

# Echtzeitsysteme

Übungen zur Vorlesung

Analyse von Ausführungszeiten

**Simon Schuster**   **Peter Wägemann**

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

09.11.2018



- 1 Rekapitulation: Worst-Case Execution Time
- 2 Ausflug: Cache-Analyse
  - Grundlagen
  - Beispiel: LRU-Cache
- 3 WCET-Analyse auf dem EZS-Board
  - GPIOs
  - aiT



## 1 Rekapitulation: Worst-Case Execution Time

### 2 Ausflug: Cache-Analyse

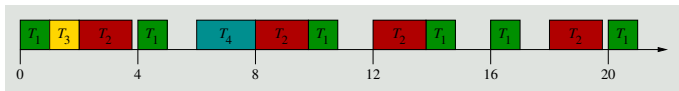
- Grundlagen
- Beispiel: LRU-Cache

### 3 WCET-Analyse auf dem EZS-Board

- GPIOs
- aiT



# Worst-Case Execution Time



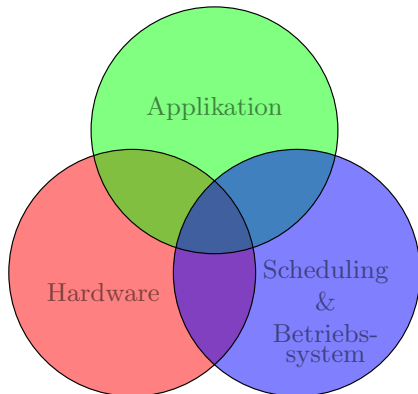
■ Eine entscheidende Größe für:

- Statische Ablaufplanung
- Planbarkeitsanalyse
- Übernahmeprüfung
- ...

☞ Es geht um den **schlimmsten Fall** (engl. *worst case*)

→ Obere Schranke für **alle** Fälle





- 1 **Applikation:** Eingabedaten, ...
- 2 **Hardware:** Caches, Pipelining, ...
- 3 **Scheduling:** Höherpriorie Aufgaben, Interrupts, Overheads, ...



```
1 void func(int a) { // entry
2   if (a % 2) {
3     f(); // if.then0
4   }
5   ++a; // if.end0
6
7   if(a % 2) {
8     g(); // if.then1
9   }
10  ... // if.end1
11 }
```

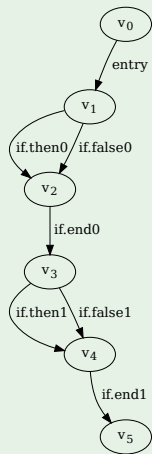
- T-Graph aus Kontrollflussgraph abgeleitet
- Worst Case == maximaler Fluss durch T-Graph



```
1 void func(int a) { // entry
2   if (a % 2) {
3     f(); // if.then0
4   }
5   ++a; // if.end0
6
7   if(a % 2) {
8     g(); // if.then1
9   }
10  ... // if.end1
11 }
```

- T-Graph aus Kontrollflussgraph abgeleitet
- Worst Case == maximaler Fluss durch T-Graph
- Nebenbedingungen des Flussproblems:
  - $freq(entry) = freq(if.then0) + freq(if.false0)$
  - $freq(if.then0) + freq(if.false0) = freq(if.end0)$
  - ...
- Nebenbedingungen werden für Integer Linear Program (ILP) verwendet:  
Zielfunktion:

$$max : cost(entry) \cdot freq(entry) + cost(if.then0) \cdot freq(if.then0) + \dots$$

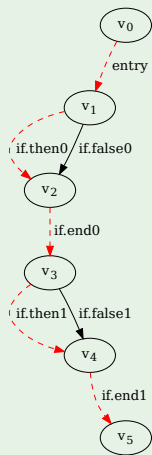


# WCET-Analyse – Flusssensitive Informationen

Pessimistische Annahmen der IPET

```
1 void func(int a) { // entry
2   if (a % 2) {
3     f(); // if.then0
4   }
5   ++a; // if.end0
6
7   if(a % 2) {
8     g(); // if.then1
9   }
10  ... // if.end1
11 }
```

- Für jeden Basis Block: WCET notwendig
- Schleifengrenzen notwendig
- Struktureller Ansatz: *nicht kontextsensitiv*
- Im Beispiel: *beide Pfade* aufgenommen  
⇒ **pessimistische Annahme**
- *Nachträgliche* Reduktion dieser Überabschätzung  
☞ **abstrakte Interpretation**  $\rightsquigarrow$  VEZS





## Grundproblem: Ausführungszyklen von Instruktionen zählen

```
_getop :  
  link    a6,#0           // 16 Zyklen  
  moveml  #0x3020 ,sp@-   // 32 Zyklen  
  movel   a6@(8) ,a2      // 16 Zyklen  
  movel   a6@(12) ,d3     // 16 Zyklen
```

Quelle: Peter Puschner [2]

■ Ergebnis:  $e_{\_getop} = 80$  Zyklen

■ Annahmen:

- Obere Schranke für jede Instruktion
- Obere Schranke der Sequenz durch Summation





## Grundproblem: Ausführungszyklen von Instruktionen zählen

```
_getop :  
  link    a6,#0           // 16 Zyklen  
  moveml  #0x3020 ,sp@-   // 32 Zyklen  
  movel   a6@(8) ,a2      // 16 Zyklen  
  movel   a6@(12) ,d3     // 16 Zyklen
```

Quelle: Peter Puschner [2]

■ Ergebnis:  $e_{\text{getop}} = 80$  Zyklen

■ Annahmen:

- Obere Schranke für jede Instruktion
- Obere Schranke der Sequenz durch Summation



## Äußerst pessimistisch und zum Teil falsch

■ Falsch für Prozessoren mit Laufzeitanomalien

- WCET der Sequenz  $>$  Summe der WCETs aller Instruktionen

■ Pessimistisch für moderne Prozessoren

- Pipeline, Cache, Branch Prediction, Prefetching, ... haben großen Anteil an der verfügbaren Rechenleistung heutiger Prozessoren
- Blanke Summation einzelner WCETs ignoriert diese Maßnahmen





## Grundproblem: Ausführungszyklen von Instruktionen zählen

```
_getop :  
  link    a6,#0           // 16 Zyklen  
  moveml  #0x3020 ,sp@-   // 32 Zyklen  
  movel   a6@(8) ,a2      // 16 Zyklen  
  movel   a6@(12) ,d3     // 16 Zyklen
```

Quelle: Peter Puschner [2]

■ Ergebnis:  $e_{\text{getop}} = 80$  Zyklen

■ Annahmen:

- Obere Schranke für jede Instruktion
- Obere Schranke der Sequenz durch Summation



Äußerst pessimistisch und zum Teil falsch

Kein Pipelining:





## Grundproblem: Ausführungszyklen von Instruktionen zählen

```
_getop :  
link    a6,#0           // 16 Zyklen  
moveml  #0x3020 ,sp@-   // 32 Zyklen  
movel   a6@(8) ,a2      // 16 Zyklen  
movel   a6@(12) ,d3     // 16 Zyklen
```

Quelle: Peter Puschner [2]

■ Ergebnis:  $e_{\text{getop}} = 80$  Zyklen

■ Annahmen:

- Obere Schranke für jede Instruktion
- Obere Schranke der Sequenz durch Summation



Äußerst pessimistisch und zum Teil falsch

Pipelining: (dreistufige Pipeline)

Lesen	Ausführen	Schreiben			
instr1 <sub>dec</sub>	instr1 <sub>exe</sub>	instr1 <sub>wb</sub>			
	instr2 <sub>dec</sub>	instr2 <sub>exe</sub>	instr2 <sub>wb</sub>		
		instr3 <sub>dec</sub>	instr3 <sub>exe</sub>	instr3 <sub>wb</sub>	



## Grundproblem: Ausführungszyklen von Instruktionen zählen

```
_getop :  
  link    a6,#0           // 16 Zyklen  
  moveml  #0x3020 ,sp@-   // 32 Zyklen  
  movel   a6@(8) ,a2      // 16 Zyklen  
  movel   a6@(12) ,d3     // 16 Zyklen
```

Quelle: Peter Puschner [2]

■ Ergebnis:  $e_{\text{getop}} = 80$  Zyklen

■ Annahmen:

- Obere Schranke für jede Instruktion
- Obere Schranke der Sequenz durch Summation



Äußerst pessimistisch und zum Teil falsch

Pipelining mit Hazards:





## Grundproblem: Ausführungszyklen von Instruktionen zählen

```
_getop :  
  link      a6,#0           // 16 Zyklen  
  moveml   #0x3020 ,sp@-    // 32 Zyklen  
  movel    a6@(8) ,a2       // 16 Zyklen  
  movel    a6@(12) ,d3      // 16 Zyklen
```

Quelle: Peter Puschner [2]

■ Ergebnis:  $e_{\text{getop}} = 80$  Zyklen

■ Annahmen:

- Obere Schranke für jede Instruktion
- Obere Schranke der Sequenz durch Summation



## Äußerst pessimistisch und zum Teil falsch

■ Falsch für Prozessoren mit Laufzeitanomalien

- WCET der Sequenz  $>$  Summe der WCETs aller Instruktionen

■ Pessimistisch für moderne Prozessoren

- Pipeline, Cache, Branch Prediction, Prefetching, ... haben großen Anteil an der verfügbaren Rechenleistung heutiger Prozessoren
- Blanke Summation einzelner WCETs ignoriert diese Maßnahmen





Hardware-Analyse teilt sich in verschiedene Phasen

- Aufteilung ist nicht dogmenhaft festgeschrieben





Hardware-Analyse teilt sich in verschiedene Phasen

- Aufteilung ist nicht dogmenhaft festgeschrieben

- **Integration** von Pfad- und Cache-Analyse

- 1 Pipeline-Analyse

- Wie lange dauert die Ausführung der Instruktionssequenz?

- 2 Cache- und Pfad-Analyse sowie WCET-Berechnung

- Cache-Analyse wird direkt in das Optimierungsproblem integriert





Hardware-Analyse teilt sich in verschiedene Phasen

- Aufteilung ist nicht dogmenhaft festgeschrieben

## ■ Integration von Pfad- und Cache-Analyse

### 1 Pipeline-Analyse

- Wie lange dauert die Ausführung der Instruktionssequenz?

### 2 Cache- und Pfad-Analyse sowie WCET-Berechnung

- Cache-Analyse wird direkt in das Optimierungsproblem integriert

## ■ Separate Pfad- und Cache-Analyse

### 1 Cache-Analyse

- Kategorisiert Speicherzugriffe mit Hilfe einer Datenflussanalyse

### 2 Pipeline-Analyse

- Ergebnisse der Cache-Analyse werden anschließend berücksichtigt

### 3 Pfad-Analyse und WCET-Berechnung



1 Rekapitulation: Worst-Case Execution Time

2 Ausflug: Cache-Analyse

- Grundlagen
- Beispiel: LRU-Cache

3 WCET-Analyse auf dem EZS-Board

- GPIOs
- aiT



## Cache: ein kleiner, schneller Zwischenspeicher

- Zugriffszeiten variieren je nach Zustand des Caches enorm:

**Treffer** (engl. *hit*), Daten/Instruktion sind im Cache  $\leadsto e_h$

**Fehlschlag** (engl. *miss*), Daten/Instruktion sind nicht im Cache  $\leadsto e_m$



 **Cache:** ein kleiner, schneller Zwischenspeicher

- Zugriffszeiten variieren je nach Zustand des Caches enorm:

**Treffer** (engl. *hit*), Daten/Instruktion sind im Cache  $\leadsto e_h$

**Fehlschlag** (engl. *miss*), Daten/Instruktion sind nicht im Cache  $\leadsto e_m$

 **Hits** sind schneller als **Misses**:  $e_m \gg e_h$

→ Strafe liegt schnell bei  $> 100$  Taktzyklen



☞ **Cache:** ein kleiner, schneller Zwischenspeicher

- Zugriffszeiten variieren je nach Zustand des Caches enorm:

**Treffer** (engl. *hit*), Daten/Instruktion sind im Cache  $\sim e_h$

**Fehlschlag** (engl. *miss*), Daten/Instruktion sind nicht im Cache  $\sim e_m$



**Hits** sind schneller als **Misses**:  $e_m \gg e_h$

→ Strafe liegt schnell bei  $> 100$  Taktzyklen

- **Eigenschaften von Caches mit Einfluss auf deren Analyse**

**Typ** ■ Cache für **Instruktionen**

■ Cache für **Daten**

■ **kombinierter** Cache für **Instruktionen und Daten**

**Auslegung** ■ **direkt abgebildet** (engl. *direct mapped*)

■ **vollasoziativ** (engl. *fully associative*)

■ **satz- oder mengenassoziativ** (engl. *set associative*)

**Seiteneretzungsstrategie**

■ engl. *(pseudo) least recently used*, (Pseudo-)LRU

■ engl. *(pseudo) first in first out*, (Pseudo-)FIFO



- Wissen ob eine Instruktion / ein Datum im Cache ist, oder nicht:

**must**, die Instruktion ist **garantiert im Cache**

- man kann immer die schnellere Ausführungszeit  $e_h$  annehmen
  - wird für die Vorhersage von Treffern verwendet



- Wissen ob eine Instruktion / ein Datum im Cache ist, oder nicht:

**must**, die Instruktion ist **garantiert im Cache**

- man kann immer die schnellere Ausführungszeit  $e_h$  annehmen
  - wird für die Vorhersage von Treffern verwendet

**may**, die Instruktion ist **vielleicht im Cache**

- ist dies nicht der Fall, muss man die Ausführungszeit  $e_m$  annehmen
  - wird für die Vorhersage von Fehlschlägen verwendet



- Wissen ob eine Instruktion / ein Datum im Cache ist, oder nicht:

**must**, die Instruktion ist **garantiert im Cache**

- man kann immer die schnellere Ausführungszeit  $e_h$  annehmen
- wird für die Vorhersage von Treffern verwendet

**may**, die Instruktion ist **vielleicht im Cache**

- ist dies nicht der Fall, muss man die Ausführungszeit  $e_m$  annehmen
- wird für die Vorhersage von Fehlschlägen verwendet

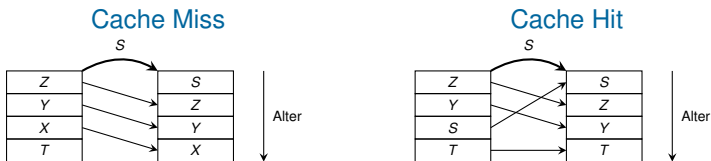
**persistent**, die Instruktion **verbleibt im Cache**

- erster Zugriff ist ein Fehlschlag, alle weiteren sind Treffer
- erster Zugriff:  $e_m$ , weitere Zugriffe:  $e_h$ 
  - ist besonders für Schleifen interessant, die den Cache „füllen“



# Beispiel: LRU-Cache, 4-fach assoziativ

LRU = „least recently used“ – Das älteste Element fliegt raus!

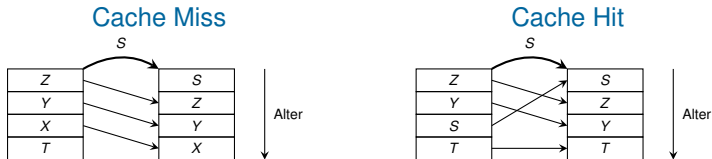


- Caches werden häufig in **Sätze** (engl. *cache set*) unterteilt
    - Ein  $n$ -fach assoziativer Cache besitzt pro Satz  $n$  Cache-Blöcke
    - Aufnahme von  $n$  konkurrierende Speicherstellen pro Satz möglich
    - Inhalt und Verwaltungsinformation (bei LRU das Alter des Blocks) werden sowohl bei Treffern als auch bei Fehlschlägen aktualisiert
- **Konkrete Semantik** des Caches



# Beispiel: LRU-Cache, 4-fach assoziativ

LRU = „least recently used“ – Das älteste Element fliegt raus!



- Caches werden häufig in **Sätze** (engl. *cache set*) unterteilt
    - Ein  $n$ -fach assoziativer Cache besitzt pro Satz  $n$  Cache-Blöcke
    - Aufnahme von  $n$  konkurrierende Speicherstellen pro Satz möglich
    - Inhalt und Verwaltungsinformation (bei LRU das Alter des Blocks) werden sowohl bei Treffern als auch bei Fehlschlägen aktualisiert
- **Konkrete Semantik** des Caches

- ☞ **must-Analyse** und **may-Analyse** approximieren diese konkrete Semantik:
  - must** Obergrenze des Alters  $\rightsquigarrow$  Unterapproximation des Inhalts
    - Obergrenze  $\leq$  Assoziativität  $\rightsquigarrow$  garantiert im Cache
  - may** Untergrenze des Alters  $\rightsquigarrow$  Überapproximation des Inhalts
    - Untergrenze  $>$  Assoziativität  $\rightsquigarrow$  garantiert nicht im Cache

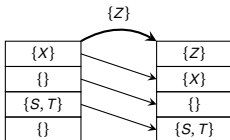


# Beispiel: LRU-Cache, Zugriff auf eine Speicherstelle

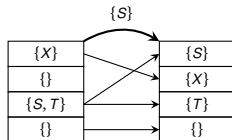
- Annäherung des Cache-Verhaltens durch must- und may-Approximation: Aktualisierung von Inhalt und Verwaltungsinformation

must-  
Approximation

Potential Cache Miss

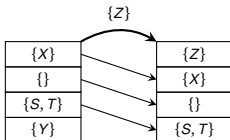


Definitive Cache Hit

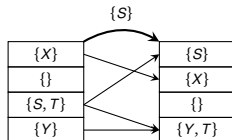


may-  
Approximation

Definitive Cache Miss



Potential Cache Hit



# Wie funktioniert nun die Cache-Analyse?

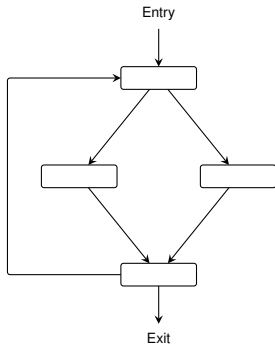
---

 Die Analyse ist eine [Datenflussanalysen](#) [1, Kapitel 8]



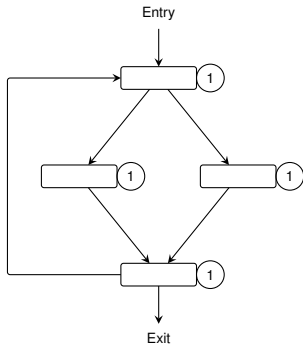
# Wie funktioniert nun die Cache-Analyse?

Die Analyse ist eine Datenflussanalysen [1, Kapitel 8]



# Wie funktioniert nun die Cache-Analyse?

Die Analyse ist eine **Datenflussanalysen** [1, Kapitel 8]

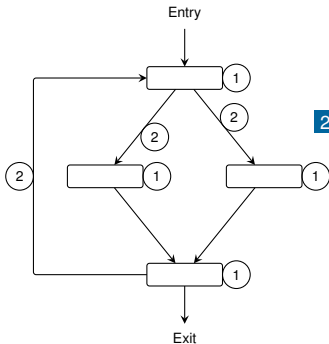


- 1 sammle Information in den Grundblöcken
  - Speicherzugriffe (s. Folie V/14)
  - man bestimmt die **Übertragungsfunktion** (engl. *transfer function*) des Grundblocks



# Wie funktioniert nun die Cache-Analyse?

Die Analyse ist eine **Datenflussanalysen** [1, Kapitel 8]

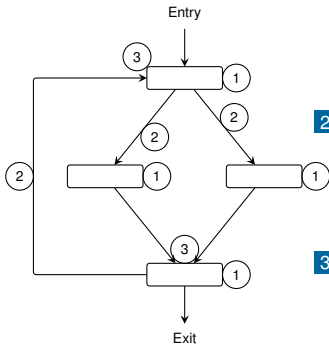


- 1 sammle Information in den Grundblöcken
  - Speicherzugriffe (s. Folie V/14)
  - man bestimmt die **Übertragungsfunktion** (engl. *transfer function*) des Grundblocks
- 2 die Information wird über ausgehende Kanten weiterverteilt
  - Eingabe für die Übertragungsfunktion der folgenden Grundblöcke



# Wie funktioniert nun die Cache-Analyse?

Die Analyse ist eine **Datenflussanalysen** [1, Kapitel 8]

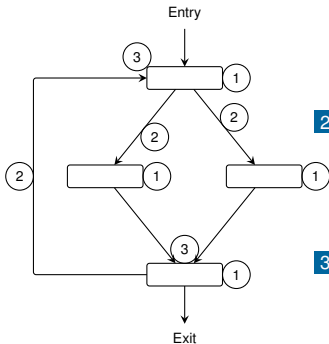


- 1 sammle Information in den Grundblöcken
  - Speicherzugriffe (s. Folie V/14)
  - man bestimmt die **Übertragungsfunktion** (engl. *transfer function*) des Grundblocks
- 2 die Information wird über ausgehende Kanten weiterverteilt
  - Eingabe für die Übertragungsfunktion der folgenden Grundblöcke
- 3 fließt der Kontrollfluss wieder zusammen, wird auch die Information verschmolzen
  - Verschmelzungsoperatoren



# Wie funktioniert nun die Cache-Analyse?

Die Analyse ist eine **Datenflussanalysen** [1, Kapitel 8]

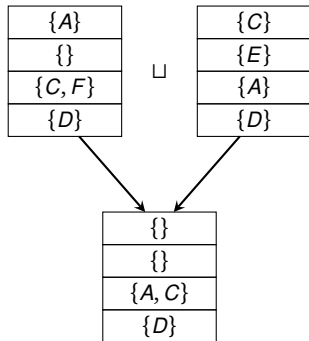


- 1 sammle Information in den Grundblöcken
  - Speicherzugriffe (s. Folie V/14)
  - man bestimmt die **Übertragungsfunktion** (engl. *transfer function*) des Grundblocks
- 2 die Information wird über ausgehende Kanten weiterverteilt
  - Eingabe für die Übertragungsfunktion der folgenden Grundblöcke
- 3 fließt der Kontrollfluss wieder zusammen, wird auch die Information verschmolzen
  - Verschmelzungsoperatoren

**Verschmelzungsoperatoren für must- und may-Analyse**



## must-Analyse

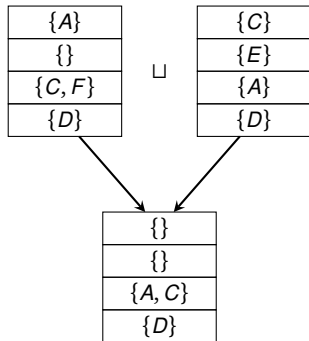


„Schnittmenge + max. Alter“



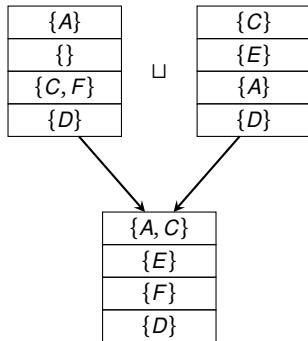
# Verschmelzungsoperatoren für must- und may-Analyse

must-Analyse



„Schnittmenge + max. Alter“

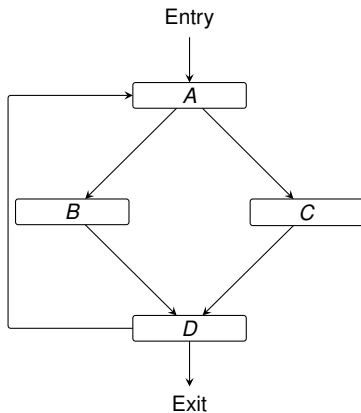
may-Analyse



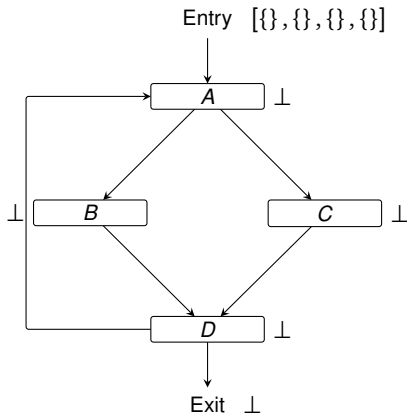
„Vereinigungsmenge + min. Alter“



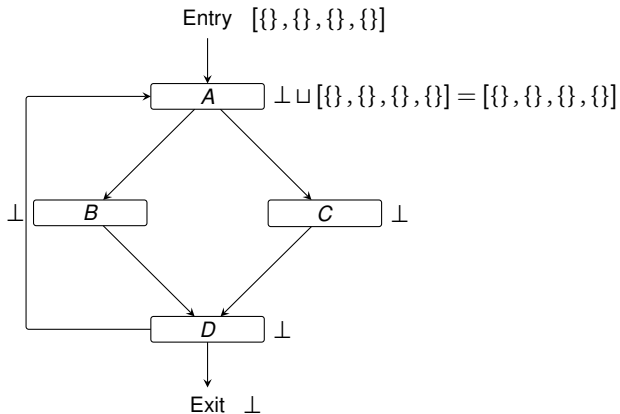
## Beispiel: must-Analyse für LRU



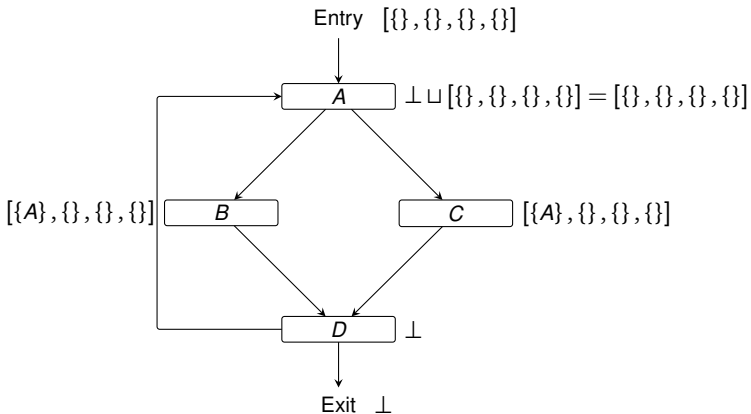
## Beispiel: must-Analyse für LRU



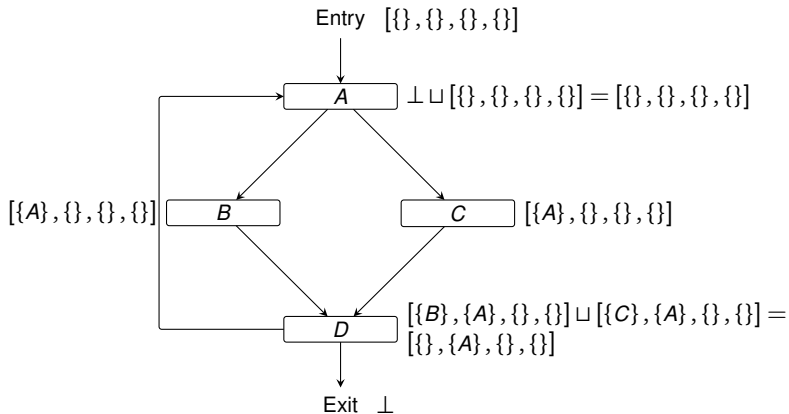
## Beispiel: must-Analyse für LRU



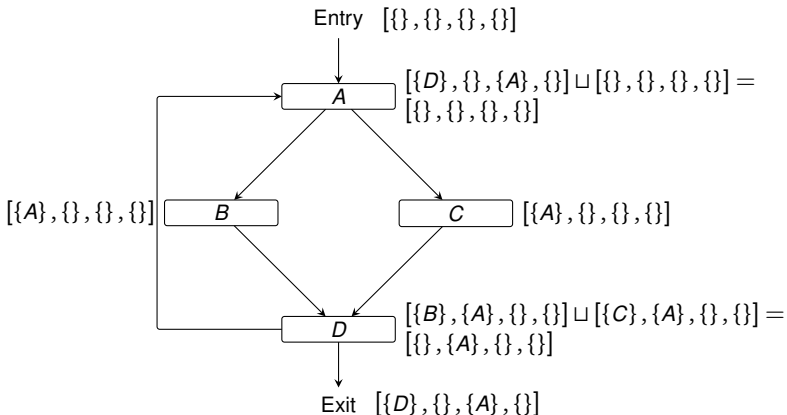
## Beispiel: must-Analyse für LRU



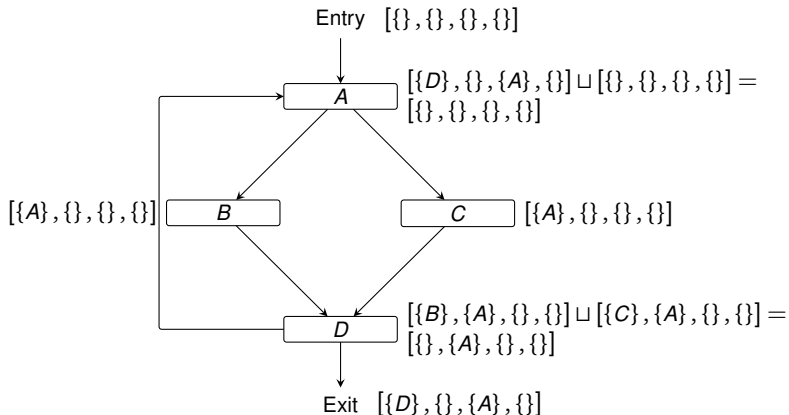
## Beispiel: must-Analyse für LRU



## Beispiel: must-Analyse für LRU



## Beispiel: must-Analyse für LRU



Hier ist leider keine Vorhersage von Treffern möglich ☹️





Cache-Analyse mithilfe einer Datenflussanalyse funktioniert für mengenassoziative Caches mit LRU sehr gut

- Zugriffe auf unterschiedliche Cache-Zeilen beeinflussen sich nicht
- Beispiel TriCore: 2-fach assoziativer LRU-Cache





Cache-Analyse mithilfe einer Datenflussanalyse funktioniert für **mengenassoziative Caches mit LRU** sehr gut

- Zugriffe auf unterschiedliche Cache-Zeilen beeinflussen sich nicht
  - Beispiel TriCore: 2-fach assoziativer LRU-Cache



Es kommen auch andere Strategien zum Einsatz:

- Im Durchschnitt ähnliche Leistung wie LRU, **weniger vorhersagbar**
  - **Pseudo-LRU**
    - Cache-Zeilen werden als Blätter eines Baums verwaltet
    - must-Analyse **eingeschränkt brauchbar**, may-Analyse **unbrauchbar**
    - Beispiel: PowerPC 750/755
  - **Pseudo-Round-Robin**
    - 4-fach mengenassoziativer Cache mit **einem** 2-bit Ersetzungszähler
    - must-Analyse **kaum**, may-Analyse **überhaupt nicht brauchbar**
    - Beispiel: Motorola Coldfire 5307



- ☞ Cache-Analyse mithilfe einer Datenflussanalyse funktioniert für **mengenassoziative Caches mit LRU** sehr gut
  - Zugriffe auf unterschiedliche Cache-Zeilen beeinflussen sich nicht
    - Beispiel TriCore: 2-fach assoziativer LRU-Cache
- ⚠ Es kommen auch andere Strategien zum Einsatz:
  - Im Durchschnitt ähnliche Leistung wie LRU, **weniger vorhersagbar**
    - **Pseudo-LRU**
      - Cache-Zeilen werden als Blätter eines Baums verwaltet
      - must-Analyse **eingeschränkt brauchbar**, may-Analyse **unbrauchbar**
      - Beispiel: PowerPC 750/755
    - **Pseudo-Round-Robin**
      - 4-fach mengenassoziativer Cache mit **einem** 2-bit Ersetzungszähler
      - must-Analyse **kaum**, may-Analyse **überhaupt nicht brauchbar**
      - Beispiel: Motorola Coldfire 5307

☞ Keine belastbaren Aussagen zum STM32F411



- 1 Rekapitulation: Worst-Case Execution Time
- 2 Ausflug: Cache-Analyse
  - Grundlagen
  - Beispiel: LRU-Cache
- 3 WCET-Analyse auf dem EZS-Board
  - GPIOs
  - aiT



## General Purpose Input/Output

- Pins eines Mikrochips zur *freien Verwendung*
- Konfigurierbar als Ein-/Ausgang
- Teilweise pegelfest bis 5 V  
~ Mikrocontroller-Handbuch lesen ☺
- Zugriff über
  - spezielle Speicheradressen
  - Spezialanweisungen

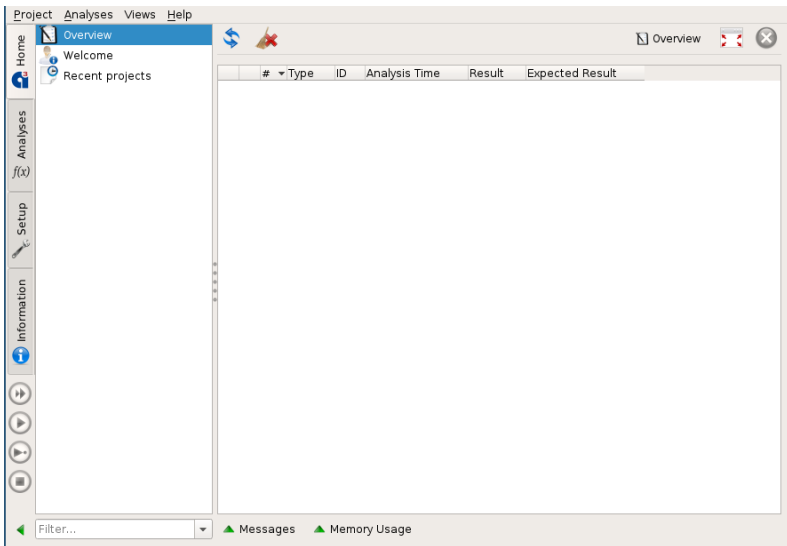
## Ansteuerung

```
void ezs_gpio_set(bool) //PD12
```

## Auswertung

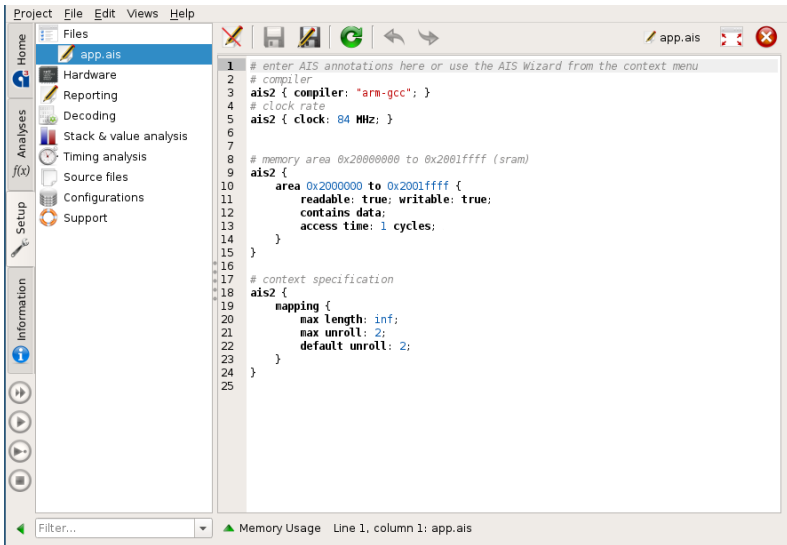
☞ Oszilloskop





The screenshot displays the 'Project' configuration window in the AbsInt aiT software. The window has a menu bar with 'Project', 'Views', and 'Help'. On the left is a sidebar with categories: Home, Analyses, Setup, and Information. The 'Files' category is selected, showing a list of project-related items: Files, Hardware, Reporting, Decoding, Stack & value analysis, Timing analysis, Source files, Configurations, and Support. The main area contains configuration fields: 'Executable:' with the value 'build/app.elf', 'AIS file:' with 'app.ais', 'Report file:', and 'XML results file:'. There is a checkbox for 'Show MD5 sums' which is currently unchecked. At the bottom, the 'Project:' path is '/home/cip/nf2011/ki08jofa/cip/ezs-auf...ben/Ausfuehrungszeit/timeAnalysis.apx' and the 'Temporary directory:' is '/tmp/a3-0UWLyB'. A 'Filter...' dropdown is at the bottom left, and a 'Memory Usage' indicator is at the bottom right.

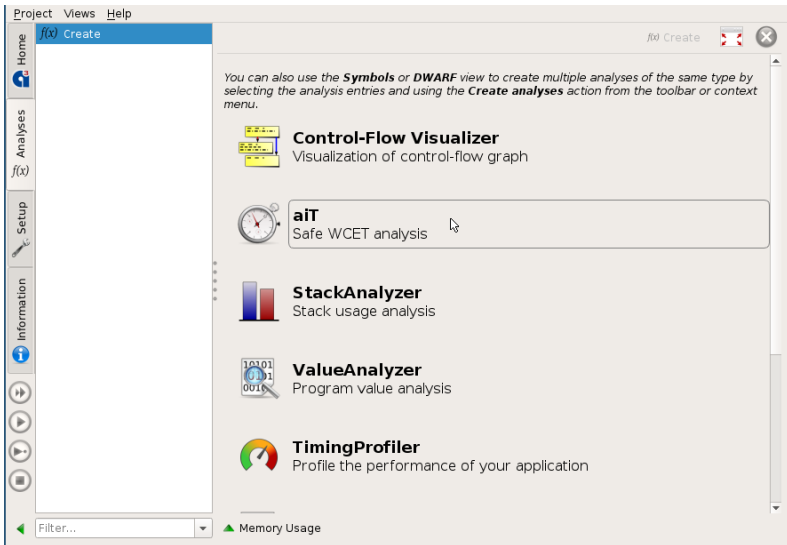




The screenshot displays the AbsInt aiT IDE interface. On the left, a sidebar contains navigation options: Home, Analyses, Setup, and Information. The 'Home' section is active, showing a file tree with 'app.ais' selected. The main editor window shows the AIS configuration file 'app.ais' with the following content:

```
1 # enter AIS annotations here or use the AIS Wizard from the context menu
2 # compiler
3 ais2 { compiler: "arm-gcc"; }
4 # clock rate
5 ais2 { clock: 84 MHz; }
6
7
8 # memory area 0x20000000 to 0x2001ffff (sram)
9 ais2 {
10     area 0x20000000 to 0x2001ffff {
11         readable: true; writable: true;
12         contains data;
13         access time: 1 cycles;
14     }
15 }
16
17 # context specification
18 ais2 {
19     napping {
20         max length: inf;
21         max unroll: 2;
22         default unroll: 2;
23     }
24 }
25
```

The status bar at the bottom indicates 'Memory Usage' and 'Line 1, column 1: app.ais'.



The screenshot shows the 'Create' dialog box in the AbsInt aiT software. The window title is 'f(x) Create'. The left sidebar contains navigation buttons: Home, Analyses, Setup, and Information. The main area displays a list of analysis types with their respective icons and descriptions:

- Control-Flow Visualizer**: Visualization of control-flow graph
- aiT**: Safe WCET analysis
- StackAnalyzer**: Stack usage analysis
- ValueAnalyzer**: Program value analysis
- TimingProfiler**: Profile the performance of your application

At the bottom of the dialog, there is a 'Filter...' dropdown menu and a 'Memory Usage' indicator with an upward-pointing green triangle.



The screenshot shows the 'Create' dialog for a new analysis in the AbsInt aiT software. The interface includes a menu bar (Project, Analysis, Views, Help), a toolbar, and a sidebar with navigation options: Home, Analyses, f(x), Setup, and Information. The main area is divided into two panes. The left pane shows a list of analyses, with 'aiT' selected. The right pane contains the configuration fields for the new analysis:

- ID:
- Comment:
- Configuration:
- Dependencies:
- Analysis start:  (highlighted in red)
- AIS file:  (with file selection and edit icons)
- Report file:  (with file selection and edit icons)
- XML report file:  (with file selection and edit icons)
- HTML report file:  (with file selection and edit icons)
- GDL output:  (with file selection and edit icons)
- Expected result:  (with a dropdown menu set to 'cycles')
- Result: n/a

At the bottom, there is a 'Filter...' dropdown, and status indicators for 'Messages' and 'Memory Usage'.

Project Analysis Views Analysis Start

f(x) Create

aiT

Regular expression

Address	Name
0x08009d8d	__addf3
0x0800a6ed	__addsf3
0x0800b139	__aeabi_atexit
0x0800a5b1	__aeabi_cdcmpeq
0x0800a5b1	__aeabi_cdcmple
0x0800a5a1	__aeabi_cdrcmpeq
0x0800a651	__aeabi_d2iz
0x0800a6a1	__aeabi_d2uiz
0x08009d8d	__aeabi_dadd
0x0800a5c1	__aeabi_dcmpeq
0x0800a5fd	__aeabi_dcmpge
0x0800a611	__aeabi_dcmpgt
0x0800a5e9	__aeabi_dcmple
0x0800a5d5	__aeabi_dcmpit
0x0800a625	__aeabi_dcmpun
0x0800a345	__aeabi_ddiv
0x0800a0f1	__aeabi_dmul
0x08009d81	__aeabi_drsub
0x08009d89	__aeabi_dsub
0x0800a049	__aeabi_f2d
0x0800a6ed	__aeabi_fadd
0x0800aa65	__aeabi_fdiv
0x0800a0fd	__aeabi_fmud

786 functions

Cancel OK

Filter... Messages Memory Usage

aiT

cycles



The screenshot shows the 'Timing analysis' configuration window in AbsInt aiT. The window has a menu bar with 'Project', 'Views', and 'Help'. On the left is a sidebar with categories: Home (Files, app.ais, Hardware, Reporting, Decoding, Stack & value analysis), Analyses (Timing analysis, Source files, Configurations, Support), Setup, and Information. The main area is titled 'Timing analysis' and contains three sections: 'Cache Analysis', 'Pipeline Analysis', and 'Path Analysis'. 'Cache Analysis' has 'Instruction cache' and 'Data cache' both set to 'Normal'. 'Pipeline Analysis' has 'WCET computation mode' set to 'Global worst-case' (with a note 'applies only to aiT analyses'), 'Threshold for applying default memory regions' set to '1024' kB, and checkboxes for 'Enable widening for cache states' (checked), 'Skip timing analysis for main entry if additional starts are defined', 'Detect timing anomalies' (with a note 'requires "Path analysis variant" to be "Prediction file ba'), and 'Generate pipeline basic block statistics'. 'Path Analysis' has 'Default loop bound' and 'Default recursion bound' both set to '4', 'Path analysis variant' set to 'ILP based', and 'ILP solver' set to 'clpsolve'. At the bottom, there is a 'Filter...' dropdown and a 'Memory Usage' indicator.

Project Views Help

Home

- Files
- app.ais
- Hardware
- Reporting
- Decoding
- Stack & value analysis

Analyses

- Timing analysis
- Source files
- Configurations
- Support

Setup

Information

Timing analysis

**Cache Analysis**

Instruction cache: Normal

Data cache: Normal

**Pipeline Analysis**

WCET computation mode: Global worst-case *applies only to aiT analyses*

Threshold for applying default memory regions: 1024 kB

Enable widening for cache states

Skip timing analysis for main entry if additional starts are defined

Detect timing anomalies *requires "Path analysis variant" to be "Prediction file ba*

Generate pipeline basic block statistics

**Path Analysis**

Default loop bound: 4

Default recursion bound: 4

Path analysis variant: ILP based

ILP solver: clpsolve

Filter... Memory Usage



The screenshot displays the AbsInt aiT software interface. The top menu bar includes 'Project', 'Analysis', 'Views', and 'Help'. The main window is titled 'aiT' and contains a left sidebar with navigation options: 'Home', 'Analyses', 'Setup', and 'Information'. The 'Analyses' section is active, showing a list of analyses with 'aiT' selected. A tooltip 'Start analysis' is visible over the 'Start analysis' button in the sidebar. The main area on the right shows configuration details for the selected analysis:

- ID: aiT
- Comment: (empty)
- Configuration: Default Configuration
- Dependencies: (empty)
- Analysis start: sample\_job
- AIS file: (empty)
- Report file: (empty)
- XML report file: (empty)
- HTML report file: (empty)
- GDL output: (empty)

Below the configuration fields, there is a section for 'Errors, warnings and info' with a 'Latest log' button. The log shows the following steps:

- aiT - aiT (0 Errors, 0 Warnings): Finished after 2 seconds
  - Control Flow Reconstruction
  - Value Analysis
  - Cache & Pipeline Analysis
  - Prediction File Optimization
  - Path Analysis (ILP Based)
  - ILP Solving
  - Applying Path Analysis Results
  - Creating GDL visualization
  - Reporting
  - Creating HTML report
  - Finished after 2 seconds with 0 errors, 0 warnings

At the bottom of the interface, there is a 'Filter...' dropdown, 'Messages' and 'Memory Usage' indicators.



The screenshot displays the AbsInt aiT software interface. The main window shows analysis results for a 'sample\_job'. A tooltip 'Display analysis results' is visible over the 'Analysis graph' icon in the toolbar. The analysis results are as follows:

- Computed Worst Case for Entry 'sample\_job': 2.12  $\mu\text{s}$
- Cache Statistics
  - L1 Instruction Cache: max 31 hits, max 5 misses

Below the statistics, a call graph visualization shows the following components:

- sample\_job: 1.62  $\mu\text{s}$
- initialize\_adc: 83.334 ns
- sample\_adc: 0.417  $\mu\text{s}$

The bottom status bar shows the following messages:

- Errors, warnings and info Latest log
- Call Graph - aiT (0 Errors, 0 Warnings): Finished after 1 second
- Control Flow Reconstruction
- Creating GDL visualization
- Finished after 1 second with 0 errors, 0 warnings

At the bottom of the interface, there are buttons for 'Messages' and 'Memory Usage'.



The screenshot shows the AbsInt aiT software interface. The main window is titled "Statistics" and displays a table of analysis results. The table has columns for "Routine", "Calls", "Self [cycles]", "Self [ns]", and "Self [ms]". The data is as follows:

Routine	Calls	Self [cycles]	Self [ns]	Self [ms]
<b>sample_job</b>	1	136	1619.05	
<b>sample_adc</b>	1	35	416.67	
<b>initialize_adc</b>	1	7	83.33	

Below the table, there is a section titled "Call Graph - aiT (0 Errors, 0 Warnings): Finished after 1 second". It contains a list of tasks with green checkmarks:

- Control Flow Reconstruction
- Creating GDL visualization
- Finished after 1 second with 0 errors, 0 warnings

The interface also shows a sidebar with navigation options like Home, Analyses, Setup, and Information. The top menu includes Project, Analysis, Statistics, Views, and Help. A tooltip "Display analysis statistics" is visible over the Statistics icon in the toolbar.



- aiT gibt Zeitmessungen zunächst nur in Takten aus
- Taktrate angeben  $\leadsto$  tatsächliche Zeit

## Beispiele:

```
clock: 84MHz;  
clock: 83.95 .. 84.05 MHz;
```



- Manche Codestücke sind nicht analysierbar

→ Ausführungszeit annotieren



Natürlich nur sinnvoll, wenn WCET bereits bekannt

## Beispiel:

```
routine "even"{
  not analyzed;
  takes: 150 cycles;
}
```

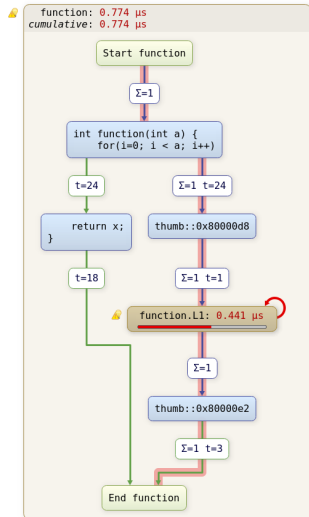
```
# exclude code as far as specified program points
instruction ProgramPoint snippet {
  continue at: ProgramPoint1 , PP2 , ... , PPn;
  not analyzed;
  takes: 10 cycles;
}
```



- Genaue Anzahl von Schleifendurchläufen zu bestimmen ist teuer
- ☞ aiT versucht standardmäßig nur zwei Durchläufe zu interpretieren
- Lohnt sich jedoch manchmal
- ☞ aiT mehr Freiheiten für die Analyse geben

### Beispiel:

```
loop "function.L1" mapping {  
    default unroll: 100;  
}
```



- Grenzen von Hand spezifizieren

### Beispiele:

```
loop "function.L1" { bound: 0 .. 10 end; }  
loop "function.L1" { bound: 10 begin; }  
loop "function.L1" { bound: 10 .. inf end; }  
loop "function.L1" { takes 20 ms; }
```

- Grenzen in Abhängigkeit von Registern spezifizieren

### Beispiele:

```
loop "function.L1" {  
    bound: 0 .. floor((reg("r0") - reg("r1")) / 4);  
}
```



The screenshot shows a code editor with a line of code: `ais2 { loop "function.L1" { bound: 0 ... <int>; #mapping defa...`. A lightbulb icon is next to the code. A context menu is open over the code, listing actions such as Copy, Copy part, Show in call graph, Show in disassembly, Show in file, Add annotation (highlighted), Show all folded messages of this type, Show all folded messages, Reset state of all folded messages, Clear all, Expand recursively, Collapse recursively, Expand all, and Collapse all.

Below the code editor, a log window titled "Errors, warnings and info" shows the following output:

- aiT - aiT (0 Errors, 1 Warning): Finished on 2018-11-08 at 09:30:35 after analyzing for 3 seconds
  - Control Flow Reconstruction
  - Value Analysis
  - Cache & Pipeline Analysis
  - Prediction File Optimization
  - Path Analysis (ILP Based)
- #7172: For loop 'function.L1' the default loop bound of 4 is used.
  - Last process took 0 s and used not more than 0 MB (RSS 0 MB) of memory
  - ILP Solving
  - Reporting
  - Creating HTML report
- Finished on 2018-11-08 at 09:30:35 after analyzing for 3 seconds with 0 errors, 1 warning

Das Herausforderung ist nicht die Syntax, sondern das Finden (präziser) Schleifengrenzen



### Problem:

```
if (C) {  
    A(); // Vorbedingungen für Schleife in R()  
    R();  
} else {  
    B(); // Andere Vorbedingungen für R()  
    R();  
}
```

### Lösung:

```
// Annotations-"Variable" rmax definieren  
routine "A" { enter with: user("rmax") = 10; }  
routine "B" { enter with: user("rmax") = 20; }  
// "Variable" in Annotation nutzen  
loop "R.L1" { bound: 0 .. user("rmax"); }
```



aiT ist oft nicht in der Lage  
Rekursionen zu analysieren  
→ Grenzen von Hand spezifizieren

## Problem:

```
int fib(int n) {  
    if (n <= 1)  
        return n;  
    return fib(n-1)  
        + fib(n-2);  
}
```

## Beispiele:

```
routine "fib" { recursion bound: 0 .. 10; }  
routine "fib" { recursion bound: 10; }  
routine "fib" { recursion bound: 5 .. 10; }
```

- Weitere Annotationen im Hilfe-Menü des aiT

→ „*AIS2 quick reference*“



# Besprechung der Übungsaufgabe

## „Ausführungszeit“



- [1] Steven S. Muchnick.  
*Advanced compiler design and implementation.*  
Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1997.
- [2] Peter Puschner.  
*Zeitanalyse von Echtzeitprogrammen.*  
PhD thesis, Technische Universität Wien, Institut für Technische Informatik, Treitlstr. 1-3/182-1,  
1040 Vienna, Austria, 1993.
- [3] Reinhard Wilhelm.  
Embedded systems.  
<http://react.cs.uni-sb.de/teaching/embedded-systems-10-11/lecture-notes.html>,  
2010.  
Lecture Notes.

