

Verlässliche Echtzeitsysteme

Übungen zur Vorlesung

Statische Stackbedarfsanalyse

Phillip Raffeck, Simon Schuster, Peter Wägemann

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

Wintersemester 2021





```

1  /* Objective function */
2  max: +16 md5_orig_init +64 md5_update \
3      +64 md5_final +16 md5_memset \
4      +208 md5_transform +16 md5_encode ... ;
5
6
7  /* Constraints */
8  +main = 1;
9  +md5_init +md5_main <= +main;
10 ...
  
```

Beispiel: md5-Summe¹

Vorgehen

1. Callgraph bestimmen
2. Stackbedarf einzelner Funktionen (gcc -fstack-usage)
3. ILP² aufstellen (Nebenbedingungen aus 1., Kosten aus 2. verwenden)
4. ILP z.B. mittels lp_solve \sim **maximaler Stackbedarf**

¹<https://github.com/tacle/tacle-bench/>

²Integer Linear Program (dt. ganzzahliges lineares Programm)

- 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:

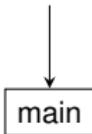
```
max : +64  n_f1  +48  n_f2  +42  n_f3 ;
```

Semantik

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

Formalisierung

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



lp_solve -Syntax

```
n_main <= 1;
```



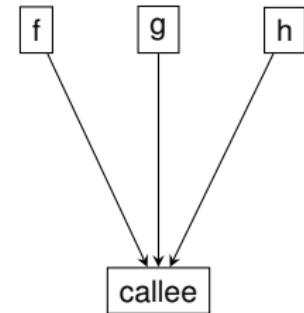
Semantik

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

Formalisierung

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

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



lp_solve -Syntax

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



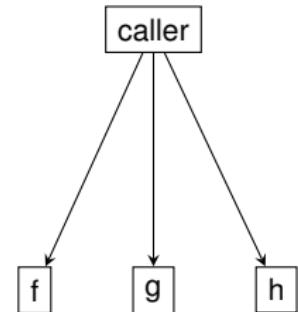
Semantik

Jede Funktionsinkarnation ruft gleichzeitig jeweils maximal eine weitere Funktion auf

Formalisierung

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

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



lp_solve -Syntax

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



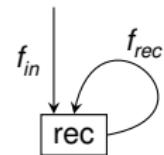
Semantik

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

Formalisierung

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

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



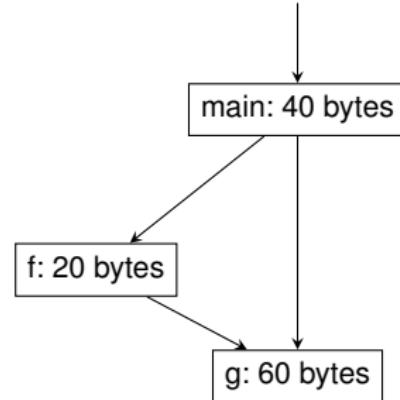
lp_solve -Syntax

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

Beispiel

- Problemformulierung in Ipsolve:

```
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;
```



Beispiel

- Problemformulierung in lp_solve:

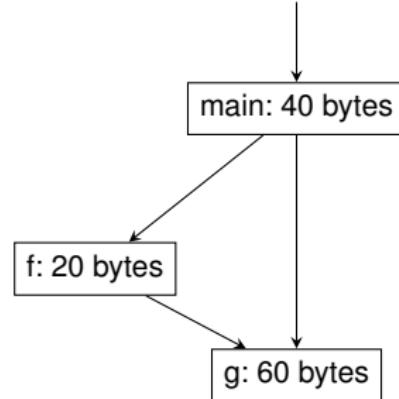
```
max: +40 n_main +20 n_f +60 n_g;  
n_main <= 1;  
+f_main_f +f_main_g <= n_main;  
n_f <= +f_main_f;  
+f_f_g <= n_f;  
n_g <= +f_f_g +f_main_g;
```

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

³<http://lpsolve.sourceforge.net/5.5/Infeasible.htm>

```
$ 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;	max: x + y + z;	x: 5000
z <= y + 1;	z <= y + 1;	y: 20
y <= 20;	y <= 20;	z: 21
	x <= 5000;	
	y <= 5000;	
	z <= 5000;	



- lp_solve ist auf die Lösung linearer Gleichungssysteme ausgelegt
- Es ist dementsprechend nicht möglich, zwei Variablen zu multiplizieren
 - $a * b \Rightarrow$ Syntaxfehler
 - max : $a b \Rightarrow$ 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;
```

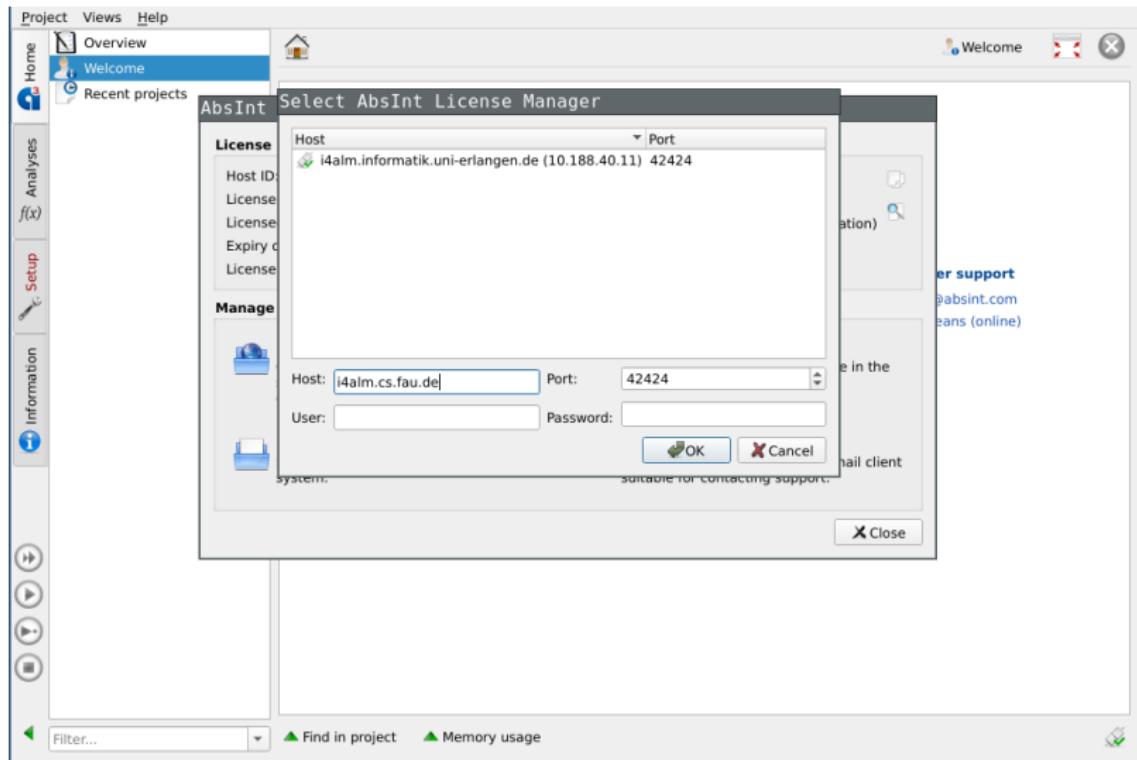
~> stackusage / lp_solvepp



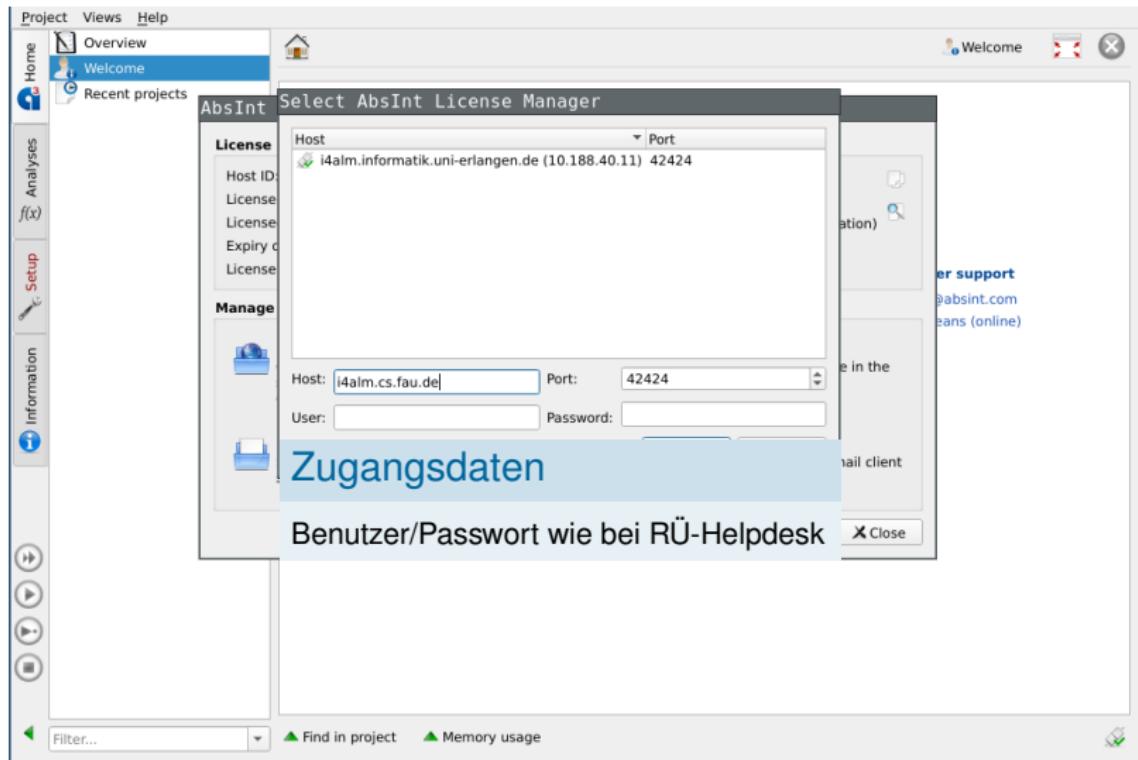
- Statische Code-Analyse mit a³ Tool-Suite
 1. aiT: WCET-Analyse
 2. Stack-Analyzer: Stackbedarf
 3. ...
- Installiert im CIP-Pool
- /proj/i4ezs/tools/a3_x86/bin/a3x86



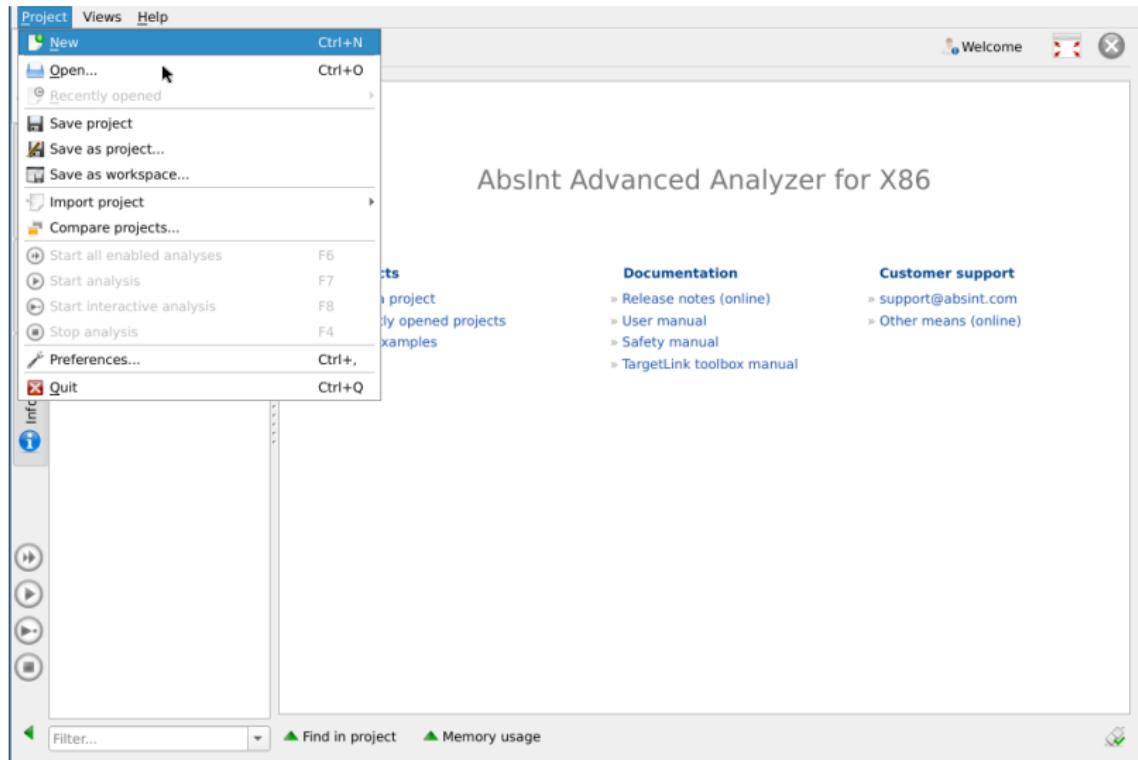
a³ Analyzer – Lizenzserver



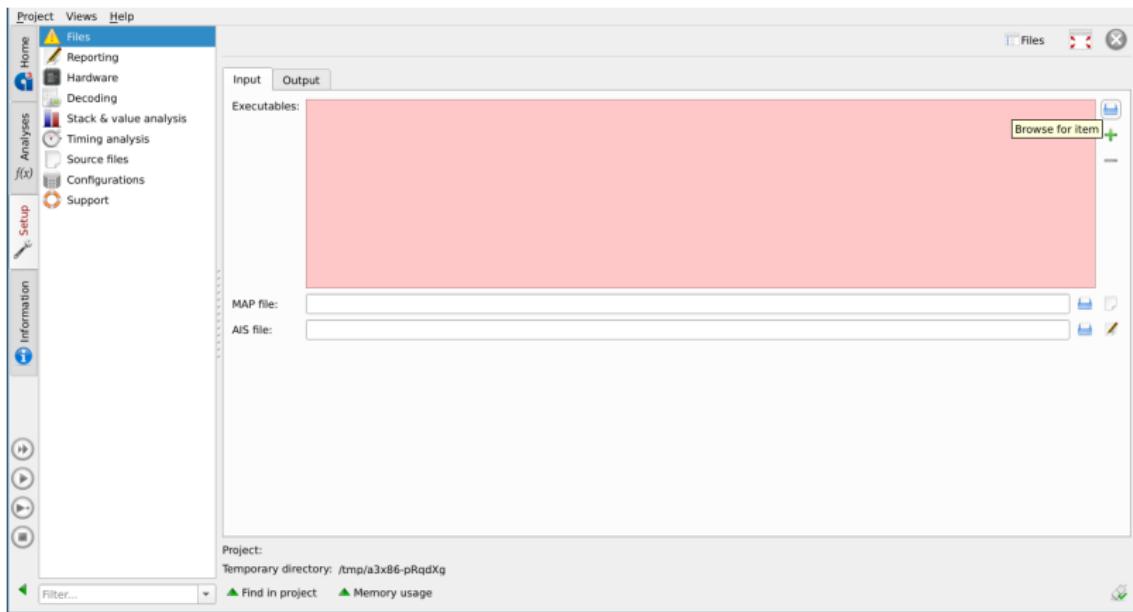
a³ Analyzer – Lizenzserver



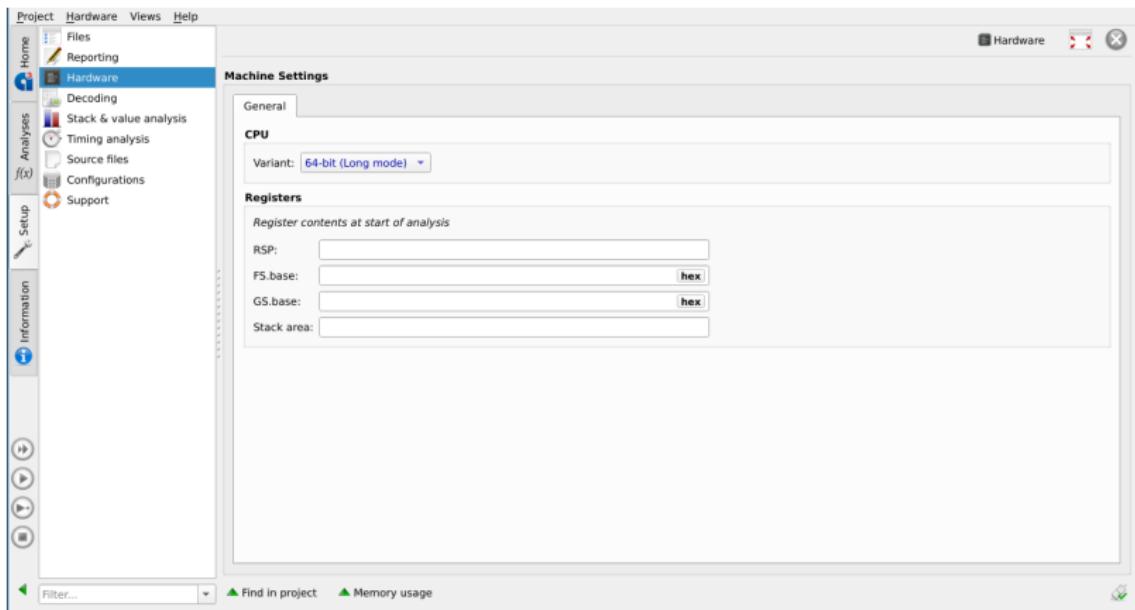
a³ Analyzer – Neues Projekt Anlegen



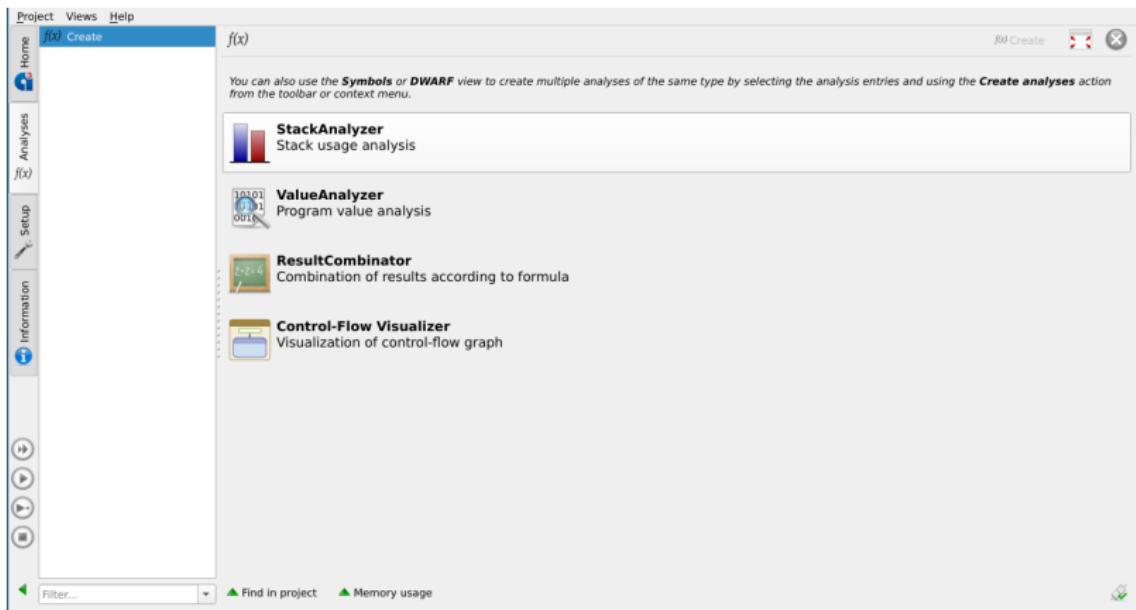
a³ Analyzer – Executable Angeben



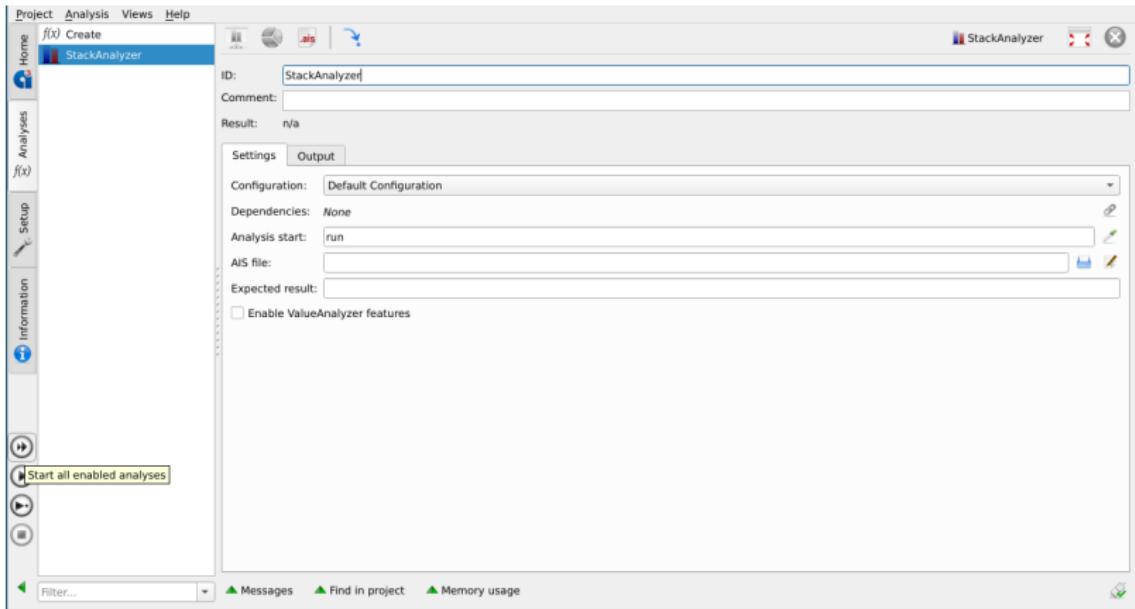
a³ Analyzer – Hardware Auswählen



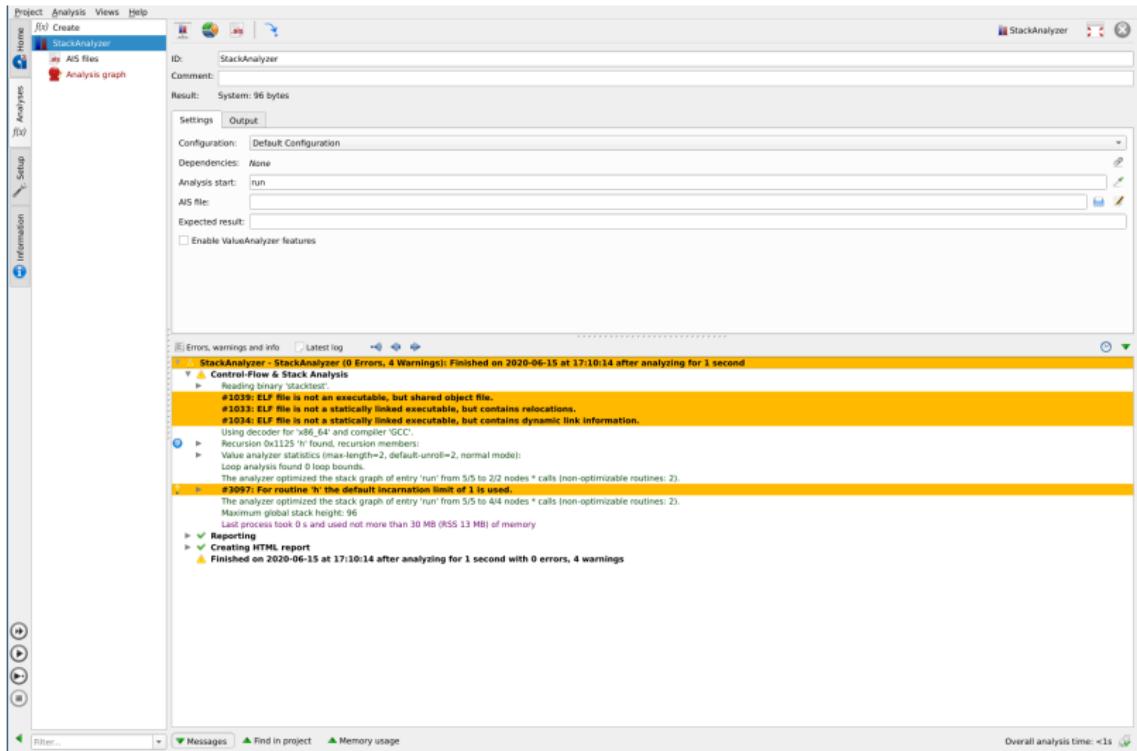
a³ Analyzer – Stack-Analyse Selektieren



a³ Analyzer – Stack-Analyse Starten



a³ Analyzer – Analyseoutput

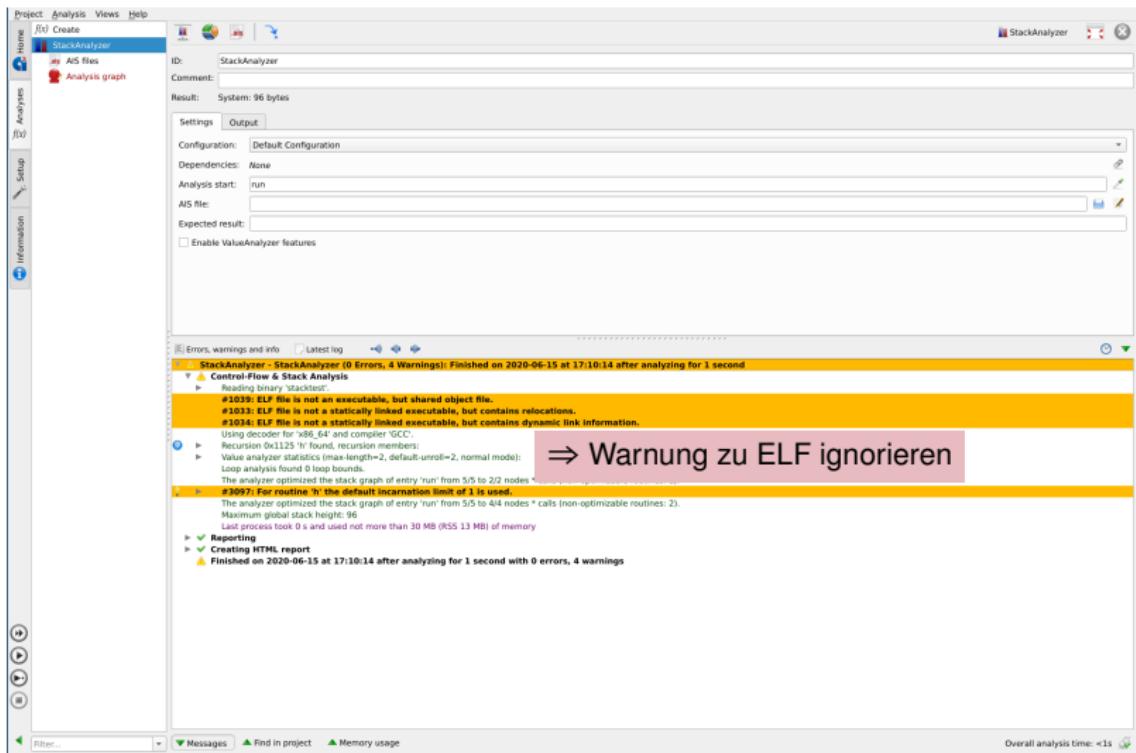


The screenshot shows the Abslnt Stack Analyzer interface with the following details:

- Project Bar:** Project, Analysis, Views, Help
- File Bar:** J2C, Create, StackAnalyzer, AIS files, Analysis graph
- Analyzer List:** Home, Analyses, StackAnalyzer (selected), Setup, Information
- Analyzer Details:**
 - ID: StackAnalyzer
 - Comment:
 - Result: System: 96 bytes
 - Settings: Output
 - Configuration: Default Configuration
 - Dependencies: None
 - Analysis start: run
 - AIS file:
 - Expected result:
 - Enable ValueAnalyzer features
- Log View:** Errors, warnings and info, Latest log. The log output is as follows:
 - StackAnalyzer - StackAnalyzer (0 Errors, 4 Warnings):** Finished on 2020-06-15 at 17:10:14 after analyzing for 1 second
 - Control-Flow & Stack Analysis:**
 - Reading binary 'StackAnalyzer'.
 - #1039: ELF file is not an executable, but shared object file.
 - #1033: ELF file is not a statically linked executable, but contains relocations.
 - #1034: ELF file is not dynamically linked executable, but contains dynamic link information.
 - Using decoder for 64 architecture "GCC".
 - Recursion 0x112 1 found, recursion memory!
 - Value analyzer statistics (max-length=2, default-unroll=2, normal mode):
Loop analysis found 0 loop bounds.
 - #2097: For routine 'run' the default incarnation limit of 1 is used.
 - The analysis optimized the stack graph of entry 'run' from 5/5 to 2/2 nodes * calls (non-optimizable routines: 2).
 - Maximum global stack height: 96
 - Last process took: 0.0 and used not more than 30 MB (955 13 MB) of memory
 - Reporting
 - Creating HTML report
 - Finished on 2020-06-15 at 17:10:14 after analyzing for 1 second with 0 errors, 4 warnings
- Bottom Bar:** Filter..., Messages, Find in project, Memory usage, Overall analysis time: <1s



a³ Analyzer – Analyseoutput



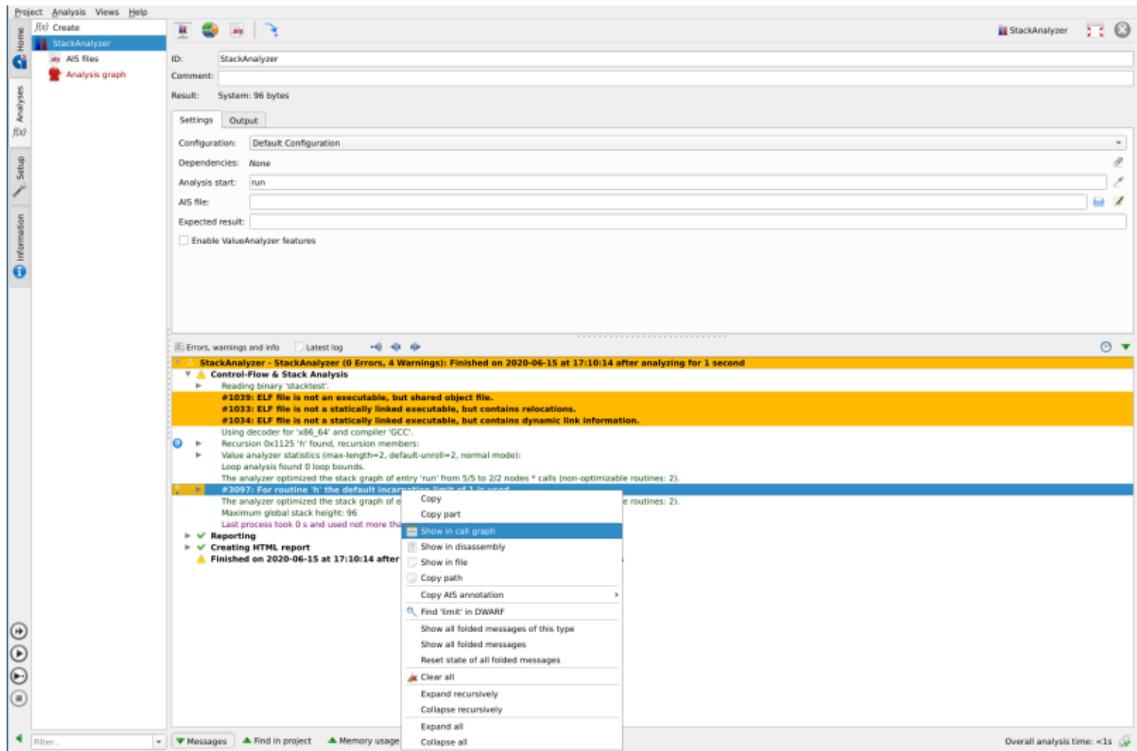
The screenshot shows the Absint Stack Analyzer interface. The main window displays a list of analysis results and errors. A prominent red box highlights a warning message: "⇒ Warnung zu ELF ignorieren".

StackAnalyzer (0 Errors, 4 Warnings): Finished on 2020-06-15 at 17:10:14 after analyzing for 1 second

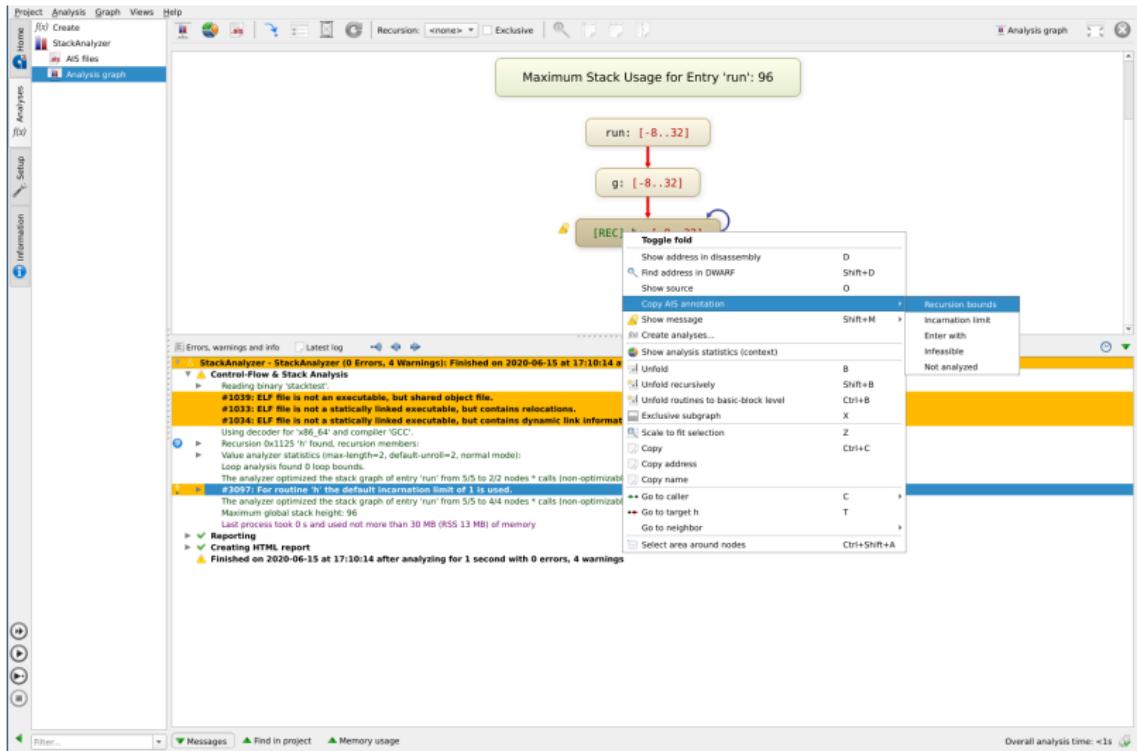
- Reading binary 'StackAnalyzer'.
- #1039: ELF file is not an executable, but shared object file.
- #1033: ELF file is not a statically linked executable, but contains relocations.
- #1034: ELF file is not dynamically linked executable, but contains dynamic link information.
- Using decision for 16 entries under "GCC".
- Recursion 0x1121 found, recursion maxdepth 1.
- Value analyzer statistics (max-length=2, deflault-roll=2, normal mode):
- Loop analysis found 0 loop bounds.
- #3097: For routine 'R' the default incarnation limit of 1 is used.
- This is the maximum size of the stack graph of entry 'R' from 5/5 to 2/2 nodes * calls (non-optimizable routines: 2).
- Maximum global stack height: 96.
- Last process took 0.0 and used not more than 30 MB (85513 MB) of memory.
- Reporting
- Creating HTML report
- Finished on 2020-06-15 at 17:10:14 after analyzing for 1 second with 0 errors, 4 warnings

Overall analysis time: <1s

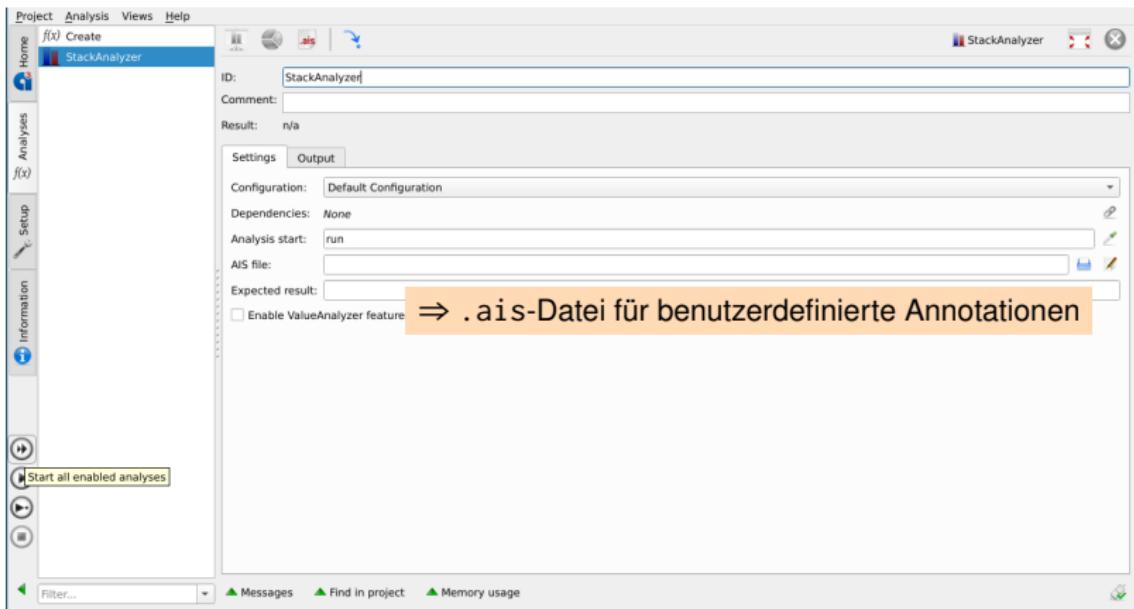
a³ Analyzer – Callgraph



a³ Analyzer – Annotationstemplate kopieren



a³ Analyzer – Stack-Analyse Starten



a³ Analyzer – Annotationstemplate kopieren

The screenshot shows the Absint Stack Analyzer (a³ Analyzer) interface. The main window displays a stack graph with nodes for 'run: [-8..32]' and 'g: [-8..32]'. A context menu is open over the 'g' node, with the title '[REC]'. The menu includes options like 'Copy AIS annotation', 'Show message', 'Create analyses...', 'Show analysis statistics (context)', 'Unfold', 'Unfold recursively', 'Exclusive subgraph', 'Scale to fit selection', 'Copy', 'Copy address', 'Copy name', 'Go to caller', 'Go to target h', 'Go to neighbor', and 'Select area around nodes'. A sub-menu for 'Recursion bounds' is also visible, containing 'Incarnation limit', 'Enter with', 'Intersible', and 'Not analyzed'. The status bar at the bottom right shows 'Overall analysis time: <1s'.

Maximum Stack Usage for Entry 'run': 96

run: [-8..32]

g: [-8..32]

[REC]

Toggle fold

Copy AIS annotation

Show message Shift+M

Create analyses...

Show analysis statistics (context)

Unfold

Unfold recursively Shift+B

Exclusive subgraph Ctrl+B

Scale to fit selection Z

Copy

Copy address

Copy name

Go to caller C

Go to target h T

Go to neighbor

Select area around nodes Ctrl+Shift+A

Recursion bounds

Incarnation limit

Enter with

Intersible

Not analyzed

Overall analysis time: <1s

StackAnalyzer - StackAnalyzer (0 Errors, 4 Warnings): Finished on 2020-06-15 at 17:10:14.8

Control-Flow & Stack Analyzer (0 Errors, 4 Warnings): Finished on 2020-06-15 at 17:10:14.8

Reading binary stackerd...

#1039: ELF file is not an executable, but shared object file.

#1033: ELF file is not a statically linked executable, but contains relocations.

#1034: ELF file is not dynamically linked executable, but contains dynamic link informat

Using optimizer for stack analysis under "GCC".

Recursion counter: 0, 42 analysis nodes.

Recursion counter: 0, 11231 found, recursion memory: 0.

Loop analysis found 0 loop bounds.

The analyzer optimized the stack graph of entry 'run' from 5/5 to 2/2 nodes * calls (non-optimized)

#2097: For routine 'h' the default incarnation limit of 1 is used.

The analyzer optimized the stack graph of entry 'run' from 5/5 to 4/4 nodes * calls (non-optimized)

Maximum global stack height: 96

Last process took: 0.4 and used not more than 30 MB (85513 MB) of memory

Reporting

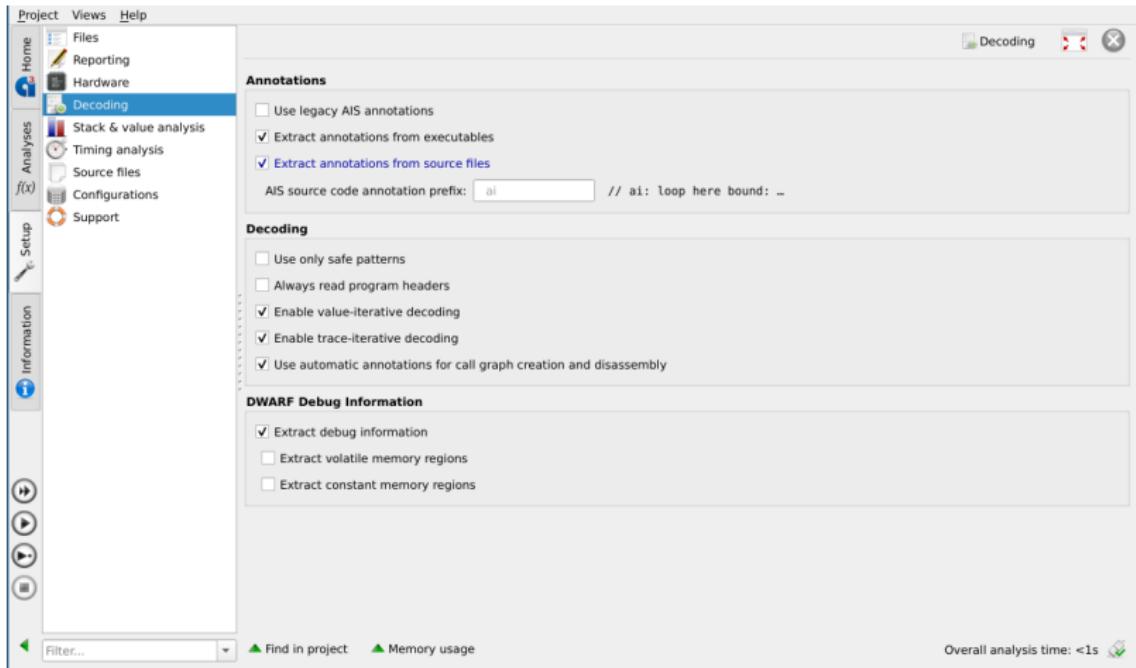
Creating HTML report

Finished on 2020-06-15 at 17:10:14 after analyzing for 1 second with 0 errors, 4 warnings

Ais-Notationen

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

a³ Analyzer – Kommentar-Parsing Aktivieren



- 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 Analyse mittels a³ Stack-Analyzer

