

# Verlässliche Echtzeitsysteme

## Übungen zur Vorlesung

### Statische Stackbedarfsanalyse

Phillip Raffeck, Tim Rheinfels, Simon Schuster, Peter Wägemann

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

Wintersemester 2022





```

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-Summe<sup>1</sup>

■ Vorgehen

1. Callgraph bestimmen
2. Stackbedarf einzelner Funktionen (gcc -fstack-usage)
3. ILP<sup>2</sup> aufstellen (Nebenbedingungen aus 1., Kosten aus 2. verwenden)
4. ILP z.B. mittels lp\_solve  $\leadsto$  **maximaler Stackbedarf**

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

<sup>2</sup>Integer Linear Program (dt. ganzzahliges lineares Programm)

## Optimierungsziel

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

## Optimierungsziel

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

In `lp_solve` -Syntax:

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



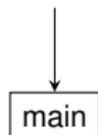
# Flussbedingung: Initialer Aufruf

## Semantik

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

## Formalisierung

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



## lp\_solve -Syntax

```
n_main <= 1;
```



# Flussbedingung: Mehrere Vorgänger

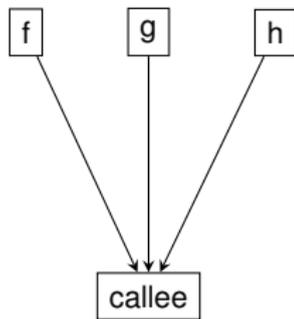
## Semantik

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

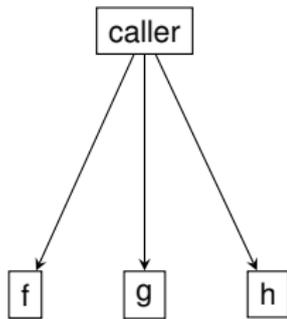
## Semantik

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

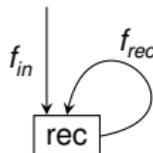
## Semantik

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;
```

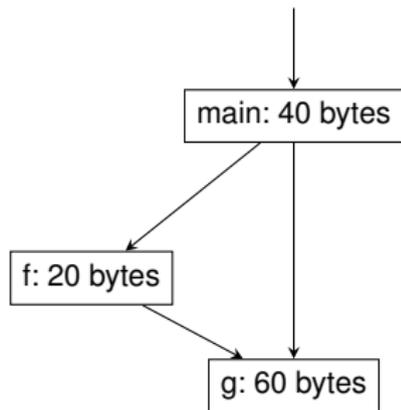
```
n_main <= 1;  
+f_main_f +f_main_g <= n_main;  
n_f <= +f_main_f;  
+f_f_g <= n_f;  
n_g <= +f_f_g +f_main_g;
```

- 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 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:<sup>3</sup>

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

<sup>3</sup><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
$z \leq y + 1$ ;	$z \leq y + 1$ ;	y: 20
$y \leq 20$ ;	$y \leq 20$ ;	z: 21
	$x \leq 5000$ ;	
	$y \leq 5000$ ;	
	$z \leq 5000$ ;	



- `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
```

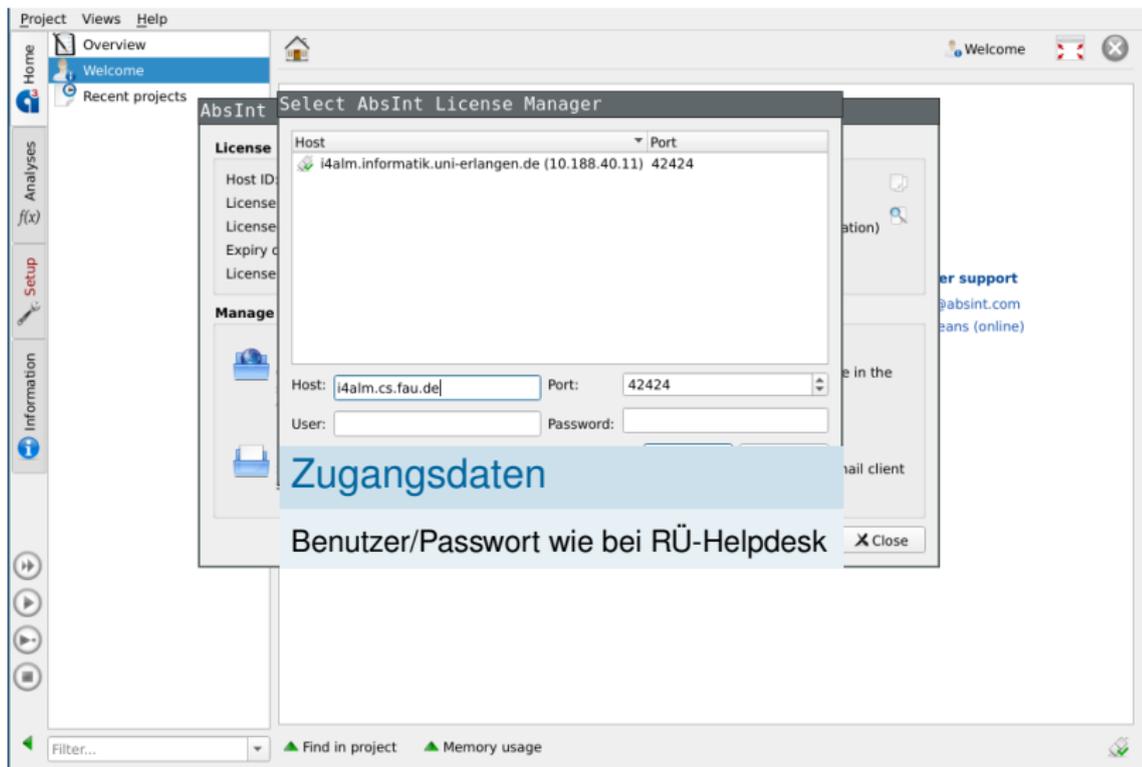
```
max: +s_main n_main +s_f n_f +s_g n_g;
```

~> `stackusage / lp_solvepp`

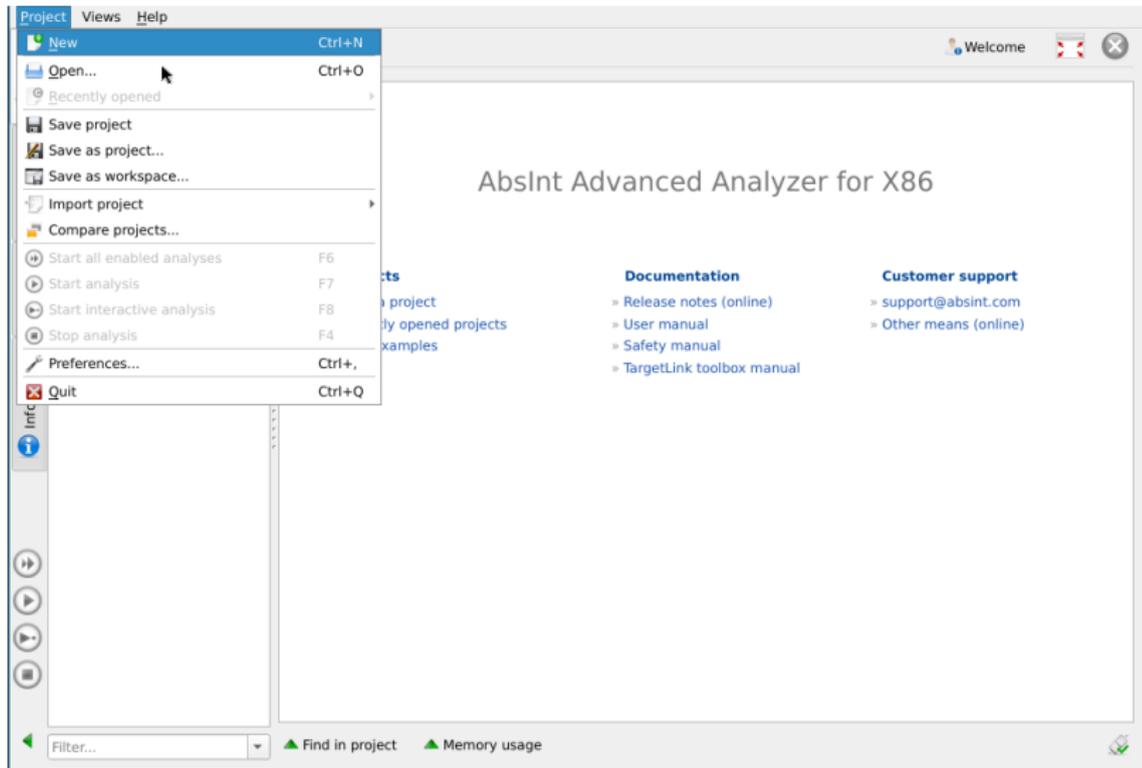


- Statische Code-Analyse mit a<sup>3</sup> Tool-Suite
  1. aiT: WCET-Analyse
  2. Stack-Analyzer: Stackbedarf
  3. ...
- Installiert im CIP-Pool
- `/proj/i4ezs/tools/a3_x86/bin/a3x86`

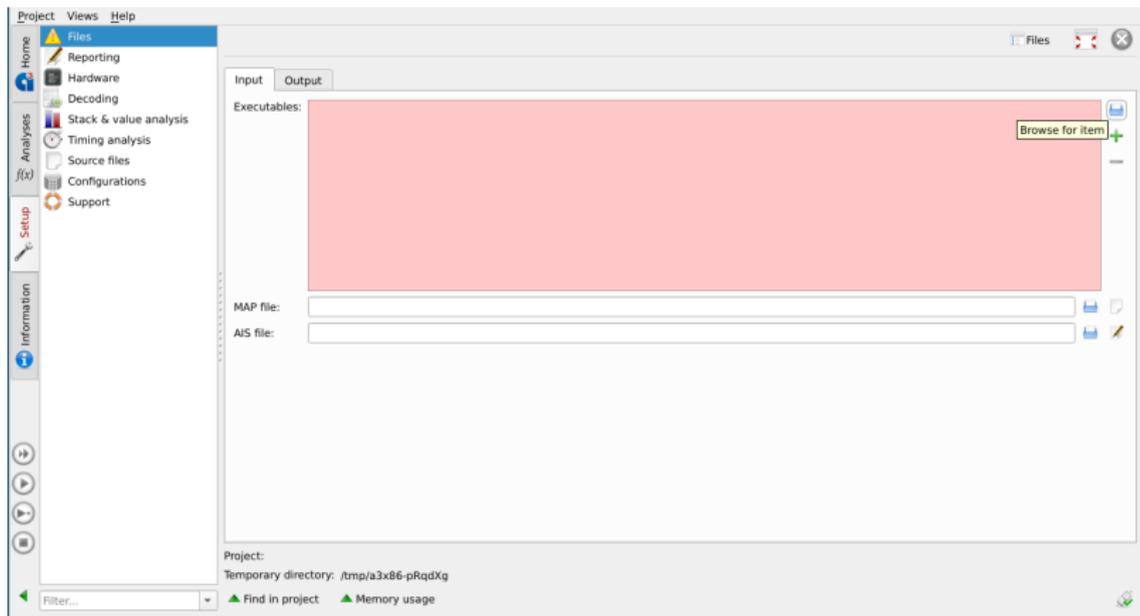




# a<sup>3</sup> Analyzer – Neues Projekt Anlegen



# a<sup>3</sup> Analyzer – Executable Angeben



# a<sup>3</sup> Analyzer – Hardware Auswählen

The screenshot shows the 'Hardware' settings window in the a3 Analyzer. The window has a menu bar with 'Project', 'Hardware', 'Views', and 'Help'. On the left is a sidebar with categories: 'Home' (Files, Reporting, Hardware, Decoding), 'Analyses' (Stack & value analysis, Timing analysis, Source files, Configurations), 'Setup' (Support), and 'Information'. The main area is titled 'Machine Settings' and contains a 'General' tab. Under 'CPU', the 'Variant' is set to '64-bit (Long mode)'. Under 'Registers', there is a section for 'Register contents at start of analysis' with input fields for 'RSP', 'FS.base', 'GS.base', and 'Stack area'. The 'FS.base' and 'GS.base' fields have a 'hex' button next to them. At the bottom of the window, there is a search bar and buttons for 'Find in project' and 'Memory usage'.



# a<sup>3</sup> Analyzer – Stack-Analyse Selektieren

The screenshot shows the a3 Analyzer application window. The title bar reads "f(x) Create". The menu bar includes "Project", "Views", and "Help". On the left, a sidebar contains icons for "Home", "Analysis", "Setup", and "Information". The main area displays a list of analysis options:

- StackAnalyzer**: Stack usage analysis
- ValueAnalyzer**: Program value analysis
- ResultCombinator**: Combination of results according to formula
- Control-Flow Visualizer**: Visualization of control-flow graph

At the bottom of the window, there is a search bar labeled "Filter...", a "Find in project" button, and a "Memory usage" button. A small green checkmark icon is visible in the bottom right corner.



# a<sup>3</sup> Analyzer – Stack-Analyse Starten

The screenshot shows the StackAnalyzer application window. The main area is titled "StackAnalyzer" and contains the following fields and controls:

- ID: StackAnalyzer
- Comment: (empty)
- Result: n/a
- Settings tab selected, Output tab also visible.
- Configuration: Default Configuration
- Dependencies: None
- Analysis start: run
- AIS file: (empty)
- Expected result: (empty)
- Enable ValueAnalyzer feature

An orange callout box points to the "Enable ValueAnalyzer feature" checkbox with the text: **⇒ .ais-Datei für benutzerdefinierte Annotationen**

On the left sidebar, the "Start all enabled analyses" button is highlighted with a yellow box.



# a<sup>3</sup> Analyzer – Analyseoutput

The screenshot shows the StackAnalyzer application window. The main area displays the analysis results for a project named 'StackAnalyzer'. The results are categorized into 'Errors, warnings and info' and 'Latest log'. A yellow highlight is placed over a warning message:

- #1039: For routine 'run' the default incarnation limit of 1 is used.**  
The analyzer optimized the stack graph of entry 'run' from 5/5 to 2/2 nodes.  
Last process took 0 s and used not more than 30 MB (RSS 13 MB) of memory

A pink callout box with an arrow points to this warning with the text: **⇒ Warnung zu ELF ignorieren**

Other visible messages include:

- #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 a statically linked executable, but contains dynamic link information.**
- #1035: ELF file is not a statically linked executable, but contains dynamic link information.**
- #1036: ELF file is not a statically linked executable, but contains dynamic link information.**
- #1037: ELF file is not a statically linked executable, but contains dynamic link information.**
- #1038: ELF file is not a statically linked executable, but contains dynamic link information.**
- #1039: ELF file is not a statically linked executable, but contains dynamic link information.**
- #1040: ELF file is not a statically linked executable, but contains dynamic link information.**
- #1041: ELF file is not a statically linked executable, but contains dynamic link information.**
- #1042: ELF file is not a statically linked executable, but contains dynamic link information.**
- #1043: ELF file is not a statically linked executable, but contains dynamic link information.**
- #1044: ELF file is not a statically linked executable, but contains dynamic link information.**
- #1045: ELF file is not a statically linked executable, but contains dynamic link information.**
- #1046: ELF file is not a statically linked executable, but contains dynamic link information.**
- #1047: ELF file is not a statically linked executable, but contains dynamic link information.**
- #1048: ELF file is not a statically linked executable, but contains dynamic link information.**
- #1049: ELF file is not a statically linked executable, but contains dynamic link information.**
- #1050: ELF file is not a statically linked executable, but contains dynamic link information.**
- #1051: ELF file is not a statically linked executable, but contains dynamic link information.**
- #1052: ELF file is not a statically linked executable, but contains dynamic link information.**
- #1053: ELF file is not a statically linked executable, but contains dynamic link information.**
- #1054: ELF file is not a statically linked executable, but contains dynamic link information.**
- #1055: ELF file is not a statically linked executable, but contains dynamic link information.**
- #1056: ELF file is not a statically linked executable, but contains dynamic link information.**
- #1057: ELF file is not a statically linked executable, but contains dynamic link information.**
- #1058: ELF file is not a statically linked executable, but contains dynamic link information.**
- #1059: ELF file is not a statically linked executable, but contains dynamic link information.**
- #1060: ELF file is not a statically linked executable, but contains dynamic link information.**
- #1061: ELF file is not a statically linked executable, but contains dynamic link information.**
- #1062: ELF file is not a statically linked executable, but contains dynamic link information.**
- #1063: ELF file is not a statically linked executable, but contains dynamic link information.**
- #1064: ELF file is not a statically linked executable, but contains dynamic link information.**
- #1065: ELF file is not a statically linked executable, but contains dynamic link information.**
- #1066: ELF file is not a statically linked executable, but contains dynamic link information.**
- #1067: ELF file is not a statically linked executable, but contains dynamic link information.**
- #1068: ELF file is not a statically linked executable, but contains dynamic link information.**
- #1069: ELF file is not a statically linked executable, but contains dynamic link information.**
- #1070: ELF file is not a statically linked executable, but contains dynamic link information.**
- #1071: ELF file is not a statically linked executable, but contains dynamic link information.**
- #1072: ELF file is not a statically linked executable, but contains dynamic link information.**
- #1073: ELF file is not a statically linked executable, but contains dynamic link information.**
- #1074: ELF file is not a statically linked executable, but contains dynamic link information.**
- #1075: ELF file is not a statically linked executable, but contains dynamic link information.**
- #1076: ELF file is not a statically linked executable, but contains dynamic link information.**
- #1077: ELF file is not a statically linked executable, but contains dynamic link information.**
- #1078: ELF file is not a statically linked executable, but contains dynamic link information.**
- #1079: ELF file is not a statically linked executable, but contains dynamic link information.**
- #1080: ELF file is not a statically linked executable, but contains dynamic link information.**
- #1081: ELF file is not a statically linked executable, but contains dynamic link information.**
- #1082: ELF file is not a statically linked executable, but contains dynamic link information.**
- #1083: ELF file is not a statically linked executable, but contains dynamic link information.**
- #1084: ELF file is not a statically linked executable, but contains dynamic link information.**
- #1085: ELF file is not a statically linked executable, but contains dynamic link information.**
- #1086: ELF file is not a statically linked executable, but contains dynamic link information.**
- #1087: ELF file is not a statically linked executable, but contains dynamic link information.**
- #1088: ELF file is not a statically linked executable, but contains dynamic link information.**
- #1089: ELF file is not a statically linked executable, but contains dynamic link information.**
- #1090: ELF file is not a statically linked executable, but contains dynamic link information.**
- #1091: ELF file is not a statically linked executable, but contains dynamic link information.**
- #1092: ELF file is not a statically linked executable, but contains dynamic link information.**
- #1093: ELF file is not a statically linked executable, but contains dynamic link information.**
- #1094: ELF file is not a statically linked executable, but contains dynamic link information.**
- #1095: ELF file is not a statically linked executable, but contains dynamic link information.**
- #1096: ELF file is not a statically linked executable, but contains dynamic link information.**
- #1097: ELF file is not a statically linked executable, but contains dynamic link information.**
- #1098: ELF file is not a statically linked executable, but contains dynamic link information.**
- #1099: ELF file is not a statically linked executable, but contains dynamic link information.**
- #1100: ELF file is not a statically linked executable, but contains dynamic link information.**

The bottom status bar shows: Overall analysis time: <1s



# a<sup>3</sup> Analyzer – Callgraph

Project Analysis Views Help

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'.
    - #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 a statically linked executable, but contains dynamic link information.
  - 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).
  - 21027 Recursion in the control flow**
    - The analyzer optimized the stack graph of e
    - Maximum global stack height: 96
    - Last process took 0 s and used not more than
  - Reporting
    - Creating HTML report
    - Finished on 2020-06-15 at 17:10:14 after

Context menu options:

- Copy
- Copy part
- Show in call graph
- Show in disassembly
- Show in file
- Copy path
- Copy AIS annotation
- Find 'lim' 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
- Collapse all

Overall analysis time: <1s



# a<sup>3</sup> Analyzer – Annotationstemplate kopieren

The screenshot displays the a3 Analyzer's Analysis graph window. At the top, a green box indicates "Maximum Stack Usage for Entry 'run': 96". Below this, a call graph shows a sequence of nodes: "run: [-B..32]", "g: [-B..32]", and "[REC]". A context menu is open over the "[REC]" node, listing various actions such as "Toggle fold", "Show address in disassembly", "Copy AIS annotation", and "Recursion bounds". The "Recursion bounds" sub-menu is also visible, showing options like "Incarnation limit", "Enter with", "Infeasible", and "Not analyzed".

Errors, warnings and info | Latest log

**StackAnalyzer - StackAnalyzer (0 Errors, 4 Warnings): Finished on 2020-06-15 at 17:10:14**

- Control-Flow & Stack Analysis
  - Reading binary 'stacktest'.
    - #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 a statically linked executable, but contains dynamic link information 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-optimized)
  - #1077: For routine "h" the default incarnation limit of 1 is used.
    - The analyzer optimized the stack graph of entry 'run' from 3/5 to 4/4 nodes \* calls (non-optimized)
    - 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

Overall analysis time: <1s

## Ais-Notationen

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



# a<sup>3</sup> Analyzer – Stack-Analyse Starten

The screenshot shows the StackAnalyzer application window. The interface includes a menu bar (Project, Analysis, Views, Help), a toolbar, and a sidebar with navigation options: Home, Analysis, Setup, and Information. The main area displays the configuration for a stack analysis named "StackAnalyzer".

Configuration details:

- ID: StackAnalyzer
- Comment: (empty)
- Result: n/a
- Settings: Output
- Configuration: Default Configuration
- Dependencies: None
- Analysis start: run
- AIS file: (empty)
- Expected result: (empty)
- Enable ValueAnalyzer feature

An orange callout box highlights the text: **⇒ .ais-Datei für benutzerdefinierte Annotationen**, pointing to the "AIS file" field.

At the bottom left of the sidebar, a button labeled "Start all enabled analyses" is highlighted with a yellow box.

At the bottom of the window, there is a "Filter..." dropdown and status indicators for Messages, Find in project, and Memory usage.



# a<sup>3</sup> Analyzer – Annotationstemplate kopieren

The screenshot displays the a3 Analyzer's control-flow graph for entry 'run'. The graph shows a sequence of nodes: 'run: [-B..32]', 'g: [-B..32]', and '[REC]'. A context menu is open over the '[REC]' node, listing various actions such as 'Toggle fold', 'Show address in disassembly', and 'Copy AIS annotation'. The 'Copy AIS annotation' option is highlighted, and a sub-menu for 'Recursion bounds' is visible, showing options like 'Incarnation limit', 'Enter with', 'Infeasible', and 'Not analyzed'. Below the graph, the console shows the following output:

```
StackAnalyzer - StackAnalyzer (0 Errors, 4 Warnings): Finished on 2020-06-15 at 17:10:14
  Control-Flow & Stack Analysis
    Reading binary 'stacktest'.
    #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 a statically linked executable, but contains dynamic link information
    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-optimized)
    #1077: 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 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
```

Overall analysis time: <1s

## Ais-Notationen

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



# a<sup>3</sup> Analyzer – Kommentar-Parsing Aktivieren

The screenshot shows the a<sup>3</sup> Analyzer interface with the 'Decoding' settings panel open. The left sidebar contains navigation options: Home, Analyses (with 'Decoding' selected), Setup, and Information. The main panel is titled 'Decoding' and contains three sections: 'Annotations', 'Decoding', and 'DWARF Debug Information'. In the 'Annotations' section, 'Extract annotations from source files' is checked, and the 'AIS source code annotation prefix' is set to 'ai'. The 'Decoding' section has 'Enable value-iterative decoding', 'Enable trace-iterative decoding', and 'Use automatic annotations for call graph creation and disassembly' checked. The 'DWARF Debug Information' section has 'Extract debug information' checked. At the bottom, there are status indicators for 'Find in project' and 'Memory usage', and a note that the 'Overall analysis time' is less than 1s.

Project Views Help

Home  
Files  
Reporting  
Hardware  
Decoding  
Analyses  
Stack & value analysis  
Timing analysis  
Source files  
Configurations  
Support

Setup  
Information

Filter...

Find in project Memory usage

Overall analysis time: <1s

### 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



- 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

