New Approaches and Visualization for Verification Coverage

Masterarbeit Abschlussvortrag

Maximilian Hailer

LMU München, Deutschland

15.06.2022 14:30 Uhr

Prüfer: Prof. Dr. Dirk Beyer
Mentor: Dr. Philipp Wendler

LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

SoSy-Lab Software Systems

1.1 Motivation – Test-Coverage

```
1   int max(int values [], int size) {
2     if (size < 1) {
3       return values [0];
4     }
5     int max = values[0];
6     for (int i = 0; i < size; i++) {
7       int candidate = values[i];
8       if (candidate > max) {
9         max = candidate;
10      }
11    }
12    return max;
13  }
14
15  int testMax() {
16    int values [3] = {2, 4, 6};
17    int maxValue = max(values, 3);
18    assertEquals (6, maxValue);
19  }
```

Beispiel C-Programm inklusive Test.

- **Problem:** Methode *max* fehlerfrei?
- **Idee:** Erstellen von Unit-Tests (siehe *testMax*)
- **Beobachtung:** Line-Coverage ist unter 100%
- **Konsequenz:** Test übersieht mögliche Fehler
- **Lösung:** Erstellen von weiteren Tests

1.2 Motivation – Verification-Coverage

example_bug.c

	Line data		Source code
1		:	int main() {
2	1	:	int i = 0;
3	1	:	int a = 0;
4		:	
5	38	:	while (1) {
6	74	:	if (i == 20) {
7	37	:	goto LOOPEND;
8		:	} else {
9	111	:	i++;
10	111	:	a++;
11		:	}
12		:	
13	74	:	if (i != a) {
14	0	:	goto ERROR;
15		:	}
16		:	}
17		:	
18	37	:	LOOPEND:
19		:	
20	74	:	if (a != 19) {
21	1	:	goto ERROR;
22		:	}
23		:	
24	37	:	return (0);
25	0	:	ERROR:
26	0	:	return (-1);
27		:	}

Coverage = 12/15 = 80%

example.c

	Line data		Source code
1		:	int main() {
2	1	:	int i = 0;
3	1	:	int a = 0;
4		:	
5	3	:	while (1) {
6	4	:	if (i == 20) {
7	2	:	goto LOOPEND;
8		:	} else {
9	6	:	i++;
10	6	:	a++;
11		:	}
12		:	
13	4	:	if (i != a) {
14	0	:	goto ERROR;
15		:	}
16		:	}
17		:	
18	2	:	LOOPEND:
19		:	
20	4	:	if (a != 20) {
21	0	:	goto ERROR;
22		:	}
23		:	
24	2	:	return (0);
25	0	:	ERROR:
26	0	:	return (-1);
27		:	}

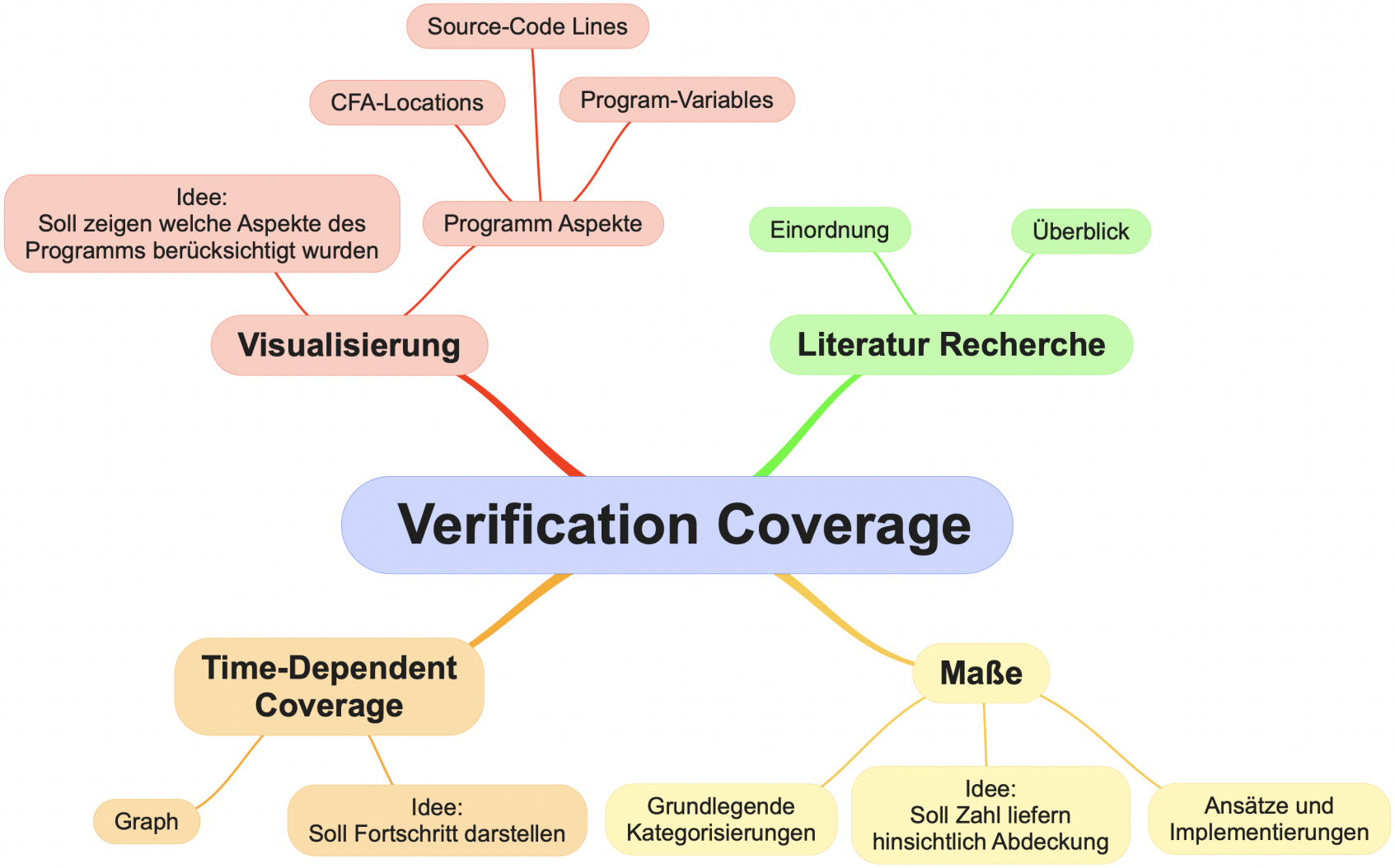
Coverage = 11/15 = 73%

Vergleich von Verification-Coverage für zwei C-Programme die mit CPAchecker mit Prädikatenanalyse mit CEGAR verifiziert worden sind und mit GCOV visualisiert.

1.3 Ziel der Arbeit

Wir wollen **Verification-Coverage Maße** und **Visualisierungen** vorschlagen, die einem Verification-Engineer helfen sollen zu überprüfen ob das ganze Programm vom Verifier berücksichtigt wurde bzw. wie weit wir gekommen sind bei einer unvollständigen Analyse.

1.4 Überblick der Arbeit



2.1 Verification-Coverage Anwendungsfälle

- **Completness of Properties:**
  - **Im Falle einer abgeschlossenen Analyse:** Wie vollständig war die gegebene Spezifikation hinsichtlich der Abdeckung des Programms?
  - **Beispiel:** Programm-Variablen, Code Zeilen etc. berücksichtigt?
- **Completness of Verification Procedure:**
  - **Im Falle einer unvollständigen Analyse:** Wie weit ist die Analyse vorangekommen?
  - **Beispiel:** Welche CFA-Locations haben wir bereits berücksichtigt (und wie oft), bzw. welche noch nicht?

2.2 Überblick zu den Visualisierungen

Implementation der Visualisierungen im HTML-Report von CPAchecker

- **Source-Code-Lines:**
  - **Idee:** Markieren aller abgedeckten Zeilen im Quellcode
- **CFA-Locations:**
  - **Idee:** Markieren aller abgedeckten CFA-Locations
- **Program-Variables:**
  - **Idee:** Markieren aller abgedeckten Variablen im Quellcode

2.2 Visualisierung Source-Code-Lines

CPA Report for bugfree.c

CFA | ARG | Source | TDG | Log | Statistics | Configurations | About | Full Screen Mode

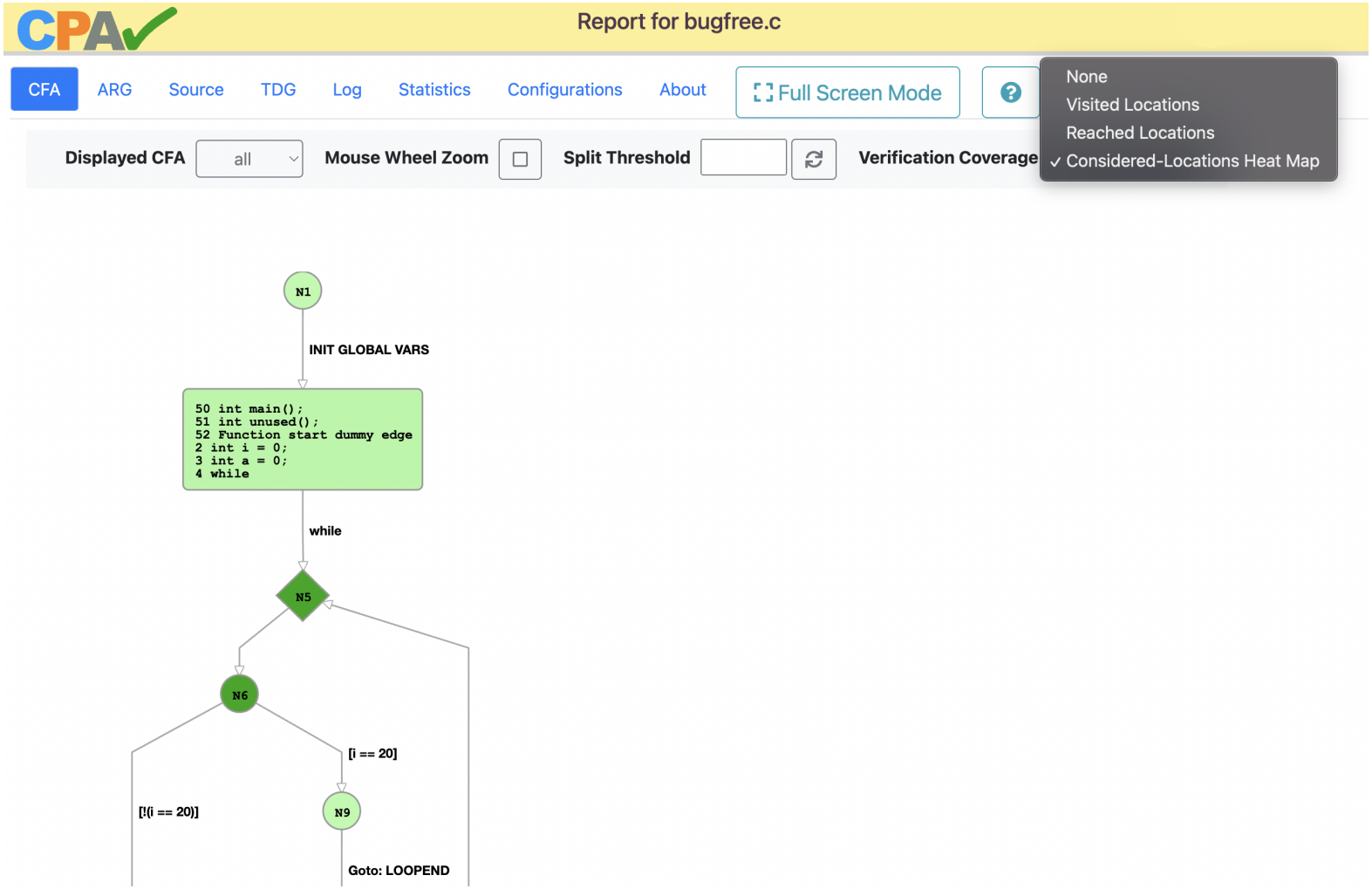
/home/hailer/Masterarbeit/development/cpachecker/run/programs/bugfree.c

- No Verification Coverage
- ✓ Visited-Lines Heat Map
- Visited Variables

```
1  int main() {
2    int i = 0;
3    int a = 0;
4
5    while (1) {
6      if (i == 20) {
7        goto LOOPEND;
8      } else {
9        i++;
10       a++;
11     }
12
13     if (i != a) {
14       goto ERROR;
15     }
16   }
17
18   LOOPEND:
19
20   if (a != 20) {
21     goto ERROR;
22   }
23
24   return (0);
25   ERROR:
26   return (-1);
27 }
```

Source-Tab im Report.html mit aktiver Visited-Lines Heat-Map Visualisierung.

2.2 Visualisierung CFA-Locations



CFA-Tab im Report.html mit aktiver Considered-Locations Heat-Map Visualisierung.

2.2 Visualisierung Program-Variables

CPA

Report for bugfree.c

CFA | ARG | Source | TDG | Log | Statistics | Configurations | About | Full Screen Mode

/home/hailer/Masterarbeit/development/cpachecker/run/programs/bugfree.c | Visited Variables

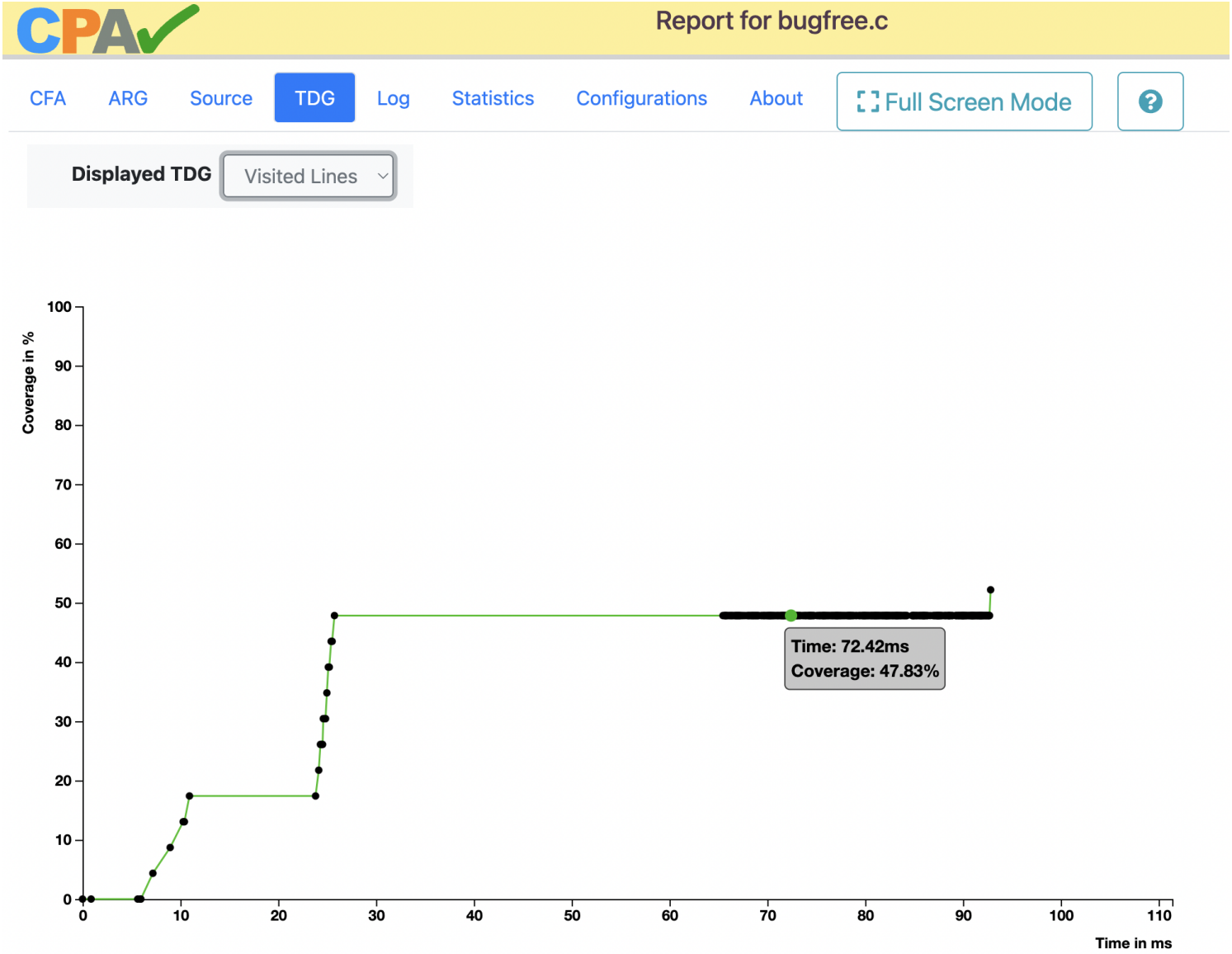
```
1  int main() {
2    int i = 0;
3    int a = 0;
4
5    while (1) {
6      if (i == 20) {
7        goto LOOPEND;
8      } else {
9        i++;
10       a++;
11     }
12
13     if (i != a) {
14       goto ERROR;
15     }
16   }
17
18   LOOPEND:
19
20   if (a != 20) {
21     goto ERROR;
22   }
23
24   return (0);
25   ERROR:
26   return (-1);
27 }
```

Source-Tab im Report.html mit aktiver Visited-Variables Visualisierung.

2.3 Time-Dependent-Coverage (TDC)

- **Motivation:** Interpretation einzelner Coverage-Werte am Ende der Analyse oftmals schwierig
- **Idee:** Sammeln von Coverage-Werten über die gesamte Analysezeit
- **Realisierung:** Tracken des Zeitpunkts und des Coverage-Werts nach jeder Transfer-Relation
- **Anwendung:** Überwiegend für Maße der Kategorie *Completness of Verification Procedure* geeignet
- **Nützlichkeit:** Hilfreich bei der Überlegung wie lange eine Analyse laufen soll (insbesondere bei unknown-Fälle)

2.3 Visualisierung TDC



TDG (Time-Dependent Graph) Tab im Report.html mit der Auswahl Visited-Lines.

3.1 Visited-Lines Coverage

```
1  int main() {
2    int i = 0;
3    int a = 0;
4    int b = 0;
5
6    while (1) {
7      if (i == 20) {
8        goto LOOPEND;
9      } else {
10       i++;
11       a++;
12     }
13
14     if (i != a) {
15       goto ERROR;
16     }
17   }
18
19   LOOPEND:
20
21   if (a != 20) {
22     goto ERROR;
23   }
24
25   return (0);
26   ERROR:
27   return (-1);
28 }
```

- **Coverage-Formel:**
  $Cov(VisitedLines) = \frac{VisitedLines}{TotalLines}$
- **Hier im Beispiel:** $Cov = \frac{13}{16}$
- **Visualisierung:**
  Heat-Map der Source-Code-Lines
- **Vorteile:** Berücksichtigt Code-Blöcke die bei der Analyse nicht herangezogen werden
- **Nachteile:** Problematisch für Analysen wie Prädikatenanalyse mit CEGAR (wegen anfangs geringer Präzision)

3.2 Visited-Variables Coverage

```
1  int main() {
2    int i = 0;
3    int a = 0;
4    int b = 1;
5
6    while (1) {
7      if (i == 20) {
8        goto LOOPEND;
9      } else {
10       i++;
11       a++;
12     }
13
14     if (i != a) {
15       goto ERROR;
16     }
17   }
18
19   LOOPEND:
20
21   if (a != 20) {
22     goto ERROR;
23   }
24
25   return (0);
26   ERROR:
27   return (-1);
28 }
```

- **Coverage-Formel:** $Cov(VisitedVariables) = \frac{VisitedVariables}{TotalVariables}$
- **Hier im Beispiel:** $Cov = \frac{3}{3}$
- **Visualisierung:** Program-Variables
- **Vorteile:** Weniger sensibel bei leichten Code-Änderungen oder bei unterschiedlichen Analysen
- **Nachteile:** Markiert auch unbenutzte Variablen

3.3 Predicate-Abstraction-Variables Coverage

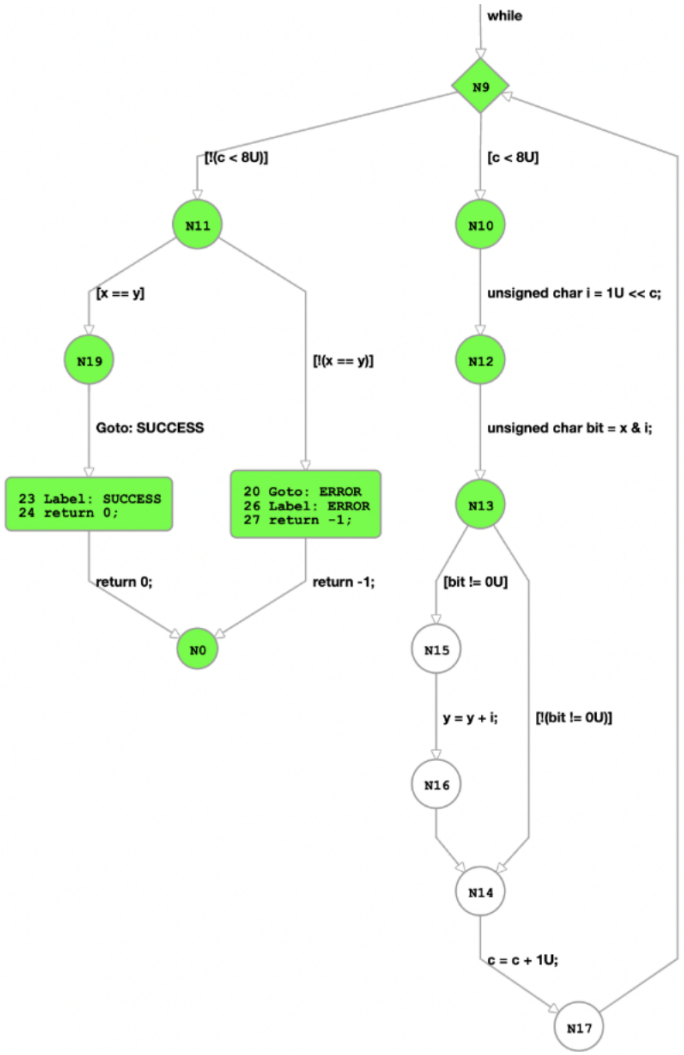
```
1  int main() {
2    int i = 0;
3    int a = 0;
4    int b = 1;
5
6    while (1) {
7      if (i == 20) {
8        goto LOOPEND;
9      } else {
10       i++;
11       a++;
12     }
13
14     if (i != a) {
15       goto ERROR;
16     }
17   }
18
19   LOOPEND:
20
21   if (a != 20) {
22     goto ERROR;
23   }
24
25   return (0);
26   ERROR:
27   return (-1);
28 }
```

- **Coverage-Formel:** $Cov(PredicateAbstractionVariables) = \frac{PredicateAbstractionVariables}{TotalVariables}$
- **Hier im Beispiel:** $Cov = \frac{2}{3}$
- **Visualisierung:** Program-Variables
- **Vorteile:** Markiert unbenutzte Variablen nicht
- **Nachteile:** Nur verfügbar bei Prädikatenanalyse und sofern die Formel der Prädikatenabstraktion Variablen beinhaltet

3.4 Predicates-Considered-Locations Coverage

- **Coverage-Formel:**
  $Cov(PredicateConsideredLocations) = \frac{PredicateConsideredLocations}{TotalLocations}$
- **Idee:** Überprüfen der Variablen in Assume-Edges mit Variablen aus Prädikatenformeln
- **Visualisierung:** CFA-Locations
- **Vorteile:** Versucht Probleme der anfangs geringen Präzision bei CEGAR zu kompensieren
- **Nachteile:**
  - Nur geeignet für Prädikatenanalyse
  - Manchmal zu strikt

3.4 Predicates-Considered-Locations Coverage



Predicate-Considered Locations Coloring

```
1  extern unsigned char __VERIFIER_nondet_uchar(void);
2  int main()
3  {
4      unsigned char x = __VERIFIER_nondet_uchar();
5      unsigned char y;
6      unsigned char c;
7
8      y = 0;
9      c = 0;
10     while (c < (unsigned char)8) {
11         unsigned char i = ((unsigned char)1) << c;
12         unsigned char bit = x & i;
13         if (bit != (unsigned char)0) {
14             y = y + i;
15         }
16         c = c + ((unsigned char)1);
17     }
18     if (x == y) {
19       goto SUCCESS;
20     } else {
21       goto ERROR;
22     }
23
24     SUCCESS: return 0;
25     ERROR: return -1;
26 }
```

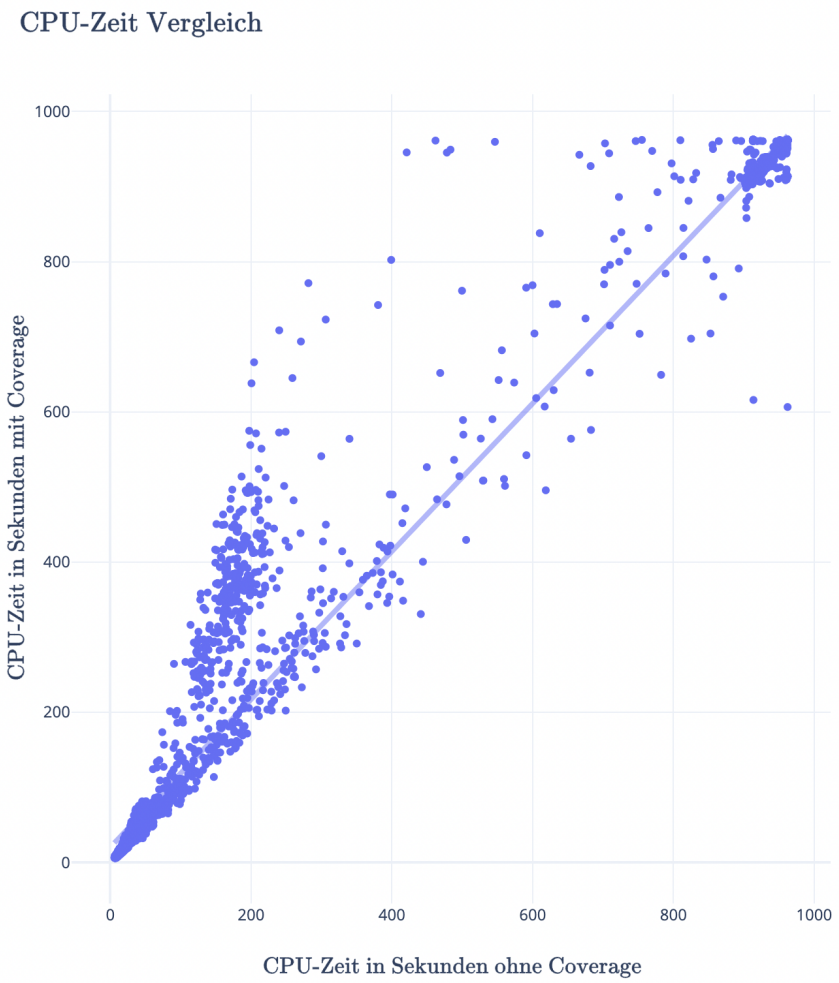
Predicate-Abstraction Variables Coloring

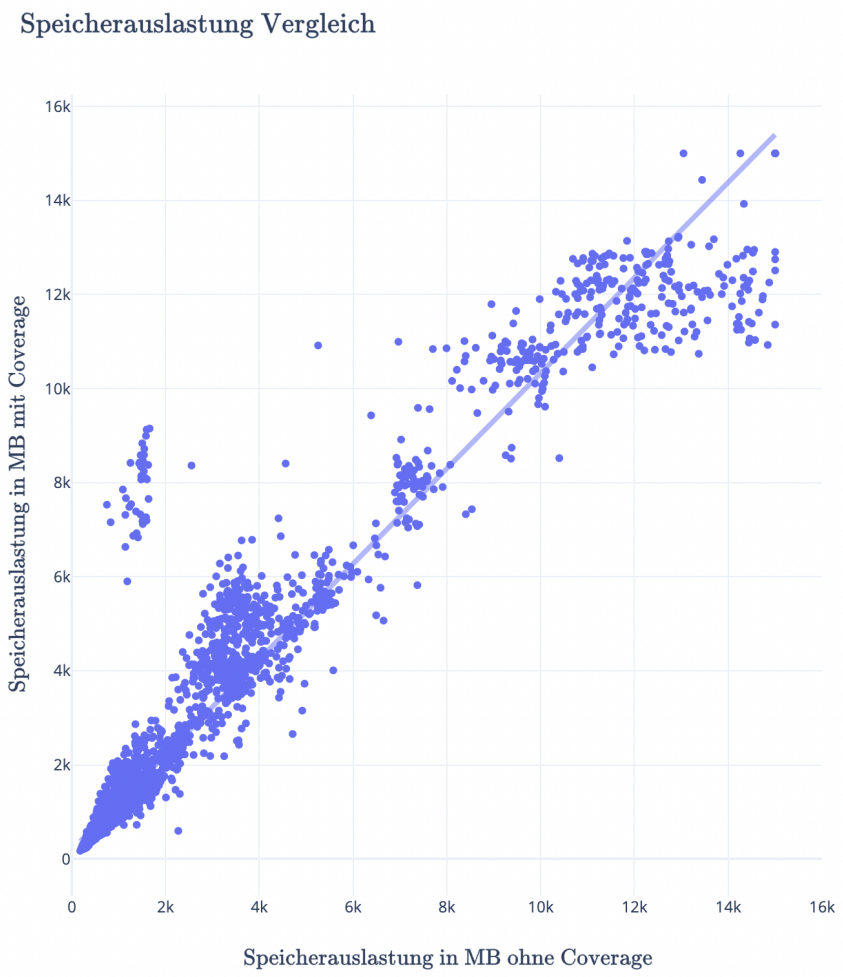
Vergleich von Coverage-Visualisierungen des Programms num_conversion_2.c.

4.1 Evaluation Setup für Performance Benchmark

- **Programme:** 5400 aus der SV-COMP 2022 *ReachSafety*-Kategorie
- **Benchmark-System:** Cluster aus 168 Xeon E3-1230 v5 @ 3.40 GHz und 32 GB RAM
- **Analyse:** Standard Prädikatenanalyse mit CEGAR
- **Zeitlimit:** 900s
- **Memory-Limit:** 15GB

4.1 Auswirkung auf Performance





4.2 Evaluation Setup für Coverage Case-Study

- **Programme:** 14 aus der SV-COMP 2022
  ReachSafety- und *SoftwareSystems*-Kategorie
- **Benchmark-System:**
  Intel Xeon CPU E5-2630 v4 @ 2.20GHz und 16GB RAM
- **Analysen:**
  - Standard Prädikatenanalyse mit CEGAR (predicate)
  - Standard Value-Analyse (value)
- **Zeitlimit:** 900s
- **Memory-Limit:** 15GB

4.2 Benutzte Programme

Programs	Total Lines	Total Functions	Total Conditions	Total Variables	Total Locations	Expected Verdict
arra	13	4	8	3	47	true (unknown)
brs1	28	4	14	4	63	false
zero	24	4	14	5	62	true (unknown)
diam	28	3	24	2	83	false
jain	11	3	2	2	32	true (unknown)
num_	17	3	6	5	40	true
kbfi	380	12	82	56	399	true
s3_s	53	2	30	4	87	true
test	65	2	44	15	126	true
mult	10	3	4	2	37	true
nest	8	2	4	1	8	false
simp	12	3	4	3	33	true
32_1	2960	35	48	222	1024	true
dirn	585	17	68	186	832	true

Überblick aller C-Programme.

4.2 Vergleich zwischen Programmen

Programs	Visited-Lines	Visited-Variables	Predicate-Abstraction-Variables	Predicate-Considered-Locations
arra	85 %	100 %	67 %	45 %
brs1	96 %	100 %	50 %	21 %
zero	67 %	100 %	60 %	23 %
diam	96 %	100 %	50 %	28 %
jain	91 %	100 %	100 %	75 %
num_	88 %	100 %	60 %	70 %
kbfi	99 %	100 %	0 %	25 %
s3_s	94 %	100 %	75 %	22 %
test	97 %	100 %	0 %	26 %
mult	80 %	100 %	100 %	78 %
nest	100 %	100 %	100 %	86 %
simp	83 %	100 %	100 %	76 %
32_1	96 %	81 %	0 %	58 %
dirn	90 %	96 %	0 %	66 %

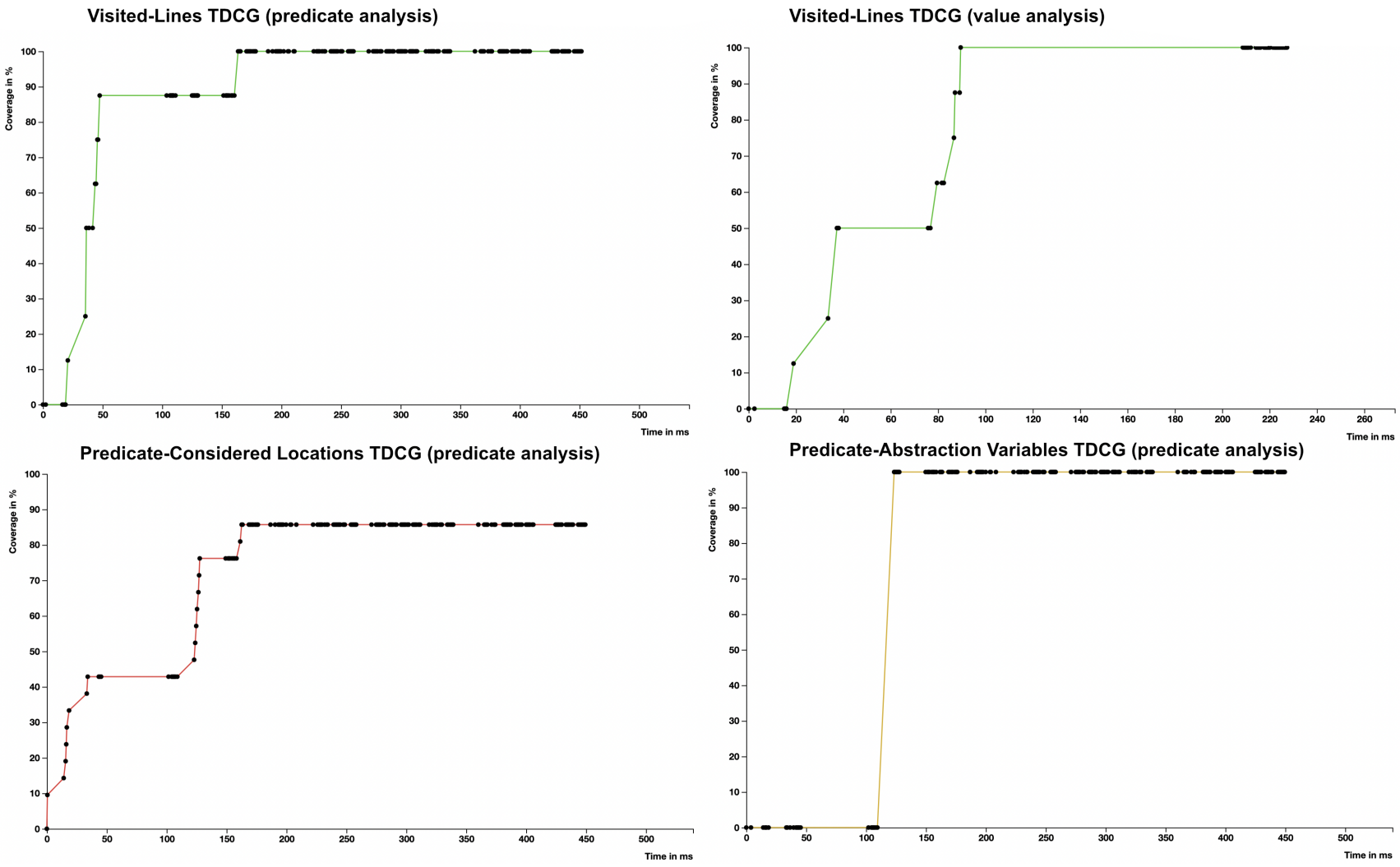
Coverage-Werte für verschiedene Programme.

4.2 Vergleich zwischen Analysen

Programs	**Visited-Lines (predicate)**	**Visited-Lines (value)**	**Visited-Variables (predicate)**	**Visited-Variables (value)**
kbfi	99 %	94 %	100 %	100 %
s3_s	94 %	91 %	100 %	100 %
nest	100 %	88 %	100 %	100 %
test	97 %	97 %	100 %	100 %
32_1	96 %	96 %	81 %	81 %
dirn	90 %	90 %	96 %	96 %

Coverage-Werte für verschiedene Analysen.

4.2 Time-Dependent Coverage Graphs (TDCGs)

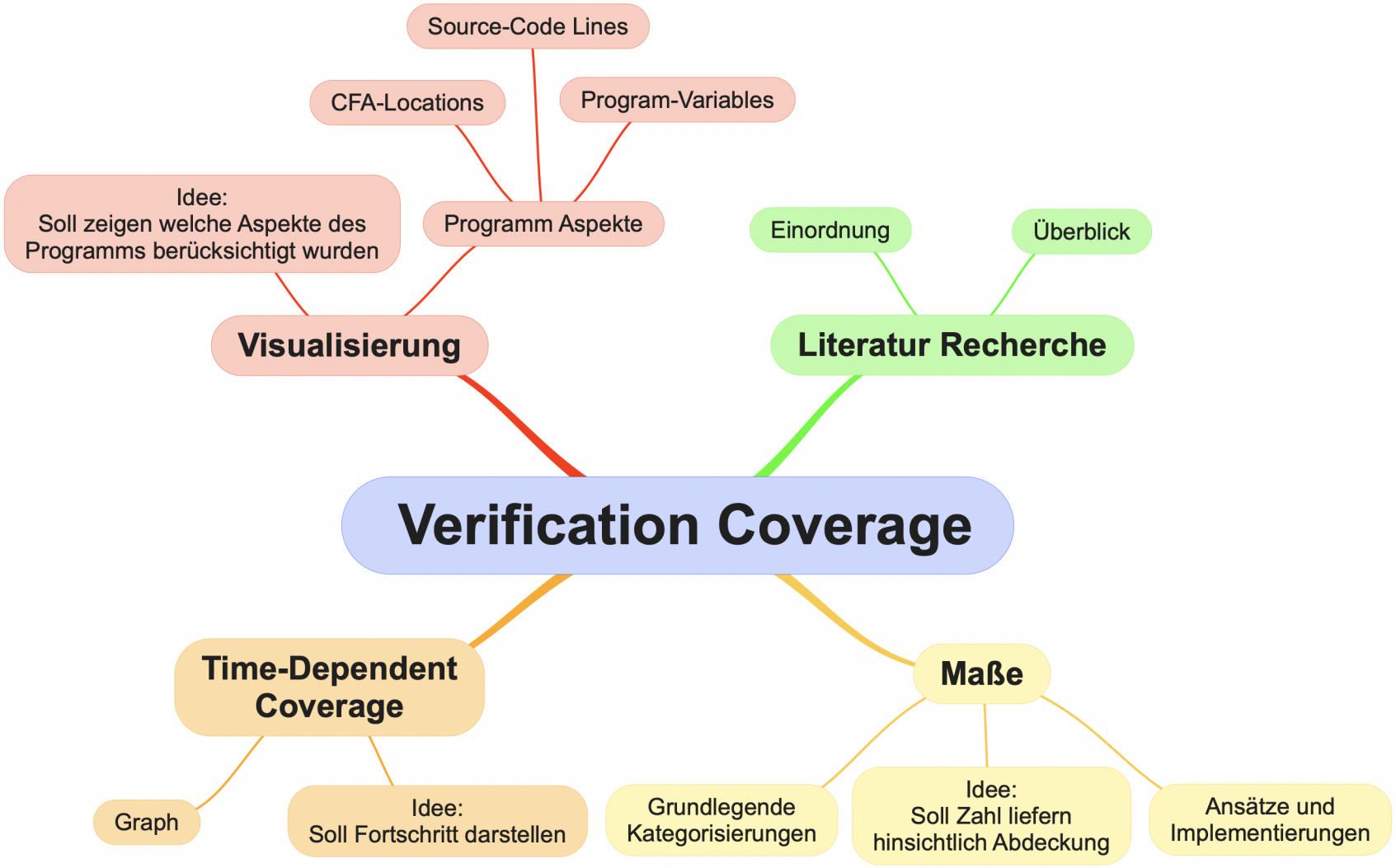


TDCGs für das C-Programm: nested_1b.c.

5 Fazit

- Funktioniert gut:
  - *Completeness of Properties* Maße je nach Szenario geeignet um Plausibilität der Spezifikation zu prüfen
  - *Completeness of Verification Procedure* Visualisierung können helfen Überblick über den Fortschritt der Analyse zu erhalten
- Neutral:
  - Vorgeschlagene Coverage Maße verringern geringfügig die Performance
- Schwierigkeiten:
  - Interpretation von *Completeness of Verification Procedure* Maße als einzelne Zahl schwierig
  - TDCGs nicht für alle Szenarien sinnvoll

6 Überblick der Arbeit



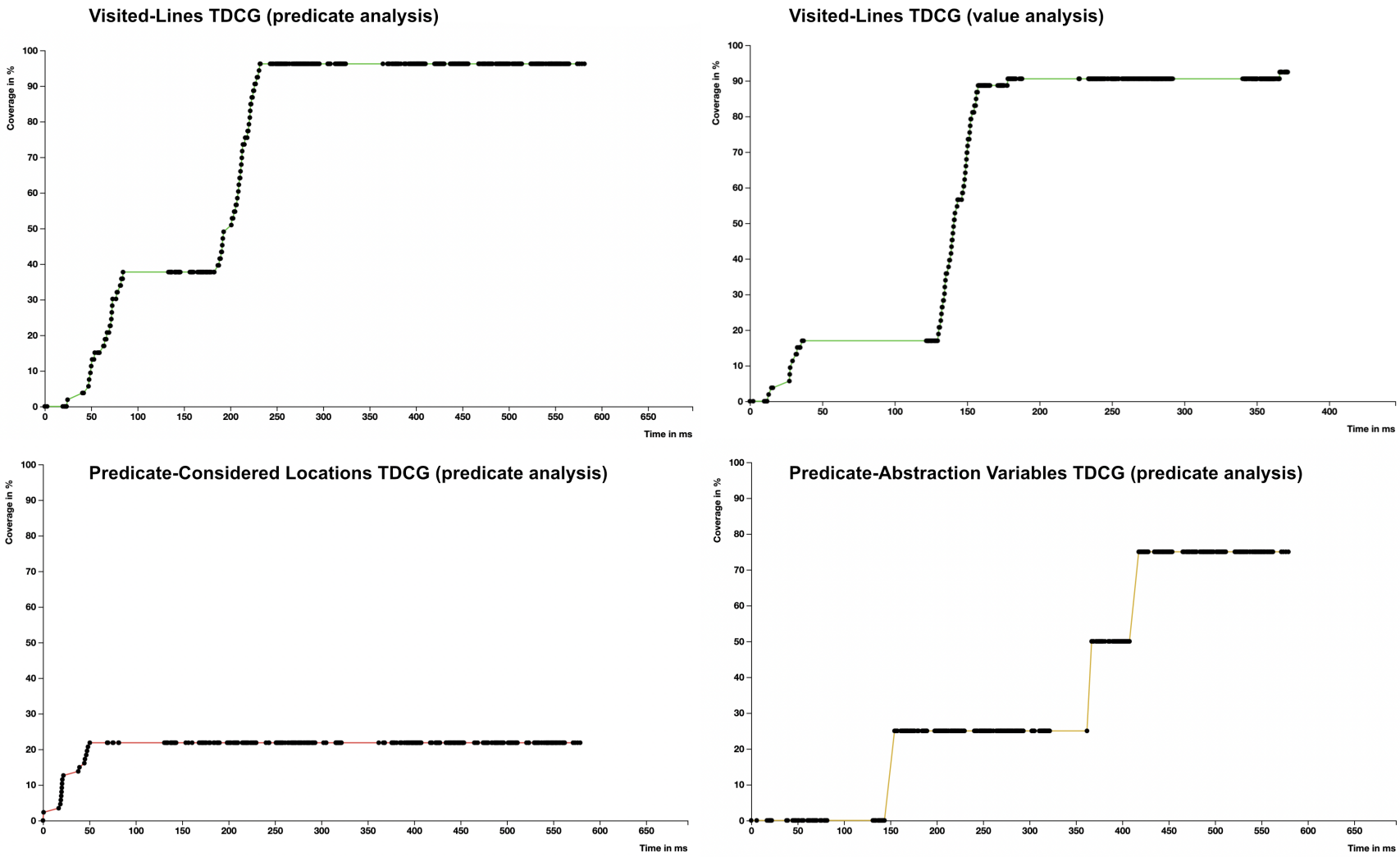
6.1 Anhang – Programm Abkürzungen

- ReachSafety-Arrays
  - array_doub_access_init_const.c (arra)
  - brs1f.c (brs1)
  - zero_sum_const1.c (zero)
- ReachSafety-BitVectors
  - diamond_2-1.c (diam)
  - jain_5-2 .c (jain)
  - num_conversion_2.c (num_)
- ReachSafety-ControlFlow
  - kbfiltr_simpl1.cil.c (kbfi)
  - s3_srvr_1b.cil.c (s3_s)
  - test_locks_7.c (test)

6.2 Anhang – Programm Abkürzungen

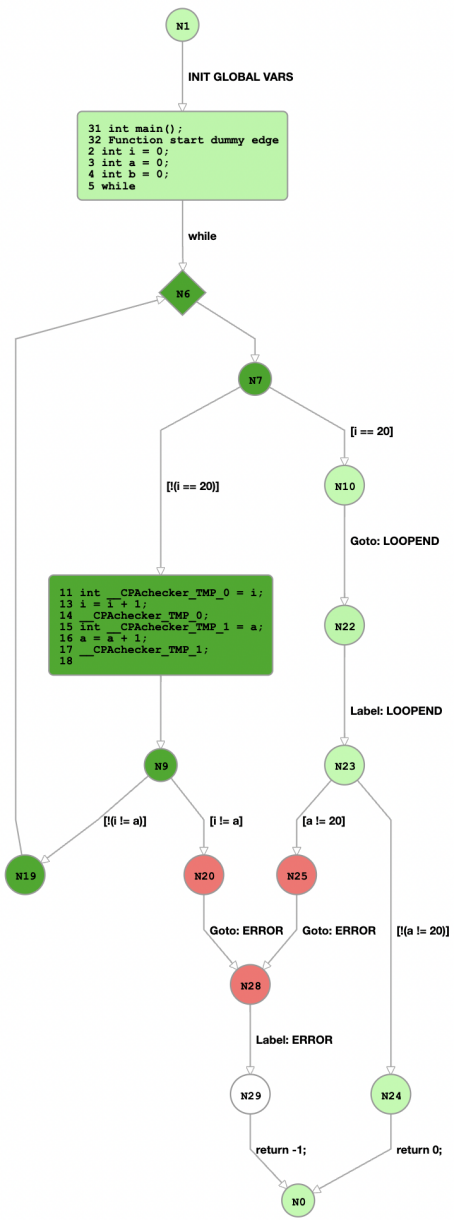
- ReachSafety-Loops
  - multivar_1-1.c (mult)
  - nested_1b.c (nest)
  - simple_vardep_1.c (simp)
- SoftwareSystems-DeviceDriversLinux64-ReachSafety
  - 32_1_cilled_ok_nondet_linux-3.4-32_1-drivers–acpi–bgrt.ko-ldv_main0_sequence_infinite_withcheck_stateful.cil.out.i (32_1)
- SoftwareSystems-BusyBox-ReachSafety
  - dirname-1.i (dirn)

6.3 Time-Dependent Coverage Graphs (TDCGs)



TDCGs für das C-Programm: s3_srvr_1b.cil.c.

6.4 Considered-Locations Heat-Map Coloring



- **Idee:**
  - Grüne Heat-Map aller Reached-Locations
  - Rote Markierung für alle Visited-Locations aber nicht Reached-Locations
  - Weiße Markierung für alle Locations die nie erreicht wurden
- **Vorteile:**
  - Man erkennt Locations die besonders häufig besucht wurden (Loops)
  - Man erkennt Locations die nie besucht wurden (z.B. unused methods)
- **Nachteile:** Problematisch für Analysen wie Prädikatenanalyse mit CEGAR (wegen anfangs geringer Präzision)