

Verlässliche Echtzeitsysteme - Übungen

Analyse

Wintersemester 2024

Eva Dengler, Peter Wägemann

Friedrich-Alexander-Universität Erlangen-Nürnberg

Lehrstuhl Informatik 4 (Systemsoftware)

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

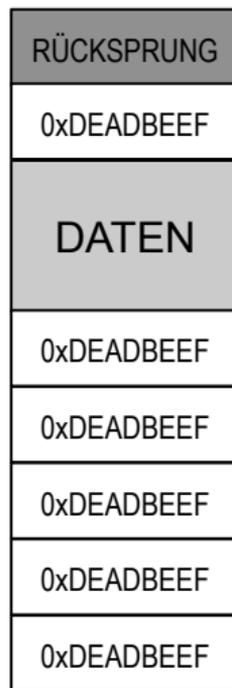
- 1 Stackbedarfsanalyse
- 2 Worst-Case Stack Usage
- 3 AbsInt Stack Analyzer
- 4 Aufgabenstellung



- 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 a³ 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 \leadsto größere Auflösung
- ⚠ Keine allgemeingültigen Aussagen
 - Liefert nur den konkreten Bedarf der Messungen
 - Vorsichtige Aussagen über Worst-Case-Verhalten
- Einsatz zur dynamischen Fehlererkennung





1. (Globalen) Stack anlegen:

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

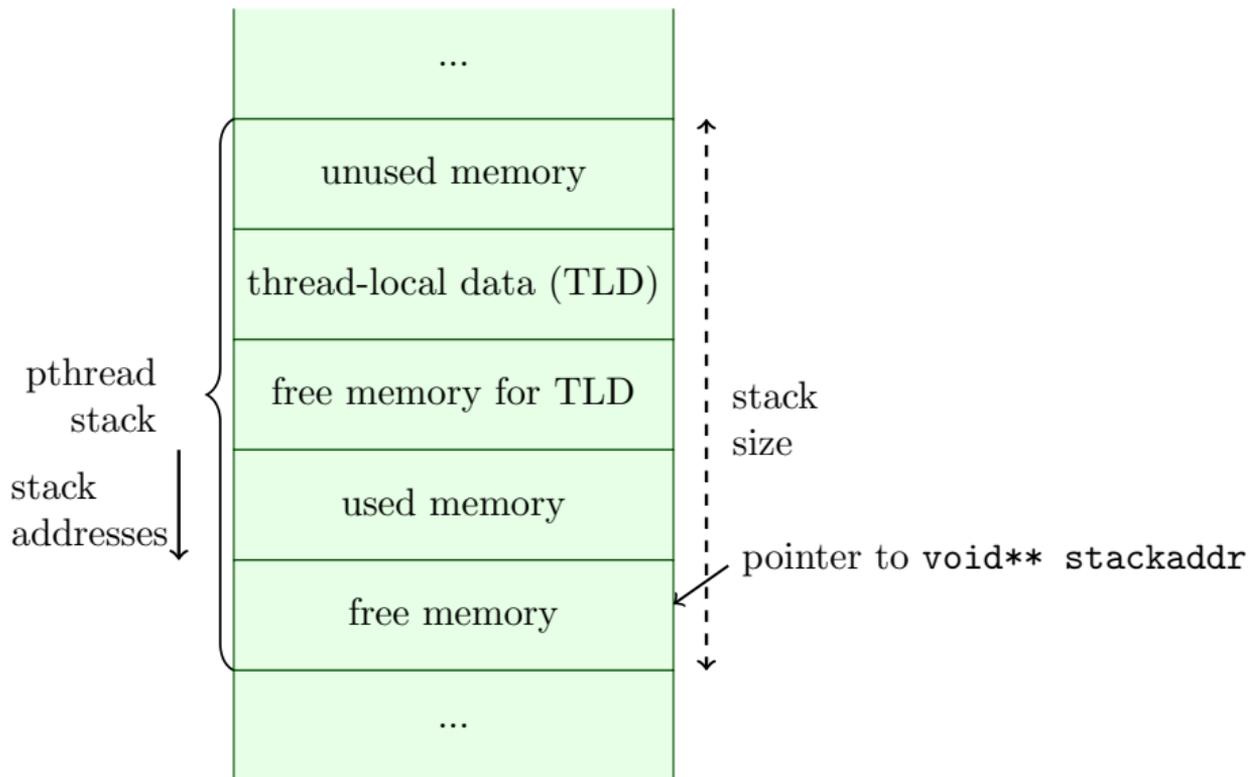
2. Thread anlegen & starten:

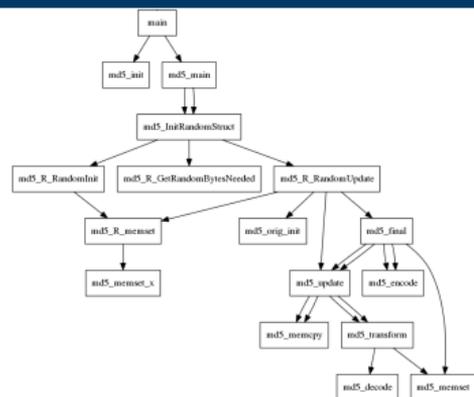
```
pthread_t thread;  
pthread_attr_t attr;  
pthread_attr_init(&attr);  
pthread_attr_setstack(&attr, &g_stack, STACK_SIZE);  
// worker function: void *run(void *param)  
int status = pthread_create(&thread, &attr, run, NULL);  
if (status != 0) { ... // handle error
```

3. Auf Thread warten:

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

pthread Stack





```
/* 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-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 \leadsto **maximaler Stackbedarf**

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

²Integer 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

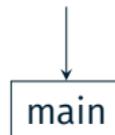
Der initiale Aufruf erfolgt maximal (wahlweise auch genau) ein mal

Formalisierung

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

lp_solve -Syntax

```
n_main <= 1;
```



Flussbedingung: Mehrere Vorgänger

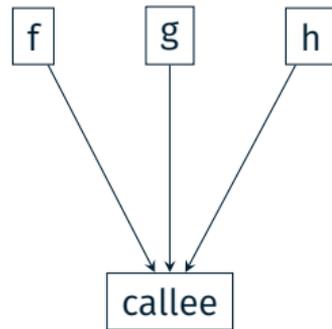
Semantik

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

Formalisierung

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

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



lp_solve -Syntax

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

Flussbedingung: Immer nur ein Nachfolger pro Funktion

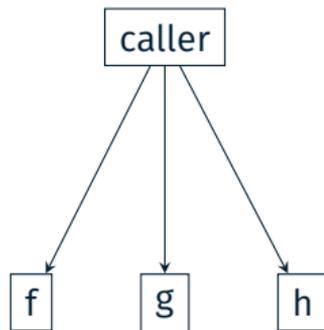
Semantik

Jede Funktionsinkarnation ruft gleichzeitig jeweils maximal eine weitere Funktion auf

Formalisierung

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

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



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.

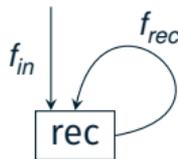
Formalisierung

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

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

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;
```

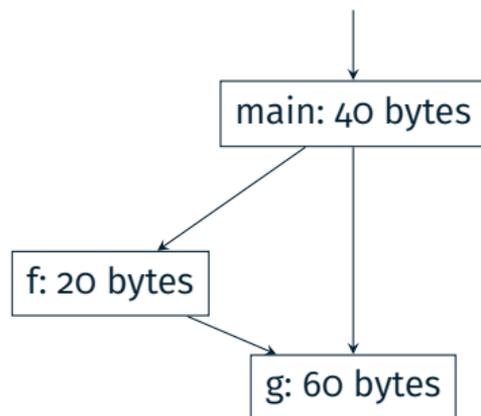
```
n_main <= 1;  
+f_main_f +f_main_g <= n_main;  
n_f <= +f_main_f;  
+f_f_g <= n_f;  
n_g <= +f_f_g +f_main_g;
```

■ 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

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

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` 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;
```

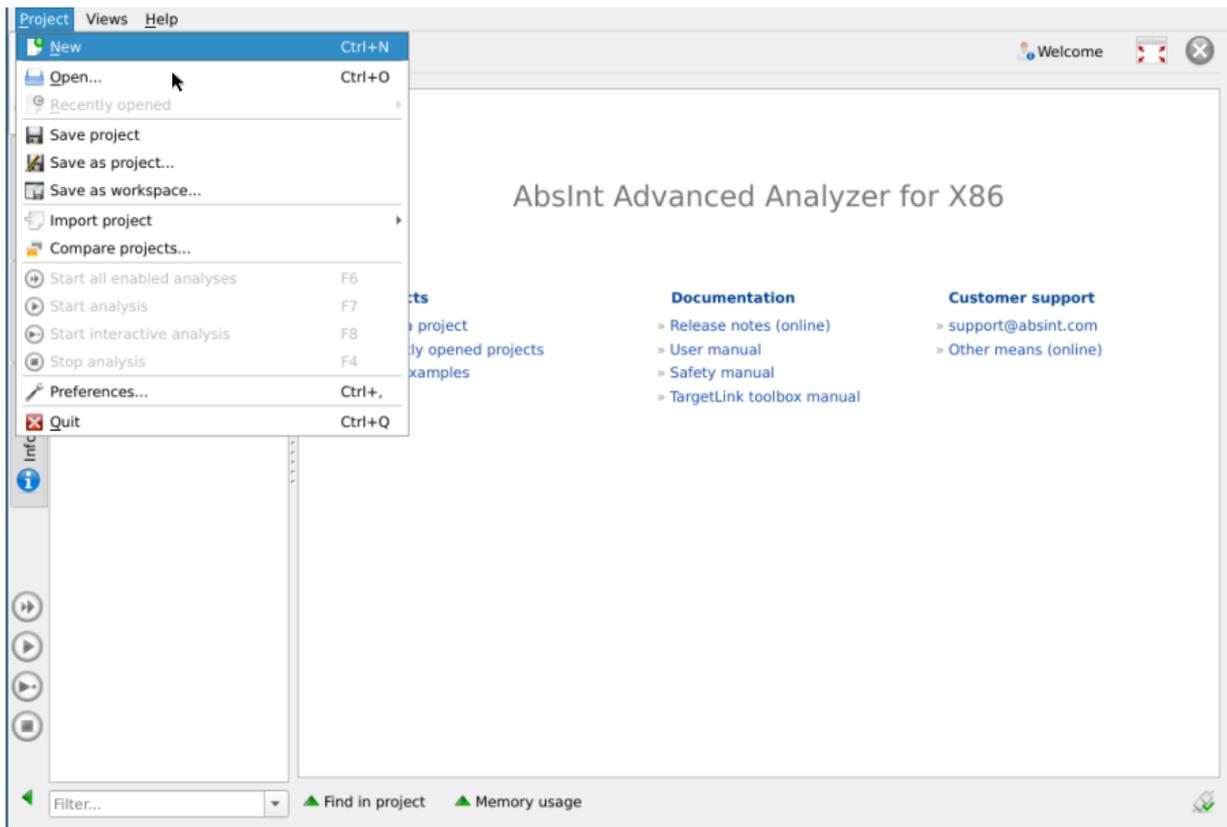
- Statische Code-Analyse mit a³ Tool-Suite
 1. aiT: WCET-Analyse
 2. Stack-Analyzer: Stackbedarf
 3. ...
- Installiert im CIP-Pool
- `/proj/i4ezs/tools/a3_x86/bin/a3x86`

a³ Analyzer – Lizenzserver

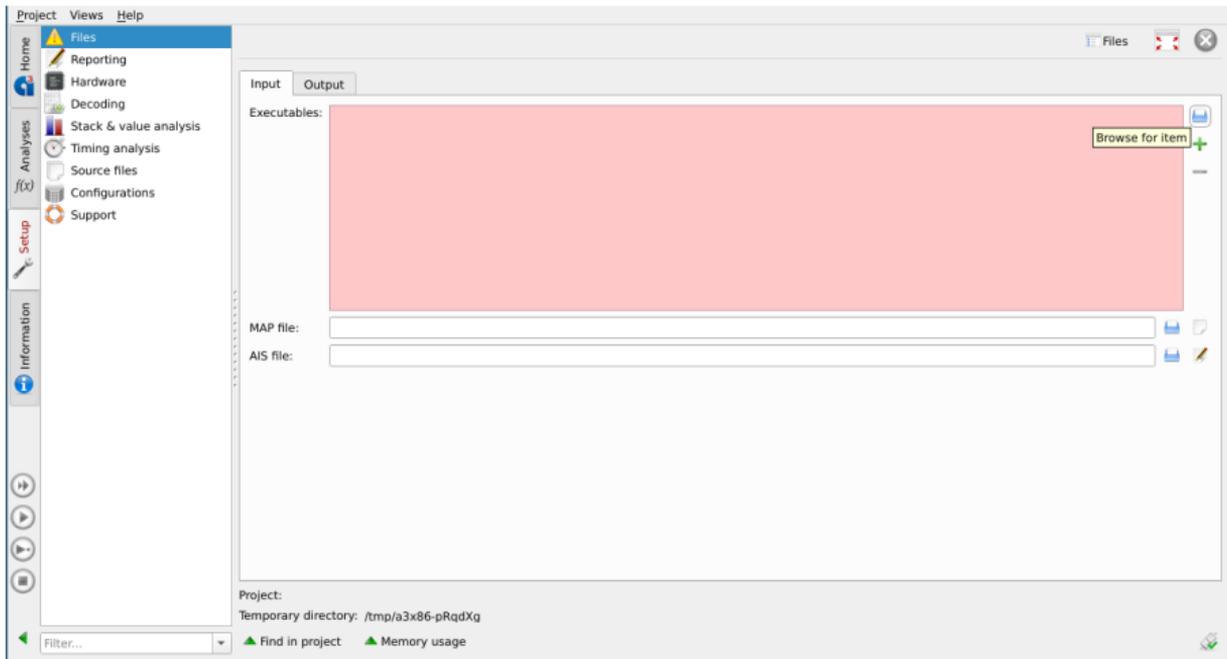
The screenshot displays the a3 Analyzer software interface. The main window has a menu bar with 'Project', 'Views', and 'Help'. The left sidebar contains 'Home' (with 'Overview', 'Welcome', and 'Recent projects' sub-items), 'Analyses' (with a 'f(x)' icon), 'Setup' (with a wrench icon), and 'Information' (with an 'i' icon). The main area shows a 'Select AbsInt License Manager' dialog box. This dialog has a 'License' section with a table listing license details and a 'Manage' section with input fields for 'Host' and 'Port'. The 'Host' field contains 'i4alm.cs.fau.de' and the 'Port' field contains '42424'. A semi-transparent blue box is overlaid on the dialog, containing the text 'Zugangsdaten Benutzer/Passwort aus initialer Mail'. A 'Close' button is visible in the bottom right of the dialog. The background window shows a 'Welcome' message and a 'Server support' link.

Zugangsdaten
Benutzer/Passwort aus initialer Mail

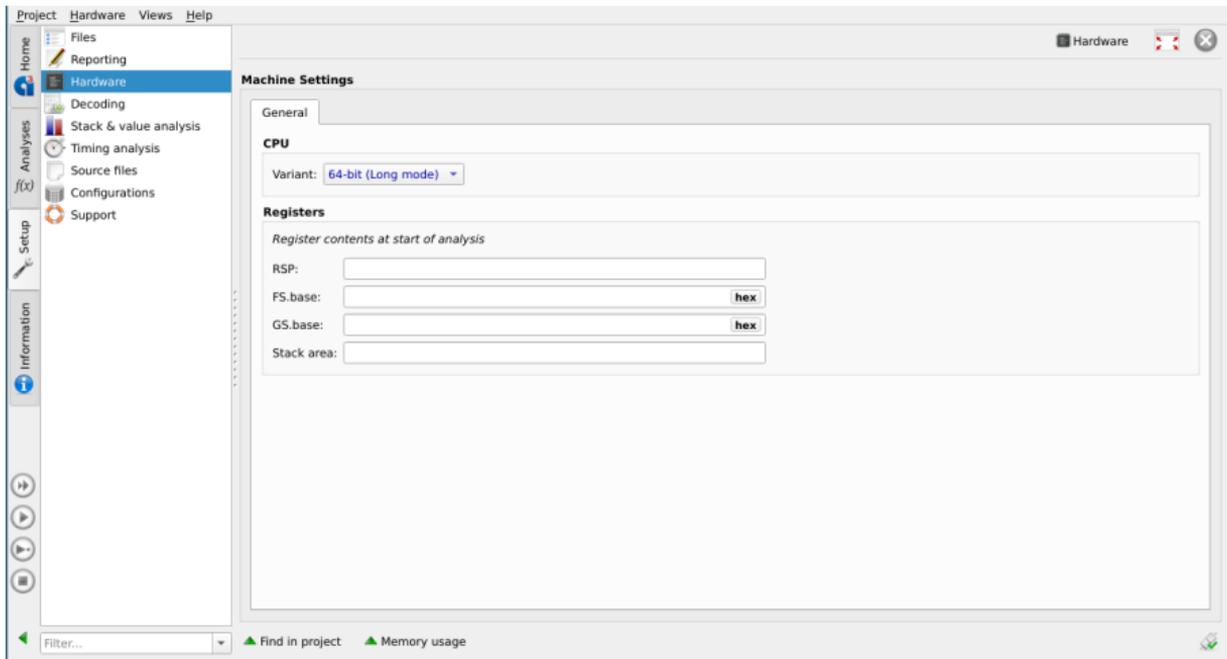
a³ Analyzer – Neues Projekt Anlegen



a³ Analyzer – Executable Angeben



a³ Analyzer – Hardware Auswählen



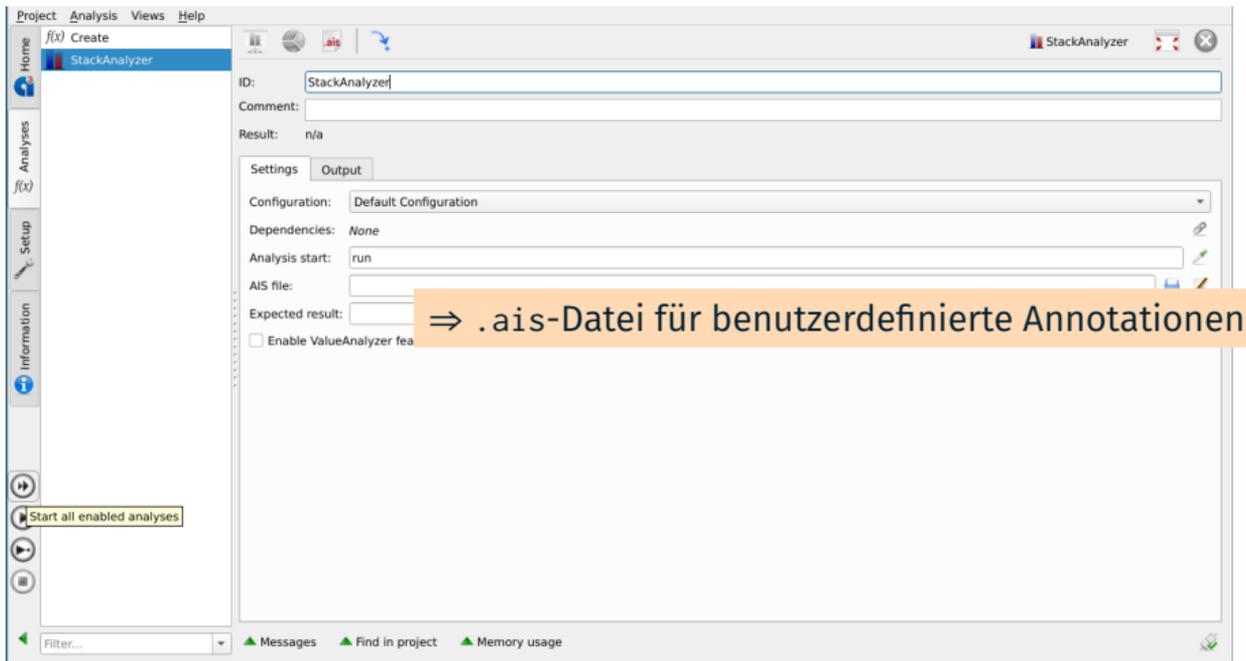
a³ Analyzer – Stack-Analyse Selektieren

The screenshot displays the a3 Analyzer application window. The title bar shows 'Project Views Help' and the current project is 'f(x) Create'. The left sidebar contains navigation options: Home, Analyses (selected), Setup, and Information. The main workspace shows a list of analysis tools under the 'f(x)' project:

- StackAnalyzer**: Stack usage analysis (represented by a bar chart icon).
- ValueAnalyzer**: Program value analysis (represented by a binary code icon).
- ResultCombinator**: Combination of results according to formula (represented by a green chalkboard icon with the equation $Z+Z=4$).
- Control-Flow Visualizer**: Visualization of control-flow graph (represented by a blue flowchart icon).

Below the list, there is a 'Filter...' dropdown menu and two active filters: 'Find in project' and 'Memory usage'. A help message at the top of the workspace reads: "You can also use the **Symbols** or **DWARF** view to create multiple analyses of the same type by selecting the analysis entries and using the **Create analyses** action from the toolbar or context menu."

a³ Analyzer – Stack-Analyse Starten



a³ Analyzer – Analyseoutput

The screenshot shows the StackAnalyzer application interface. The top menu bar includes Project, Analysis, Views, and Help. The left sidebar has buttons for Home, Analysis, Setup, and Information. The main window displays the 'StackAnalyzer' configuration and results. The 'Settings' tab is active, showing configuration options like 'Default Configuration', 'None' dependencies, and 'run' analysis start. The 'Output' tab shows the analysis results, including a summary: 'StackAnalyzer - StackAnalyzer (0 Errors, 4 Warnings): Finished on 2020-06-15 at 17:10:14 after analyzing for 1 second'. A warning is expanded under 'Control-Flow & Stack Analysis', showing messages like '#1033: ELF file is not an executable, but shared object file.' and '#1034: ELF file is not a statically linked executable, but contains dyn'. A pink callout box with an arrow points to these messages with the text '⇒ Warnung zu ELF ignorieren'. Other messages include '#1097: For routine 'h' the default incarnation limit of 1 is used.' and a 'Reporting' section indicating 'Creating HTML report' and 'Finished on 2020-06-15 at 17:10:14 after analyzing for 1 second with 0 errors, 4 warnings'. The bottom status bar shows 'Overall analysis time: <1s'.

StackAnalyzer

ID: StackAnalyzer

Comment:

Result: System: 96 bytes

Settings Output

Configuration: Default Configuration

Dependencies: None

Analysis start: run

AIS file:

Expected result:

Enable ValueAnalyzer features

Errors, warnings and info Latest log

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 'stacktest'.
 - #1033: ELF file is not an executable, but shared object file.
 - #1033: ELF file is not a statically linked executable, but contains dyn
 - #1034: ELF file is not a statically linked executable, but contains dyn
- Using decoder for 'x86_64' and compiler 'GCC'.
 - Recursion 0x1125 'h' found, recursion members:
 - Value analyzer statistics (max-length=2, default-unroll=2, normal mode):
 - Loop analysis found 0 loop bounds.
 - The analyzer optimized the stack graph of entry 'run' from 5/5 to 2/2 nodes * calls (non-optimizable routines: 2).
- #1097: 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-optimizable routines: 2).
 - Maximum global stack height: 96
 - Last process took 0 s and used not more than 30 MB (RSS 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

⇒ Warnung zu ELF ignorieren

Filter... Messages Find in project Memory usage Overall analysis time: <1s

a³ Analyzer – Callgraph

The screenshot displays the a3 Analyzer interface for the StackAnalyzer project. The top menu bar includes Project, Analysis, Views, and Help. The left sidebar shows navigation options: Home, Analysis, Setup, and Information. The main window is titled 'StackAnalyzer' and contains the following sections:

- Settings:** Configuration is set to 'Default Configuration'. Dependencies are 'None'. Analysis start is 'run'. AIS file is empty. Expected result is empty. There is an unchecked checkbox for 'Enable ValueAnalyzer features'.
- Output Log:** Shows a summary: 'StackAnalyzer - StackAnalyzer (0 Errors, 4 Warnings): Finished on 2020-06-15 at 17:10:14 after analyzing for 1 second'. Below this, a 'Control-Flow & Stack Analysis' section lists warnings: '#1030: ELF file is not an executable, but shared object file.', '#1033: ELF file is not a statically linked executable, but contains relocations.', and '#1034: ELF file is not a statically linked executable, but contains dynamic link information.' It also notes 'Using decoder for 'x86_64' and compiler 'GCC'', 'Recursion 0x1125 'r' found, recursion members:', and 'Value analyzer statistics (max-length=2, default-unroll=2, normal mode):'. A 'Reporting' section indicates 'Finished on 2020-06-15 at 17:10:14 after'.

A context menu is open over the log, with 'Show in call graph' selected. Other menu items include Copy, Copy part, Show in disassembly, Show in file, Copy path, Copy AIS annotation, Find 'limit' in DWARF, 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.

At the bottom right, the overall analysis time is shown as '<1s'.

a³ Analyzer – Annotationstemplate kopieren

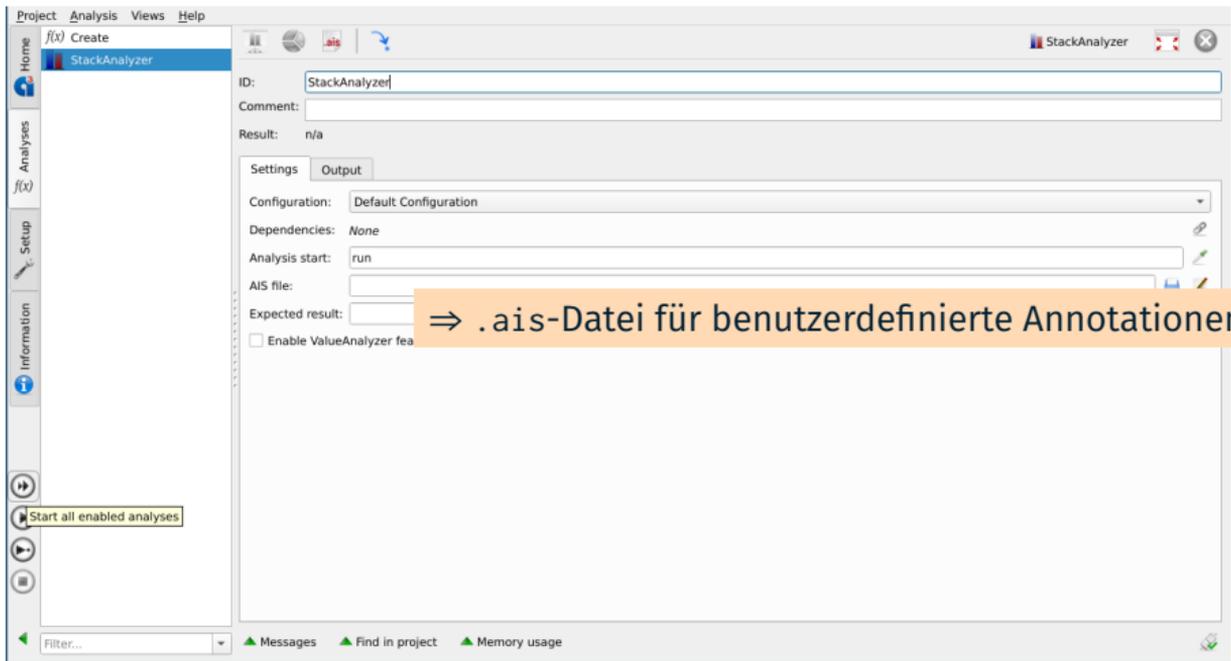
The screenshot displays the a3 Analyzer interface. At the top, a menu bar includes 'Project', 'Analysis', 'Graph', 'Views', and 'Help'. Below the menu, there are icons for 'Home', 'StackAnalyzer', 'AIS files', and 'Analysis graph'. The main window shows a stack graph with nodes: 'run: [-8..32]', 'g: [-8..32]', and '[REC]'. A context menu is open over the '[REC]' node, listing actions like 'Toggle fold', 'Show address in disassembly', 'Find address in DWARF', 'Show source', 'Copy AIS annotation', 'Show message', 'Create analyses...', 'Show analysis statistics (context)', 'Unfold', 'Unfold recursively', 'Unfold routines to basic-block level', 'Exclusive subgraph', 'Scale to fit selection', 'Copy', 'Copy address', 'Copy name', 'Go to caller', and 'Go to target h'. A sub-menu for 'Recursion bounds' is also visible, containing 'Incarnation limit', 'Enter with', 'Infeasible', and 'Not analyzed'. The bottom panel shows 'Errors, warnings and info' with a log entry: 'StackAnalyzer - StackAnalyzer (0 Errors, 4 Warnings): Finished on 2020-06-15 at 17:10:14'. The log includes warnings about ELF files and a note about the recursion limit. A semi-transparent text box in the foreground contains the following text:

Ais-Notationen

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

At the bottom of the interface, there is a 'Filter...' field, 'Messages', 'Find in project', and 'Memory usage' buttons. The overall analysis time is shown as '<1s'.

a³ Analyzer – Stack-Analyse Starten



a³ Analyzer – Annotationstemplate kopieren

The screenshot displays the a3 Analyzer interface. At the top, the menu bar includes Project, Analysis, Graph, Views, and Help. The main window shows a stack graph with nodes for 'run: [-8..32]', 'g: [-8..32]', and '[REC]'. A context menu is open over the '[REC]' node, listing actions such as 'Toggle fold', 'Show address in disassembly', 'Find address in DWARF', 'Show source', 'Copy AIS annotation', 'Show message', 'Create analyses...', 'Show analysis statistics (context)', 'Unfold', 'Unfold recursively', 'Unfold routines to basic-block level', 'Exclusive subgraph', 'Scale to fit selection', 'Copy', 'Copy address', 'Copy name', 'Go to caller', and 'Go to target h'. The bottom panel shows the 'Errors, warnings and info' section with a log of messages, including warnings about ELF files and stack optimization. A semi-transparent text box in the foreground contains the following text:

Ais-Notationen

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

Overall analysis time: <1s

a³ Analyzer – Kommentar-Parsing Aktivieren

The screenshot displays the a3 Analyzer software interface. The left sidebar contains a navigation menu with categories: Home (Files, Reporting, Hardware, Decoding), Analyses (Stack & value analysis, Timing analysis, Source files, Configurations), Setup, and Information. The main window is titled 'Decoding' and contains three sections of settings:

- Annotations:**
 - Use legacy AIS annotations
 - Extract annotations from executables
 - Extract annotations from source files

AIS source code annotation prefix: // ai: loop here bound: _
- Decoding:**
 - Use only safe patterns
 - Always read program headers
 - Enable value-iterative decoding
 - Enable trace-iterative decoding
 - Use automatic annotations for call graph creation and disassembly
- DWARF Debug Information:**
 - Extract debug information
 - Extract volatile memory regions
 - Extract constant memory regions

At the bottom of the window, there is a 'Filter...' dropdown, 'Find in project' and 'Memory usage' buttons, and a status bar indicating 'Overall analysis time: <1s'.

- 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 a³ Stack-Analyzer
 3. Optional: Zeitanalyse mit aiT