

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>

Sommersemester 2022



- 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

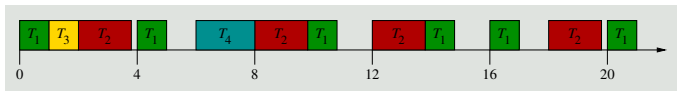
- Grundlagen
- Beispiel: LRU-Cache

3 WCET-Analyse auf dem EZS-Board

- GPIOs
- aiT



Worst-Case Execution Time



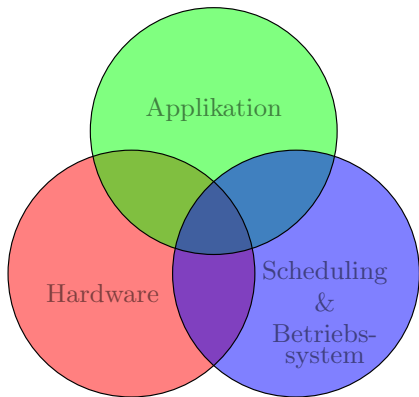
■ Eine entscheidende Größe für:

- 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 }
```

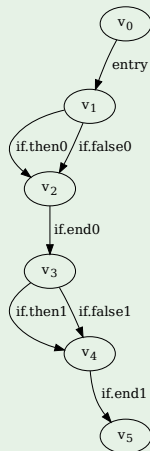
- T-Graph aus Kontrollflussgraph abgeleitet
- Worst Case == maximaler Fluss durch T-Graph



```
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 }
```

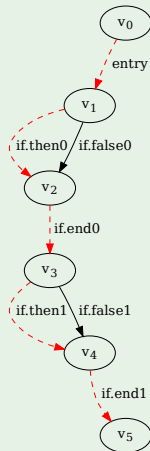
- T-Graph aus Kontrollflussgraph abgeleitet
- Worst Case == maximaler Fluss durch T-Graph
- Nebenbedingungen des Flussproblems:
 - $freq(entry) = freq(if.then0) + freq(if.false0)$
 - $freq(if.then0) + freq(if.false0) = freq(if.end0)$
 - ...
- Nebenbedingungen werden für Integer Linear Program (ILP) verwendet: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

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



```
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 [3]

■ Ergebnis: $e_{\text{getop}} = 80$ 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  
movl    a6@(8) ,a2      // 16 Zyklen  
movl    a6@(12) ,d3     // 16 Zyklen
```

Quelle: Peter Puschner [3]

■ 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 mit Laufzeitanomalien behaftete Systeme

- (intuitive) Annahmen über Worst-Case-Verhalten verletzt
- Lokales Maximum führt nicht zwingend zu globalem Maximum

■ 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  
movl    a6@(8),a2       // 16 Zyklen  
movl    a6@(12),d3      // 16 Zyklen
```

Quelle: Peter Puschner [3]

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

■ Annahmen:

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



Äußerst pessimistisch und zum Teil falsch

Laufzeitanomalie: Beispiel

- Initialer Cachezustand zu Beginn der untersuchten Aufgabe?





Grundproblem: Ausführungszyklen von Instruktionen zählen

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

Quelle: Peter Puschner [3]

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

■ Annahmen:

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



Äußerst pessimistisch und zum Teil falsch

Laufzeitanomalie: Beispiel

- Initialer Cachezustand zu Beginn der untersuchten Aufgabe?
- Leerer Cache?





Grundproblem: Ausführungszyklen von Instruktionen zählen

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

Quelle: Peter Puschner [3]

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

■ Annahmen:

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



Äußerst pessimistisch und zum Teil falsch

Laufzeitanomalie: Beispiel

- Initialer Cachezustand zu Beginn der untersuchten Aufgabe?
- Leerer Cache?



Abhängig von konkretem Cacheverhalten



Falsche Annahme bei FIFO-Cache [1]





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 [3]

■ 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 mit Laufzeitanomalien behaftete Systeme

- (intuitive) Annahmen über Worst-Case-Verhalten verletzt
- Lokales Maximum führt nicht zwingend zu globalem Maximum

■ 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  
movl    a6@(8),a2       // 16 Zyklen  
movl    a6@(12),d3      // 16 Zyklen
```

Quelle: Peter Puschner [3]

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

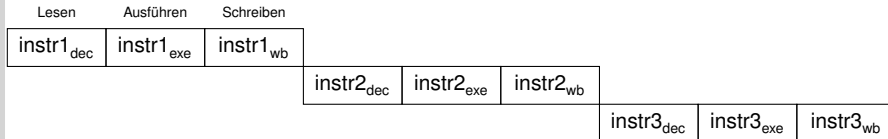
■ Annahmen:

- 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  
movl    a6@(8) ,a2      // 16 Zyklen  
movl    a6@(12) ,d3     // 16 Zyklen
```

Quelle: Peter Puschner [3]

■ 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 _{dec}	instr1 _{exe}	instr1 _{wb}		
	instr2 _{dec}	instr2 _{exe}	instr2 _{wb}	
		instr3 _{dec}	instr3 _{exe}	instr3 _{wb}





Grundproblem: Ausführungszyklen von Instruktionen zählen

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

Quelle: Peter Puschner [3]

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

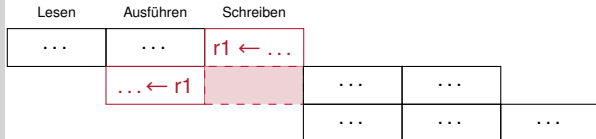
■ Annahmen:

- 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 [3]

■ 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 mit Laufzeitanomalien behaftete Systeme

- (intuitive) Annahmen über Worst-Case-Verhalten verletzt
- Lokales Maximum führt nicht zwingend zu globalem Maximum


■ 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 $\leadsto e_h$

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

- 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*)

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

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

Zeilenersetzungsstrategie

■ 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

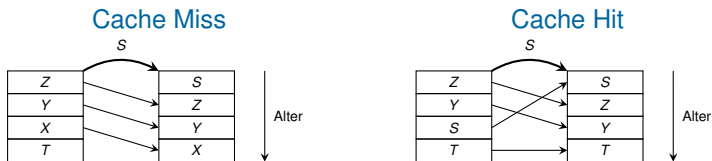
persistent, die Instruktion **verbleibt im Cache**

- 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!

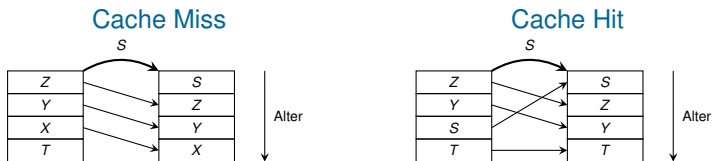


- Caches werden häufig in **Sätze** (engl. *cache set*) unterteilt
 - Ein *n*-fach assoziativer Cache besitzt pro Satz *n* Cache-Blöcke
 - Aufnahme von *n* konkurrierende Speicherstellen pro Satz möglich
 - Inhalt und Verwaltungsinformation (bei LRU das Alter des Blocks) werden sowohl bei Treffern als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

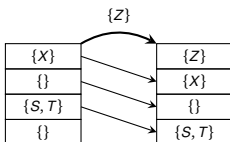


Beispiel: LRU-Cache, Zugriff auf eine Speicherstelle

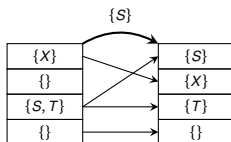
- Annäherung des Cache-Verhaltens durch must- und may-Approximation: Aktualisierung von Inhalt und Verwaltungsinformation

must-
Approximation

Potential Cache Miss

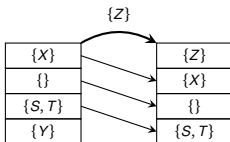


Definitive Cache Hit

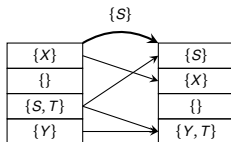


may-
Approximation

Definitive Cache Miss



Potential Cache Hit



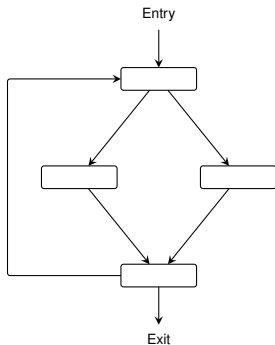
Wie funktioniert nun die Cache-Analyse?

 Die Analyse ist eine [Datenflussanalysen](#) [2, Kapitel 8]



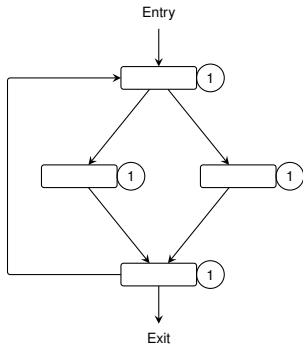
Wie funktioniert nun die Cache-Analyse?

Die Analyse ist eine **Datenflussanalysen** [2, Kapitel 8]



Wie funktioniert nun die Cache-Analyse?

Die Analyse ist eine **Datenflussanalysen** [2, Kapitel 8]

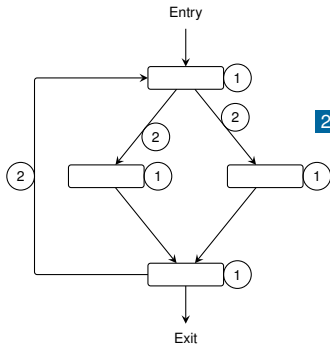


- 1 sammle Information in den Grundblöcken
 - Speicherzugriffe (s. Folie V/14)
 - man bestimmt die **Übertragungsfunktion** (engl. *transfer function*) des Grundblocks



Wie funktioniert nun die Cache-Analyse?

Die Analyse ist eine **Datenflussanalysen** [2, Kapitel 8]

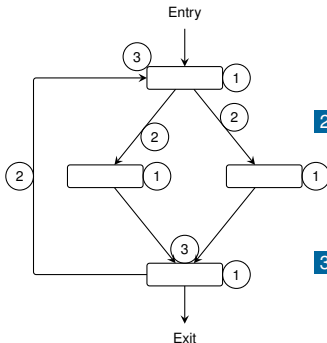


- 1 sammle Information in den Grundblöcken
 - Speicherzugriffe (s. Folie V/14)
 - man bestimmt die **Übertragungsfunktion** (engl. *transfer function*) des Grundblocks
- 2 die Information wird über ausgehende Kanten weiterverteilt
 - Eingabe für die Übertragungsfunktion der folgenden Grundblöcke



Wie funktioniert nun die Cache-Analyse?

Die Analyse ist eine **Datenflussanalysen** [2, Kapitel 8]

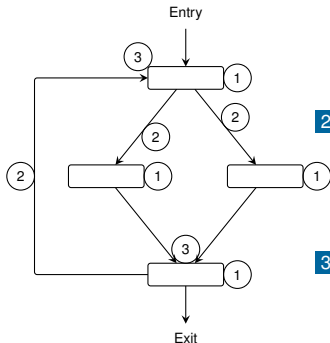


- 1 sammle Information in den Grundblöcken
 - Speicherzugriffe (s. Folie V/14)
 - man bestimmt die **Übertragungsfunktion** (engl. *transfer function*) des Grundblocks
- 2 die Information wird über ausgehende Kanten weiterverteilt
 - Eingabe für die Übertragungsfunktion der folgenden Grundblöcke
- 3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



Wie funktioniert nun die Cache-Analyse?

Die Analyse ist eine **Datenflussanalysen** [2, Kapitel 8]

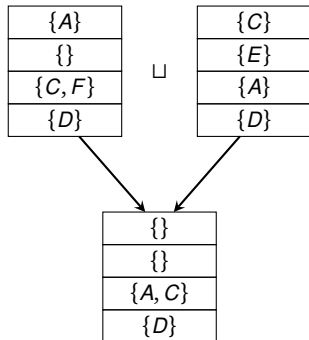


- 1 sammle Information in den Grundblöcken
 - Speicherzugriffe (s. Folie V/14)
 - man bestimmt die **Übertragungsfunktion** (engl. *transfer function*) des Grundblocks
- 2 die Information wird über ausgehende Kanten weiterverteilt
 - Eingabe für die Übertragungsfunktion der folgenden Grundblöcke
- 3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

Verschmelzungsoperatoren für must- und may-Analyse



must-Analyse

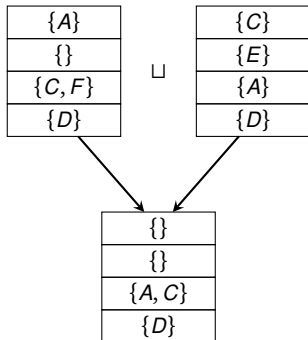


„Schnittmenge + max. Alter“



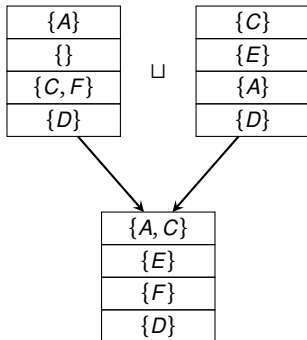
Verschmelzungsoperatoren für must- und may-Analyse

must-Analyse



„Schnittmenge + max. Alter“

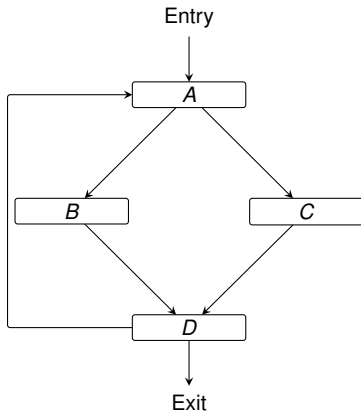
may-Analyse



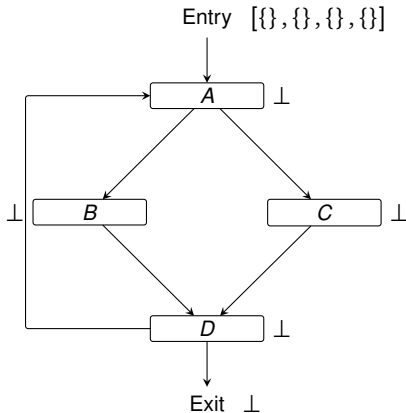
„Vereinigungsmenge + min. Alter“



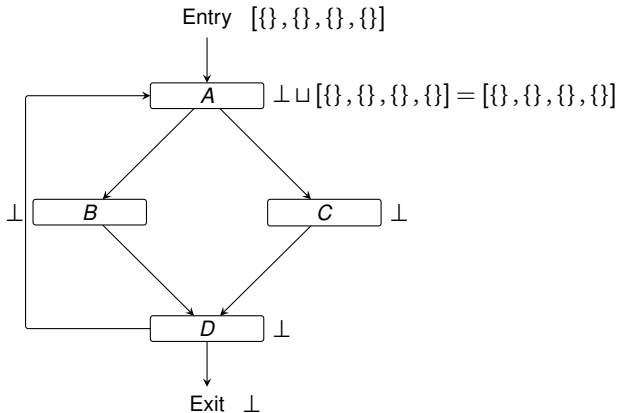
Beispiel: must-Analyse für LRU



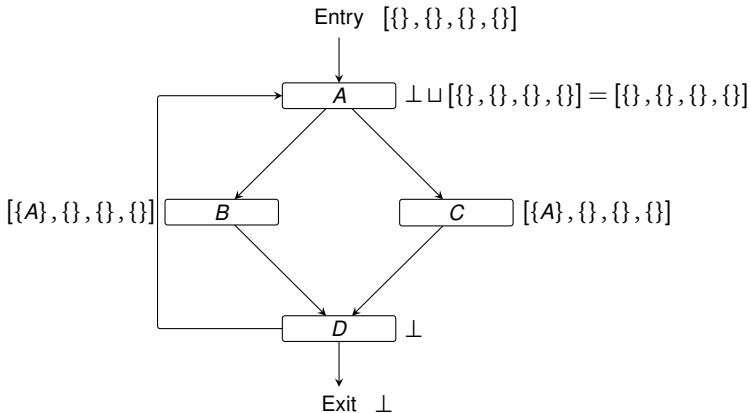
Beispiel: must-Analyse für LRU



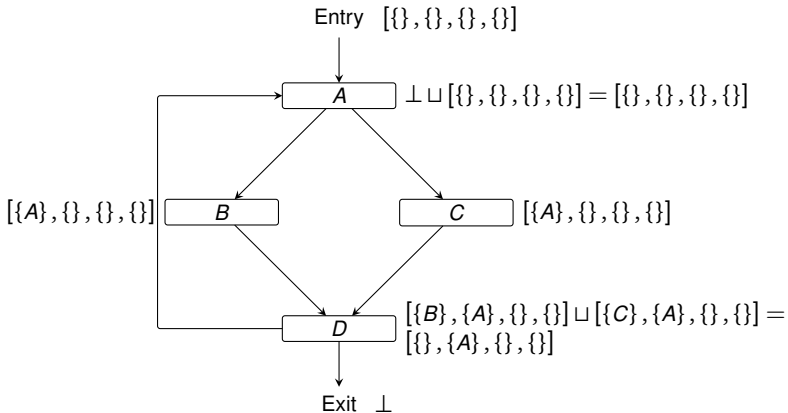
Beispiel: must-Analyse für LRU



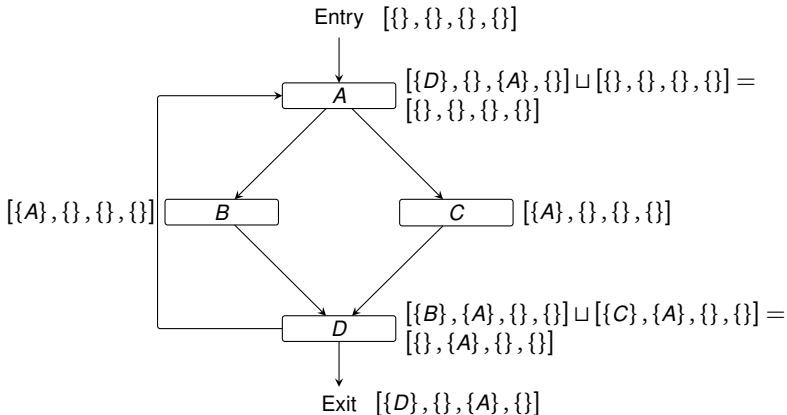
Beispiel: must-Analyse für LRU



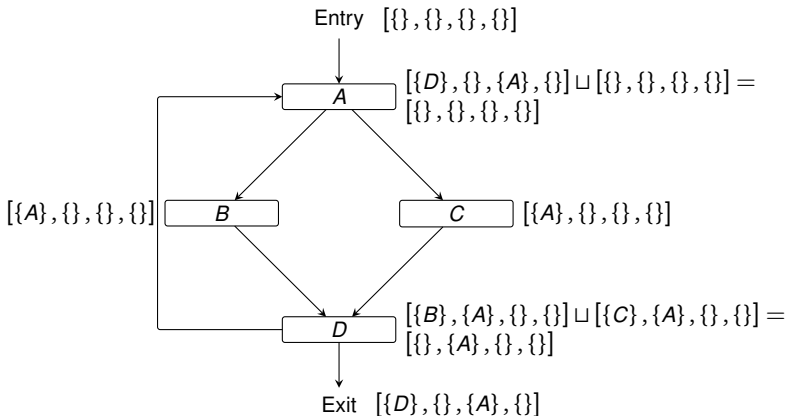
Beispiel: must-Analyse für LRU



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 STM32F429



- 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

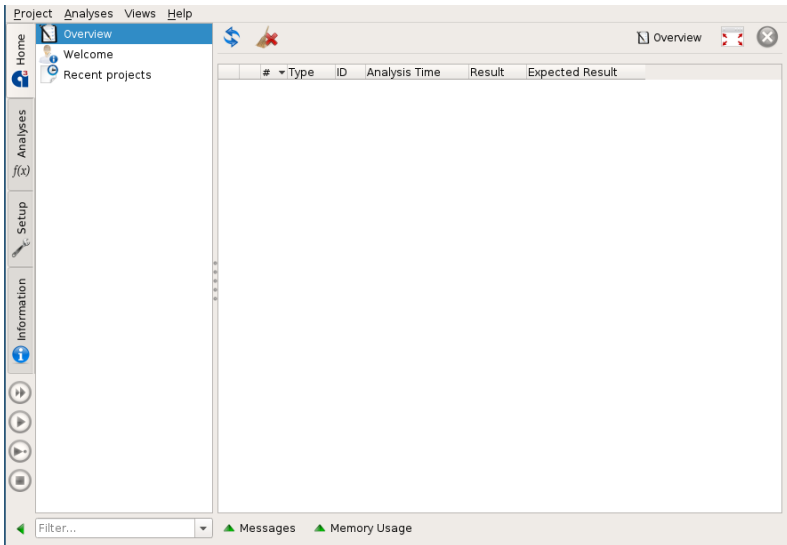
Ansteuerung

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

Auswertung

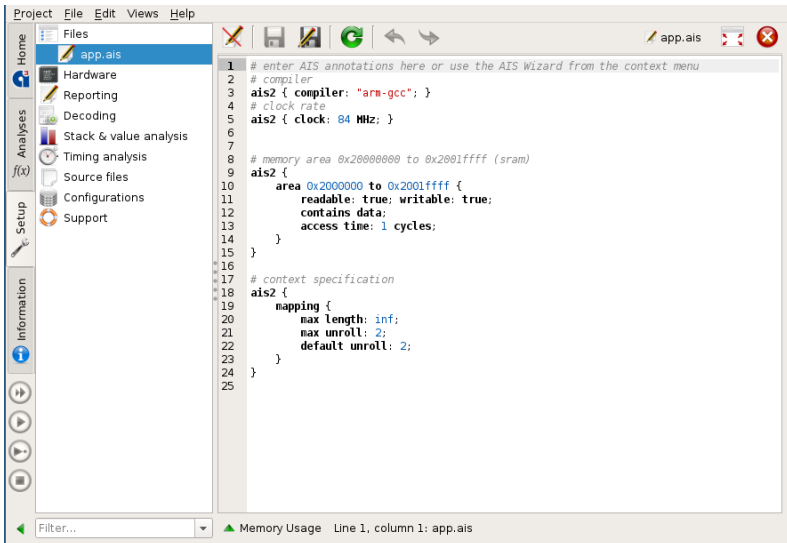
➡ Oszilloskop





The screenshot displays the 'Project' 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' (containing 'Files', 'Hardware', 'Reporting', 'Decoding'), 'Analyses' (containing 'Stack & value analysis', 'Timing analysis', 'Source files'), 'Configurations' (containing 'Support'), 'Setup', and 'Information'. The 'Files' category is selected, showing a list of project files. The main area contains configuration fields: 'Executable:' (build/app.elf), 'AIS file:' (app.ais), 'Report file:', and 'XML results file:'. There is a checkbox for 'Show MD5 sums'. At the bottom, it shows 'Project: /home/cip/nf2011/ki08jofa/cip/ezs-auf...ben/Ausfuehrungszeit/timeAnalysis.apx' and 'Temporary directory: /tmp/a3-0UWLyB'. A 'Filter...' dropdown and a 'Memory Usage' indicator are also visible.

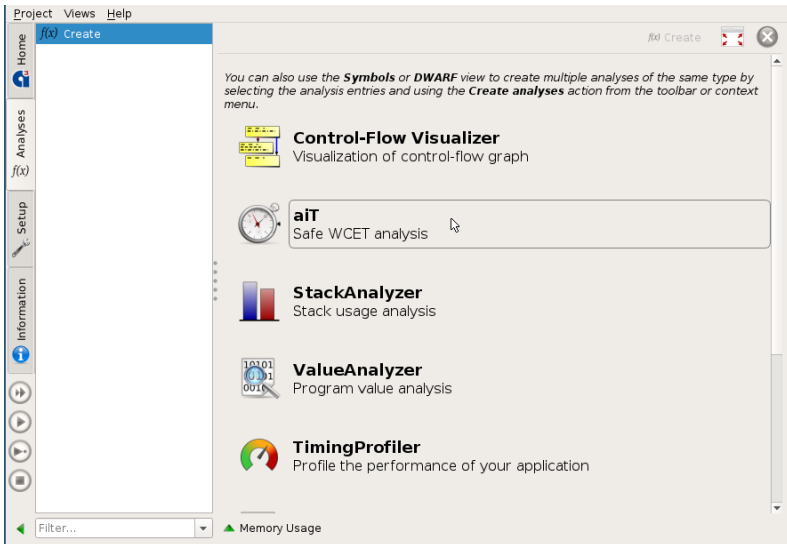




The screenshot displays the AbsInt aiT IDE interface. The top menu bar includes 'Project', 'File', 'Edit', 'Views', and 'Help'. The left sidebar contains a 'Home' view with a file tree showing 'app.ais' selected, and a vertical toolbar with icons for Home, Analyses (Decoding, Stack & value analysis, Timing analysis, Source files, Configurations, Support), Setup, and Information. The main editor window shows the content of 'app.ais' with line numbers 1 through 25. The code defines an AIS2 annotation for a compiler, clock rate, memory area, and context specification.

```
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 0x2000000 to 0x2001ffff {
11         readable: true; writable: true;
12         contains data;
13         access time: 1 cycles;
14     }
15 }
16
17 # context specification
18 ais2 {
19     mapping {
20         max length: inf;
21         max unroll: 2;
22         default unroll: 2;
23     }
24 }
25
```

At the bottom of the IDE, there is a 'Filter...' dropdown and a status bar showing 'Memory Usage Line 1, column 1: app.ais'.



The screenshot shows the 'Create' dialog in the AbsInt aiT software. The window title is 'f(x) Create'. The left sidebar contains a 'Home' button and a list of analysis types: 'Analyses', 'Setup', and 'Information'. The main area displays a list of analysis options:

- Control-Flow Visualizer**: Visualization of control-flow graph
- aiT**: Safe WCET analysis (highlighted with a mouse cursor)
- 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.



The screenshot shows the 'Create' dialog in the AbsInt aiT software. The window title is 'Project Analysis Views Help' and the main title is 'f(x) Create'. The 'aiT' analysis is selected in the left sidebar. The main area contains the following fields:

- 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, 'Messages' and 'Memory Usage' indicators, and a red circular arrow icon in the bottom-left corner.

Project Analysis Views Analysis Start

Regular expression

Address	Name
0x08009d8d	__adddf3
0x0800a6ed	__addsf3
0x0800b139	__aeabi_atexit
0x0800a5b1	__aeabi_cdcmpeq
0x0800a5b1	__aeabi_cdcmple
0x0800a5a1	__aeabi_cdrcmple
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
0x0800a6f4	__aeabi_fmul

786 functions

Cancel OK

Home f(x) Create aiT

Analyses f(x)

Setup

Information

Filter... Messages Memory Usage

aiT

cycles



The screenshot shows the 'Timing analysis' configuration window in AbsInt aiT. The window has a menu bar (Project, Views, Help) and a sidebar with navigation options: Home, Analyses (selected), f(x), Setup, and Information. The main area is titled 'Timing analysis' and contains several analysis configuration sections:

- Cache Analysis:** Instruction cache: Always hit; Data cache: Always hit.
- Pipeline Analysis:** WCET computation mode: Global worst-case (applies only to aiT analyses); Threshold for applying default memory regions: 1024 kB; Skip timing analysis for main entry if additional starts are defined (unchecked); Generate pipeline basic block statistics (unchecked); Interactive pipeline visualization (checked).
- Path Analysis:** Default loop bound: 4; Default recursion bound: 4; Path analysis variant: ILP based; ILP solver: clpsolve.
- TimeWeaver:** Restrict loop bounds with trace results (unchecked); Process complete traces only (unchecked).

At the bottom, there is a search bar with 'Filter...' and buttons for 'Search' and '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 several sections:

- Left Panel:** A vertical sidebar with icons for 'Home', 'Analyses', 'Setup', and 'Information'. The 'Analyses' section is active, showing a list of analyses with 'aiT' selected.
- Configuration Panel:** A form for configuring the analysis. Fields include:
 - 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)
- Log Panel:** A scrollable area showing the execution log. The log entry for 'aiT - aiT (0 Errors, 0 Warnings): Finished after 2 seconds' is expanded, showing a list of completed tasks:
 - 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
- Bottom Panel:** A status bar with a 'Filter...' dropdown, 'Messages' icon, and 'Memory Usage' icon.

A mouse cursor is pointing to the 'Start analysis' button in the 'Information' section of the left sidebar.



The screenshot displays the AbsInt aiT software interface. The main window shows the results of an analysis. At the top, a toolbar contains a button labeled "Display analysis results". Below this, a text box reports: "Computed Worst Case for Entry 'sample_job': 2.12 μ s". Underneath, "Cache Statistics" are shown: "- L1 Instruction Cache: max 31 hits, max 5 misses". A call graph visualization shows three nodes: "sample_job: 1.62 μ s" at the top, which branches into "initialize_adc: 83.334 ns" on the left and "sample_adc: 0.417 μ s" on the right. At the bottom, a log window titled "Errors, warnings and info" shows the following messages:

- 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



Project Analysis Statistics Views Help

Home f(x) Create
aiT
Control-Flow graph
Analysis graph
Disassembly
Statistics

Analyses f(x)
Setup
Information

WCET
WCET (context)
Variable usage
Variable usage (context)
Object size
Infeasibility
Sources

Display analysis statistics

Filter: 3 of 3 visible

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

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

Filter... Messages Memory Usage



- 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;
}
```

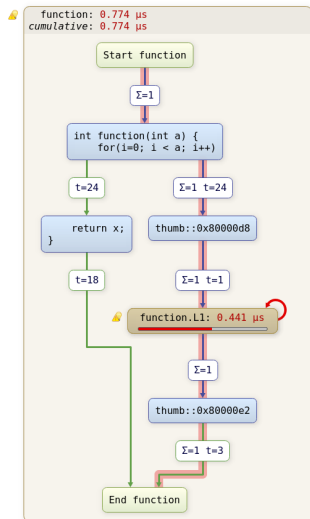
```
# exclude code as far as specified program points
instruction ProgramPoint snippet {
  continue at: ProgramPoint1 , PP2 , ... , PPn;
  not analyzed;
  takes: 10 cycles;
}
```



- 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 loop annotation: `ais2 { loop "function.L1" { bound: 0 ... <int>; #mapping defa...`. Below the code is a log window titled "Errors, warnings and info" with a "Latest log" button. The log contains several entries, including a warning about the default loop bound of 4. A context menu is open over the warning entry, listing actions such as "Copy", "Copy part", "Show in call graph", "Add annotation", and "Expand all".

👉 Die 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] Franck Cassez, René Rydhof Hansen, and Mads Chr Olesen.
What is a timing anomaly?
In *Proceedings of the 12th International Workshop on Worst-Case Execution Time Analysis (WCET '12)*. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2012.
- [2] Steven S. Muchnick.
Advanced compiler design and implementation.
Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1997.
- [3] Peter Puschner.
Zeitanalyse von Echtzeitprogrammen.
PhD thesis, Technische Universität Wien, Institut für Technische Informatik, Treitlstr. 1-3/182-1, 1040 Vienna, Austria, 1993.
- [4] Reinhard Wilhelm.
Embedded systems.
<http://react.cs.uni-sb.de/teaching/embedded-systems-10-11/lecture-notes.html>,
2010.
Lecture Notes.

