (1)

```
% cat test.txt
```

```
hallo
<<<<<<< HEAD
welT!     meine Version
=======
Welt!     Version in origin/main
>>>>>>> 8ec6e9309fa37677e2e7ffcf9553a6bebf8827d6
```

- Konflikt auflösen:
  - Manuelles Bearbeiten
  - Übernahme des eigenen, Verwerfen des fremden Standes:
    `git checkout --ours test.txt`
  - Übernahme des fremden, Verwerfen des eigenen Standes:
    `git checkout --theirs test.txt`

Konflikt beheben (2)

- Konfliktlösung an `git` signalisieren:

```
% git add test.txt && git commit
```

```
[main 4d21871] Merge branch 'main' of /tmp/test
```

```
% git push origin main
```

```
Counting objects: 5, done.
Writing objects: 100% (3/3), 265 bytes, done.
Total 3 (delta 0), reused 0 (delta 0)
Unpacking objects: 100% (3/3), done.
To /tmp/test.git
   8ec6e93..278c740  main -> main
```

- *Juhu!*

`git`-Kommandos: Austausch von Quellcode (1)

- initiales *Klonen*:

  ```
  % git clone https://www4.cs.fau.de/...
  ```

- Einspielen entfernter Änderungen:

  ```
  % git pull
  ```

  ⇒ äquivalent zu

  ```
  % git fetch && git merge
  ```

- Mehrere Repositories registrieren:

  ```
  % git remote add 32-stable git://git.kernel.org/.../...
  ```

- registrierte Remotes untersuchen:

  ```
  % git remote -v
  ```

`git`-Kommandos: Austausch von Quellcode (2)

- alle Remotes nachladen (aktueller Branch wird nicht verändert)

  ```
  % git remote update
  ```

- lokalen Branch aus dem neuen ‘"Remote“ anlegen:

  ```
  % git checkout -b work 32-stable/main
  ```

- Unterschiede zwischen lokalem und entferntem Branch untersuchen:

  ```
  % git log ..origin/main
  ```

- aktuelle Änderungen auf dem entfernten Branch neu aufspielen:

  ```
  % git pull --rebase
  ```

- die neuste Änderung untersuchen:

  ```
  % git show
  ```

`git`-Kommandos: Austausch von Quellcode (3)

- herausfinden wer für welche Zeilen einer Datei verantwortlich ist:

```
% git blame
```

`git`-Kommandos: Austausch von Patches

- die letzten drei Änderungen als Patch:

  ```
  % git format-patch HEAD~~
  ```

- Sendeziel für Patchversand per E-Mail vorgeben:

  ```
  % git config sendemail.to=...@...
  ```

- Patchset letzten drei Änderungen per E-Mail senden:

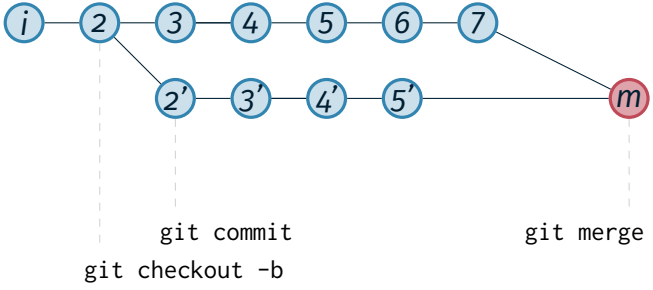
  ```
  % git send-email --compose HEAD~~
  ```

- einen Patch aus einer Mailbox anwenden:

  ```
  % git am < <Datei>
  ```

Verzweigungen und Zusammenführungen

Beispiel für parallele Entwicklung:



Arbeitsablauf mit Branches

In den meisten Versionsverwaltungssystemen

1. Featurebranch anlegen
2. Feature im Branch implementieren, testen
3. Featurebranch mit `main` verschmelzen
4. ggf. Featurebranch löschen

Naiver Ansatz

$\rightsquigarrow$ skaliert nicht!

Warum `branch/edit/merge` nicht skaliert

Aufgaben von Versionsverwaltung

1. Codeschreiben unterstützen
2. Konfigurationsmanagment/Branches
   ⇝ z. B. Release-Version, HEAD-Version …

⇝ Konflikt

1. braucht Checkpoint-Commits
   - möglichst oft einchecken

   ⇝ skaliert nicht
2. braucht Stable-Commits
   - nur einchecken, wenn Commit perfekt

   ⇝ nicht praktikabel

Lösung mit `git`: öffentlicher vs. privater Branch

Öffentlicher Branch $\rightsquigarrow$ verbindliche Geschichte

Commits sollen $\left\{\begin{array}{l}\text{atomar}\\ \text{gut dokumentiert}\\ \text{linear}\\ \text{unveränderlich}\end{array}\right\}$ sein

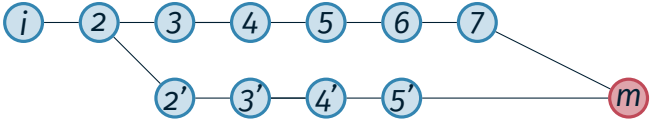
Privater Branch $\rightsquigarrow$ Schmierpapier

- für einzelnen Entwickler
- möglichst lokal
- wenn im zentralen Repo $\rightsquigarrow$ auf Privatheit einigen

Aufräumen

- verschmelze nie direkt privaten mit öffentlichem Branch
  - Historie wird sonst unübersichtlich

  ⇝ nicht einfach `git merge` im `main` machen



- vorher immer erst `git`
  - `rebase` ⇝ Commits auf Branch anwenden
  - `merge --squash` ⇝ einzelnen Commit aus Branch-Commits

    oder: `rebase --interactive` ⇝ Commits umgruppieren
  - `commit --amend` ⇝ letzten Commit überarbeiten
- Ziel: öffentlicher Commit ≡ Kapitel eines Buches

Michael Crichton

Great books aren't written – they're rewritten.

Arbeitsablauf für kleinere Änderungen

- `git merge --squash` ⇝ Änderungen aus Branch in aktuellen Index

Branch

`% git checkout -b private_feature_branch` (Branch anlegen)

`% touch file1.txt file2.txt`

`% git add file1.txt; git commit -am "WIP1"` (file1.txt einchecken)

`% git add file2.txt; git commit -am "WIP2"` (file2.txt einchecken)

Merge

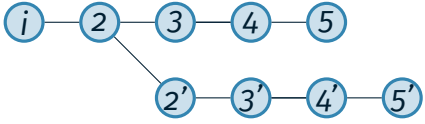
`% git checkout main` (nach `main` wechseln)

`% git merge –squash private_feature_branch` (Änderungen auf `main`)

`% git commit -v` (Änderungen einchecken)

git rebase <branch>

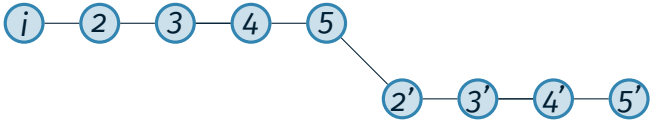
- Aufsetzen auf bestehenden `<branch>`



- Patches aus „unterem“ Zweig werden auf „oberen“ aufgespielt
- Die Historie ist nun linear
- Linearisierte Änderungen lassen sich häufig einfacher bewerten
- Vorsicht!
  - Verzweigungen von alten Zweig können nicht mehr zusammengeführt werden
  - Keine gemeinsamen Vorgänger mehr
  - Visualisierung der Historie ist nun bestenfalls verwirrend

git rebase <branch>

- Aufsetzen auf bestehenden `<branch>`



- Patches aus „unterem“ Zweig werden auf „oberen“ aufgespielt
- Die Historie ist nun linear
- Linearisierte Änderungen lassen sich häufig einfacher bewerten
- Vorsicht!
  - Verzweigungen von alten Zweig können nicht mehr zusammengeführt werden
  - Keine gemeinsamen Vorgänger mehr
  - Visualisierung der Historie ist nun bestenfalls verwirrend

git rebase –interactive <commit>

- schreibt Geschichte um

`git rebase –interactive ccd6e62^`

pick ⇝ übernimmt Commit

```
pick ccd6e62 Work on back button
pick 1c83feb Bug fixes
pick f9d0c33 Start work on toolbar
```

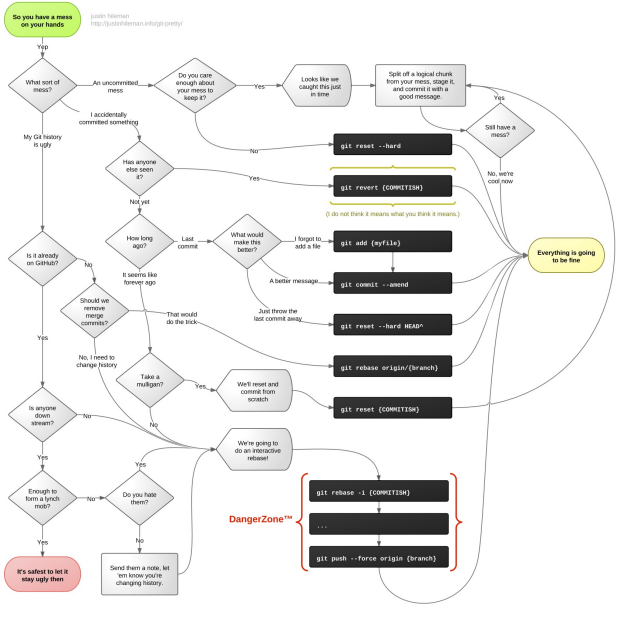
fixup ⇝ verschmilzt Commit mit Vorgänger

```
pick ccd6e62 Work on back button
fixup 1c83feb Bug fixes # mit Vorgaenger verschmelzen
pick f9d0c33 Start work on toolbar
```

reword ⇝ Beschreibung editieren

edit ⇝ kompletten Commit editieren

Geschichte neuschreiben



Wenn der Feature-Branch im Chaos versinkt?

↝ aufgeräumten Branch anlegen

1. auf Branch `main` wechseln
   ```
   % git checkout main
   ```
2. Branch aus `main` erzeugen
   ```
   % git checkout -b cleaned_up_branch
   ```
3. Branch-Änderungen in den Index und die Working Copy ziehen
   ```
   % git merge --squash private_feature_branch
   ```
4. Index zurücksetzen
   ```
   % git reset
   ```

- danach Commits neu zusammenbauen

↝ Auf der Konsole: `git add -p`

↝ Graphisch: `git cola`

git reflog

- Historie des `HEAD`-Zeigers
- Historie aller Befehle, die `HEAD` verändern

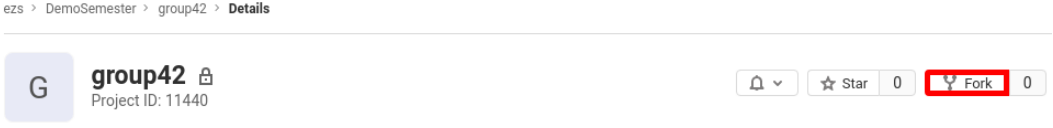
`git reflog`

```
8afd010 HEAD@{0}: rebase -i (finish): returning to refs/heads/main
8afd010 HEAD@{1}: checkout: moving from main to 8afd010ae2ab48246d5
7f97fab HEAD@{2}: commit: Pentax K20D fw version 1.04.0.11 wb presets
8c37332 HEAD@{3}: rebase -i (finish): returning to refs/heads/main
8c37332 HEAD@{4}: checkout: moving from main to 8c373324ca196c337dd
9d66ec9 HEAD@{5}: clone: from git://github.com/darktable-org/darkt...
```

- `git reset --hard HEAD@{2}` stellt alten Zustand wieder her

Gruppenrepositories

- Selbstverwaltet auf `https://gitos.rrze.fau.de/`
- Login über Single-Sign-On mit IDM-Kennung
- Regeln auf der Hauptseite beachten!
- Abgaberepository pro Gruppe:
  `https://gitos.rrze.fau.de/i4/teaching/vezs/WS24/group<groupid>`
- Arbeitskopie als Fork erstellen



- Abgabe per Merge-Request gegen
  `https://gitos.rrze.fau.de/i4/teaching/vezs/WS24/group<groupid>`

SSH-Schlüssel konfigurieren

- SSH Schlüssel erzeugen:

⇝ `% ssh-keygen -t rsa -f ~/.ssh/gitlab`

- SSH Schlüssel für Authentifizierung hinterlegen:

⇝ Kopieren: `% xclip -selection clipboard .ssh/gitlab.pub`

⇝ Einfügen im Gitlab unter: Benutzer → Einstellungen → SSH-Keys

- `https://gitos.rrze.fau.de/help/user/ssh`

- SSH-Config anpassen, damit der neue Schlüssel auch verwendet wird:

`$HOME/.ssh/config`

```
Host gitos.rrze.fau.de
  IdentityFile ~/.ssh/gitlab
```

Dateien ignorieren mit `git`

- Dateien erscheinen nicht in der Ausgabe von bspw. `git status`
- Änderungen an Dateien werden ignoriert

.gitignore

```
# Ignore LaTex temporary files
*.aux
*.log

# except this one
!important.log

# everything under directory
solutions/
```

- Auch global möglich: `$HOME/.gitconfig`

`git`-Konfiguration des Repositories

Per Befehlszeile

```
% git remote set-url origin git@gitos.rrze.fau.de:<user>/<projekt>.git
% git remote add vorgabe git@gitos.rrze.fau.de:ezs/vezs25-vorgabe.git
```

`.git/config`

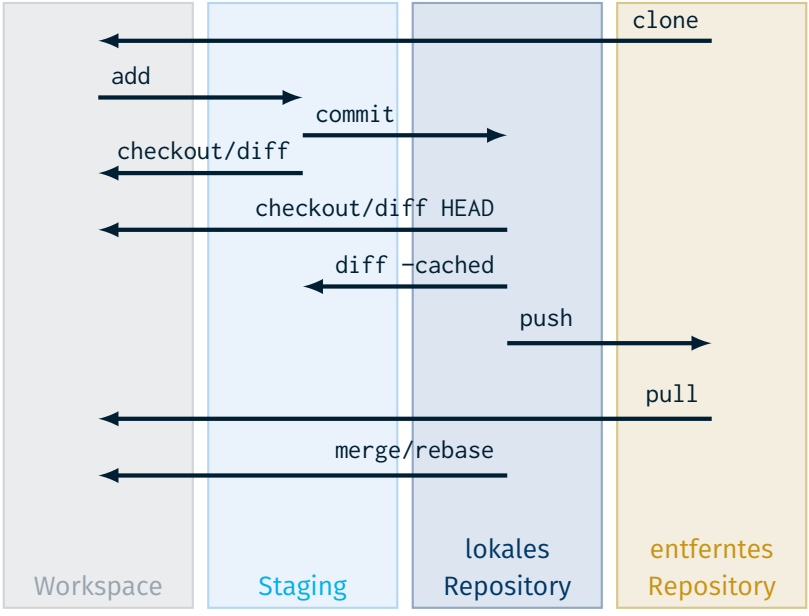
```
[remote "origin"]
  fetch = +refs/heads/*:refs/remotes/origin/*
  url = git@gitos.rrze.fau.de:<username>/<projektname>.git

[remote "vorgabe"]
  fetch = +refs/heads/*:refs/remotes/origin/*
  url = git@gitos.rrze.fau.de:ezs/vezs25-vorgabe.git
```

Globale `git`-Konfiguration des Systems

`$HOME/.gitconfig`

```
[user]
  name = Max Mustermann
  email = max.mustermann@fau.de
[core]
  editor = <maxs-lieblingseditor>
[commit]
  verbose = true
[color]
  ui = auto
[alias]
  s       = status
  a       = add
  c       = commit
  co      = checkout
  b       = branch -a -v
  unstage = reset HEAD --
  visual  = !gitk
  lg      = log --graph \
                --abbrev-commit \
                --date=relative
```

Überblick: Dateien mit GIT verwalten



Git Cheatsheet

git init	initiales Anlegen eines Repositories
git clone <*url*>	initiales Kopieren von einer Quelle
git pull	kurz für Holen und Zusammenfügen
git push	Übertragen in entfernte Quelle
git add <*file*>	Datei als Kandidat für nächsten *commit* markieren
git add -p	feingranular...
git add -i	... oder interaktiv
git diff	unversionierte Änderungen anzeigen
git diff –cached	Änderungen des nächsten Commits anzeigen
git commit	Änderungen versionieren
git clean -d <*path*>	alles, was nicht im git ist, löschen
git clean -n -d <*path*>	anzeigen, was gelöscht werden würde

Git Cheatsheet (II)

git show	neuste (versionierte) Änderungen anzeigen
git status	Änderungen zum Vorgänger anzeigen
git log	Historie anzeigen
git checkout – <*path*>	geänderte, aber nicht eingecheckte Datei zurücksetzen
git checkout <*branch*>	zu einem Branch wechseln
git revert <*id*>	Commit rückgängig machen
git branch <name>	Branch anlegen
git branch -a	alle Branches anzeigen
man git-<*cmd*>	Hilfe anzeigen, z.B. `man git-add`

Lesenswertes zu `git`

- `http://gitready.com`
- `http://book.git-scm.com/`
- `http://eagain.net/articles/git-for-computer-scientists/`
- `http://sandofsky.com/blog/git-workflow.html`
- `http://365git.tumblr.com/`

- Zum Ausprobieren:
  - `https://learngitbranching.js.org/`
  - `https://github.com/git-game/git-game`
  - `man git-tutorial`, `man git-tutorial2`

Verlässliche Echtzeitsysteme - Übungen

Filter

Wintersemester 2024

Eva Dengler, Peter Wägemann

Friedrich-Alexander-Universität Erlangen-Nürnberg
Lehrstuhl Informatik 4 (Systemsoftware)
https://sys.cs.fau.de

Überblick

1. C-Quiz Teil I
2. Festkommaarithmetik
3. Softwareentwurf
4. Hinweise zur Aufgabe: Filter

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 3 [2]

Angenommen: `int x = 1;` **Zu was wird** `(unsigned short)x > -1` **ausgewertet?**

1. `0`
2. 1
3. nicht definiert

Erklärung

- vor dem Vergleich beide Operanden nach `int` umgewandelt
- weil dies ohne Wertverlust geschehen kann

⇝ hier werden zwei `signed`-Werte verglichen

⇝ ein `unsigned int` würde nicht umgewandelt werden!

Frage 4 [2]

Zu was wird `-1L > 1U` auf x86-64 ausgewertet? Auf x86?

1. beides `0`
2. beides `1`
3. `0` auf x86-64, `1` auf x86
4. `1` auf x86-64, `0` auf x86

Erklärung

- auf x86-64 ist `int` kürzer als `long`

⇝ `unsigned int` wird zu `long` ⇝ `-1L > 1L` ⇒ `0`

- auf x86 entspricht `int` dem Datentyp `long`

⇝ `UINT_MAX > 1U` ⇒ `1`

Frage 5 [2]

Zu was wird `SCHAR_MAX == CHAR_MAX` ausgewertet?

1. `0`
2. `1`
3. implementation defined

Erklärung

- C99 schreibt nicht vor ob `char` vorzeichenbehaftet ist
- auf x86 und x86-64 ist `char` für gewöhnlich vorzeichenbehaftet

Frage 6 [2]

Zu was wird `UINT_MAX + 1` ausgewertet?

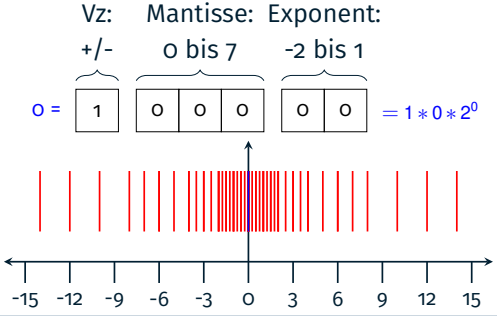
1. `0`
2. `1`
3. `INT_MIN`
4. `UINT_MIN`
5. nicht definiert

Erklärung

Der C-Standard garantiert, dass `UINT_MAX + 1 == 0`

Inhalt

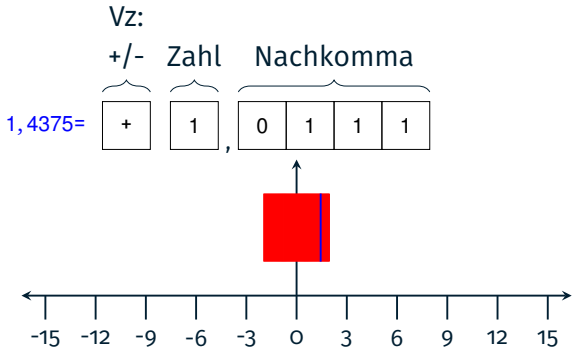
Fließkommazahlen



IEEE 754

- Noch komplexer:
  - normalisierte/denormalisierte Darstellung
  - Rundung, Fehlersemantik, …
  - NaN, ∞, …
- `https://ieeexplore.ieee.org/document/4610935/`

Festkommazahlen: Grundlagen



C-Standard und Zahlendarstellung

- Zahlendarstellung im Standard nicht festgelegt:
  - Einerkomplement
  - Vorzeichen und Magnitude
  - Zweierkomplement
- Heute meist Zweierkomplement $\rightsquigarrow$ kein dediziertes Vorzeichenbit

Festkommaarithmethik – Motivation

```
float func(void){
  volatile float a = 23.42;
  volatile float b = 12.34;
  return a * b;
}
```

```
func:
  push  {r7, lr}
  sub   sp, #8
  add   r7, sp, #0
  ldr   r3, [pc, #28] ; float a
  str   r3, [r7, #4]
  ldr   r3, [pc, #28] ; float b
  str   r3, [r7, #0]
  ldr   r2, [r7, #4]
  ldr   r3, [r7, #0]
  adds  r0, r2, #0 ; Param 1
  adds  r1, r3, #0 ; Param 2
  bl    3a6c <__aeabi_fmul>
  adds  r3, r0, #0
  adds  r0, r3, #0
  mov   sp, r7
  add   sp, #8
  pop   {r7, pc}
```

- Setup
  - Plattform: ARM Cortex-M0+
  - Compiler: arm-gcc
- Funktion `__aeabi_fmul` : 300 Zeilen Assembler
- Keine Fließkommaeinheit (engl. floating-point unit, FPU) vorhanden
- Emulation der **Fließkommaarithmetik in Software**

Festkommaarithmetik – Q-Notation

- Mikrocontroller ohne *Fließkommaeinheit*
- *Kein EAN* für Fließkommazahlen
  - $\rightsquigarrow$ *Festkommaarithmethik* mit Ganzzahlen
- Zahlenformat häufig in Q-Notation [1] angegeben
- *Qm.n* $\rightsquigarrow$ Festkommazahl mit
  - m Bit vor dem Komma, n nach dem Komma, ein Vorzeichenbit
  - Wertebereich: $[-2^m, 2^m - 2^{-n}]$
  - Auflösung: 2^{-n}
- Implementierung für Übungsaufgabe *vorgegeben*

Implementierung als Integer

$\rightsquigarrow$ passendes Q-Format ist **anwendungsspezifisch**

Q-Notation – Beziehung zu Fließkommazahlen

von Fließkomma nach Qm.n

1. Multiplikation mit 2^n
2. Runden auf die nächste Ganzzahl

von Qm.n nach Fließkomma

1. Umwandlung in Fließkommazahl $\rightsquigarrow$ cast
2. Multiplikation mit 2^{-n}

Operationen – Addition/Subtraktion

- Addition und Subtraktion wie bei Ganzzahlen

Addition

```
int32_t       a = ...;
int32_t       b = ...;
int32_t result = a + b;
```

```
    2 , 8 0
+ 1 3 , 3 7
  (1)
= 1 6 , 1 7
```

Subtraktion

```
int32_t       a = ...;
int32_t       b = ...;
int32_t result = a - b;
```

```
  1 6 , 1 7
-   2 , 8 0
  (1)
= 1 3 , 3 7
```

Operationen – Multiplikation/Division

- Braucht Zwischenergebnis von doppelter Bitbreite

Multiplikation

```
#define K   (1 << (n - 1))
int32_t     a = ...;
int32_t     b = ...;
int64_t temp = (int64_t) a * (int64_t) b;
        temp += K;
int32_t result= temp >> n;
```

$$
\begin{aligned}
& a \cdot b \\
\overset{Q.n}{=} & (a \cdot 10^n) \cdot (b \cdot 10^n) \\
= & (a \cdot b) \cdot 10^{2n} \\
\neq & (a \cdot b) \cdot 10^n
\end{aligned}
$$

Division

```
int32_t      a = ...;
int32_t      b = ...;
int64_t   temp = (int64_t) a << n;
          temp += b / 2;
int32_t result = temp / b;
```

$$
\begin{aligned}
& \frac{a}{b} \overset{Q.n}{=} \frac{a \cdot 10^n}{b \cdot 10^n} \\
= & \frac{a}{b} \neq \frac{a}{b} \cdot 10^n
\end{aligned}
$$

- Siehe Implementierung in `fixedpoint.c`
- Vorsicht: Rundungsfehler durch Transformationen

Inhalt

Spezifikation: Simples Textstatistikprogramm

Anforderungen

- liest ASCII-Text über Standardeingabe ein
- zählt vorkommende Zeichen, Wörter und Zeilen
- Ausgabe: *<Anzahl Zeilen> <Anzahl Wörter> <Anzahl Zeichen>*

Umsetzungsversuch 1

```
static int e,n,j,o,y;int main(){for(++o;(n=-~getchar());e+=11==n,y++)
 o=n>0xe^012>n&&'`'^n^65?!n:!o?++j:o;printf("%8d%8d%8d\n",e^n,j+=!o&&y,y);}
```

- ✚ erfüllt Anforderungen
- ✗ schwer zu lesen
- ✗ noch schwerer zu verstehen

https://www.ioccc.org/2019/burton/prog.clean.c

Umsetzungsversuch 2

```c
#include <stdio.h>
typedef size_t CharCountTy; typedef size_t WordCountTy; typedef size_t LineCountTy;
static void inc_char_count(CharCountTy *c) {*c += 1;}
static void inc_word_count(WordCountTy *w) {*w += 1;}
static void inc_line_count(LineCountTy *l) {*l += 1;}
typedef struct {int character; int error; int done;} ReadResTy;
static int isWordTerminator(int c) {return c == ' ' || (c >= '\t' && c <= '\r');}
static int isLineTerminator(int c) {return c == '\n';}
static ReadResTy getCharacter(void) { ReadResTy r;
 r.character = getchar(); r.done = 0; r.error = 0;
 if(r.character == EOF) {if(feof(stdin)) {r.done = 1;} else {r.error = 1;}} return r;
}
int main(void) {
 CharCountTy char_count = 0; WordCountTy word_count = 0;
 LineCountTy line_count = 0; int in_word = 0; ReadResTy input;
 while((input = getCharacter()), !input.error && !input.done) {
  inc_char_count(&char_count);
  if(isWordTerminator(input.character) && in_word) {
   inc_word_count(&word_count); in_word = 0;
  } else if(!isWordTerminator(input.character)) {in_word = 1;}
  if(isLineTerminator(input.character)) {inc_line_count(&line_count);}
 }
 if(input.error) {return -1;} // Something went wrong...
 printf("%8lu%8lu%8lu\n", line_count, word_count, char_count); return 0;
}
```

✗ vielleicht etwas zu viel des Guten …

Gelungener Softwareentwurf

> *All problems in computer science can be solved by another level of indirection, except for the problem of too many layers of indirection.*
>
> *—David J. Wheeler*

- guter Softwareentwurf ist mehr Kunst als Wissenschaft
- keine Patentlösung

Ziele des Softwareentwurfs

Modifizierbarkeit: lokale Veränderbarkeit

- ⇝ Änderungen an Anforderungen umsetzbar
- ⇝ Fehler korrigierbar

Effizienz: optimaler Betriebsmittelbedarf

- wird häufig zu früh berücksichtigt

Verlässlichkeit: lange Zeit funktionsfähig ohne menschlichen Eingriff

- gutmütiges Ausfallverhalten
- muss von Anfang an eingeplant sein!

Verständlichkeit: Isolierung von

- Daten
- Algorithmen

Prinzipien des Softwareentwurfs

Abstraktion: wichtige Details hervorheben

Kapselung: unnötige Details verbergen

Einheitlichkeit: konsistente Notation

Vollständigkeit: alle wichtigen Aspekte berücksichtigt

Testbarkeit: muss von Anfang an eingeplant werden

C macht es einem hier nicht leicht

$\rightsquigarrow$ **disziplinierte Herangehensweise** notwendig!

Inhalt

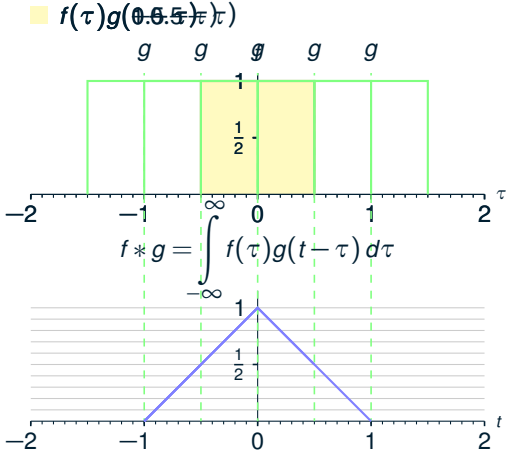
Hinweise zur Aufgabe Implementierung Filter

- Objekte identifizieren (z.B. Eingaben)
- Implementierung der Filterung durch **Faltung** (engl. convolution) mit Impulsantwort
  - f Signalwerte, g Filterwerte

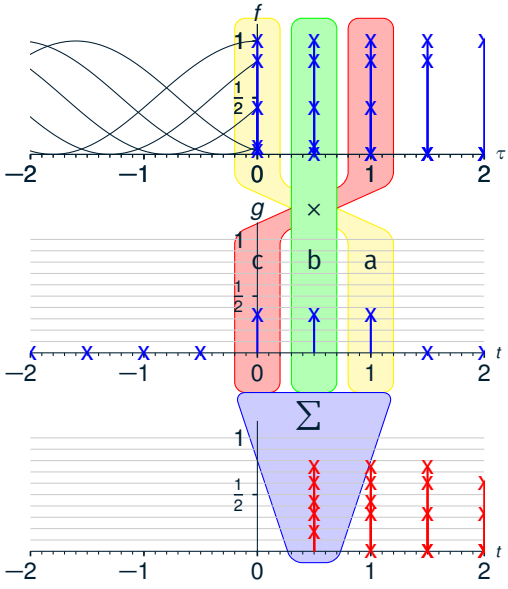
$$(f * g)[n] = \sum_{m=-\infty}^{\infty} f[m] \cdot g[n-m] \tag{1}$$

- Zunächst Verwendung von `float`, anschließend **Festkommaformat**

Beispiel: Faltung



Beispiel: Diskrete Faltung



$$(f * g)[n] = \sum_{m=-\infty}^{\infty} f[m] \cdot g[n-m]$$

Exkurs: Prüfsummen

- Verkürzte Repräsentation eines Datensatzes (Bsp.: Git Commithash)
- Aus Eingabedaten errechnet
- Nutzen: Überprüfung der Datenintegrität
- Anforderungen
  - Stabil
  - Effizient berechenbar
  - Zuverlässige Fehlererkennung
- Beispiel: Quersumme mit Zehner-Restklasse:

$$\mathrm{CHECK}(12345) = (1+2+3+4+5) \mod 10 = 5$$

$$\mathrm{CHECK}(12355) = (1+2+3+5+5) \mod 10 = 6$$

  $\rightsquigarrow$ Fehlermodell: Schützt vor allen Einzifferfehlern

  $\rightsquigarrow$ jedoch bspw. kein Schutz gegen Vertauschung:

$$\mathrm{CHECK}(12354) = (1+2+3+5+4) \mod 10 = 5$$

- Wichtig: Fehlermodell, welche Arten von Bitfehlern werden erkannt?

Hinweise zur Aufgabe

- **Einzelnen Filterschritt** implementieren (kein Burst-Filter)
- Verwendung von **Q-Notation**
- *Wichtig für die späteren Experimente*
  1. Berechnungsdauer möglichst kurz (Fehlerinjektion dauert sehr lange)
  2. wollen keine verpflichtende Nutzung von `floats`
  3. Später dann: Keine Verwendung von `printf` bei Fehlerinjektion
  4. Konzeption einer einfachen Prüfsumme
- Aspekte:
  - Nutzung abstrakter Schnittstellen
  - Einfluss von Schnittstellen auf Verlässlichkeit
  - Entwurfsentscheidungen und -abwägung in der Systemimplementierung

Literatur (1)


E. L. Oberstar.
Fixed-point representation & fractional math.
Technical report, Oberstar Consulting, August 2007.


J. Regher.
A quiz about integers in c.

Verlässliche Echtzeitsysteme - Übungen

Triple Modular Redundancy

Wintersemester 2024

Eva Dengler, Peter Wägemann

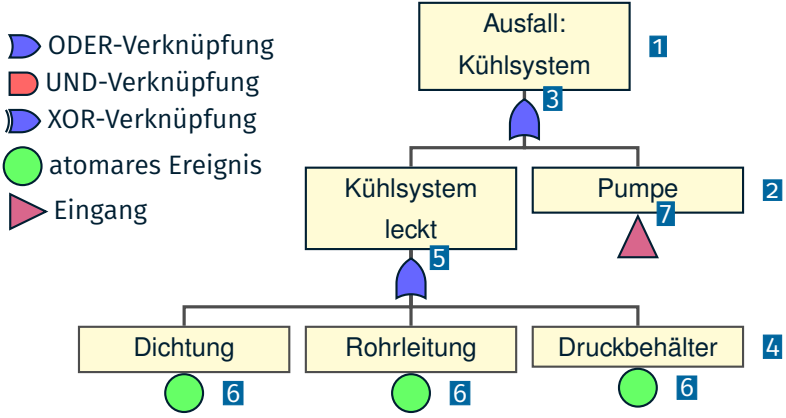
Friedrich-Alexander-Universität Erlangen-Nürnberg
Lehrstuhl Informatik 4 (Systemsoftware)
https://sys.cs.fau.de

Überblick

1. Wiederholung: Grundlagen Fehlerbäume
2. Wiederholung: Triple Modular Redundancy
3. Ausblick: Rechnerarchitektur, Replikation und Redundanz
4. Replikation von Code

Überblick

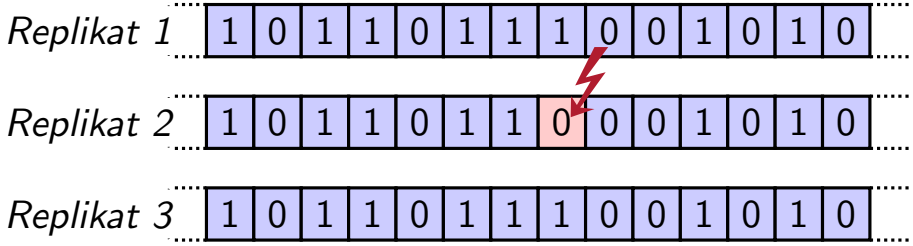
Fehlerbäume – Wiederholung



1. Schadensereignis
2. Ereignisse auf Ebene 2
3. Logische Verknüpfung
4. Ereignisse auf Ebene 3
5. Logische Verknüpfung
6. Atomare Ereignisse
7. Eingänge zerlegen den Fehlerbaum ↦ Neuer Teilbaum

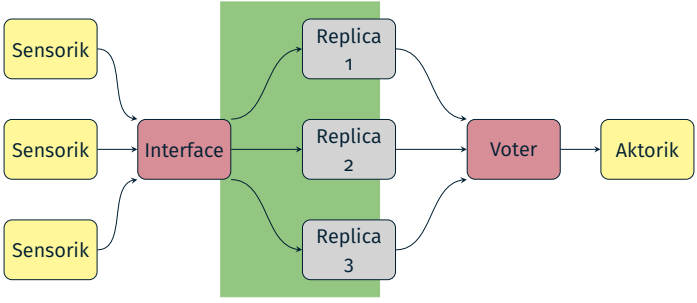
Überblick

Fehlerhypothese



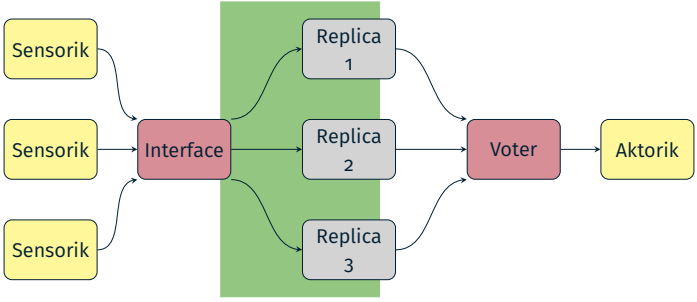
- **Wie viele Replikate** benötigt man zur Fehlermaskierung?
- Arten des Fehlverhaltens (von n Replikaten sind f fehlerhaft)
  1. fail-silent $\mapsto$ # Replikate: $n = f + 1$
  2. fail-consistent $\mapsto$ # Replikate: $n = 2f + 1$
  3. malicious $\mapsto$ # Replikate: $n = 3f + 1$, bösartige verteilte Systeme

Triple Modular Redundancy



- Schnittstelle sammelt Eingangsdaten (Replikdeterminismus)
- Insbesondere: Mögl. Mehrheitsentscheid für redundante Sensordaten
- Verteilt Daten und aktiviert Replikate
- Mehrheitsentscheider (Voter) wählt Ergebnis
- Ergebnis wird an Aktuator versendet

Triple Modular Redundancy



Redundanzbereich

Ausschließlich Replikatausführung

- Mehrheitsentscheid über Berechnungsergebnisse
- Erweiterung der Ausgangsseite mit Informationsredundanz

Replikdeterminismus

Replikat 1

```
void repl_1(void *p){
  ticks_t time =
    ezs_get_time();

  ...
}
```

Replikat 2

```
void repl_2(void *p){
  ticks_t time =
    ezs_get_time();

  ...
}
```

Replikat 3

```
void repl_3(void *p){
  ticks_t time =
    ezs_get_time();

  ...
}
```

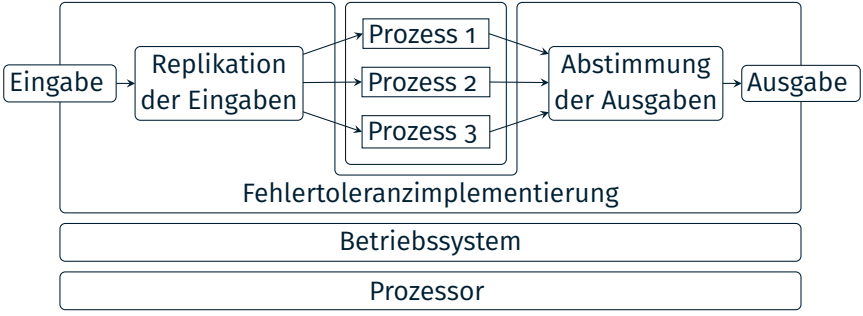
Sicherstellung Replikdeterminismus

- Globale diskrete Zeitbasis
- Einigung über Eingabewerte
- Statische Kontrollstruktur der Replikate
- Deterministische Algorithmen

Sicherstellung Systemverhalten

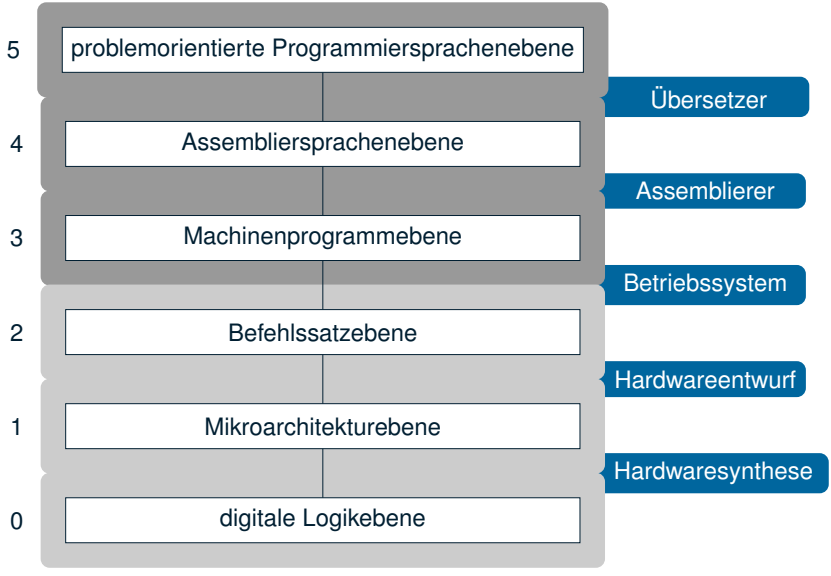
☞ Replikate müssen *innerhalb bestimmter Zeitspanne* terminieren

Process-Level Redundancy



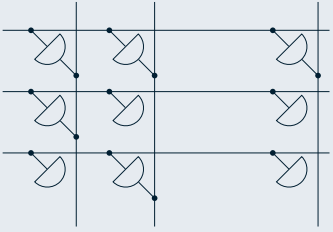
Überblick

Ebenen

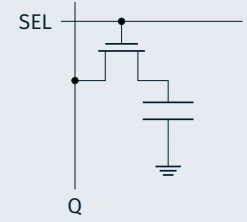

5
problemorientierte Programmiersprachenebene
Übersetzer
4
Assembliersprachenebene
Assemblierer
3
Machinenprogrammebene
Betriebssystem
2
Befehlssatzebene
Hardwareentwurf
1
Mikroarchitekturebene
Hardwaresynthese
0
digitale Logikebene

Digitale Logikebene

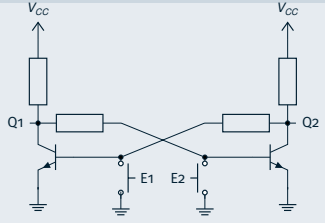
ROM



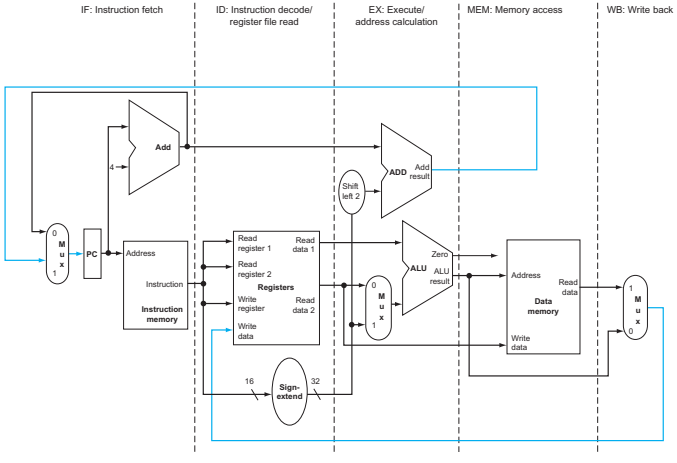
DRAM



RS - FlipFlop

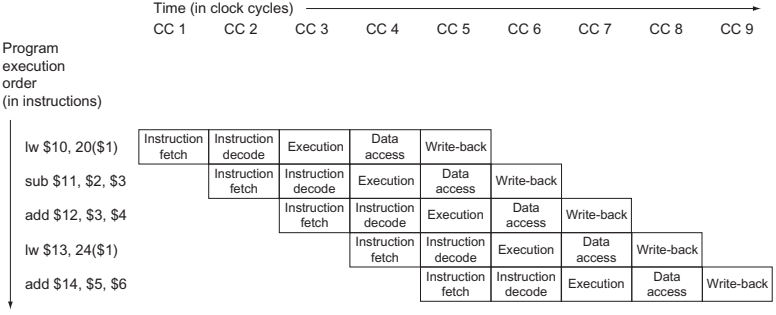


MIPS: Single-Cycle



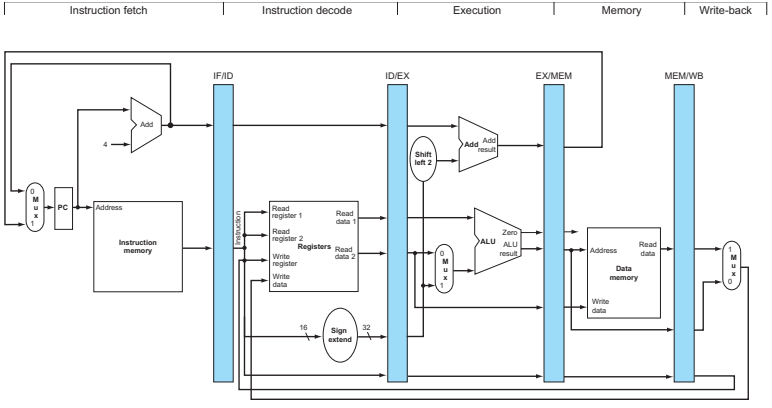
Source: D. A. Patterson und J. L. Hennessy, Computer organization and design: the hardware/software interface, 4th ed., 2012

MIPS: Pipelining



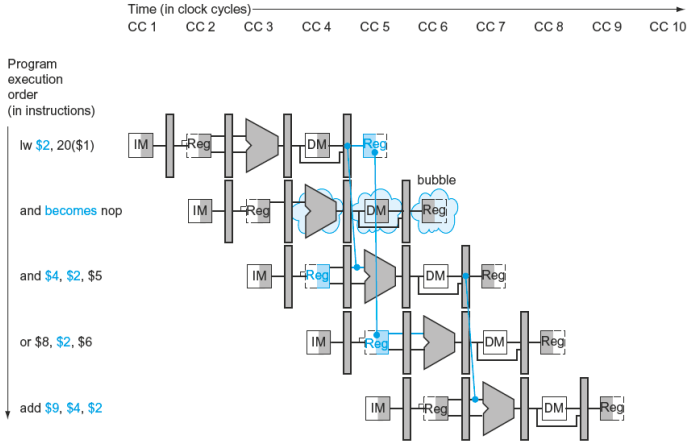
Source: D. A. Patterson und J. L. Hennessy, Computer organization and design: the hardware/software interface, 4th ed., 2012

MIPS: Pipelining



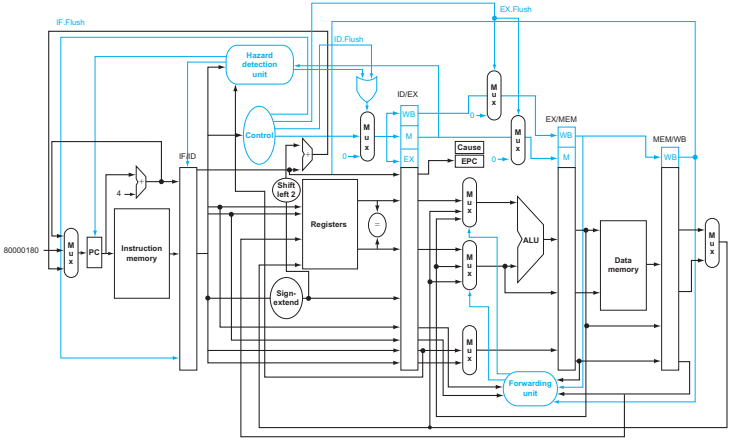
Source: D. A. Patterson und J. L. Hennessy, Computer organization and design: the hardware/software interface, 4th ed., 2012

MIPS: Pipelining



Source: D. A. Patterson und J. L. Hennessy, Computer organization and design: the hardware/software interface, 4th ed., 2012

MIPS: Pipelining



Source: D. A. Patterson und J. L. Hennessy, Computer organization and design: the hardware/software interface, 4th ed., 2012

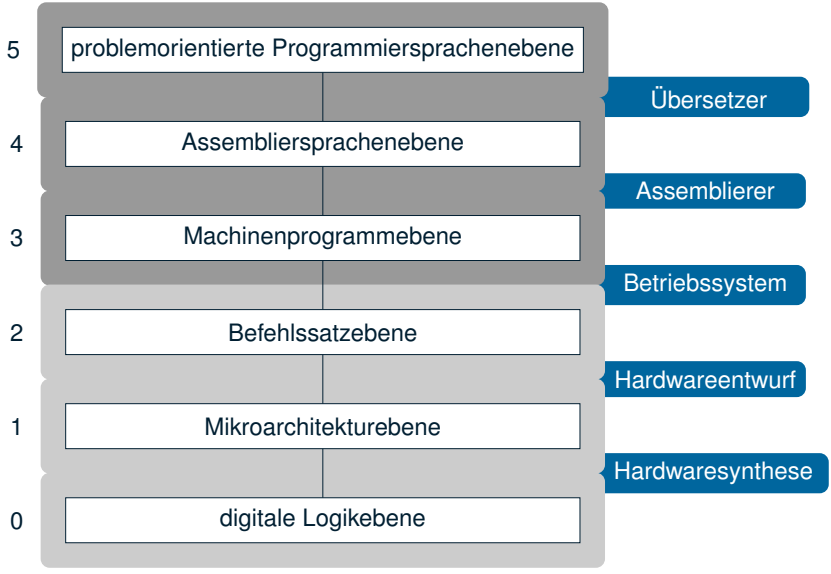
Eigenschaften von CPU-Architekturen

- Mikroprogrammierbar vs. Fixed-Function
- Out-of-Order-Prozessoren
- Sprungvorhersage
- Transaktionaler Speicher
- Superskalarität
- Mehrkernarchitekturen
- Hyperthreading
- ...

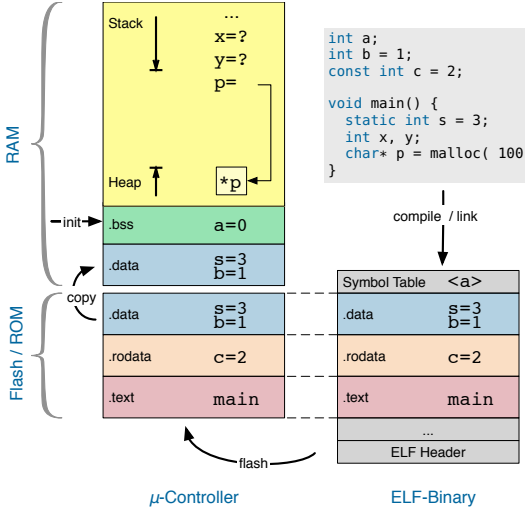
☞ All diese zusätzlichen Fehlerpunkte müssen im Fehlermodell berücksichtigt werden

☞ Ein Ein-Bit-Fehler in einer dieser Komponenten kann zu komplexen Mehrbitfehlern auf ISA-Ebene führen

Ebenen


5
problemorientierte Programmiersprachenebene
Übersetzer
4
Assembliersprachenebene
Assemblierer
3
Machinenprogrammebene
Betriebssystem
2
Befehlssatzebene
Hardwareentwurf
1
Mikroarchitekturebene
Hardwaresynthese
0
digitale Logikebene

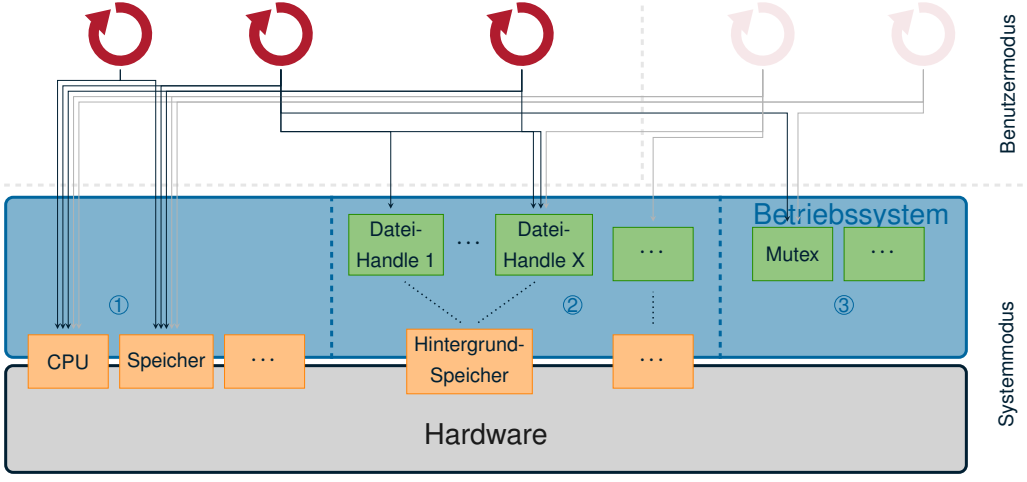
Speicherorganisation auf einem Mikrocontroller



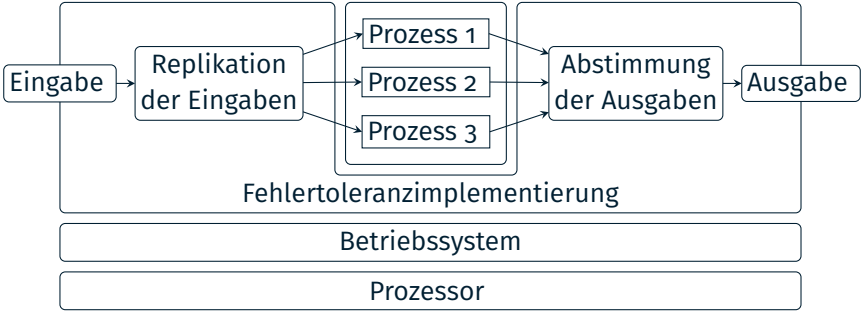

```c
int a;                          // a: global, uninitialized
int b = 1;                      // b: global, initialized
const int c = 2;                // c: global, const

void main() {
  static int s = 3;             // s: local, static, initialized
  int x, y;                     // x: local, auto; y: local, auto
  char* p = malloc( 100 );      // p: local, auto; *p: heap (100 byte)
}
```

Quellprogramm

Betriebssystem



Process-Level Redundancy



C-Code vs. Assembler-Code

C-Code

```
int a;
int b = 1;
const int c = 2;
void main() {
  static int s = 3;
  int x, y;
  char* p =
       malloc(100);
}
```

Assembler-Code

```
4004f0 <main>:
  4004f0: push   %rbp
  4004f1: mov    %rsp,%rbp
  4004f4: sub    $0x10,%rsp
  4004f8: movabs $0x64,%rdi
  400502: callq  4003e0 <malloc@plt>
  400507: mov    %rax,-0x10(%rbp)
  40050b: add    $0x10,%rsp
  40050f: pop    %rbp
  400510: retq
```

Wo können Datenfehler auftreten?

1. RAM: `-0x10(%rbp)`
   - ⇝ Stack, globale Daten, Heap, (Programmcode)
2. Allgemeine CPU-Register: `%rsp`
3. Sonstige CPU-Register: `%rip`, `%rflags`

Überblick

Code-Replikation: Stringification

Stringification von CPP

```
#define CMP_FUNC(pre, repl, type, op) \
    type pre##repl(type a, type b) {  \
        return a op b ? a : b;        \
    }

CMP_FUNC(max, 1, int, >); // Funktion ?max1
CMP_FUNC(max, 2, int, >); // Funktion ?max2
...
CMP_FUNC(min, 1, int, <); // Funktion ?min1
```

- Verwendung des C-Präprozessors (CPP)
- ##: „Token Pasting Operator“
- Konkatenieren zweier Token zu einem
- Aufruf & Deklaration müssen erstellt werden

  ☞ Es geht eleganter ...

C++ Templates

C++ Template

```
template <typename T>
T max(T x, T y) {
  T value;
  if (x < y) value = y;
  else value = x;
  return value;
}
double md = max<double>(2.3, 4.2);
auto   mi = max<int>(23U, 42);
```

- Templates ermöglichen generische Programmierung
- Wiederverwendung durch **Parametrisierung**
- Unterscheidung von Funktions- & Klassen-Templates
- Expansion zur Compilezeit $\rightsquigarrow$ Quelltext muss verfügbar sein (Header)
- „Code Bloat" beim Compilieren $\rightarrow$ **nutzbar für Replikation von Code**

C++ Templates

- Explizite Typen als Templateparameter möglich
- Nutzbar zum „Zählen” von Templates

Indizierte Template-Spezialisierung

```
template <typename T, unsigned INDEX>
T max(T x, T y) {
  T value;
  if (x < y)
    value = y;
  else
    value = x;
  return value;
}
...
auto m0 = max<int, 0>(23, 42);
auto m1 = max<int, 1>(23, 42);
```

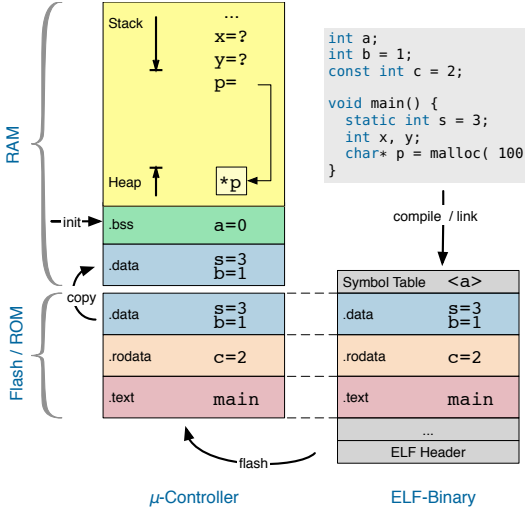
Einbinden von C++-Code in C

C Linkage

```
// C++ code
extern "C" void f(int); // one way
extern "C" {            // another way
  int g(double);
  double h();
};
void code(int i, double d)
{
  f(i);
  int ii = g(d);
  double dd = h();
  // ...
}
```

- C++ betreibt name mangling
  ⇒ `int test(int)` ↦ `_Z4testi`
- Name mangling verhindern
  ⇒ C++-Code aus C-Code aufrufbar

Speicherorganisation und Sektionsgrößen




```c
int a;                          // a: global, uninitialized
int b = 1;                      // b: global, initialized
const int c = 2;                // c: global, const

void main() {
  static int s = 3;             // s: local, static, initialized
  int x, y;                     // x: local, auto; y: local, auto
  char* p = malloc( 100 );      // p: local, auto; *p: heap (100 byte)
}
```

Quellprogramm

```
$ size --format=berkeley a.out
text data bss  dec hex filename
2592    8   4 2604 a2c     a.out
```

Sektionsgrößen in Byte,
„`text`“ umfasst hier: `.text` + `.rodata`

Umgang mit Assembly-Code I

nm: Ausgabe der Symboltabelle

```
0000000000004028 D var_initialized
...
0000000000201028 B __bss_start
0000000000201028 b var_uninitialized
...
000000000000061a T main
0000000000000580 t register_tm_clones
0000000000000510 T _start
...
000000000000066d t int max<int,0u>(int,int)
000000000000067c t int max<int,1u>(int,int)
```

- Nützliche Optionen
  - `-C, --demangle`: Dekodieren der C++-Namensmangelung:

    `_Z3maxIiLj0EET_S0_S0_` ⇒ `int max<int, 0u>(int, int)`
  - `-S, --print-size`: Ausgabe der Symbolgrößen

    `000000000000061 a 00000028 T main`

Umgang mit Assembly-Code II

`objdump`: Ausgabe von Informationen über Objektdateien

```
int main(){
 4007cd:   55                    push   %rbp
 4007ce:   48 89 e5              mov    %rsp,%rbp
 4007d1:   48 83 ec 10           sub    $0x10,%rsp
int a = max<int>(23U, 42);
 4007d5:   be 2a 00 00 00        mov    $0x2a,%esi
 4007da:   bf 17 00 00 00        mov    $0x17,%edi
 4007df:   e8 04 01 00 00        callq
                                 4008e8 <_Z3maxIiET_S0_S0_>
 4007e4:   89 45 fc              mov    %eax,-0x4(%rbp)
std::cout << a << "\n";
 4007e7:   8b 45 fc              mov    -0x4(%rbp),%eax
 ...
```

- Nützliche Optionen
  - `-S`: Ausgabe von Quell-Code im Assembly-Code (Debug-Symbole notwendig)
  - `-D`: alle Sektionen disassemblieren

Verlässliche Echtzeitsysteme - Übungen

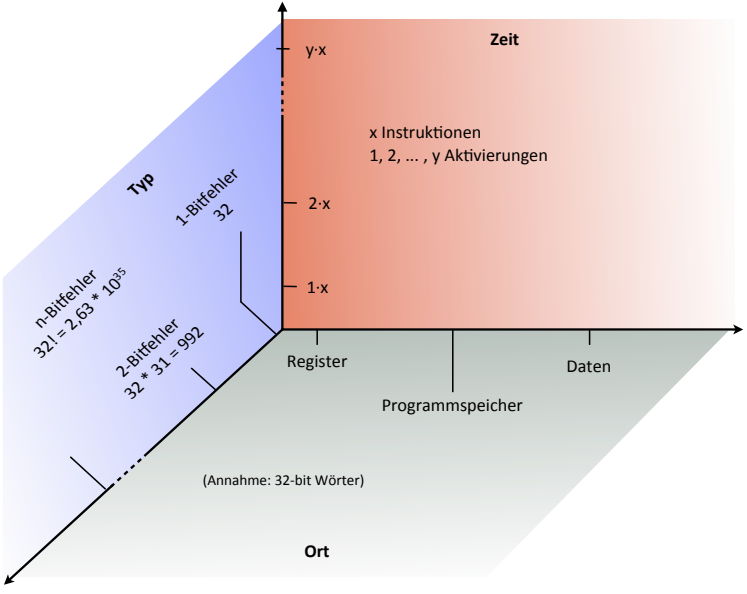
Arbeiten mit Binärdateien und Assembler, Codierung

Wintersemester 2024

Eva Dengler, Peter Wägemann

Friedrich-Alexander-Universität Erlangen-Nürnberg
Lehrstuhl Informatik 4 (Systemsoftware)
https://sys.cs.fau.de

Fehlerraum


Zeit
y·x
x Instruktionen
1, 2, … , y Aktivierungen
2·x
1·x
Typ
1-Bitfehler
32
n-Bitfehler
32! = 2,63 * 10^35
2-Bitfehler
32 * 31 = 992
Register
Daten
Programmspeicher
(Annahme: 32-bit Wörter)
Ort

C-Code vs. Assembler-Code

C-Code

```
int a;
int b = 1;
const int c = 2;
void main() {
  static int s = 3;
  int x, y;
  char* p =
       malloc(100);
}
```

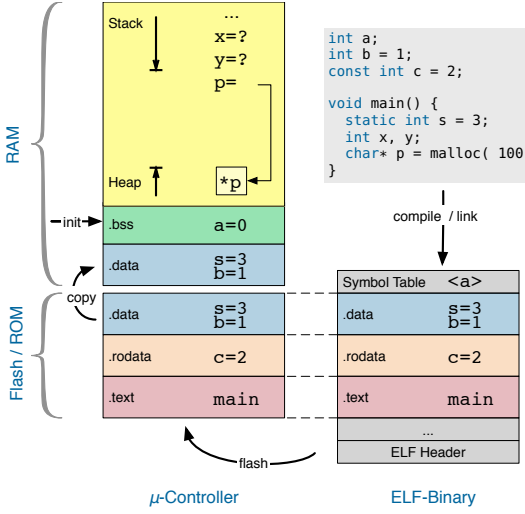
Assembler-Code

```
4004f0 <main>:
  4004f0: push   %rbp
  4004f1: mov    %rsp,%rbp
  4004f4: sub    $0x10,%rsp
  4004f8: movabs $0x64,%rdi
  400502: callq  4003e0 <malloc@plt>
  400507: mov    %rax,-0x10(%rbp)
  40050b: add    $0x10,%rsp
  40050f: pop    %rbp
  400510: retq
```

Wo können Datenfehler auftreten?

1. RAM: `-0x10(%rbp)`
   - ⇝ Stack, globale Daten, Heap, (Programmcode)
2. Allgemeine CPU-Register: `%rsp`
3. Sonstige CPU-Register: `%rip`, `%rflags`

Speicherorganisation und Sektionsgrößen




```c
int a;                          // a: global, uninitialized
int b = 1;                      // b: global, initialized
const int c = 2;                // c: global, const

void main() {
  static int s = 3;             // s: local, static, initialized
  int x, y;                     // x: local, auto; y: local, auto
  char* p = malloc( 100 );      // p: local, auto; *p: heap (100 byte)
}
```

Quellprogramm

```
$ size --format=berkeley a.out
text data bss  dec hex filename
2592    8   4 2604 a2c     a.out
```

Sektionsgrößen in Byte,
„`text`“ umfasst hier: `.text` + `.rodata`

Umgang mit Assembly-Code I

nm: Ausgabe der Symboltabelle

```
0000000000004028 D var_initialized
...
0000000000201028 B __bss_start
0000000000201028 b var_uninitialized
...
000000000000061a T main
0000000000000580 t register_tm_clones
0000000000000510 T _start
...
000000000000066d t int max<int,0u>(int,int)
000000000000067c t int max<int,1u>(int,int)
```

- Nützliche Optionen
  - `-C, --demangle`: Dekodieren der C++-Namensmangelung:

    `_Z3maxIiLj0EET_S0_S0_` ⇒ `int max<int, 0u>(int, int)`
  - `-S, --print-size`: Ausgabe der Symbolgrößen

    `000000000000061 a 00000028 T main`

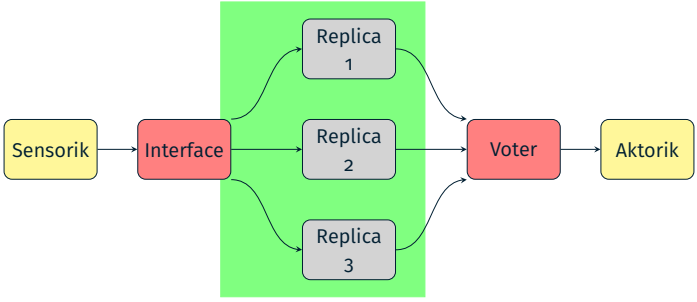
Umgang mit Assembly-Code II

`objdump`: Ausgabe von Informationen über Objektdateien

```
int main(){
 4007cd:   55                    push   %rbp
 4007ce:   48 89 e5              mov    %rsp,%rbp
 4007d1:   48 83 ec 10           sub    $0x10,%rsp
int a = max<int>(23U, 42);
 4007d5:   be 2a 00 00 00        mov    $0x2a,%esi
 4007da:   bf 17 00 00 00        mov    $0x17,%edi
 4007df:   e8 04 01 00 00        callq
                                 4008e8 <_Z3maxIiET_S0_S0_>
 4007e4:   89 45 fc              mov    %eax,-0x4(%rbp)
std::cout << a << "\n";
 4007e7:   8b 45 fc              mov    -0x4(%rbp),%eax
 ...
```

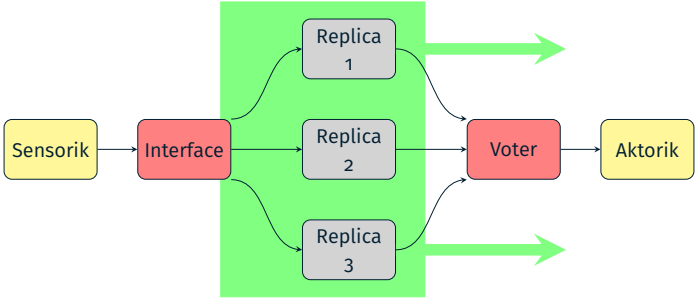
- Nützliche Optionen
  - `-S`: Ausgabe von Quell-Code im Assembly-Code (Debug-Symbole notwendig)
  - `-D`: alle Sektionen disassemblieren

Klassische “Triple Modular Redundancy” (TMR)



- Schnittstelle sammelt Eingangsdaten (Replikdeterminismus)
- Verteilt Daten und aktiviert Replikate
- Mehrheitsentscheider (Voter) wählt Ergebnis
- Ergebnis wird an Aktuator versendet

Klassische “Triple Modular Redundancy” (TMR)



Redundanzbereich

Ausschließlich Replikatausführung

- ⇝ Erweiterung der Ausgangsseite mit Informationsredundanz
- ⇝ Mehrheitsentscheid über codierte Prüfsumme

Erweiterte arithmetische Codierung

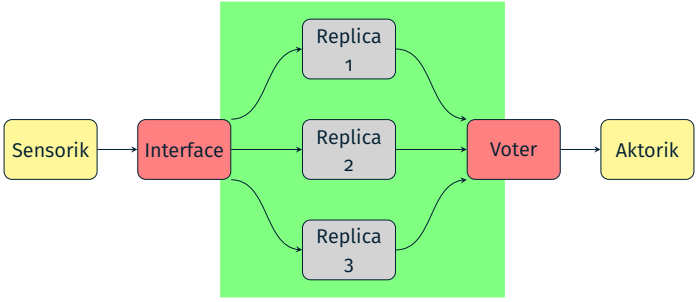
nach Forin 1989: “Vital coded microprocessor principles and application for various transit systems”

- Arithmetisch codierter Wert V_C
- Ausgangswert
- Schlüssel
- Variablenspezifische Signatur
- Zeitstempel

Wertebereichseinschränkungen

- Schlüssel A sollte so groß wie möglich sein:
  - $\rightsquigarrow$ Möglichst geringe Restfehlerwahrscheinlichkeit ($P = 1/A$)
- Wertebereich des dynamischen Zeitstempels
  - $D = \{x | x \in \mathbb{N}_0 \wedge x \leq D_{max}\}$
  - Zeitstempel darf überlaufen: $D_{max} + 1 = 0$
- Für jede Signatur $B_* \in \mathbb{N}$ muss dann gelten
  - $B_* + D_{max} < A$
  - Die minimale Distanz zwischen jeweils zwei Signaturen im System muss größer D_{max} sein: $\forall i, j : |B_i - B_j| > D_{max}$
  - Sämtliche Distanzen zwischen jeweils zwei Signaturen müssen unterschiedlich sein: $\forall i : \forall j, k : |B_i - B_j| = |B_k - B_j| \Rightarrow i = k$

Erweiterung I – codierte Ausgangswerte



- Replikate liefern arithmetisch codierte Ergebnisse
- Mehrheitsentscheid auf codierten Prüfsummen
- Übertragung codierter Ergebnisse

EAN Vergleichsoperator

Vereinfachung für diese Übungsaufgabe

- Kein Zeitstempel

- Voting basiert auf codierter Vergleichsoperation:

$\rightsquigarrow X_C \equiv Y_C \Rightarrow X * A + B_X \equiv Y * A + B_Y$

- Im fehlerfreien Fall gilt:

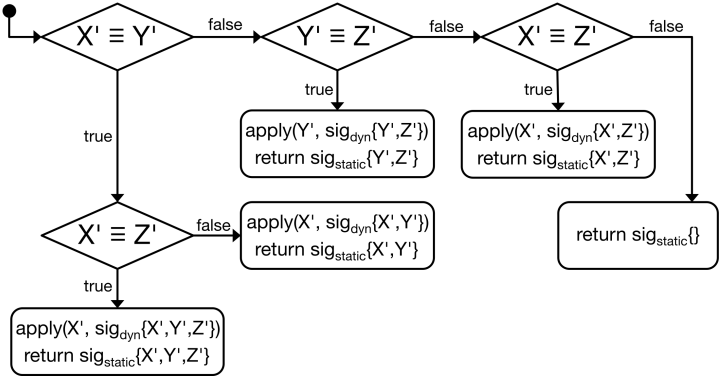
$X = Y$, $A = A$ aber $B_X \neq B_Y$!

  - *Rohwerte* sind identisch
  - Schlüssel ist per Definition identisch
  - Signaturen sind unterschiedlich (aber konstant!)

Bestimmung der Gleichheit durch Differenzbildung:

$\rightsquigarrow X_C - Y_C = B_X - B_Y = const.$

Codierter Mehrheitsentscheid



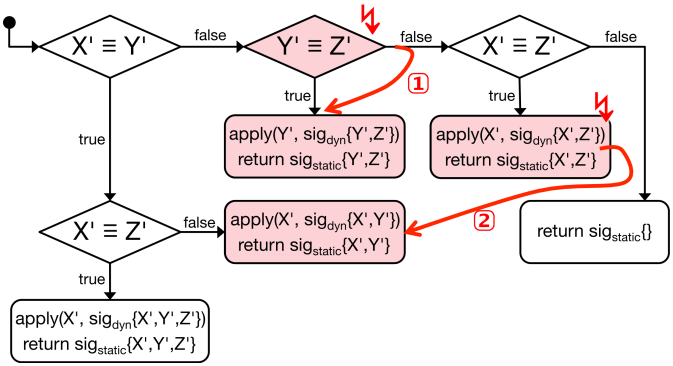
- Bestimmung von dynamischer und statischer Signatur:
  - → $sig_{dyn}(X', Y') : X' \equiv Y' \Rightarrow X' - Y'$
  - → $sig_{static}(X', Y') : X' \equiv Y' \Rightarrow B_X - B_Y$
- Für die Signaturen muss gelten: $B_X > B_Y > B_Z$

Codierter Mehrheitsentscheid (Forts.)

1. Vergleichsoperation wird durchgeführt (z. B. $X' \equiv Y' \wedge X' \equiv Z'$)
   - Berechnung von sig_{dyn}
   - Vergleich mit sig_{static}
2. Verzweigungsentscheidung wird nachberechnet:
   - Wiederholte (redundante) Berechnung von sig_{dyn}
   - Erneuter Vergleich: $sig_{dyn} = sig_{static}$
   - Addiere sig_{dyn} (*apply*) zum gewählten Ergebnis
3. Konstante Signatur des durchlaufenen Zweiges identifiziert Gewinner (Rückgabewert: sig_{static})
   - Aktor wählt entsprechendes Replikatergebnis
   - Führt inverse Operation zu *apply* durch

Im Voter wurde die *dynamisch berechnete Signatur der Verzweigungsentscheidung* hinzu addiert. Im Aktor wird mit der entsprechenden *konstanten Signatur zurückgerechnet*.

Codierter Mehrheitsentscheid - Fehlerfall



1. Falsche Verzweigungsentscheidung: ($Y' \not\equiv Z'$)
   - Y' wird als korrekt angenommen, sig_{dyn} wird berechnet
   - allerdings ist sig_{dyn} tatsächlich $\neq sig_{static}$
   - Fehler wird vor dem *apply* erkannt
2. Falscher (plötzlicher) Sprung

Ausgangslage

- Wähle $A = 601$
- Initiale Ergebniskodierung in den Replikaten:

	X	Y	Z
Wert	7	5	7
B	37	23	5
Kodiert	$X' = A * 7 + 37$ $= 4244$	$Y' = A * 5 + 23$ $= 3028$	$Z' = A * 7 + 5$ $= 4212$

- Berechnung der statischen Signaturen **vorab, statisch**:

$$sig_{static}\{X', Y', Z'\} = (B_X - B_Y) + (B_X - B_Z) = 46$$
$$sig_{static}\{X', Y'\} = (B_X - B_Y) = 14$$
$$sig_{static}\{Y', Z'\} = (B_Y - B_Z) = 18$$
$$sig_{static}\{X', Z'\} = (B_X - B_Z) = 32$$

Regulärer Durchlauf

- Das eigentliche Voting geschieht dann zur Laufzeit:

1. $X' = Y'$?

$$X' - Y' \stackrel{?}{=} sig_{static}\{X', Y'\} \Leftrightarrow 4244 - 3028 = 1216 \stackrel{?}{=} 14 \Leftrightarrow \textit{false}$$

2. $Y' = Z'$?

$$Y' - Z' \stackrel{?}{=} sig_{static}\{Y', Z'\} \Leftrightarrow 3028 - 4212 = -1184 \stackrel{?}{=} 18 \Leftrightarrow \textit{false}$$

3. $X' = Z'$?

$$X' - Z' \stackrel{?}{=} sig_{static}\{X', Z'\} \Leftrightarrow 4244 - 4212 = 32 \stackrel{?}{=} 32 \Leftrightarrow \textit{true}$$

4. Berechnung der dynamischen Signaturen zur Laufzeit:

$$sig_{dyn}\{X', Z'\} = (X' - Z') = (4244 - 4212) = 32$$

Regulärer Durchlauf (II)

5. $sig_{dyn}\{X', Z'\} \stackrel{?}{=} sig_{static}\{X, Z\} \iff 32 \stackrel{?}{=} 32 \iff \textit{true}$
6. $X' = apply\left(X', sig_{dyn}\{X', Z'\}\right) = 4276$
7. $B_E \leftarrow sig_{static}\{X', Z'\} = 32$
8. $return(32)$
9. Nachschlagen der zugrundeliegenden Variable mittels $B_{dyn} = 32$ (erste Variable der Konsensmenge), basierend auf den vorberechneten statischen Werten:

$$\left(result_{variable}, result_{extrasignature}\right) \leftarrow (X', B_X)$$

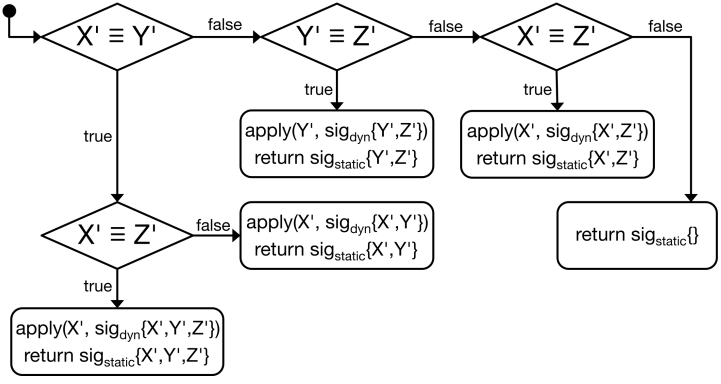
10. $inv\_apply\left(result_{variable}, B_{dyn}\right) = inv\_apply\left(4276, 32\right) = 4276 - 32 = 4244$

Regulärer Durchlauf (III)

11. Signaturverifikation:

$$check(4244, A, B_X)\text{: } \frac{4244 - B_X}{A} = 7 \text{ Rest: } 0$$

12. Ergebnis erfolgreich dekodiert: 7



Es hat somit eine erfolgreiche Einigung auf die Konsensmenge $\{X', Z'\}$ stattgefunden.

Ausgangslage (unverändert)

- Wähle $A = 601$
- Initiale Ergebniskodierung in den Replikaten:

	X	Y	Z
Wert	7	5	7
B	37	23	5
Kodiert	$X' = A * 7 + 37$ $= 4244$	$Y' = A * 5 + 23$ $= 3028$	$Z' = A * 7 + 5$ $= 4212$

- Berechnung der statischen Signaturen **vorab, statisch**:

$$sig_{static}\{X', Y', Z'\} = (B_X - B_Y) + (B_X - B_Z) = 46$$
$$sig_{static}\{X', Y'\} = (B_X - B_Y) = 14$$
$$sig_{static}\{Y', Z'\} = (B_Y - B_Z) = 18$$
$$sig_{static}\{X', Z'\} = (B_X - B_Z) = 32$$

Fehlerszenario ①

1. $X' = Y'$?

$$X' - Y' \stackrel{?}{=} sig_{static}\{X', Y'\} \Leftrightarrow 4244 - 3028 = 1216 \stackrel{?}{=} 14 \Leftrightarrow \textit{false}$$

2. $Y' = Z'$?

$$Y' - Z' \stackrel{?}{=} sig_{static}\{Y', Z'\} \Leftrightarrow 3028 - 4212 = -1184 \stackrel{?}{=} 18 \Leftrightarrow \textit{false}$$

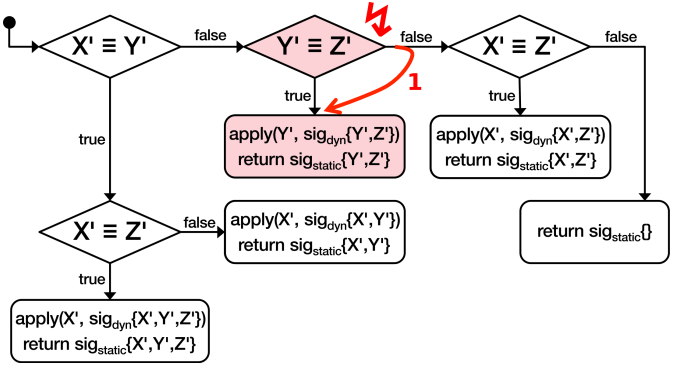
3. Hier tritt nun der Operatorfehler ein, die falsche Verzweigung ① wird aus gewählt
4. Berechnung der dynamischen Signaturen zur Laufzeit:

$$sig_{dyn}\{Y', Z'\} = (Y' - Z') = (3028 - 4212) = -1184$$

5. $sig_{dyn}\{Y', Z'\} \stackrel{?}{=} sig_{static}\{Y, Z\} \Leftrightarrow -1184 \stackrel{?}{=} 18 \Leftrightarrow \textit{false}$

Fehlerszenario ① (II)

6. Fehler erfolgreich detektiert: Vergleich zwischen sig_{dyn} und sig_{static} für Konsensmenge $\{Y', Z'\}$ fehlgeschlagen



Ausgangslage (unverändert)

- Wähle $A = 601$
- Initiale Ergebniskodierung in den Replikaten:

	X	Y	Z
Wert	7	5	7
B	37	23	5
Kodiert	$X' = A * 7 + 37$ $= 4244$	$Y' = A * 5 + 23$ $= 3028$	$Z' = A * 7 + 5$ $= 4212$

- Berechnung der statischen Signaturen **vorab, statisch**:

$$sig_{static}\{X', Y', Z'\} = (B_X - B_Y) + (B_X - B_Z) = 46$$
$$sig_{static}\{X', Y'\} = (B_X - B_Y) = 14$$
$$sig_{static}\{Y', Z'\} = (B_Y - B_Z) = 18$$
$$sig_{static}\{X', Z'\} = (B_X - B_Z) = 32$$

Fehlerszenario ②

1. $X' = Y'$?

$$X' - Y' \stackrel{?}{=} sig_{static}\left\{X', Y'\right\} \Leftrightarrow 4244 - 3028 = 1216 \stackrel{?}{=} 14 \Leftrightarrow \textit{false}$$

2. $Y' = Z'$?

$$Y' - Z' \stackrel{?}{=} sig_{static}\left\{Y', Z'\right\} \Leftrightarrow 3028 - 4212 = -1184 \stackrel{?}{=} 18 \Leftrightarrow \textit{false}$$

3. $X' = Z'$?

$$X' - Z' \stackrel{?}{=} sig_{static}\left\{X', Z'\right\} \Leftrightarrow 4244 - 4212 = 32 \stackrel{?}{=} 32 \Leftrightarrow \textit{true}$$

4. Berechnung der dynamischen Signaturen zur Laufzeit:

$$sig_{dyn}\left\{X', Z'\right\} = (X' - Z') = (4244 - 4212) = 32$$

Fehlerszenario ②

5. $sig_{dyn}\{X', Z'\} \stackrel{?}{=} sig_{static}\{X, Z\} \Leftrightarrow 32 \stackrel{?}{=} 32 \Leftrightarrow \textit{true}$
6. $X' = apply\left(X', sig_{dyn}\{X', Z'\}\right) = 4276$
7. Hier tritt nun der Kontrollflussfehler ② ein
8. $B_E \leftarrow sig_{static}\{X', Y'\} = 14$
9. $return(14)$
10. Nachschlagen der zugrundeliegenden Variable mittels $B_{dyn} = 14$ (erste Variable der Konsensmenge), basierend auf den vorberechneten statischen Werten:

$$\left(result_{variable}, result_{extrasignature}\right) \leftarrow (X', B_X)$$

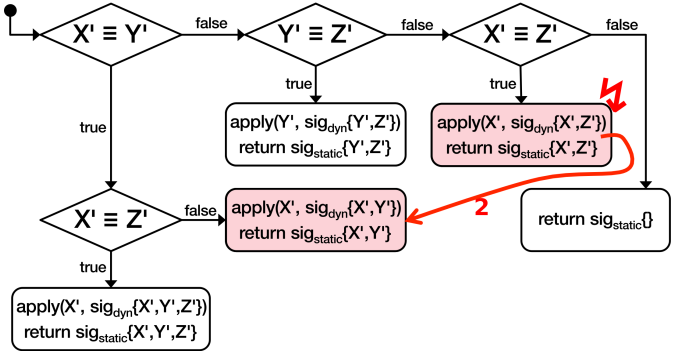
11. $inv\_apply\left(result_{variable}, B_{dyn}\right) = inv\_apply\left(4276, 14\right) = 4276 - 14 = 4262$

Fehlerszenario ② (II)

12. Signaturverifikation:

$$check(4262, A, B_X): \frac{4262 - B_X}{A} = 7 \text{ Rest: } 18$$

13. Fehler erfolgreich detektiert: Dekodieren schlägt fehl



Hinweise zur Aufgabenbearbeitung

Analyse der Toleranzmaßnahmen

- Fehlerraumanalyse
  - Einfluss des Speichers
  - Einfluss der Laufzeit

- Aufrufgraph

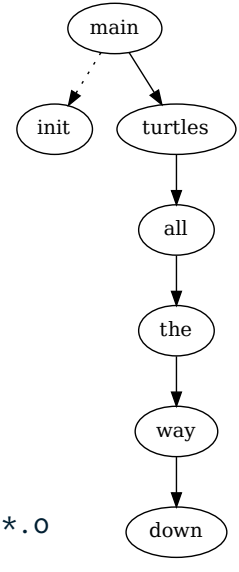
  `make callgraph_{baseline,tmr,ean}`

- ELF

  `build/{baseline,tmr,ean}.elf`

- Objektdateien

  `build/CMakeFiles/{baseline,tmr,ean}.dir/src/*.o`



Aufgabenstellung

Aufgabe

- Absichern des Voters per EAN mittels CoRed-Ansatz
- Berechnungen mit codierten Werten
  - Berücksichtigung der (statischen) Signaturen

  ⇝ Eigene Operationen mit konstanten Signaturwerten notwendig

  ⇝ bspw. eigenes `equals` anstatt ==

Verlässliche Echtzeitsysteme - Übungen

Testen

Wintersemester 2024

Eva Dengler, Peter Wägemann

Friedrich-Alexander-Universität Erlangen-Nürnberg
Lehrstuhl Informatik 4 (Systemsoftware)
https://sys.cs.fau.de

Überblick

Defensives Programmieren

- C bietet viele subtile Fehlermöglichkeiten
- Wie verhält man sich als Programmierer richtig?
- *Defensives Programmieren*

$\rightsquigarrow$ beispielhaft anhand von Ganzzahloperationen

Addition

Was soll da schon schiefgehen...

```
unsigned int func(unsigned int a, unsigned int b) {
  return a + b;
}
```

Vorbedingungstest

```
#include <limits.h>
unsigned int func(unsigned int a, unsigned int b) {
  if (UINT_MAX - a < b) { raise("wraparound"); }
  return a + b;
}
```

Nachbedingungstest

```
unsigned int func(unsigned int a, unsigned int b) {
  unsigned int ret = a + b;
  if (ret < a) { raise("wraparound"); }
  return ret;
}
```

Subtraktion

Was soll da schon schiefgehen...

```
unsigned int func(unsigned int a, unsigned int b) {
  return a - b;
}
```

Vorbedingungstest

```
unsigned int func(unsigned int a, unsigned int b) {
  if (a < b) { raise("wraparound"); }
  return a - b;
}
```

Nachbedingungstest

```
unsigned int func(unsigned int a, unsigned int b) {
  unsigned int ret = a - b;
  if (ret > a) { raise("wraparound"); }
  return ret;
}
```

Multiplikation

Was soll da schon schiefgehen...

```
unsigned int func(unsigned int a, unsigned int b) {
  return a * b;
}
```

Vorbedingungstest

```
#include <limits.h>
unsigned int func(unsigned int a, unsigned int b) {
  if (a == 0 or b == 0) { return 0; }
  if (UINT_MAX / a < b) { raise("wraparound"); }
  return a * b;
}
```

Explizite Typumwandlung

Was soll da schon schiefgehen...

```
unsigned int func(signed int a) {
  return (unsigned int) a; /* keine Compilerwarnung wg. Cast */
}
```

Vorbedingungstest

```
unsigned int func(signed int a) {
  if (a < 0) { raise("wraparound"); }
  return (unsigned int) a;
}
```

Explizite Typumwandlung

Was soll da schon schiefgehen...

```
unsigned char func(unsigned long int a) {
  return (unsigned char) a; /* keine Compilerwarnung wg. Cast */
}
```

Vorbedingungstest

```
unsigned char func(unsigned long int a) {
  if (a > UCHAR_MAX) { raise("overflow"); }
  return (unsigned char) a; /* keine Compilerwarnung wg. Cast */
}
```

Explizite Typumwandlung

Was soll da schon schiefgehen...

```
signed char func(unsigned long int a) {
  return (signed char) a; /* keine Compilerwarnung wg. Cast */
}
```

Vorbedingungstest

```
#include <limits.h>
signed char func(unsigned long int a) {
  if (a > SCHAR_MAX) { raise("overflow"); }
  return (signed char) a;
}
```

Explizite Typumwandlung

Was soll da schon schiefgehen...

```
signed char func(signed long int a) {
  return (signed char) a; /* keine Compilerwarnung wg. Cast */
}
```

Vorbedingungstest

```
#include <iso646.h>
#include <limits.h>
signed char func(signed long int a) {
  if (a < SCHAR_MIN or SCHAR_MAX < a) { raise("overflow"); }
  return (signed char) a; /* keine Compilerwarnung wg. Cast */
}
```

Addition

Was soll da schon schiefgehen...

```
signed int func(signed int a, signed int b) {
  return a + b;
}
```

Vorbedingungstest

```
#include <iso646.h>
#include <limits.h>
signed int func(signed int a, signed int b) {
  if ((b > 0 and a > INT_MAX - b)
   or (b < 0 and a < (INT_MIN - b))) { raise("overflow"); }
  return a + b;
}
```

Subtraktion

Was soll da schon schiefgehen...

```
signed int func(signed int a, signed int b) {
  return a - b;
}
```

Vorbedingungstest

```
#include <iso646.h>
#include <limits.h>
signed int func(signed int a, signed int b) {
  if ((b > 0 and a < INT_MIN + b)
   or (b < 0 and a > INT_MAX + b)) { raise("overflow"); }
  return a - b;
}
```

Division

Was soll da schon schiefgehen...

```
signed long func(signed long a, signed long b) {
  return a / b;
}
```

Vorbedingungstest

```
#include <iso646.h>
#include <limits.h>
signed long func(signed long a, signed long b) {
  if (b == 0) { raise("division by 0"); }
  return a / b;
}
```

Division

- Reicht das schon?

Was soll da schon schiefgehen...

```
signed long func(signed long a, signed long b) {
  if (b == 0) { raise("division by 0"); }
  return a / b;
}
```

Vorbedingungstest

```
#include <iso646.h>
#include <limits.h>
signed long func(signed long a, signed long b) {
  if (b == 0) { raise("division by zero"); }
  if (a == LONG_MIN and b == -1) { raise("overflow"); }
  return a / b;
}
```

Modulo

Was soll da schon schiefgehen...

```
signed long func(signed long a, signed long b) {
  return a % b;
}
```

Vorbedingungstest

```
#include <iso646.h>
#include <limits.h>
signed long func(signed long a, signed long b) {
  if (b == 0) { raise("division by zero"); }
  if (a == LONG_MIN and b == -1) { raise("overflow"); }
  return a % b;
}
```

Negation

Was soll da schon schiefgehen...

```
signed long func(signed long a) {
  return -a;
}
```

Vorbedingungstest

```
#include <limits.h>
signed long func(signed long a) {
  if (a == LONG_MIN) { raise("overflow"); }
  return -a;
}
```

Multiplikation

Was soll da schon schiefgehen...

```
signed int func(signed int a, signed int b) {
  return a * b;
}
```

Vorbedingungstest

```
#include <iso646.h>
#include <limits.h>
signed int func(signed int a, signed int b) {
  if (a == 0 or b == 0) { return 0; }
  if (a > 0 and b > 0 and a > INT_MAX / b) { raise("overflow"); }
  if (a > 0 and b < 0 and b < INT_MIN / a) { raise("overflow"); }
  if (a < 0 and b > 0 and a < INT_MIN / b) { raise("overflow"); }
  if (a < 0 and b < 0 and b < INT_MAX / a) { raise("overflow"); }
  return a * b;
}
```

Weitere Maßnahmen (I)

Konstruktiver Ausschluss

- Einhalten der Grenzbereiche durch Verifikation sichergestellt
- *beweisbare* Sicherheit

Garantiertes Ausnahmeverhalten

- auf Sprachebene
  - Rust: Operationen mit Überprüfung (bspw. `checked_add`)
  - D: Operationen mit Überprüfung: `checkedint`
  - Ada: Constraint_Error bei Überläufen
- durch die Hardware $\leadsto$ MIPS

Weitere Maßnahmen (II)

Softwareseitige Maßnahmen

- compilergestützt
  - gcc built-in functions
    - `__builtin_{add,sub,mul}_overflow`
  - spezielle Warnungen nutzen
    - `-W-sign-compare, -W-sign-conversion`
    - `-W-strict-overflow, -W-shift-overflow`
- mittels Bibliotheken (bspw. *Safe Numerics* von boost.org)

Keine Patentlösung

- abhängig von Anwendung und System
- muss beim *Systementwurf* bedacht werden
- zieht sich durch die *gesamte Systementwicklung*
- C macht es einem hier nicht einfach

Überblick

1. Abfangen von Integer-Fehlern
2. Testen
   - Vollständig Testen
   - Peer-Review
3. Übungsaufgabe

Spezifikation einer Warteschlange (1)

Die Warteschlange soll folgenden Anforderungen genügen: Es soll möglich sein, eine beliebige Anzahl von Warteschlangen zu erstellen, und die maximale Anzahl der Elemente einer solchen Warteschlange soll nur durch den verfügbaren freien Speicher und die Zahl der Prioritätsebenen begrenzt sein. Jede Prioritätsebene soll innerhalb einer Warteschlange maximal einem Element gleichzeitig zugewiesen sein. 0 entspricht hierbei der höchsten Priorität. Die Warteschlange soll nicht-vorzeichenbehaftete 64 Bit-Ganzzahlen verwalten.

Versucht der Benutzer ein Element mit einer Priorität einzufügen, die in der Warteschlange bereits vergeben ist, so ist dies ein Fehler. Dieser soll nicht durch eine Ausgabe auf dem Bildschirm, sondern durch einen Rückgabewert angezeigt werden. Geht während des Aufrufs einer Warteschlangenoperation der Hauptspeicher aus, so ist dies ein Fehler, über den der Benutzer über die API benachrichtigt wird. Die Reaktion auf den Fehler ist dem Aufrufer überlassen.

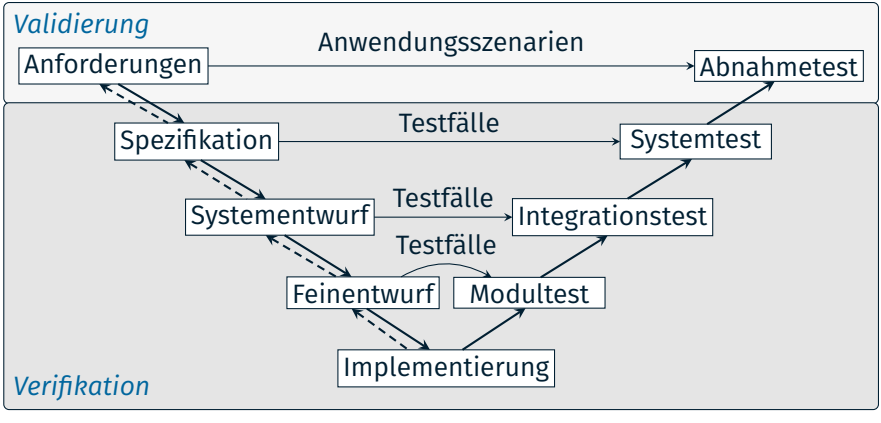
Spezifikation einer Warteschlange (2)

Die Warteschlange soll die Möglichkeit bieten, das höchstpriore Element zu entfernen und hierbei den Wert dieses Elements dem Benutzer zugänglich zu machen. Ferner soll es möglich sein, die Anzahl der Elemente, die sich derzeit in der Warteschlange befinden, zu erfragen.

Im Fehlerfall soll sich die Warteschlange allgemein gutmütig verhalten, d. h. ein Speicherzugriffsfehler oder nicht definiertes Verhalten sind beispielsweise nicht akzeptabel. Sofern das Verhalten der Warteschlange vom Laufzeitsystem abhängig ist, soll dies dokumentiert werden.

Nebenläufige Operationen auf der Warteschlange müssen **nicht** unterstützt werden. Es ist also keine Synchronisation erforderlich.

Einordnung in den Entwicklungsprozess



Testen

- Erste Grundregeln:
  - Testbarkeit von vornherein einplanen
    - ⇝ Feingranulare Testfälle
    - ⇝ *Separate Testfälle für jede einzelne Funktion!*
  - Teste Datentypen an ihren Wertebereichsgrenzen
  - Fehlerfälle explizit testen
  - *Minimale Testabdeckung* erreichbarer Codeüberdeckung
- Hilfsmittel:
  - Überdeckungsmetriken
  - Automatisierte Testinfrastruktur

Vorsicht!

- Testfälle können nur die Anwesenheit von Fehlern zeigen
- Nicht deren Abwesenheit! (→ vgl. formale Verifikation)

⇝ Alle *Randfälle* erkennen und abdecken

Peer-Review und Quality-Assurance

1. ✉ Betreff: “Euer QA-Team wartet”
2. `make doxy`: Dokumentation lesen und implementieren
3. `make pack-review`
4. ✉ “[illegible]” (Feedback)

To: i4ezsmux+projectX-dev@i4.cs.fau.de
Subject: Mögliche Fehler

Hallo liebes Dev-Team,

bei uns schlagen die folgenden Tests mit euerer Implementierung fehl:
- unordered_insert:
  Der Test fügt in inverser Reihenfolge ein und prüft den folgenden Satz der Spezifikation “...”
  Ausgabe: ...
- ...

`make pack-review`, `ALIEN_TEST`, ... im CIP ausführen
⇝ Referenzumgebung für alle

Sag

Überblick

1. Abfangen von Integer-Fehlern
2. Testen
   - Vollständig Testen
   - Peer-Review
3. Übungsaufgabe

Aufgabe 5 – Testen

- Verwendung von GNU/Linux (kein eCos mehr)
- Ziele
  1. Testfokusierter Softwareentwurf
  2. Testfallentwurf
     - Vollständige Pfadüberdeckung
     - Abdecken aller Randfälle
  3. Implementierung von Software und Testfällen
     - Getrennte Implementierung von Software und Testfällen
     - Möglichst durch verschiedene Übungsteilnehmer

     ⇝ Peer-Review
- Implementiert werden soll eine *Prioritätswarteschlange*
- Einfügen, Entfernen, *Iterieren*

  ⇝ `for (... x = ...; x = ...; ++x) ...`!
  - Implementierung?

Iterator – 1. Versuch

- *Datenstruktur als Array* im Header vereinbaren
- Zugriff durch Zeigerarithmetik

```
typedef struct Element { ... } Element;
Element elements[ELEMENTS_SIZE];
...
  for (size_t i = 0; i < ELEMENTS_SIZE; ++i)
    { use(elements[i]); }
```

- *Vorteile:*
  - Einfache Implementierung
  - Für den Compiler leicht zu optimieren
- *Nachteil:* Implementierung offen gelegt
  $\rightsquigarrow$ Verpflichtung gegenüber Benutzer

Iterator – 2. Versuch

- *Iterator als Teil des Objekts*
- Header:

```
typedef struct Elements Elements;
void El_reset_iterator(Elements *self);
void El_next(Elements *self);
bool El_isAtEnd(Elements *self);
int64_t El_iterator_value(Elements *self);
```

- Verwendung:

```
El_reset_iterator(dings);
while(!El_isAtEnd(dings)) {
  use(El_iterator_value(dings));
  El_next(dings);
}
```

Iterator – 2. Versuch

- Implementierung:

```
typedef struct Element { int64_t value; } Element;
struct Elements {
  Element elements[ELEMENTS_SIZE];
  Element *it;
};
void El_reset_iterator(Elements *self)
  { self->it = &self->elements }
void El_next(Elements *self)
  { self->it = self->it + 1; }
bool El_isAtEnd(Elements *self)
  { return self->it
      == &(self->elements[ELEMENTS_SIZE]); }
int64_t El_iterator_value(Elements *self)
  { return self->it->value; }
```

- *Vorteil:* Kapselung sehr gut
- *Nachteile:*
  - Für den Compiler evtl. nicht mehr optimierbar (Schleife ausrollen)
  - So nur ein Iterator gleichzeitig möglich

Iterator – 3. Versuch

- *Iterator als eigenes Objekt*
- Header:

```
typedef struct Elements Elements;
typedef struct El_Iterator El_Iterator;

El_Iterator *El_begin(Elements *self);
void El_Iterator_destroy(El_Iterator *self);
void El_Iterator_next(El_Iterator *self);
bool El_Iterator_isAtEnd(El_Iterator *self);
int64_t El_Iterator_value(El_Iterator *self);
```

- Verwendung:

```
El_Iterator *it;
for (it = El_begin(dings);
     not El_Iterator_isAtEnd(it);
     El_Iterator_next(it)) {
  use(El_Iterator_value(it))
}
El_Iterator_destroy(it);
```

Iterator – 3. Versuch

- Implementierung:

```
typedef struct Element { int64_t value; } Element;
struct Elements { Element elements[ELEMENTS_SIZE]; };
struct El_Iterator {
  Element *position;
  Element *end;
};

El_Iterator *El_begin(Elements *self) {
  El_Iterator *ret = malloc(sizeof(El_Iterator));
  if (ret == NULL) { return NULL; }
  ret->position = self->elements;
  ret->end = &self->elements[ELEMENTS_SIZE];
  return ret;
}

void El_Iterator_next(El_Iterator *self)
  { self->position += 1; }
bool El_Iterator_isAtEnd(El_Iterator *self) { ... }
int64_t El_Iterator_value(El_Iterator *self) { ... }
void El_Iterator_destroy(El_Iterator *self) { ... }
```

Iterator – 3. Versuch

- *Vorteile:*
  - Vollständige Kapselung
  - Beliebig viele Iteratoren möglich
- *Nachteil:*
  - Iterator muss nach Gebrauch beseitigt werden
  - Compiler hat evtl. Probleme zu optimieren

$\Rightarrow$ Verwendung in der Übung

Verlässliche Echtzeitsysteme - Übungen

Analyse

Wintersemester 2024

Eva Dengler, Peter Wägemann

Friedrich-Alexander-Universität Erlangen-Nürnberg
Lehrstuhl Informatik 4 (Systemsoftware)
https://sys.cs.fau.de

Überblick

1. Stackbedarfsanalyse
2. Worst-Case Stack Usage
3. AbsInt Stack Analyzer
4. Aufgabenstellung

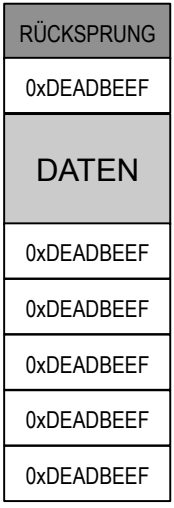
Analyse Harter Echtzeitsysteme

- Harte, verlässliche Echtzeitsysteme
  - Garantien über Ressourcenbedarf notwendig

    ☞ statische Analyse unabdingbar
- Mögliche Ressourcen: Speicherbedarf, Laufzeit, etc.
- Übung: Analyse des Stackverbrauchs einer Feldbibliothek
- Stack-Analyse
  1. Dynamisch: Wasserstandstechnik
  2. Statisch: „Eigenbau“ und aiT (Stack-Analyzer der a^3 Suite)
- WCET-Analyse mittels aiT (bereits in EZS behandelt)

Dynamische Analyse des Stapelspeicherbedarfs

- *Messung zur Laufzeit*: Wasserstandsmessung
- Grundidee: Einfügen von **Stack Canaries**
- Explizite Verwaltung des Stapelspeichers notwendig
- pthread-Bibliothek ermöglicht Verwaltung
- Mögliche Canaries
  - Lesbare Bitmuster: `0xDEADBEEF`
  - Unwahrscheinliche Bitmuster: `0b101010101010...`
  - Kleinere Bitmuster $\rightsquigarrow$ größere Auflösung
- ⚠ Keine allgemeingültigen Aussagen
  - Liefert nur den konkreten Bedarf der Messungen
  - Vorsichtige Aussagen über Worst-Case-Verhalten
- Einsatz zur dynamischen Fehlererkennung



pthread-Bibliothek

1. (Globalen) Stack anlegen:

```
static unsigned int g_data[DATA_SIZE];
```

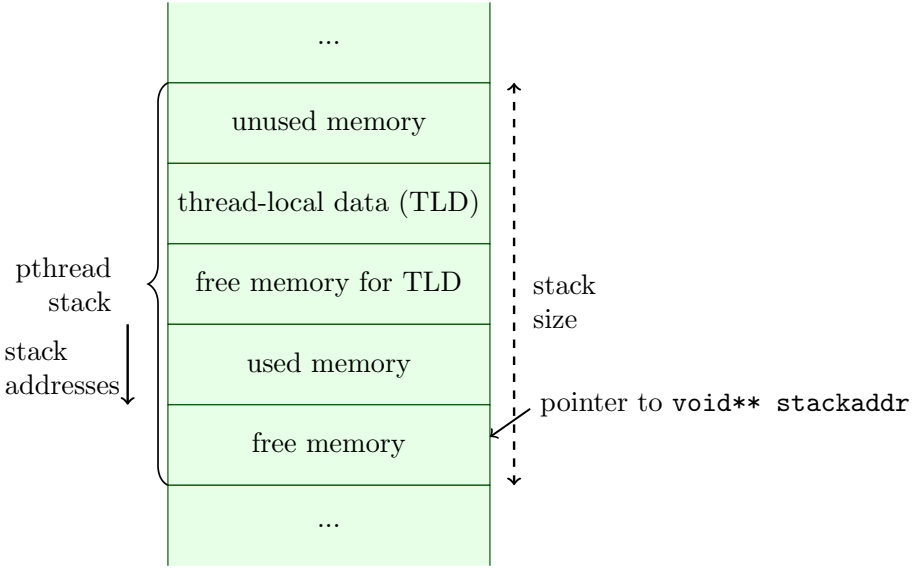
2. Thread anlegen & starten:

```
pthread_t thread;
pthread_attr_t attr;
pthread_attr_init(&attr);
pthread_attr_setstack(&attr, &g_stack, STACK_SIZE);
// worker function: void *run(void *param)
int status = pthread_create(&thread, &attr, run, NULL);
if (status != 0) { ... // handle error
```

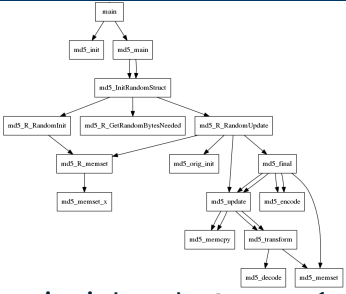
3. Auf Thread warten:

```
pthread_join(thread, &ret);
```

pthread Stack



Statische Stackbedarfsanalyse




```
/* Objective function */
max: +16 md5_orig_init +64 md5_update \
      +64 md5_final +16 md5_memset \
      +208 md5_transform +16 md5_encode ...;

/* Constraints */
+main = 1;
+md5_init +md5_main <= +main;
...
```

- Beispiel: md5-Summe[1]
- Vorgehen
  1. Callgraph bestimmen
  2. Stackbedarf einzelner Funktionen (`gcc -fstack-usage`)
  3. ILP[2] aufstellen (Nebenbedingungen aus 1., Kosten aus 2. verwenden)
  4. ILP z.B. mittels `lp_solve` $\rightsquigarrow$ **maximaler Stackbedarf**

[1] `https://github.com/tacle/tacle-bench/`

[2] Integer Linear Program (dt. ganzzahliges lineares Programm)

Optimierungsziel

- Jeder Stapelrahmen einer Funktion *f* hat eine Größe *size*
- Jede Funktion kann auf einem Pfad ein- oder mehrfach (Rekursion), insgesamt *n*-fach auf dem Stapel vorkommen
- Gesucht: Fluss durch den Aufrufgraphen, welcher Stapelbedarf maximiert
- Dabei müssen **Flussbedingungen** eingehalten werden
  - Aufruferbeziehung
  - Alternativen
  - ...

Optimierungsziel

$$max \sum_{\text{Funktion } f} size_f \cdot n_f$$

In `lp_solve` -Syntax: `max : +64 n_f1 +48 n_f2 +42 n_f3 ;`

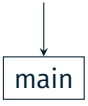
Flussbedingung: Initialer Aufruf

Semantik

Der initiale Aufruf erfolgt maximal (wahlweise auch genau) ein mal

Formalisierung

$$n_{\texttt{main}} \leq 1$$



`lp_solve` -Syntax

```
n_main  <=  1;
```

Flussbedingung: Mehrere Vorgänger

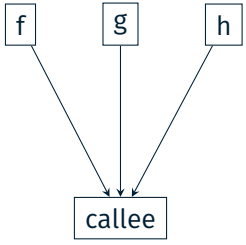
Semantik

Jede Funktion kann nur so oft ausgeführt werden, wie sie von den Vorgängern aus aufgerufen wird

Formalisierung

Sei $f_{a \to b}$ die Anzahl der Aufrufe von b durch a:

$$n_{callee} \leq \sum_{p \in \text{Aufrufer(callee)}} f_{p \to callee}$$



`lp_solve` -Syntax

```
n_callee <= + f_f_callee + f_g_callee + f_h_callee ;
```

Flussbedingung: Immer nur ein Nachfolger pro Funktion

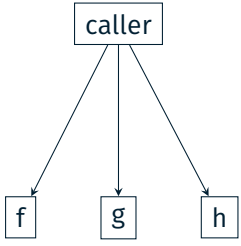
Semantik

Jede Funktionsinkarnation ruft gleichzeitig jeweils maximal eine weitere Funktion auf

Formalisierung

Sei $f_{a \to b}$ die Anzahl der Aufrufe von b durch a:

$$\sum_{c \in \text{Aufgerufene(caller)}} f_{caller \to c} \leq n_{\text{caller}}$$



`lp_solve` -Syntax

```
+ f_caller_f  + f_caller_g  + f_caller_h  <=  n_caller ;
```

Flussbedingung: Rekursion

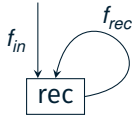
Semantik

Rekursive Funktionen können pro Aufruf von außen bis zu ihrer maximalen Rekursionstiefe (d) oft ausgeführt werden.

Formalisierung

$$f_{rec} \leq d_{rec} \cdot f_{in}$$

$$n_{rec} \leq f_{in} + f_{rec}$$



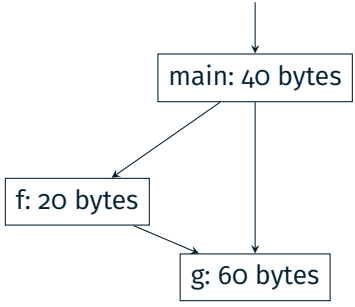
`lp_solve` -Syntax

```
  f_rec  <=  +42  f_in ;
n_rec  <=  f_in  +  f_rec ;
```

Beispiel

- Problemformulierung in lpsolve:

```
max: +40 n_main +20 n_f +60 n_g;

n_main <= 1;
+f_main_f +f_main_g <= n_main;
n_f <= +f_main_f;
+f_f_g <= n_f;
n_g <= +f_f_g +f_main_g;
```



- Ausgabe von `lp_solve` :

```
Value of objective function: 120.00000000

Actual values of the variables:
n_main                         1
n_f                            1
n_g                            1
f_main_f                       1
f_main_g                       0
f_f_g                          1
```

LP-Solve Fallstricke: Infeasible model

```
$ lp_solve infeasible.lp
This problem is infeasible
```

Infeasible Models

Logischer Widerspruch in Nebenbedingungen

Leider bietet `lp_solve` selbst direkt keine Hilfestellung zur Lokalisation. Die Entwickler empfehlen das Einführen von “slack”-Variablen:[3]

```
max: x + y;
x + 1 <= x;
y > y + 1;
x <= 20;
y <= 20;
```

```
max: x + y
    -1000 e_1
    -1000 e_2;
x + 1 - e_1 <= x;
y + e_2 > y + 1;
x <= 20;
y <= 20;
```

```
x:    20
y:    20
e_1:   1
e_2:   1
```

[3] http://lpsolve.sourceforge.net/5.5/Infeasible.htm

LP-Solve Fallstricke: Unbounded model

```
$ lp_solve unbounded.lp
This problem is unbounded
```

Unbounded Models

Eine oder mehrere der Variablen sind nach oben unbeschränkt

Durch künstliche Beschränkung aller Variablen im System (auf einen sehr großen Wert) lassen sich unbeschränkte Variablen detektieren:

```
max: x + y + z;
z <= y + 1;
y <= 20;
```

```
max: x + y + z;
z <= y + 1;
y <= 20;
x <= 5000;
y <= 5000;
z <= 5000;
```

```
x:  5000
y:    20
z:    21
```

LP-Solve Fallstricke: Syntax

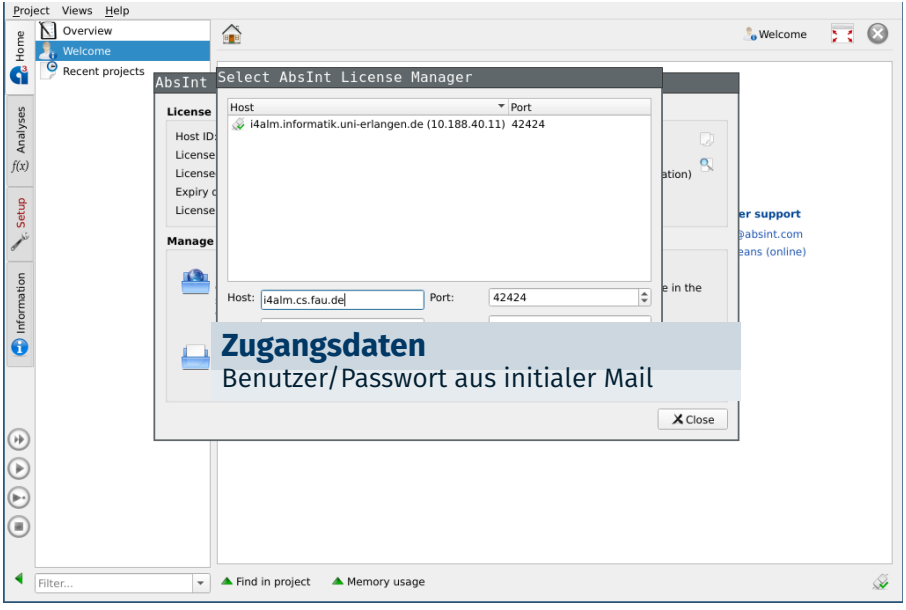
- `lp_solve` ist auf die Lösung linearer Gleichungssysteme ausgelegt
- Es ist dementsprechend nicht möglich, zwei Variablen zu multiplizieren
  - `a * b` ⇒ Syntaxfehler
  - `max: a b` ⇒ optimiert $a + b$
- Lösung in VEZS für Konstanten (Stapelrahmengrößen):
  C-Präprozessor:

```
#define s_main 40
#define s_f    20
#define s_g    60

max: +s_main n_main +s_f n_f +s_g n_g;
```

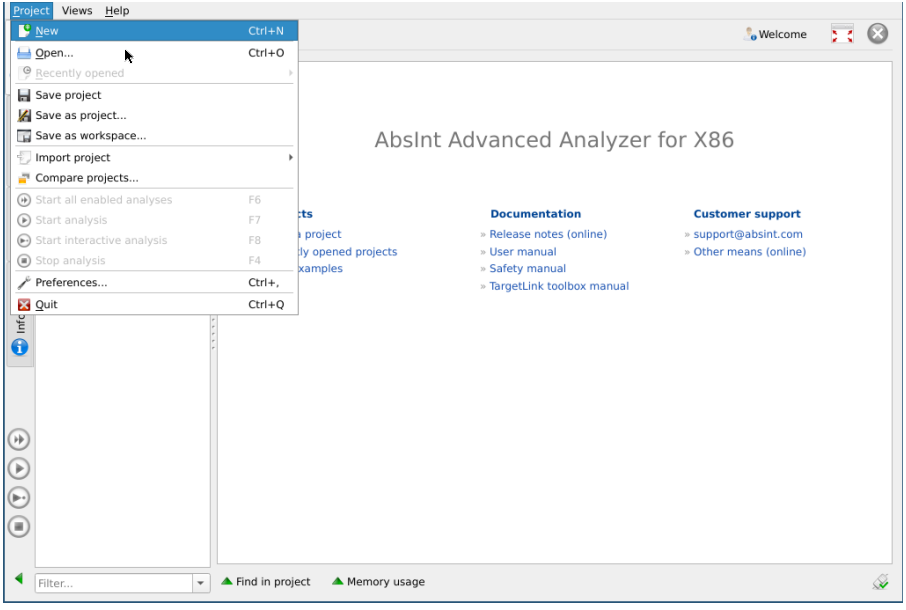
Statische Stackbedarfsanalyse

- Statische Code-Analyse mit a^3 Tool-Suite
  1. aiT: WCET-Analyse
  2. Stack-Analyzer: Stackbedarf
  3. ...
- Installiert im CIP-Pool
- `/proj/i4ezs/tools/a3_x86/bin/a3x86`

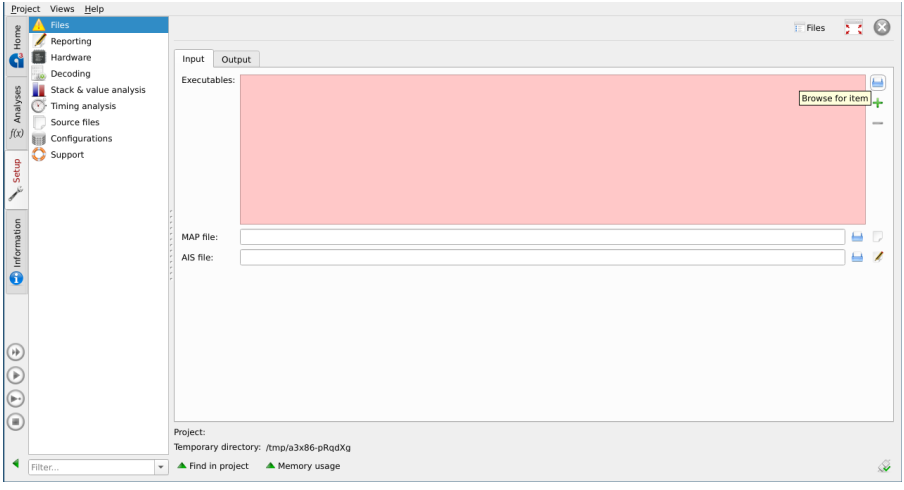
a³ Analyzer – Lizenzserver



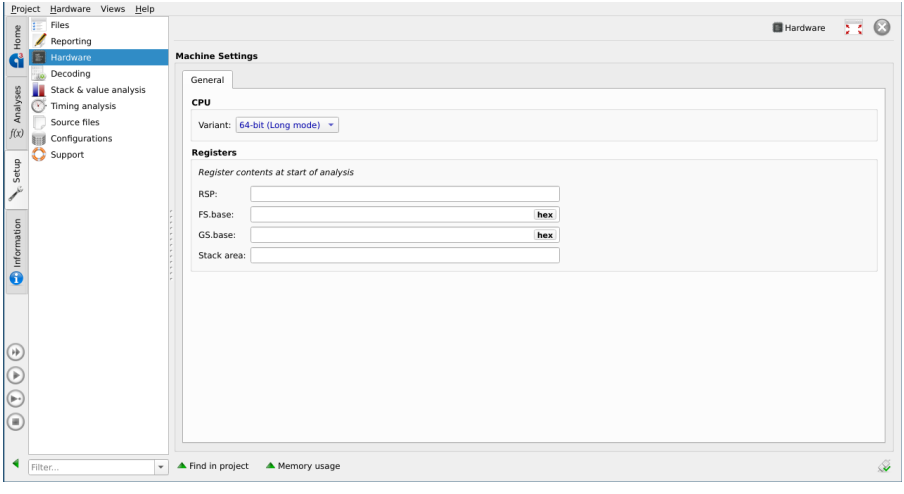
a³ Analyzer – Neues Projekt Anlegen



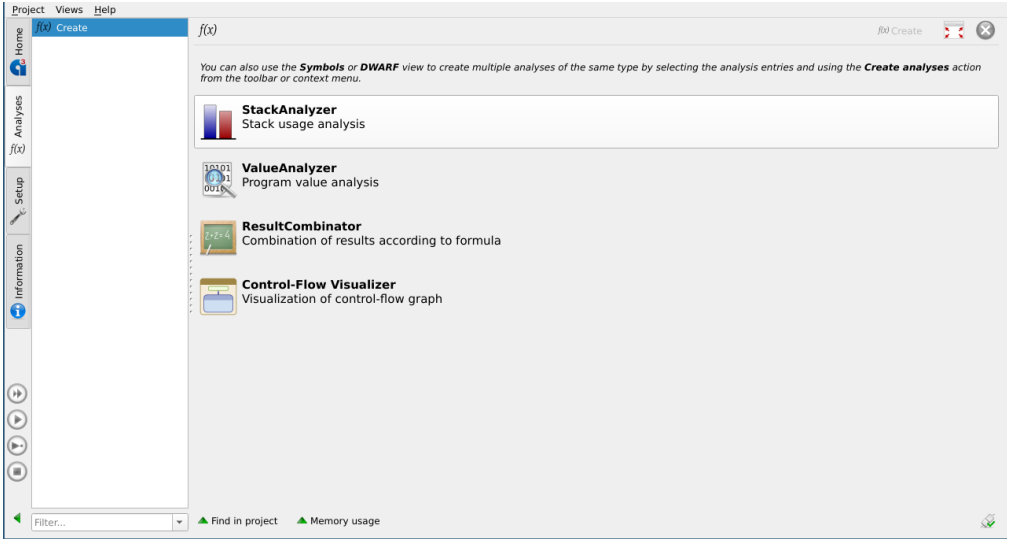
a³ Analyzer – Executable Angeben



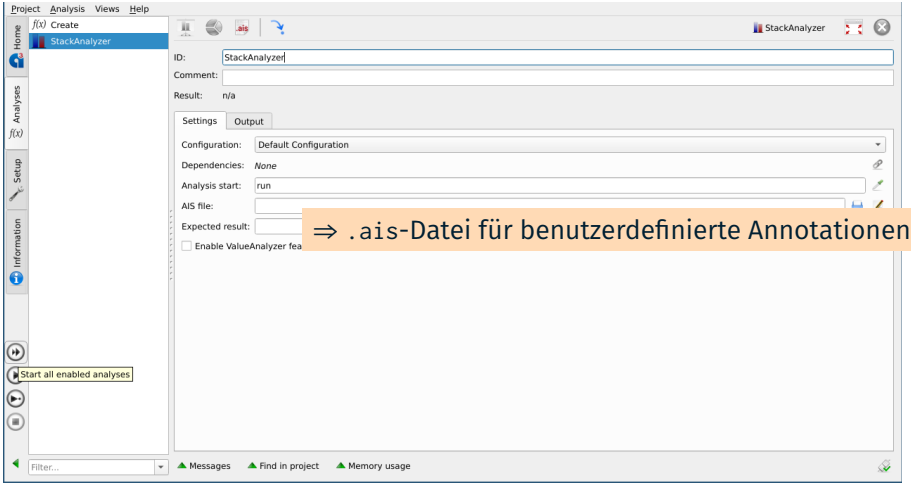
a³ Analyzer – Hardware Auswählen



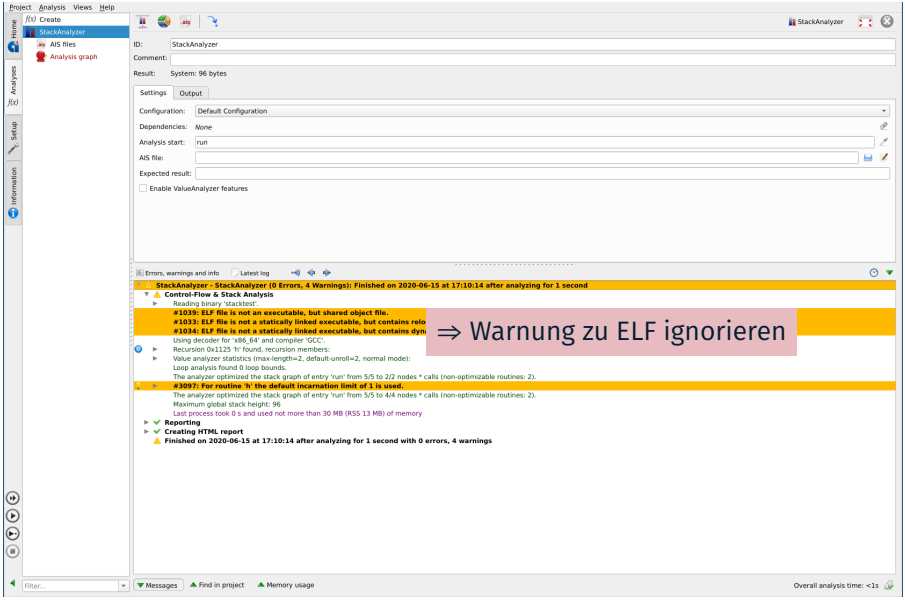
a³ Analyzer – Stack-Analyse Selektieren



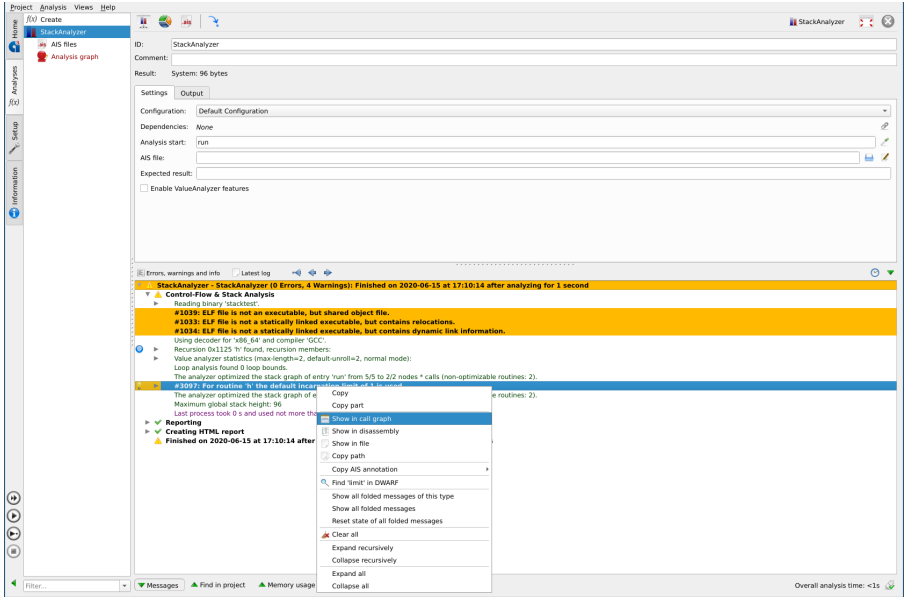
a^3 Analyzer – Stack-Analyse Starten



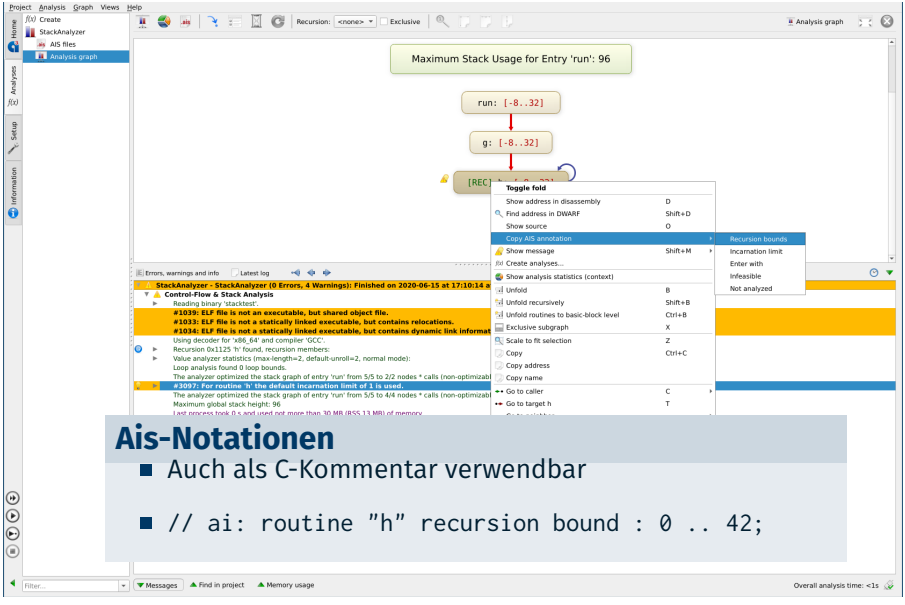
a³ Analyzer – Analyseoutput



a^3 Analyzer – Callgraph



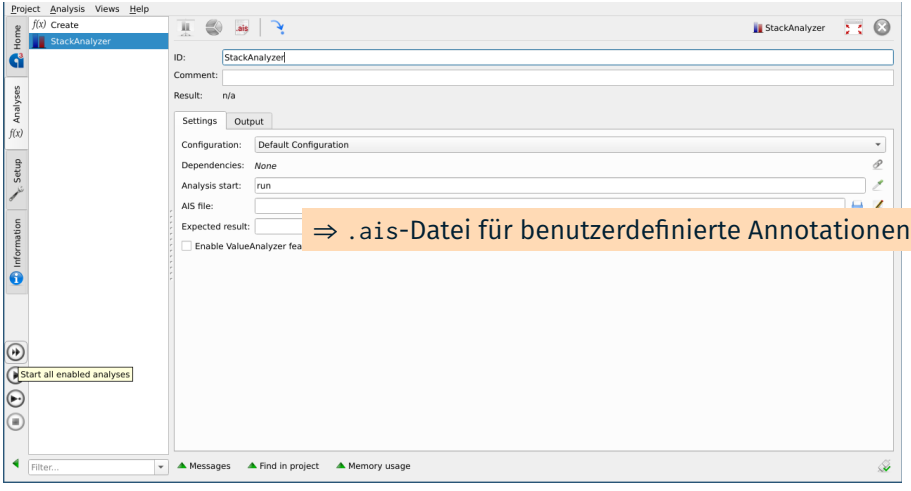
a³ Analyzer – Annotationstemplate kopieren



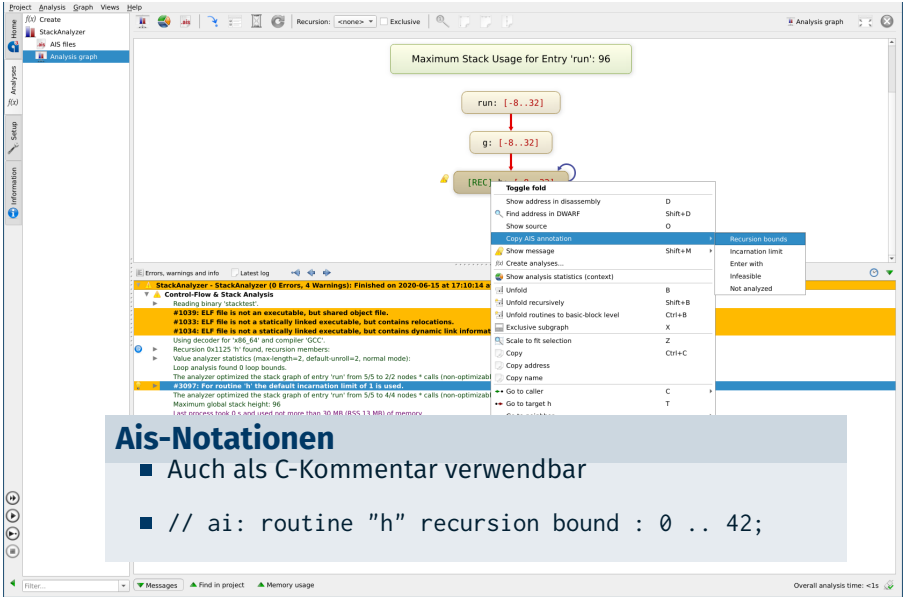
Ais-Notationen

- Auch als C-Kommentar verwendbar
- `// ai: routine "h" recursion bound : 0 .. 42;`

a³ Analyzer – Stack-Analyse Starten



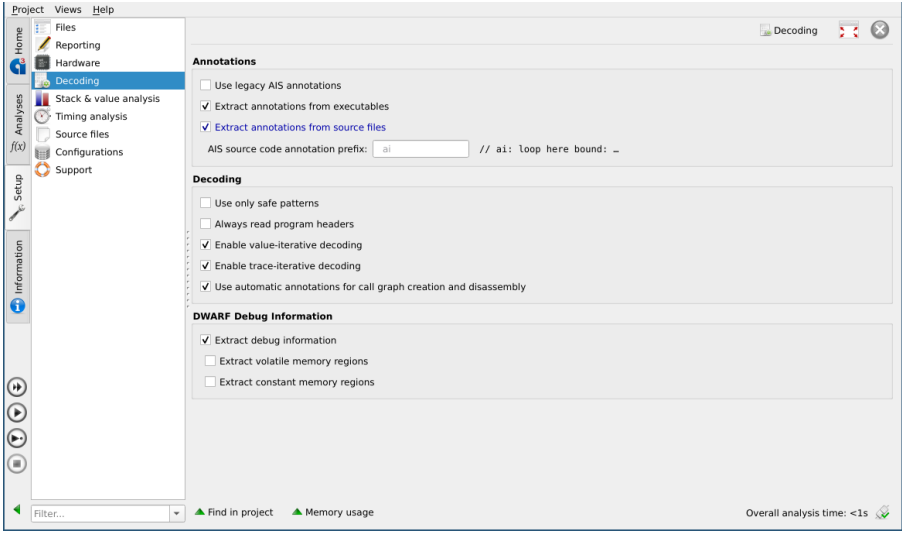
a³ Analyzer – Annotationstemplate kopieren



Ais-Notationen

- Auch als C-Kommentar verwendbar
- `// ai: routine "h" recursion bound : 0 .. 42;`

a³ Analyzer – Kommentar-Parsing Aktivieren



Aufgabenstellung

- Existierende Implementierung: Array-Datenstruktur
- Vorgegebene Funktionen: Sortieren, Maximumssuche, ...
- Aufgaben
  1. Dynamische Analyse
     - 1.1 Thread erstellen
     - 1.2 Stack initialisieren
     - 1.3 Programm (mit Eingabedaten) ausführen
     - 1.4 Stackverbrauch messen
  2. Statische Analyse
     - 2.1 ILP aus Aufrufgraph aufstellen
     - 2.2 Mittels `lp_solve` lösen
     - 2.3 Verwendung a^3 Stack-Analyzer
  3. Optional: Zeitanalyse mit aiT

Verlässliche Echtzeitsysteme - Übungen

Abstrakte Interpretation & Verifikation

Wintersemester 2024

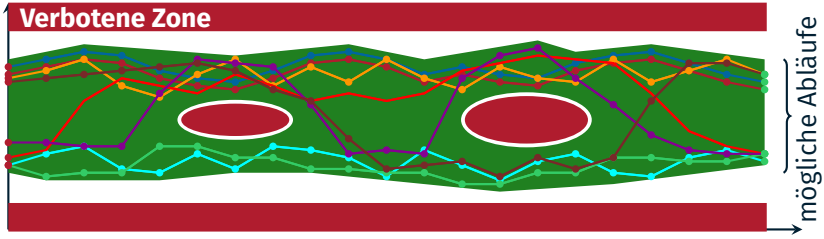
Eva Dengler, Peter Wägemann

Friedrich-Alexander-Universität Erlangen-Nürnberg
Lehrstuhl Informatik 4 (Systemsoftware)
https://sys.cs.fau.de

Überblick

1. Abstrakte Interpretation mit Astrée
2. Formale Verifikation mit Frama-C
3. Aufgabenstellung
4. α-β-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 ⇒ konkrete Semantik ist sicher

Astrée [1]

- Ziel: Nachweis der Abwesenheit von Laufzeitfehlern
- Findet *alle* potentiellen Laufzeitfehler
- Leider auch *falsch-positive*
  $\rightsquigarrow$ **Gödelsches Unvollständigkeitstheorem**

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
- 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

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))`

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

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

- Analyzer erzeugt Alarm, falls `B` *nicht immer wahr ist*
- Analyzer 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;
}
```

Stubs

`__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

volatile

`__ASTREE_volatile_input((V))`

- Zeigt an, dass `V` sich jederzeit ändern kann

$\rightsquigarrow$ Modelliert Eingabe von außen

`__ASTREE_volatile_input((Vp, r))`

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

Exkurs: Beschränkte Laufzeit

- Viele Echtzeitsysteme endlosschleifenbasiert
- Allerdings durch andere Umstände begrenzt
- 1kHz Ausführungsfrequenz, Neustart nach 7 Tagen $\rightsquigarrow$ 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
  }
}
```

$\Rightarrow$ 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()));
}
```

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

Analyse untersuchen

__ASTREE_a

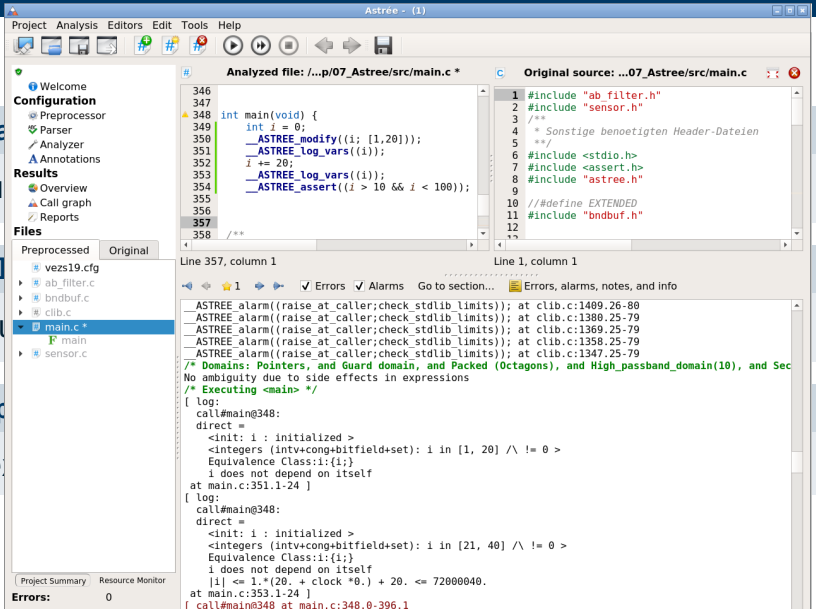
- Gibt Zu

__ASTREE_I

- Zeigt Zu

__ASTREE_p

- Gibt Te



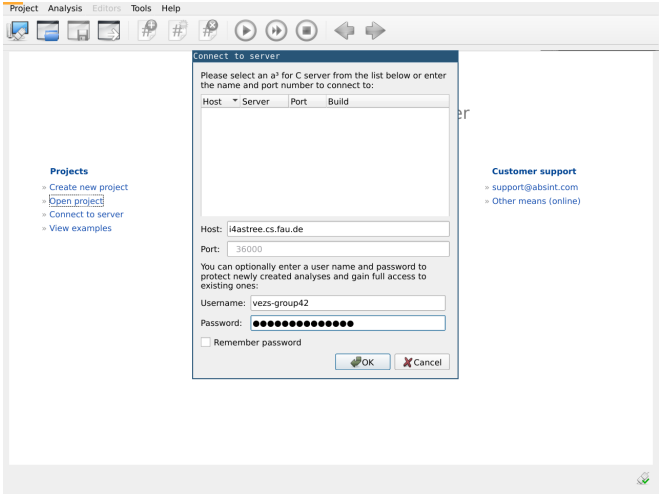
Astrée verwenden

- 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



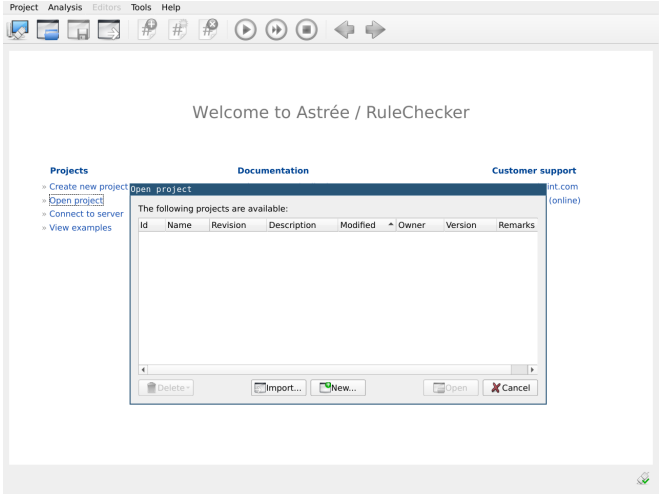
Host i4astree.cs.fau.de

Port 36000

Benutzer vezs-groupXX

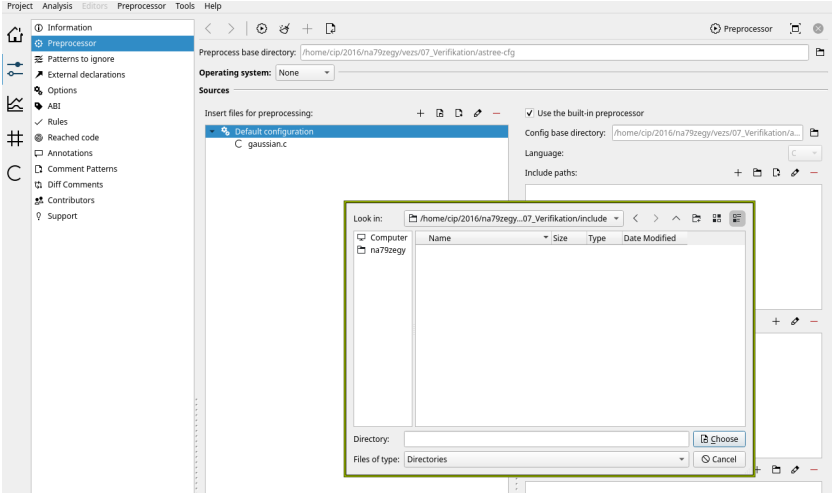
Passwort Beliebig wählbar

Projekt anlegen



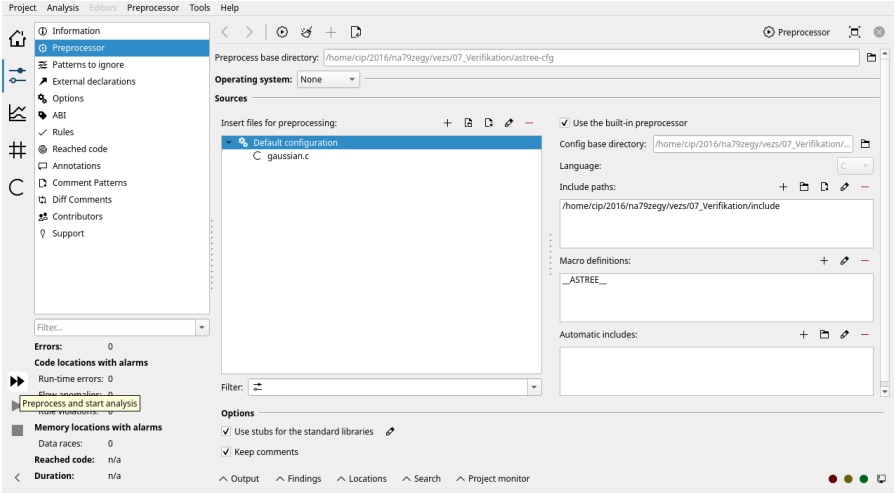
- „Import…“
- Projekt aus Vorgabedatei `astree-cfg/vezs.dax` importieren

Quelldateien konfigurieren



- Quelltextdateien und Includepfade definieren

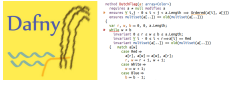
Analyse starten



Programme zur Softwareverifikation

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

frama C
Software Analyzers



Idris

WP-Kalkül [4]

- 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

  $\rightarrow$ 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;
```

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

```
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 :

Auswertungsreihenfolge

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

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

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

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

Frama-C: Syntax

- erlaubt Nachweise funktionaler Eigenschaften mittels wp-Kalkül
  - Prädiaktenlogik erster Stufe
    - ∀: `\forall`
    - ∃: `\exists`
    - ⇒: 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

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 : $\result \geq a \wedge \result \geq b \wedge (\result == a \vee \result == b)$

```
/*@
  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

/*@ 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

Schleifen

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

- Wann ist die Berechnung korrekt?
  - Stellt Q her $\rightsquigarrow$ Schleifen**invariante**
  - Terminiert $\rightsquigarrow$ 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` ≥ 0
- Schleifeninvariante: `0 <= i <= 30`

Die Schleifenvariante gilt vor, während und nach der Schleife

Nach der Schleife ist die Schleifenbedingung falsch:

$\rightsquigarrow \neg(i < 30) \wedge (0 \leq i \leq 30)$

$\rightsquigarrow i == 30$

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];
    }
  }
  return max;
}
```

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

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

```
/*@
  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;
}
```

Frama-C (III): Prädikate

- „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

Frama-C: Behaviors

- 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;
}
```

Frama-C: Fallstricke

- 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

  $\rightsquigarrow$ Frama-C assert verwenden: `//@ assert x == 1;`
- Mehrere Annotationen immer in einen gemeinsamen Block `/*@ */`
- Weitere Informationen:
  - Frauenhofer 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`

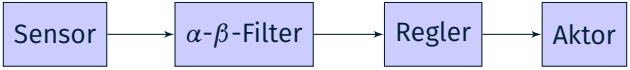
Überblick

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

Inhalt

α-β-Filter [3]

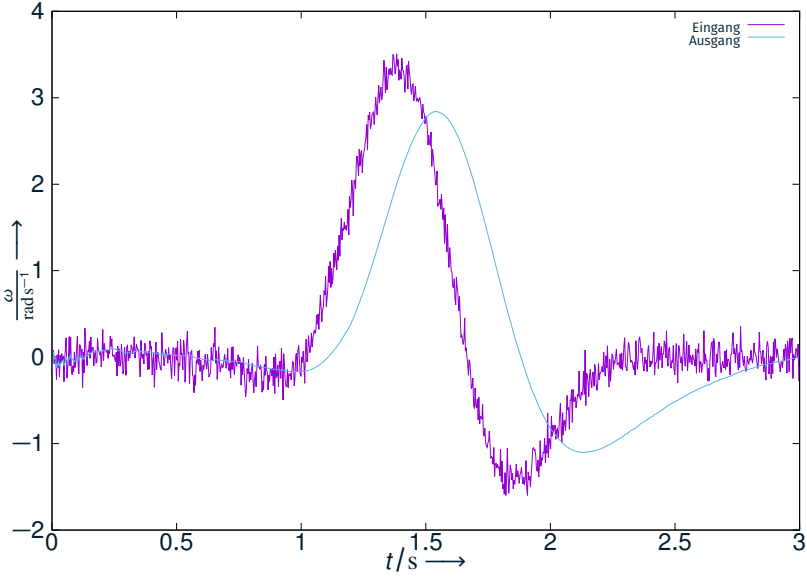


- Rauschunterdrückungsfilter
- geeignet zur Schätzung von physikalischen Größen
  - mit Ableitung $\neq 0$
  - z. B. Position eines Flugzeugs, Lagewinkel …
  - im I4Copter
    - liefern Sensoren häufiger Werte als diese später verarbeitet werden

    $\rightsquigarrow$ α-β-Filter für *Ratenwandlung* verwendet

    $\rightsquigarrow$ nutzt gewonnene Information vollständig

Beispiel α-β-Filter



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 $\rightsquigarrow$ hier beispielhaft nur eines [6, 5]

Filterparameter

T Abtastintervall
$\rightsquigarrow$ in welchem Zeitabstand gemessen wird

σ_w^2 Prozessvarianz
$\rightsquigarrow$ wie lebhaft der gemessene Prozess ist

σ_v^2 Rauschvarianz
$\rightsquigarrow$ wie verrauscht das Signal ist

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

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

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

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

Initialisierungsphase

- Zu Beginn hat der Filter keinen gültigen Zustand

$\rightsquigarrow$ Einschwingphase, in der der Filter „lernt“

- n startet bei 1 und wächst in jeder Runde um 1

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

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

- 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

- α-β-Filter stabil, wenn gilt

$$0 < \alpha \leq 1 \tag{10}$$

$$0 < \beta \leq 2 \tag{11}$$

$$0 < 4 - 2\alpha - \beta \tag{12}$$

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

$$0 < \beta < 1 \tag{13}$$

- 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	$y[\kappa] \in [-10 \dots 10]$

$\rightsquigarrow$ Korrektheit mindestens für diese Wertebereiche zeigen!

Ringpuffer – Implementierung

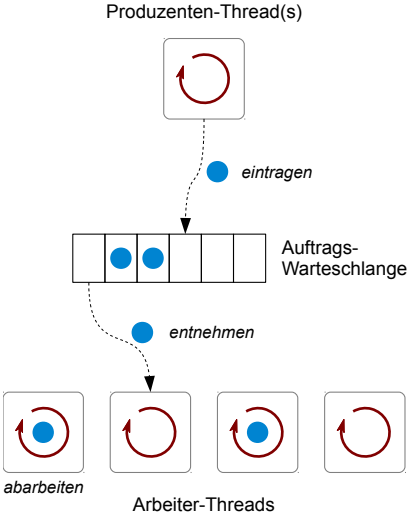
- Beispiel für dynamisch angelegten Ringpuffer

```
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



Literatur (1)

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

Literatur (2)


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.

Literatur (3)

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