

Verlässliche Echtzeitsysteme

Übungen zur Vorlesung

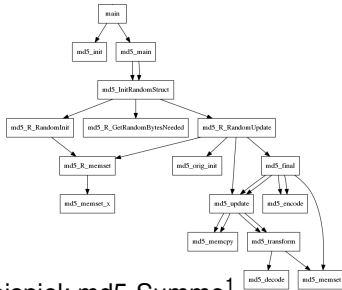
Statische Stackbedarfsanalyse

Phillip Raffeck, Florian Schmaus, Simon Schuster

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

Sommersemester 2020





```

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 \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

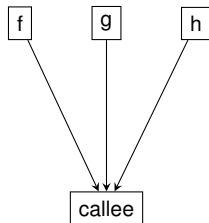
Semantik

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



Flussbedingung: Immer nur ein Nachfolger pro Funktion

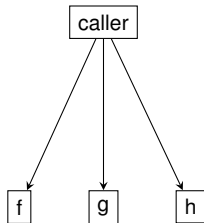
Semantik

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 ;
```



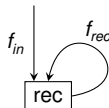
Semantik

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 ;
```



■ Problemformulierung in Ipsolve:

max: +40 n_main +20 n_f +60 n_g;

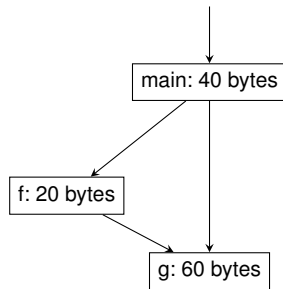
n_main <= 1;

+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;



■ Problemformulierung in lpsolve:

max: +40 n_main +20 n_f +60 n_g;

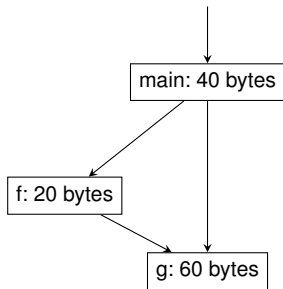
```
n_main <= 1;  
+f_main_f +f_main_g <= n_main;  
n_f <= +f_main_f;  
+f_f_g <= n_f;  
n_g <= +f_f_g +f_main_g;
```

■ 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: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 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
```

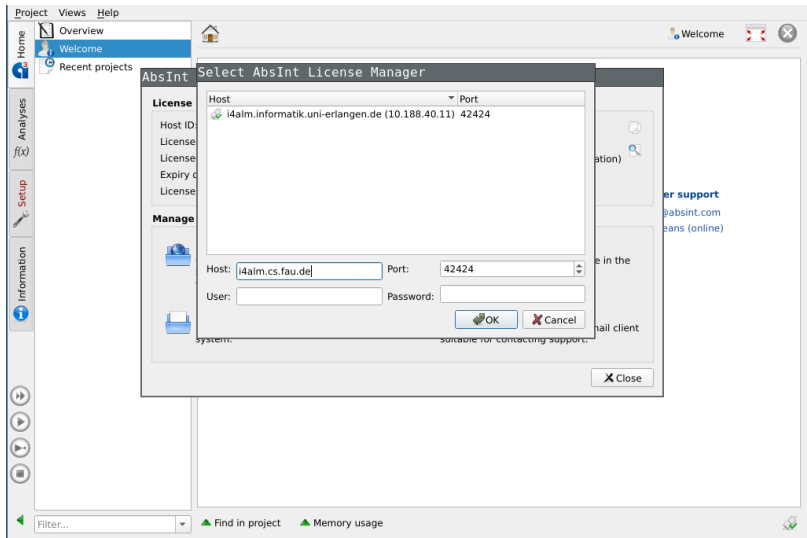
```
max: +s_main n_main +s_f n_f +s_g n_g;
```

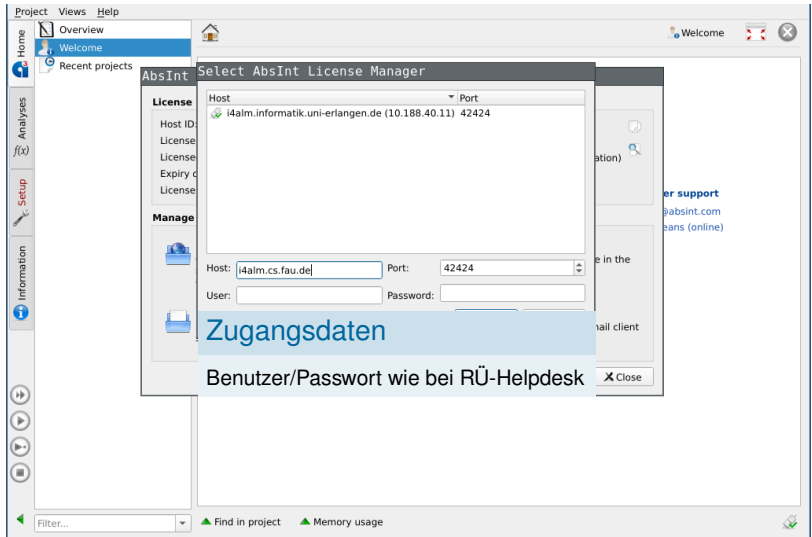
→ `stackusage / lp_solvepp`



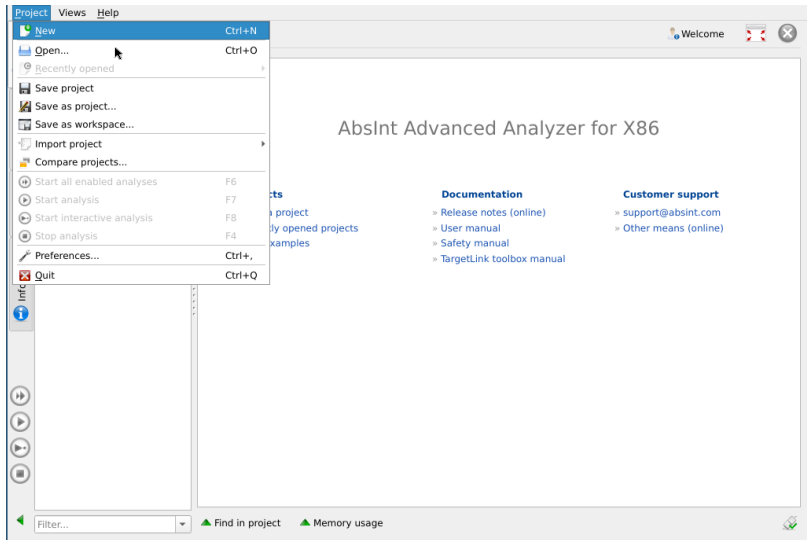
- Statische Code-Analyse mit a³ Tool-Suite
 1. aiT: WCET-Analyse
 2. Stack-Analyzer: Stackbedarf
 3. ...
- Installiert im CIP-Pool
- `/proj/i4ezs/tools/a3_x86/bin/a3x86`



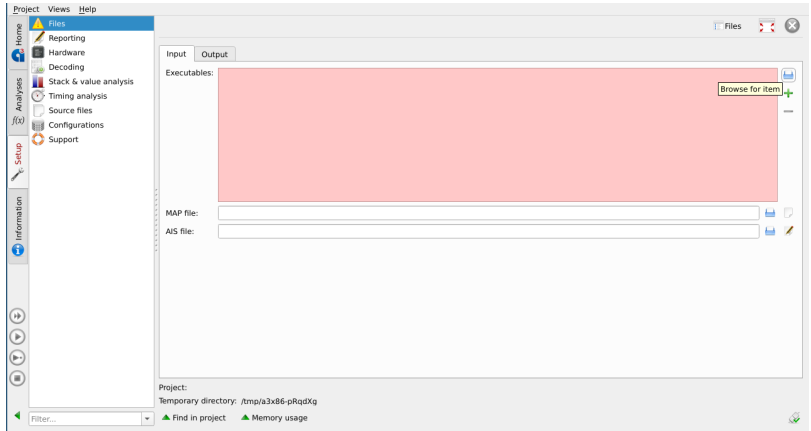




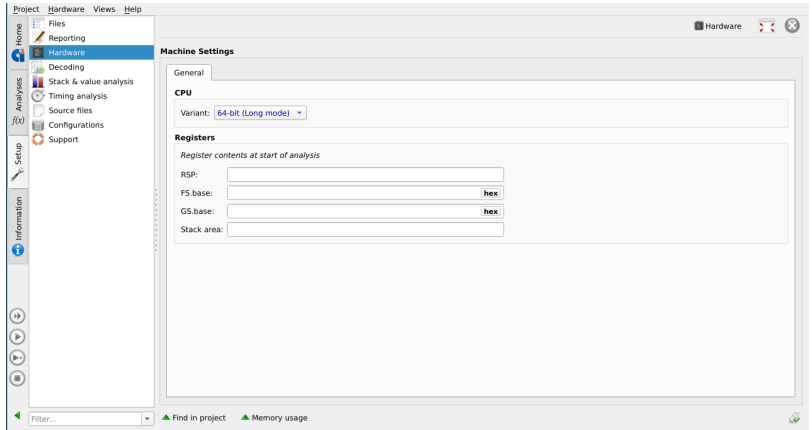
a³ Analyzer – Neues Projekt Anlegen



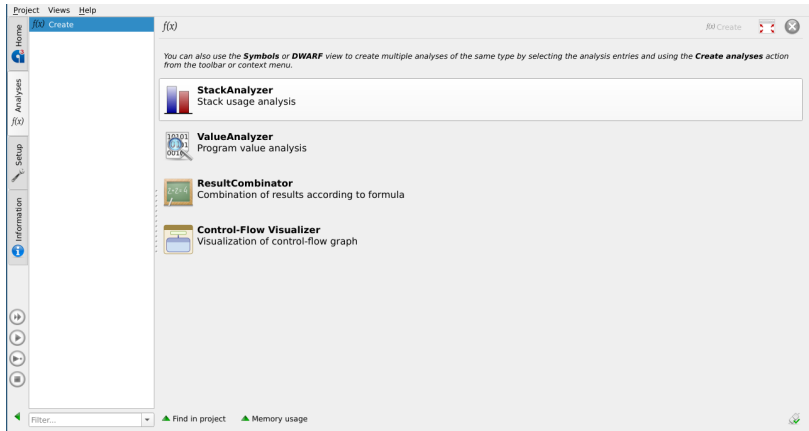
a³ Analyzer – Executable Angeben



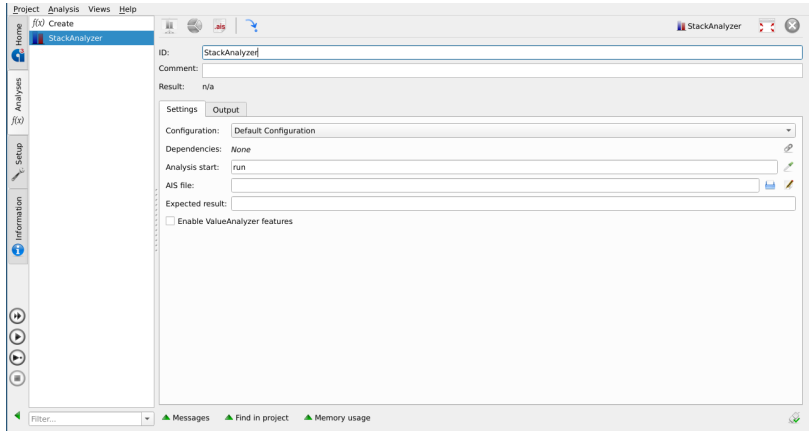
a³ Analyzer – Hardware Auswählen



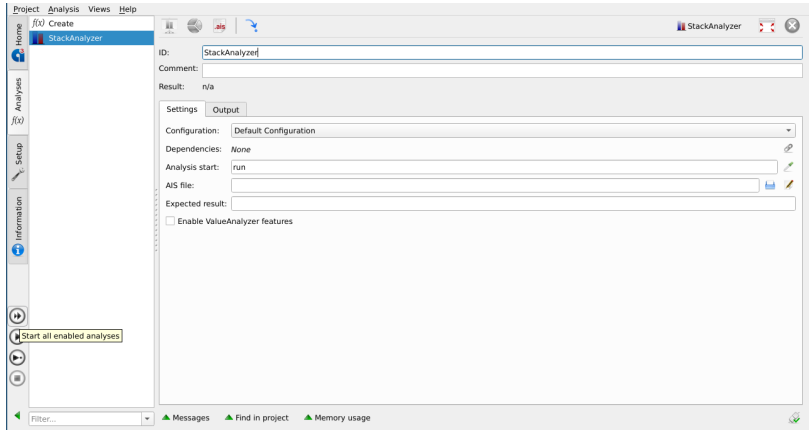
a³ Analyzer – Stack-Analyse Selektieren



a³ Analyzer – Funktion Auswählen



a³ Analyzer – Stack-Analyse Starten



a³ Analyzer – Analyseoutput

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

- ID:** StackAnalyzer
- Comment:**
- Result:** System: 96 bytes
- Settings:** Configuration: Default Configuration; Dependencies: None; Analysis start: run; AIS file: ; Expected result: ; ☐ Enable ValueAnalyzer features

The 'Errors, warnings and info' section shows the following messages:

- Control-Flow & Stack Analysis:** Finished on 2020-06-15 at 17:10:14 after analyzing for 1 second
 - 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.
 - #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 2/2 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

The bottom status bar shows 'Overall analysis time: <1s' and navigation buttons for 'Filter...', 'Messages', 'Find in project', and 'Memory usage'.

a³ Analyzer – Analyseoutput

The screenshot shows the a3 Analyzer interface. The sidebar on the left has tabs for 'Home', 'Analysis', 'Setup', and 'Information'. The 'Analysis' tab is active, showing the 'StackAnalyzer' configuration. The configuration includes fields for ID, Comment, Result, Settings, and Output. The 'Output' tab is selected, showing a list of errors and warnings. A pink callout box highlights a warning about ELF files.

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.
 - #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 2/2 nodes.
- 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

The screenshot displays the StackAnalyzer application window. The top menu bar includes 'Project', 'Analysis', 'Views', and 'Help'. The left sidebar contains icons for 'Home', 'Analysis', 'f3d', 'Setup', and 'Information'. The main window is titled 'StackAnalyzer' and shows a project named 'StackAnalyzer' with a comment field and a result of 'System: 96 bytes'. The 'Settings' tab is active, showing configuration options like 'Configuration: Default Configuration', 'Dependencies: None', 'Analysis start: run', and 'AIS file:'. A 'Report' section is expanded, showing a 'Control-Flow & Stack Analysis' report. The report includes a summary of errors and warnings, a list of messages, and a detailed analysis of the stack graph. A context menu is open over the report, offering actions such as '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', and 'Collapse all'. The bottom status bar indicates 'Overall analysis time: <1s'.

Project Analysis Views Help

f3d Create

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).

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

The analyzer 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 s and used not more than 0 bytes of memory.

Reporting

Creating HTML report

Finished on 2020-06-15 at 17:10:14 after

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

Filter...

Messages Find in project Memory usage

Overall analysis time: <1s



22-24

a³ Analyzer – Annotationstemplate kopieren

The screenshot displays the a3 Analyzer's main window. At the top, a title bar reads 'Project Analysis Graph Views Help'. Below it, a menu bar includes 'File', 'Edit', 'View', 'Tools', 'Window', and 'Help'. A toolbar contains various icons for file operations and analysis. The main area is divided into two panes. The left pane, titled 'Project', shows a tree view with 'Create', 'StackAnalyzer', 'AIS files', and 'Analysis graph'. The right pane, titled 'Analysis graph', shows a stack graph for entry 'run'. The graph has three 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', 'Go to target h', 'Go to neighbor', and 'Select area around nodes'. The bottom pane shows a log of errors and warnings, including messages about ELF files and recursion bounds. A status bar at the bottom right indicates 'Overall analysis time: <1s'.

Maximum Stack Usage for Entry 'run': 96

run: [-8..32]

g: [-8..32]

[REC]

Toggle fold

Show address in disassembly D

Find address in DWARF Shift+D

Show source O

Copy AIS annotation

Show message Shift+M

Create analyses...

Show analysis statistics (context)

Unfold B

Unfold recursively Shift+B

Unfold routines to basic-block level Ctrl+B

Exclusive subgraph X

Scale to fit selection Z

Copy Ctrl+C

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

Incursion limit

Enter with

Infeasible

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-optimizable)

#1037: Recursion in the default incursion limit of 4 is used.

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

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

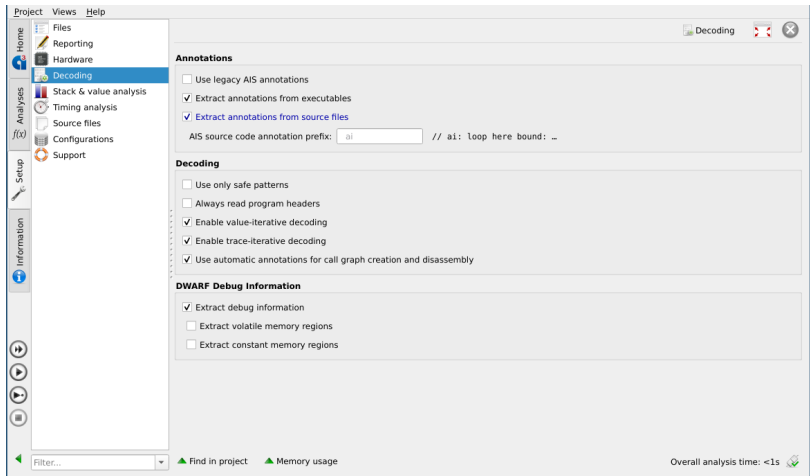
Finished on 2020-06-15 at 17:10:14 after analyzing for 1 second with 0 errors, 4 warnings

Overall analysis time: <1s

Ais-Notationen

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

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

