

Cendrillon Sort

Shape Analysis

Pascal Jung* Thorsten Kober†

30. September 2015

Zusammenfassung

...

*358XXXX

†3583619

Inhaltsverzeichnis

| | | |
|----------|---|-----------|
| 1 | Einführung | 3 |
| 2 | Allgemeiner Aufbau der Analyse | 3 |
| 2.1 | Kernprädikate | 3 |
| 2.2 | Actions | 3 |
| 3 | Meilenstein A | 4 |
| 3.1 | Eingabestrukturen | 5 |
| 3.2 | TVP | 5 |
| 3.3 | Besonderheiten | 5 |
| 3.4 | Ausgabestrukturen | 5 |
| 4 | Meilenstein B | 6 |
| 4.1 | Eingabestrukturen | 6 |
| 4.2 | TVP | 6 |
| 4.3 | Besonderheiten | 7 |
| 4.4 | Ausgabestrukturen | 7 |
| 4.5 | Anpassungen | 7 |
| 5 | Meilenstein C | 7 |
| 5.1 | Eingabestrukturen | 7 |
| 5.2 | TVP | 7 |
| 5.3 | Besonderheiten | 8 |
| 5.4 | Ausgabestrukturen | 8 |
| 6 | Anhang | 10 |
| 6.1 | Analysebericht Meilenstein A | 10 |
| 6.2 | Analysebericht Meilenstein B | 22 |
| 6.3 | Analysebericht Meilenstein B (verbessert) | 44 |
| 6.4 | Analysebericht Meilenstein C | 57 |

1 Einführung

2 Allgemeiner Aufbau der Analyse

2.1 Kernprädikate

...

istGut* und *istSchlecht

...

```
%p istGut(v)
%p istSchlecht(v)
```

isArbitrary

...

```
%p isArbitrary(v)
```

2.2 Actions

...

Set_Ist_Gut_Halb*, *Set_Ist_Gut_False* und *Set_Ist_Gut_True

...

```
%action Set_Ist_Gut_Halb(lhs) {
    %t lhs + "->istGut = 1/2"
    {
        istGut(v) = (lhs(v) & 1/2) | (!lhs(v) & istGut(v))
    }
}

%action Set_Ist_Gut_False(lhs) {
    %t lhs + "->istGut = false"
    {
        istGut(v) = (lhs(v) & 0) | (!lhs(v) & istGut(v))
    }
}

%action Set_Ist_Gut_True(lhs) {
    %t lhs + "->istGut = true"
    {
        istGut(v) = (lhs(v) & 1) | (!lhs(v) & istGut(v))
    }
}
```

Set_Ist_Schlecht_Opposite

...

```
%action Set_Ist_Schlecht_Opposite(lhs) {
  %t lhs + "->istSchlecht = !" + lhs + "->istGut "
  {
    istSchlecht(v) = (!lhs(v) & istSchlecht(v)) | (lhs(v) &
      !istGut(v) & !isArbitrary(v))
  }
}
```

Is_Good und Is_Not_Good

...

```
%action Is_Good(lhs) {
  %t lhs + "->istGut == true"
  %f { lhs(v) & istGut(v) }
  %p E(v) lhs(v) & istGut(v)
}

%action Is_Not_Good(lhs) {
  %t lhs + "->istGut == false"
  %f { lhs(v) & istSchlecht(v) }
  %p A(v) !(lhs(v) & istGut(v))
}
```

Free_Arbitrary

...

```
%action Free_Arbitrary() {
  %t "Free_Arbitrary()"
  {
    t[n](v_1, v_2) = t[n](v_1, v_2)
    foreach(z in PVar) {
      r[n,z](v) = r[n,z](v)
    }
    is[n](v) = is[n](v)
  }
  %retain !isArbitrary(v) | |/{r[n, z](v): z in PVar}
}
```

3 Meilenstein A

In diesem ersten Meilenstein soll die Erzeugung einer einfach-verketteten Liste mit beliebig vielen Linsenobjekten analysiert werden. Insbesondere die folgenden Eigenschaften des Speichers nach dem Abarbeiten des entsprechenden Programnteils sind von Interesse:

- Keine Objekte sind verloren gegangen und liegen unerreichbar auf dem Heap. Ihre Analyse soll zeigen, dass nach dem entsprechenden Programmteil auf dem Heap lediglich Linsenobjekte existieren, die von der Programmvariable *LinsenTopf* aus erreichbar sind.
- Alle Linsenobjekte sind korrekt erzeugt, d.h eine Linse ist entweder gut oder schlecht. Niemals hat sie beide oder keine der Eigenschaften. Aus den von Ihrer Analyse erzeugten Strukturen muss hervorgehen, dass obige Eigenschaft gilt, also genau eines der beiden Attribute *istGut* und *istSchlecht* den Wahrheitswert true besitzt.

3.1 Eingabestrukturen

Da der zu untersuchende Programmabschnitt für die Erzeugung der Eingabestruktur von Meilenstein B zuständig ist, wird für dessen Analyse lediglich die leere Eingabestruktur benötigt.

3.2 TVP

...

| | |
|---|---|
| L1 Malloc_L(linse) | L1a // linse = malloc(sizeof(Linse)); |
| L1a uninterpreted() : false; | L2 // linse->istGut = (rand() % 2) ? true |
| L1a uninterpreted() | L2a |
| L2 Set_Ist_Gut_True(linse) | L3 |
| L2a Set_Ist_Gut_False(linse) | L3 |
| L3 Set_Ist_Schlecht_Opposite(linse) | L4 // linse->istSchlecht = !linse->istGut |
| ; Copy_Var_L(linsenTopf, linse) | L5 // linsenTopf = linse; |
| L5 Copy_Var_L(topf, linsenTopf) | L6 // topf = linsenTopf; |
| L6 uninterpreted() | L7 // while(i <= n) { |
| L6 uninterpreted() | exit |
| L7 Malloc_L(linse) | L7a // linse = malloc(sizeof(Linse)); |
| L7a uninterpreted() true : false; | L8 // linse->istGut = (rand() % 2) ? |
| L7a uninterpreted() | L8a |
| L8 Set_Ist_Gut_True(linse) | L9 |
| L8a Set_Ist_Gut_False(linse) | L9 |
| L9 Set_Ist_Schlecht_Opposite(linse) | L10 // linse->istSchlecht = !linse-> |
| istGut; | L11 // topf->next = linse; |
| L10 Set_Next_L(topf, linse) | L6 // topf = linse; |
| L11 Copy_Var_L(topf, linse) | // } |
| exit Set_Null_L(linse) | exitA |
| exitA Set_Null_L(topf) | exitB |
| exitB Assert_No_Good_n_Bad() | error |
| exitB Assert_ListInvariants(linsenTopf) | error |
| exitB Assert_No_Leak(linsenTopf) | error |

3.3 Besonderheiten

...

3.4 Ausgabestrukturen

...

4 Meilenstein B

In diesem Meilenstein soll der Abbau einer einfach-verketteten Liste mit beliebig vielen Linsenobjekten sowie der parallele Aufbau zweier disjunkter, die Linsenobjekte in gute und schlechte Linsen aufteilende einfach-verkettete Listen analysiert werden.

- Alle Linsen befinden sich am Ende des Sortiervorgangs in genau einem der beiden Töpfe für gute bzw. schlechte Linsen. D.h. jedes Linsenobjekt ist entweder von Programmzeiger *guteLinsen* oder *schlechteLinsen* aus erreichbar.
- Alle Linsen, die von *guteLinsen* aus erreichbar sind, haben definitive Werte für *istGut* und *istSchlecht*: *istGut* ist wahr und *istSchlecht* falsch für alle diese Linsenobjekte.
- Alle Linsen, die von *schlechteLinsen* aus erreichbar sind, haben definitive Werte für *istGut* und *istSchlecht*: *istGut* ist falsch und *istSchlecht* wahr für alle diese Linsenobjekte.

4.1 Eingabestrukturen

...

4.2 TVP

...

| | |
|--|---|
| L0 Set_Null_L(linsenTopf) | L1 // Linse* linsenTopf = NULL; |
| L1 Malloc_L(linse) | L1a // linse = malloc(sizeof(Linse)); |
| L1a uninterpreted() | L2 // linse->istGut = (rand() % 2) ? true |
| : false; | |
| L1a uninterpreted() | L2a |
| L2 Set_Ist_Gut_True(linse) | L3 |
| L2a Set_Ist_Gut_False(linse) | L3 |
| L3 Set_Ist_Schlecht_Opposite(linse) | L4 // linse->istSchlecht = !linse->istGut |
| ; | |
| L4 Copy_Var_L(linsenTopf, linse) | L5 // linsenTopf = linse; |
| L5 Copy_Var_L(topf, linsenTopf) | L6 // topf = linsenTopf; |
| L6 uninterpreted() | L7 // while (i <= n) { |
| L6 uninterpreted() | L12 |
| L7 Malloc_L(linse) | L7a // linse = malloc(sizeof(Linse)); |
| L7a uninterpreted() | L8 // linse->istGut = (rand % 2) ? true: |
| false; | |
| L7a uninterpreted() | L8a |
| L8 Set_Ist_Gut_True(linse) | L9 |
| L8a Set_Ist_Gut_False(linse) | L9 |
| L9 Set_Ist_Schlecht_Opposite(linse) | L10 // linse->istSchlecht = !linse-> |
| istGut; | |
| L10 Set_Next_L(topf, linse) | L11 // topf->next = linse; |
| L11 Copy_Var_L(topf, linse) | L6 // topf = linse; |
| | // } |
| L12 Copy_Var_L(topf, linsenTopf) | L13 // topf = LinsenTopf; |
| L13 Is_Null_Var(topf) | exit // while (topf) { |
| L13 Is_Not_Null_Var(topf) | L14 |
| L14 Copy_Var_L(linse, topf) | L15 // linse = topf; |
| L15 Get_Next_L(topf, topf) | L16 // topf = topf->next; |
| L16 Is_Good(linse) | L17 // if (linse->istGut) { |
| L16 Is_Not_Good(linse) | L20 |
| L17 Set_Next_Null_L(linse) | L18 // |
| L18 Set_Next_L(linse, guteLinsen) | L19 // |
| L19 Copy_Var_L(guteLinsen, linse) | L13 // linse->next = guteLinsen; |
| | guteLinsen = linse; |
| | // } else { |
| L20 Set_Next_Null_L(linse) | L21 // |
| L21 Set_Next_L(linse, schlechteLinsen) | L22 // |
| L22 Copy_Var_L(schlechteLinsen, linse) | L13 // linse->next = schlechteLinsen; |
| | schlechteLinsen = linse; |
| | // } |
| | // } |
| exit Set_Null_L(linse) | exitA |

| | |
|---|-------|
| exitA Set_Null_L(linsenTopf) | exitB |
| exitB Assert_No_Reachable_Arbitrary() | error |
| exitB Assert_No_Good_n_Bad() | error |
| exitB Assert_Good_Linsen(guteLinsen) | error |
| exitB Assert_Bad_Linsen(schlechteLinsen) | error |
| exitB Assert_No_Leak_End(guteLinsen, schlechteLinsen) | error |

4.3 Besonderheiten

...

4.4 Ausgabestrukturen

...

4.5 Anpassungen

...

| | |
|---------------------------------|---------------------------------------|
| L0 Set_Null_L(linsenTopf) | L0a // Linse* linsenTopf = NULL; |
| // Program fix | |
| L0a Set_Null_L(guteLinsen) | L0b // Linse* guteLinsen = NULL; |
| L0b Set_Null_L(schlechteLinsen) | L1 // Linse* schlechteLinsen = NULL; |
| L1 Malloc_L(linse) | L1a // linse = malloc(sizeof(Linse)); |

5 Meilenstein C

Im Quellcode ist eine alternative, elegantere Implementierung der Linsentopferzeugung (auskommentiert) vorhanden. Ersetzen Sie Ihre, bereits verifizierte Implementierung durch die Kürzere und stellen Sie sicher, dass Ihre Analyse weiterhin die Verifikationsziele der Meilensteine A und B erfüllt.

5.1 Eingabestrukturen

...

5.2 TVP

...

| | |
|---|---|
| L0 Set_Null_L(linsenTopf) | L0a // Linse* linsenTopf = NULL; |
| // This corrects the program | |
| L0a Set_Null_L(guteLinsen) | L0b // Linse* guteLinsen = NULL; |
| L0b Set_Null_L(schlechteLinsen) | L1 // Linse* schlechteLinsen = NULL; |
| L1 uninterpreted() | L3 // jump into loop as condition is i <= |
| n and i is for the first iteration 1 and n at least 1 | |
| L2 uninterpreted() | optional |
| L2 uninterpreted() | L3 // while (i <= n) { |
| L3 Malloc_L(linse) | L4 // linse = malloc(sizeof(Linse)); |
| L4 uninterpreted() | L5 // linse->istGut = (rand() % 2) ? |
| true: false; | |
| L4 uninterpreted() | L6 |
| L5 Set_Ist_Gut_True(linse) | L7 |
| L6 Set_Ist_Gut_False(linse) | L7 |
| L7 Set_Ist_Schlecht_Opposite(linse) | L8 // linse->istSchlecht = !linse-> |
| istGut; | |
| L8 Set_Next_L(linse, linsenTopf) | L9 // linse->next = linsenTopf; |
| L9 Copy_Var_L(linsenTopf, linse) | L2 // linsenTopf = linse; |
| | // } |
| optional Set_Null_L(linse) | L12 // clean result structures |

| | |
|---|---------------------------------------|
| L12 Copy_Var_L(topf, linsenTopf) | L13 // topf = LinsenTopf; |
| L13 Is_Null_Var(topf) | exit // while (topf) { |
| L13 Is_Not_Null_Var(topf) | L14 |
| L14 Copy_Var_L(linse, topf) | L15 // linse = topf; |
| L15 Get_Next_L(topf, topf) | L16 // topf = topf->next; |
| L16 Is_Good(linse) | L17 // if (linse->istGut) { |
| L16 Is_Not_Good(linse) | L20 |
| L17 Set_Next_Null_L(linse) | L18 // linse->next = NULL; // |
| Additional statement for analysis, | no semantic change |
| L18 Set_Next_L(linse, guteLinsen) | L19 // linse->next = guteLinsen; |
| L19 Copy_Var_L(guteLinsen, linse) | L13 // guteLinsen = linse; |
| | // } else { |
| L20 Set_Next_Null_L(linse) | L21 // linse->next = NULL; // |
| Additional statement for analysis, | no semantic change |
| L21 Set_Next_L(linse, schlechteLinsen) | L22 // linse->next = schlechteLinsen; |
| L22 Copy_Var_L(schlechteLinsen, linse) | L13 // schlechteLinsen = linse; |
| | // } |
| | // } |
| exit Set_Null_L(linse) | exitA //clean result structures |
| exitA Set_Null_L(linsenTopf) | exitB |
| exitB Assert_No_Reachable_Arbitrary() | error |
| exitB Assert_No_Good_n_Bad() | error |
| exitB Assert_Good_Linsen(guteLinsen) | error |
| exitB Assert_Bad_Linsen(schlechteLinsen) | error |
| exitB Assert_No_Leak_End(guteLinsen, schlechteLinsen) | error |

5.3 Besonderheiten

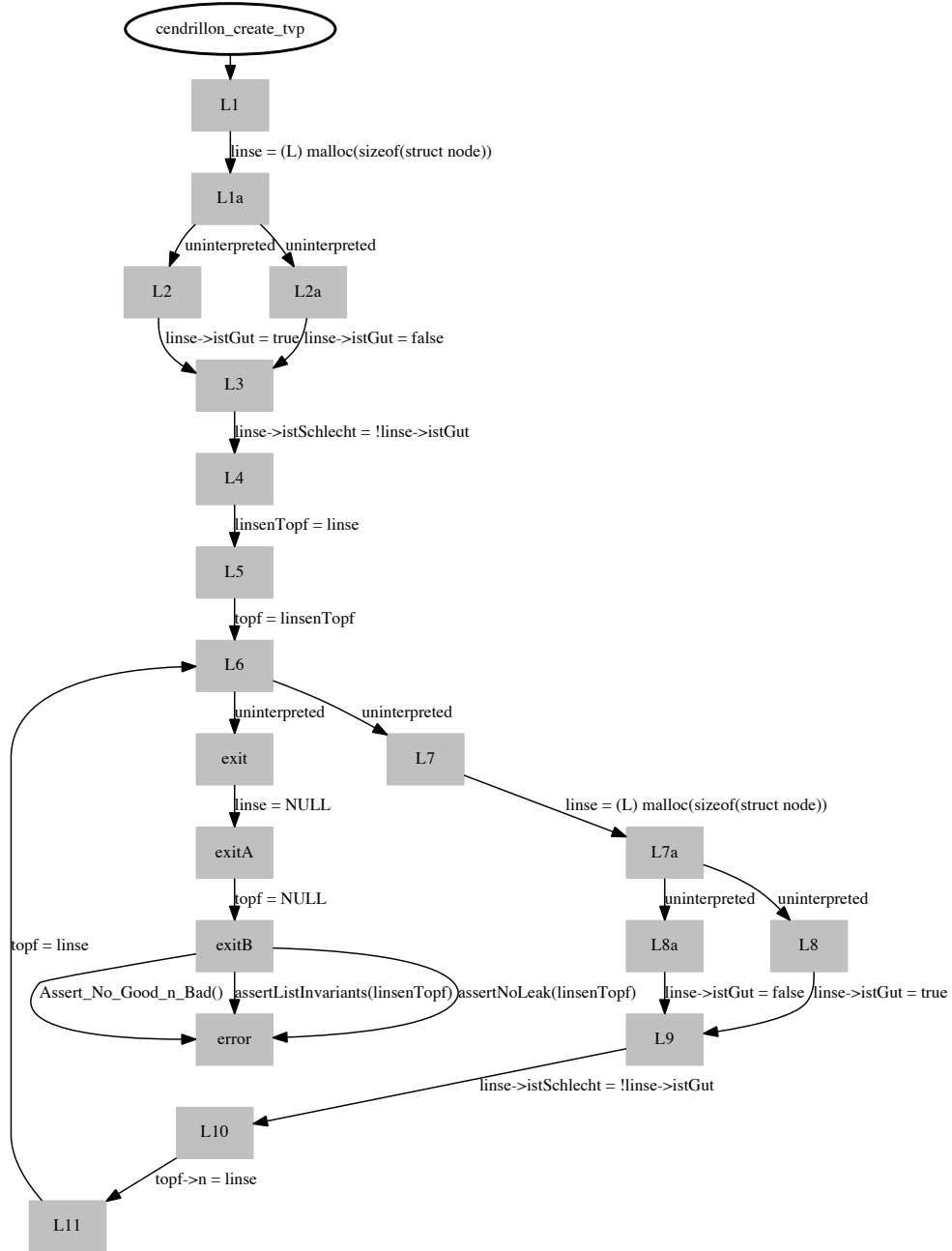
...

5.4 Ausgabestrukturen

...

6 Anhang

6.1 Analysebericht Meilenstein A

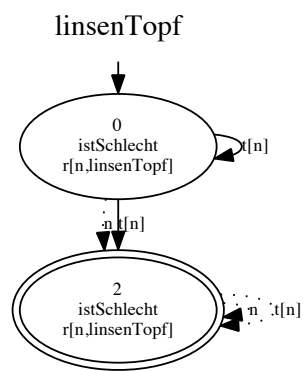


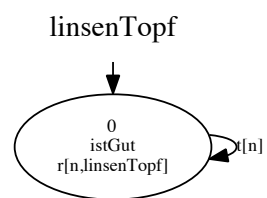
Program Location

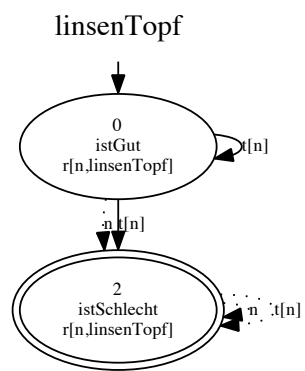
L1

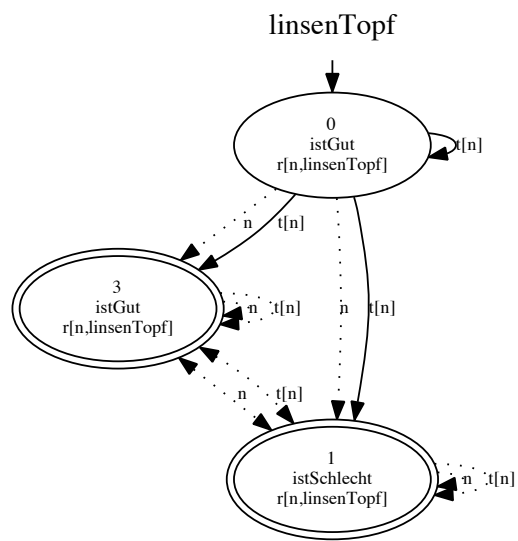
Structure with empty universe

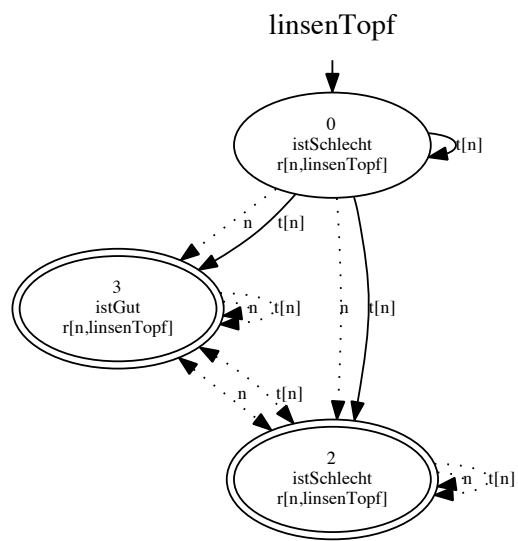
Program Location
exitB

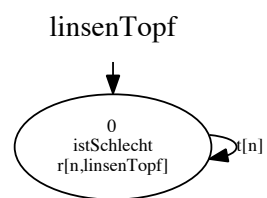


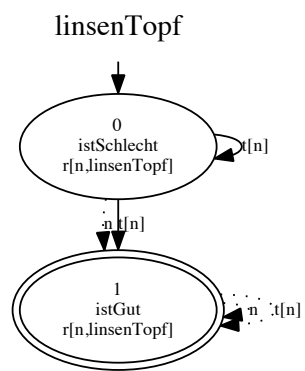


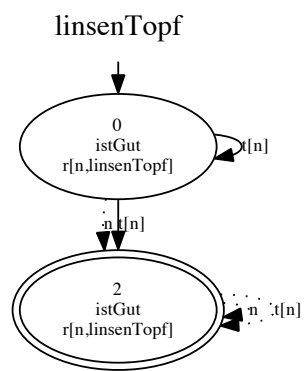




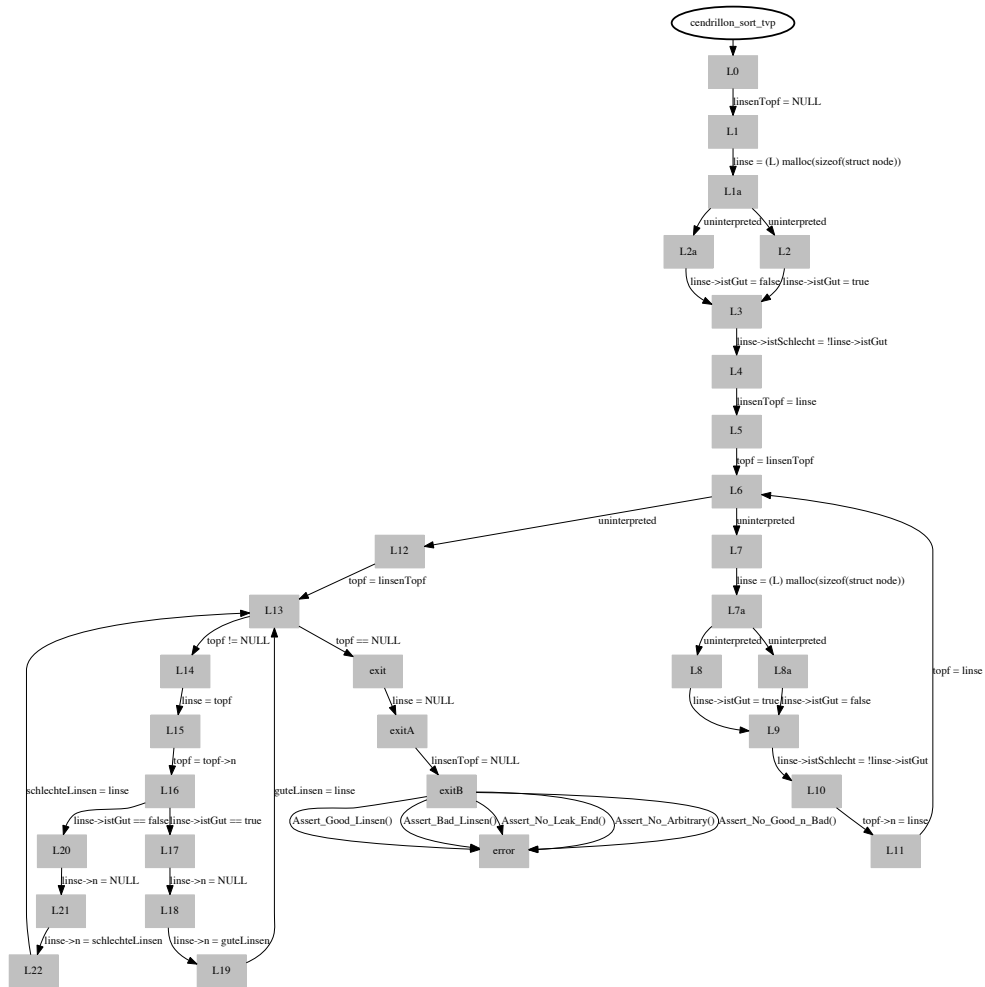




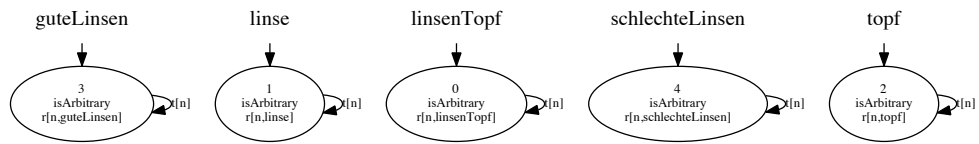




6.2 Analysebericht Meilenstein B

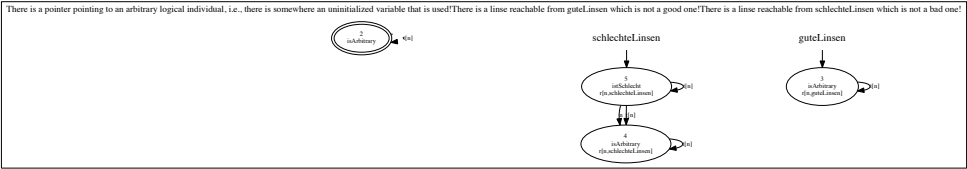


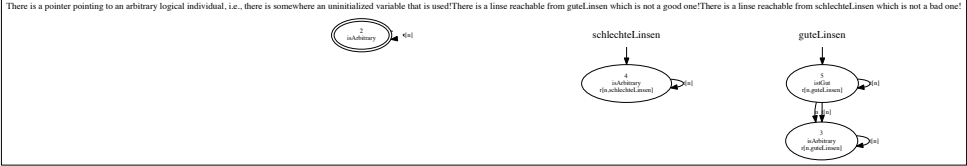
Program Location L0

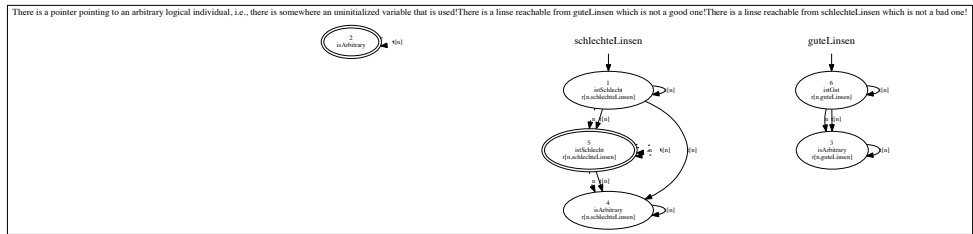


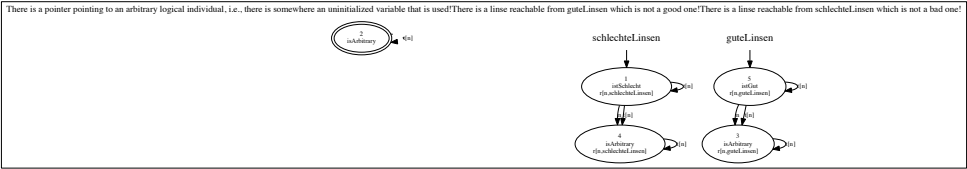
Program Location
exitB

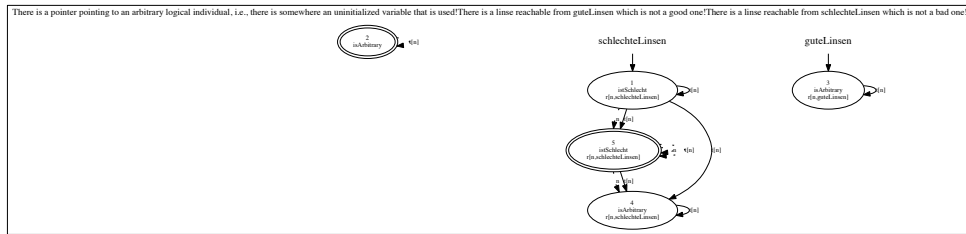
Messages for exitB

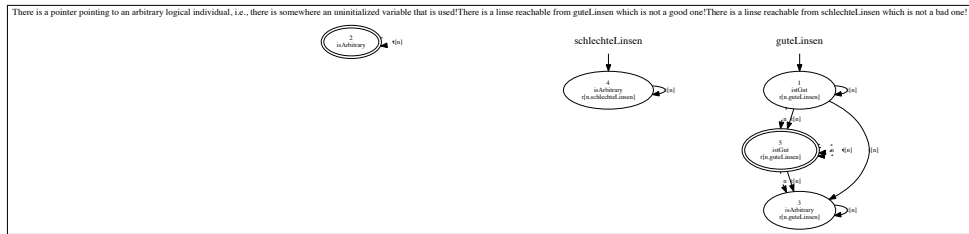


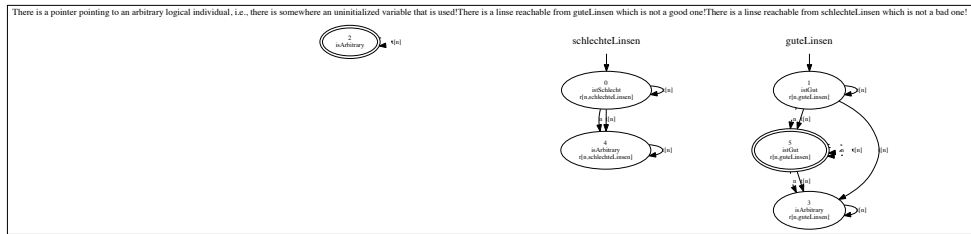


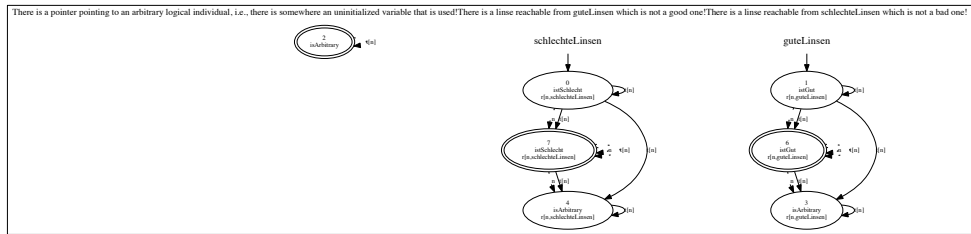


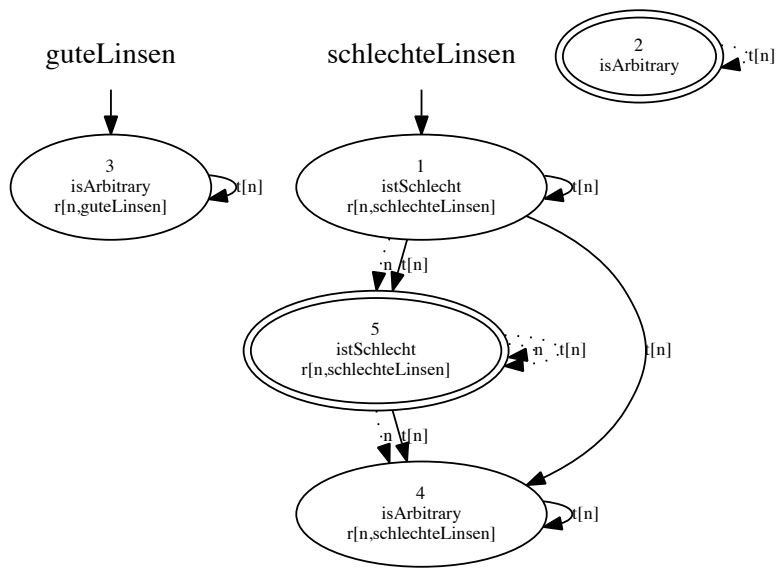


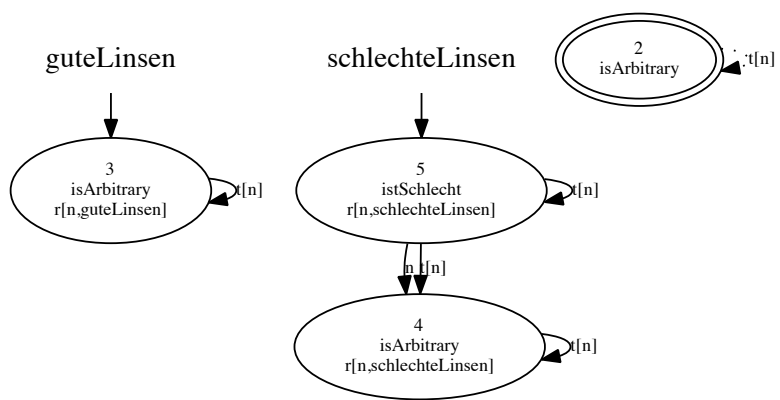


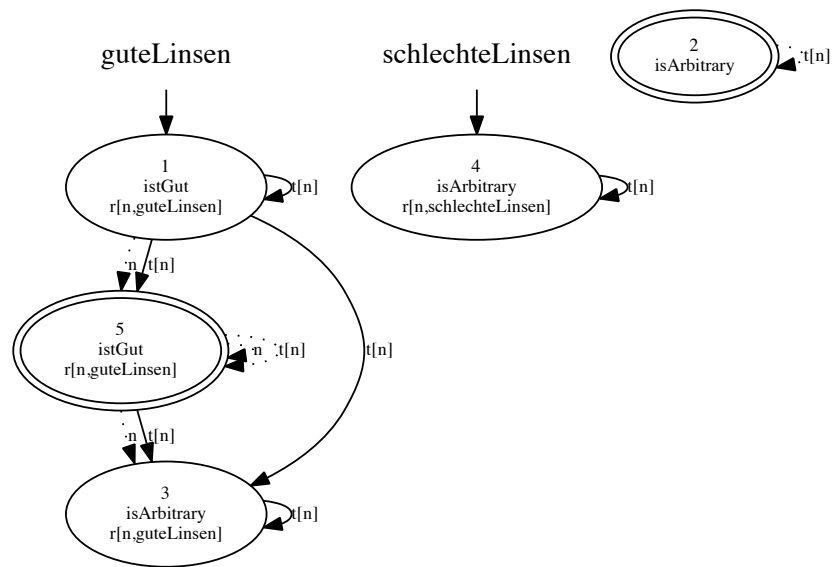


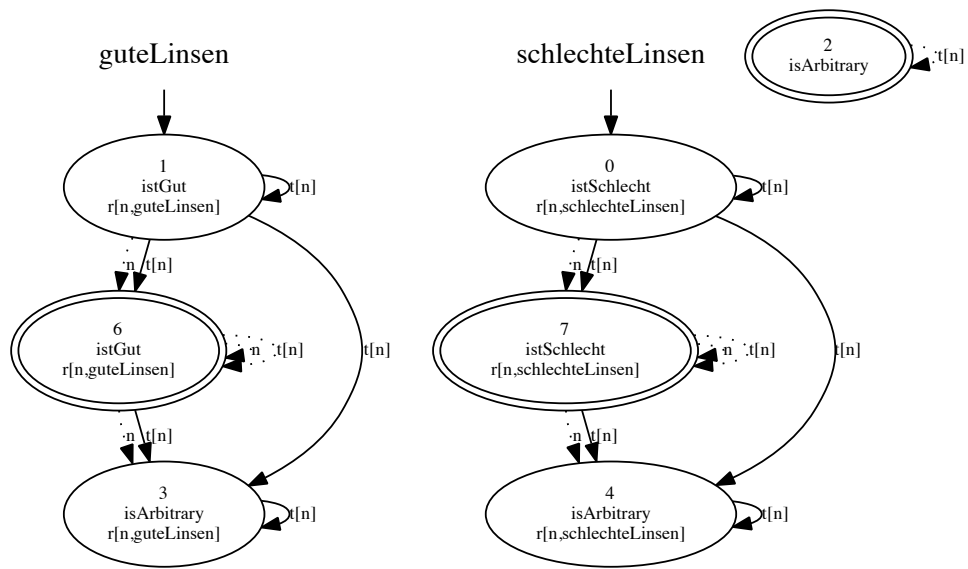


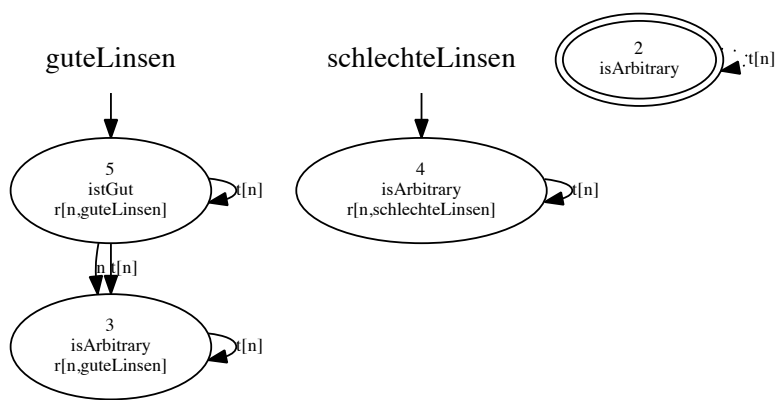


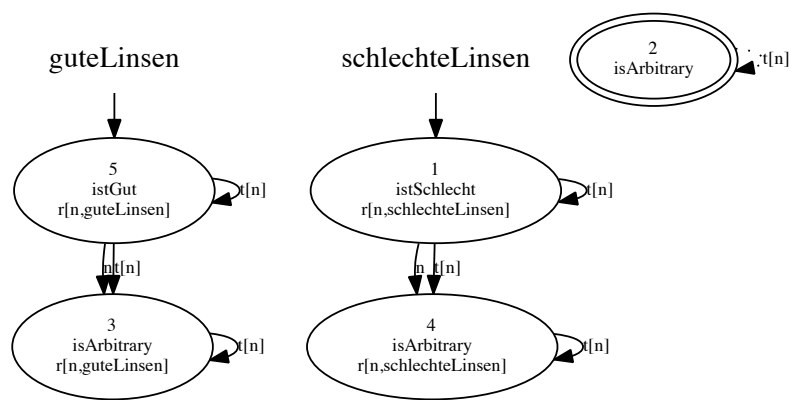


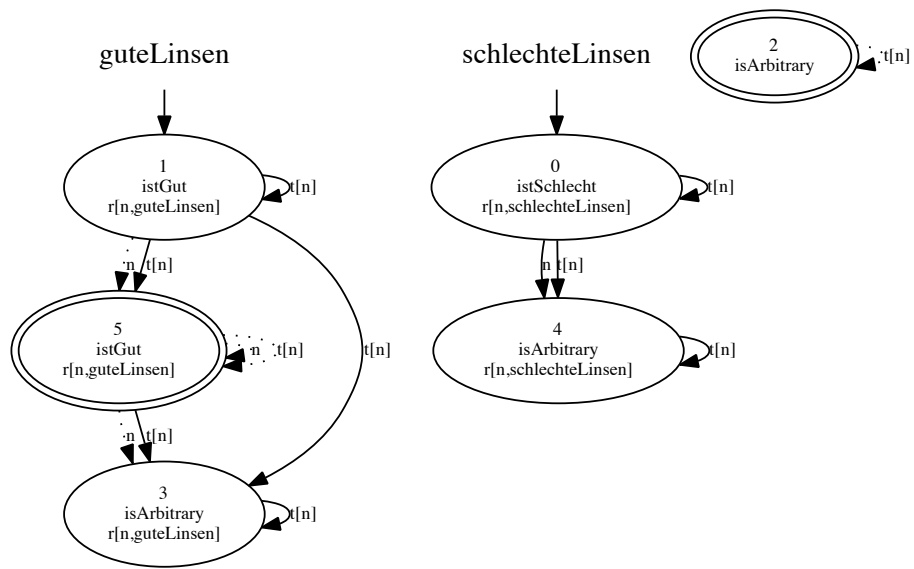


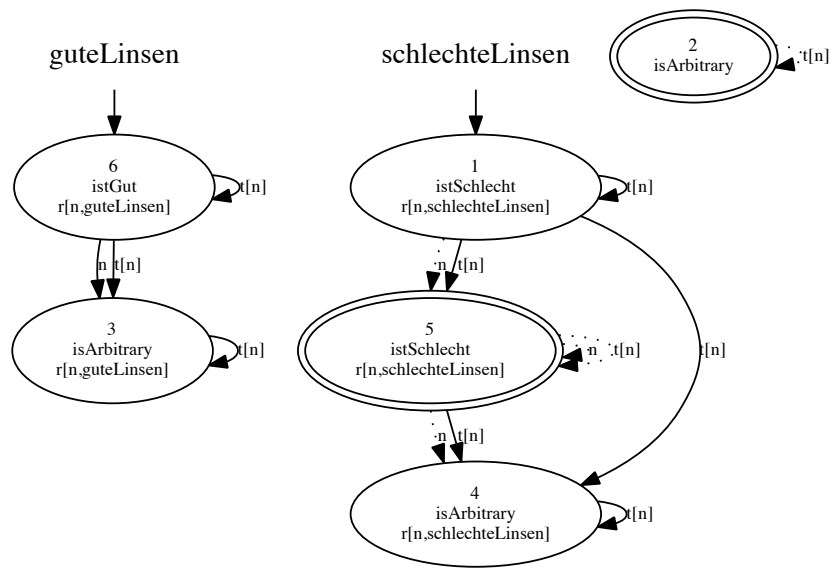






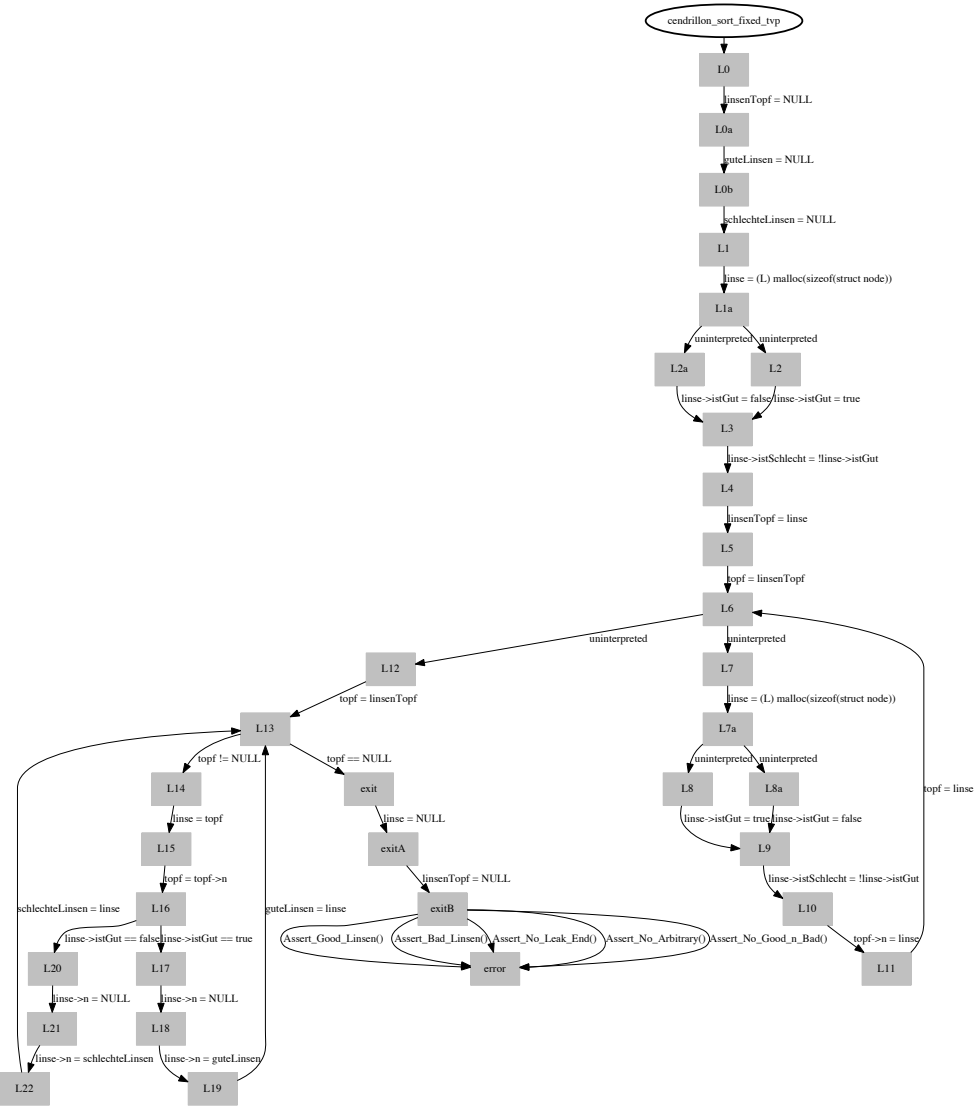




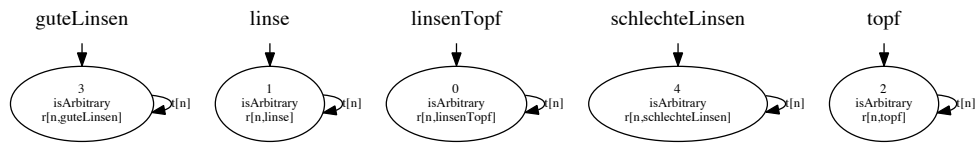


Program Location error

6.3 Analysebericht Meilenstein B (verbessert)

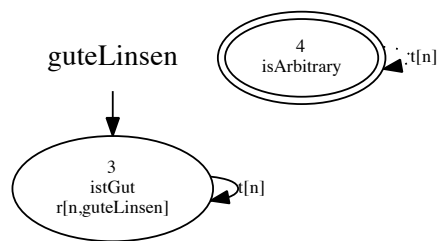


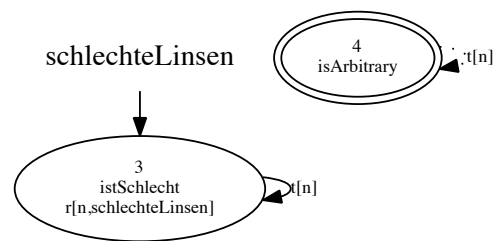
Program Location L0

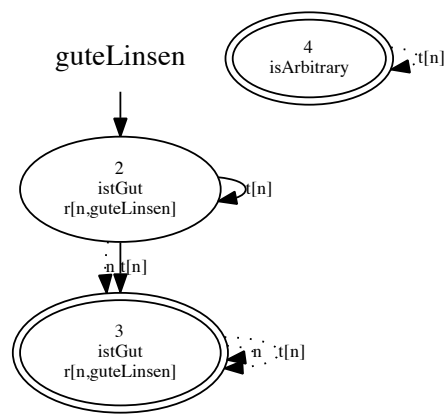


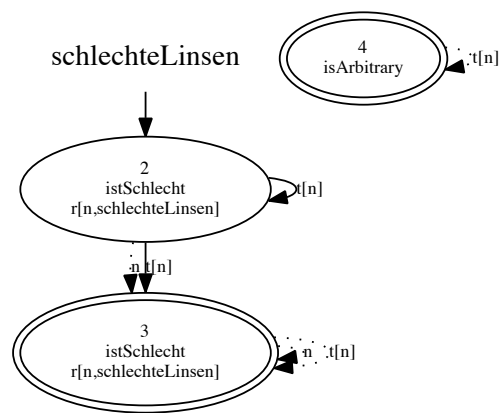
Program Location

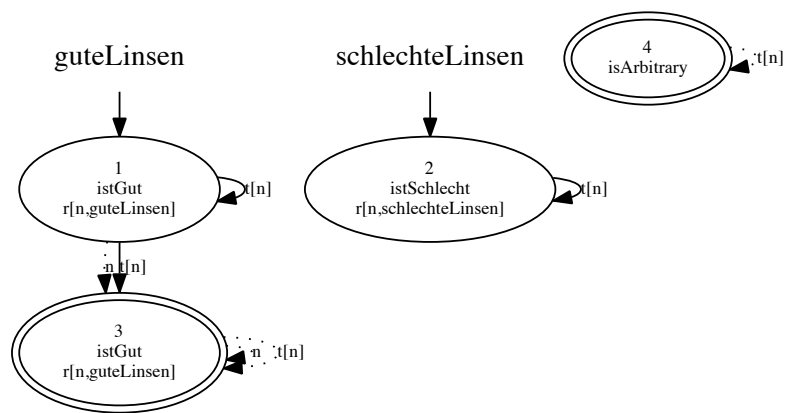
exitB

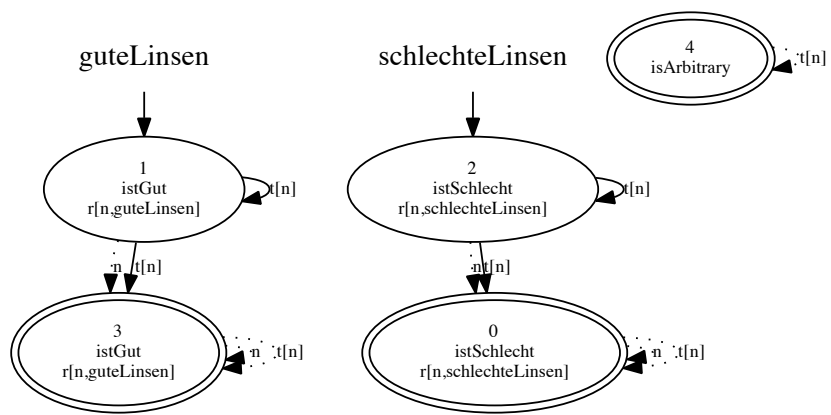


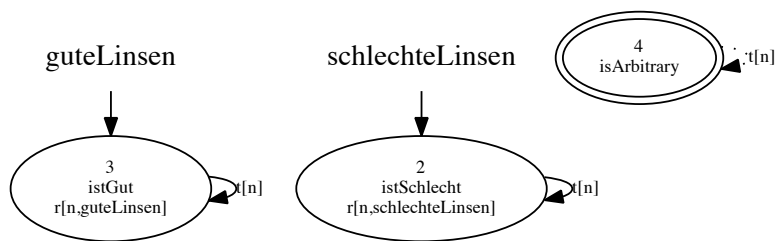


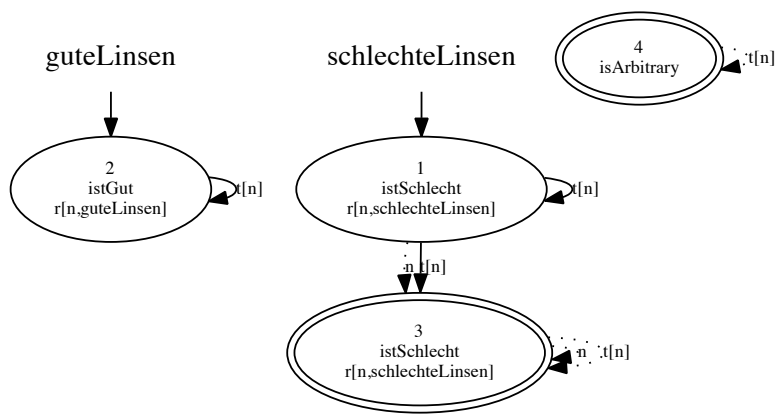






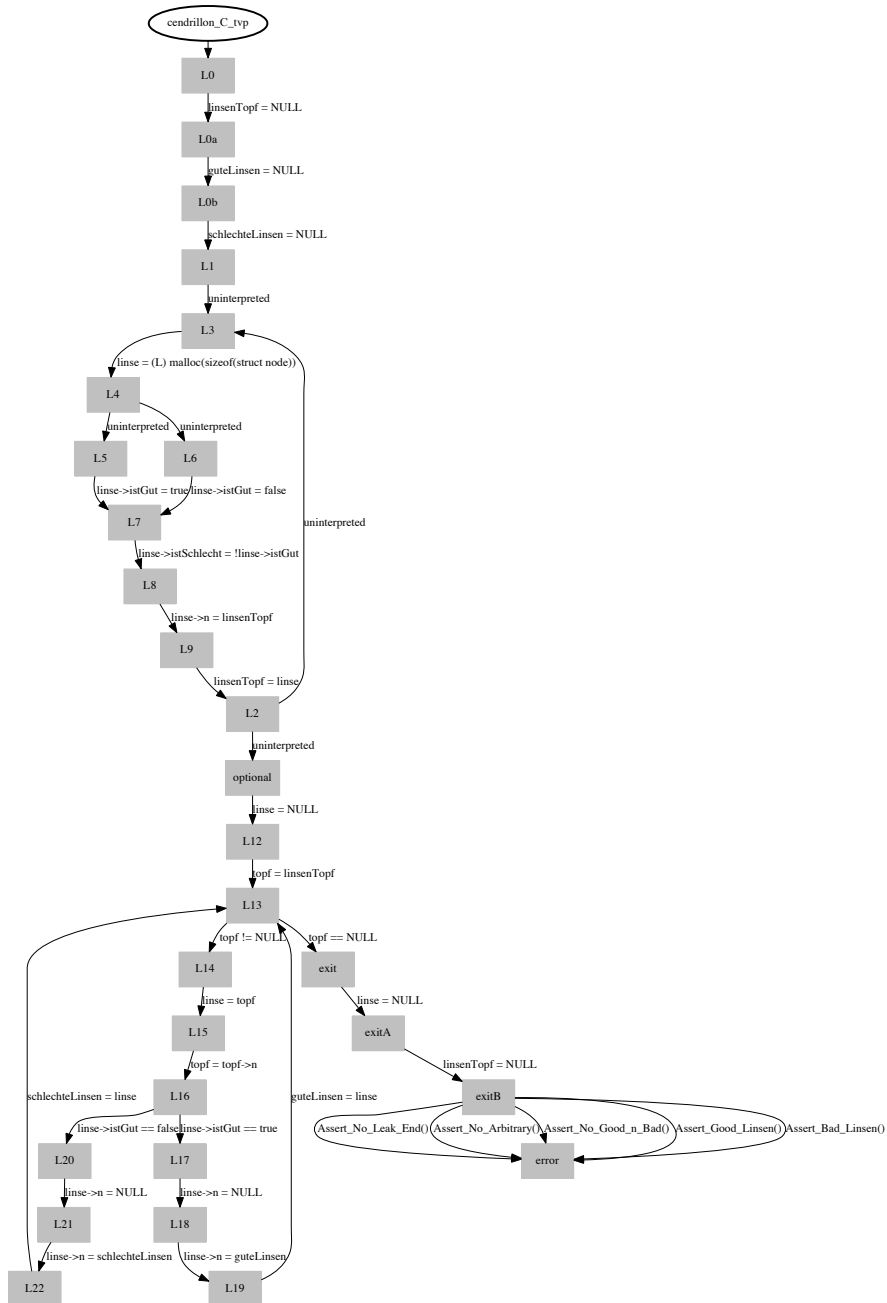




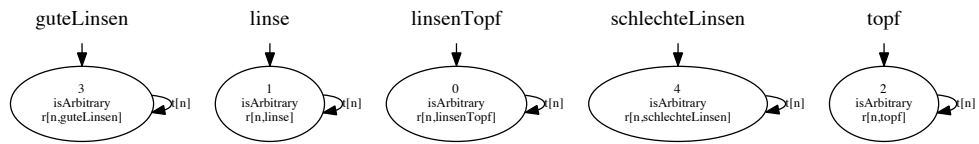


Program Location error

6.4 Analysebericht Meilenstein C

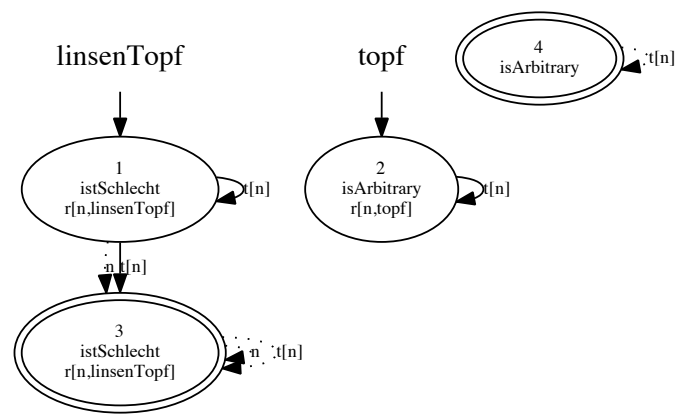


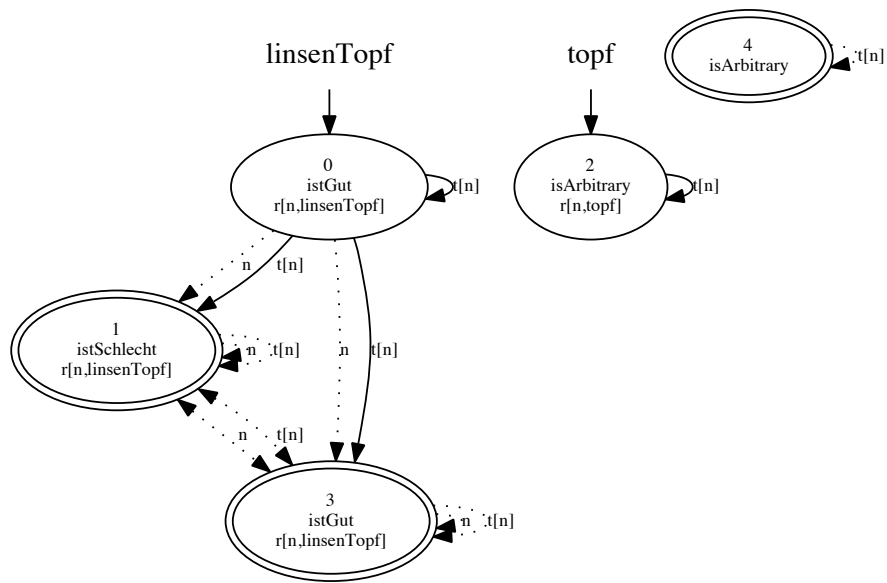
Program Location L0

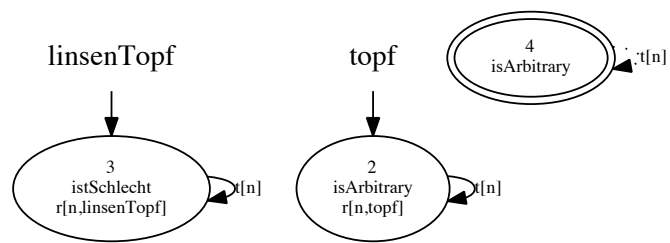


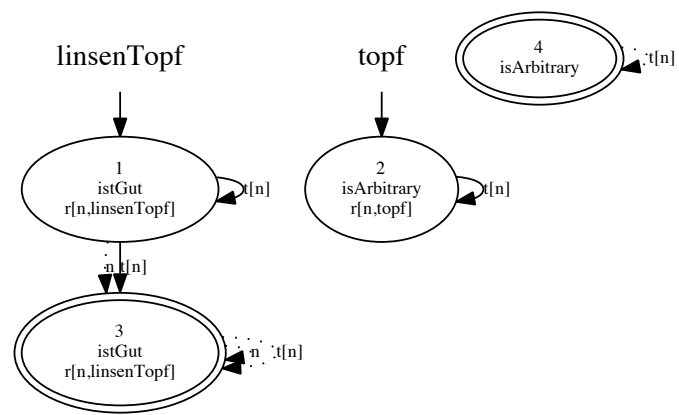
Program Location

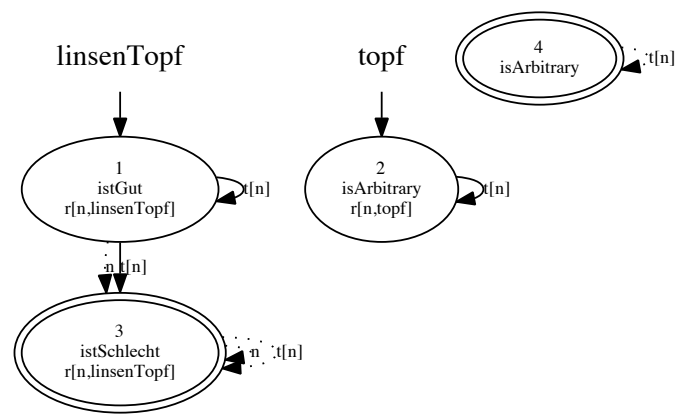
L12

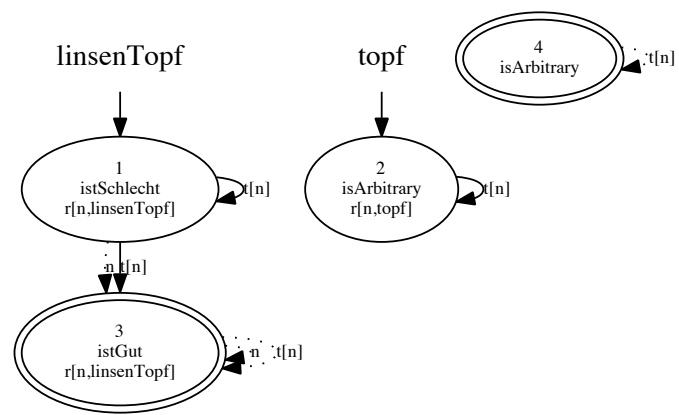


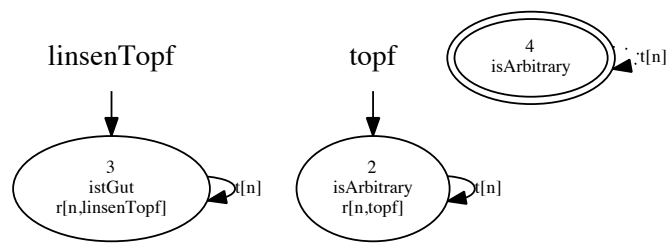


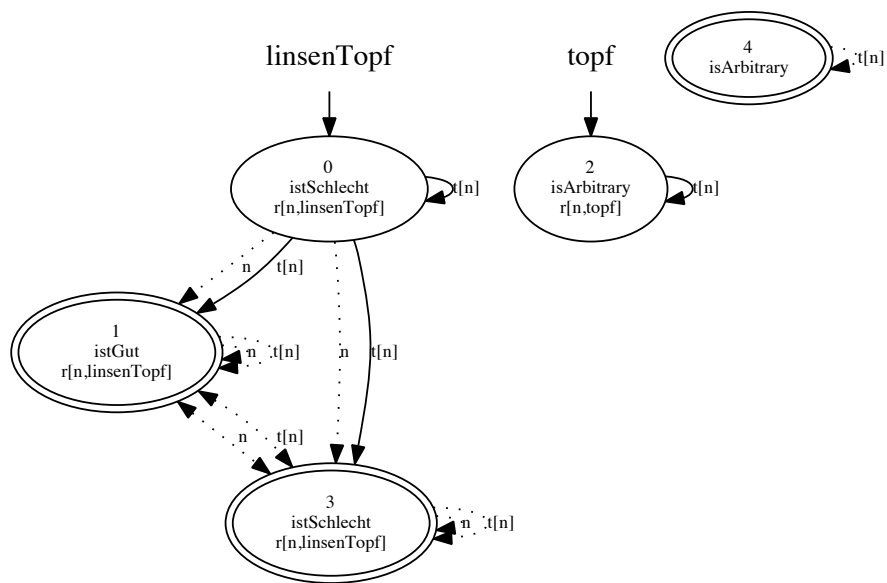




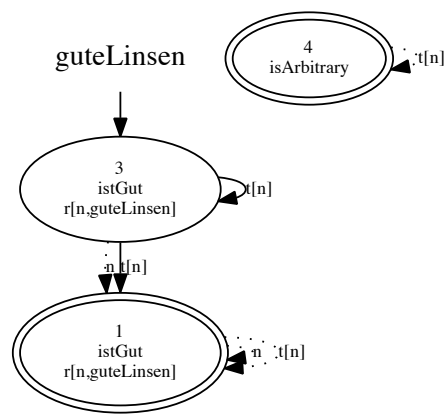


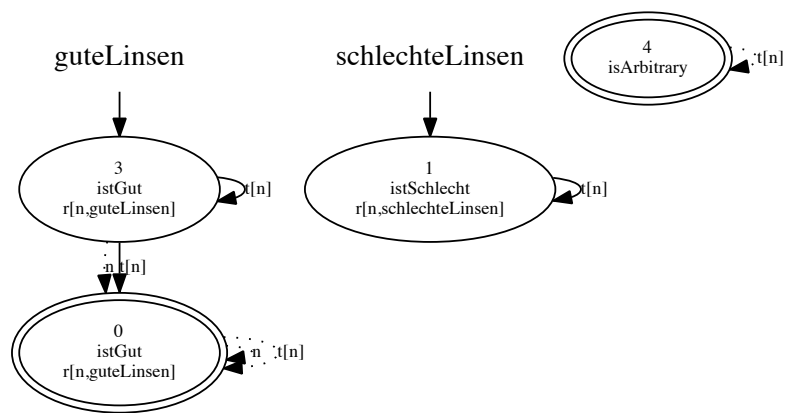


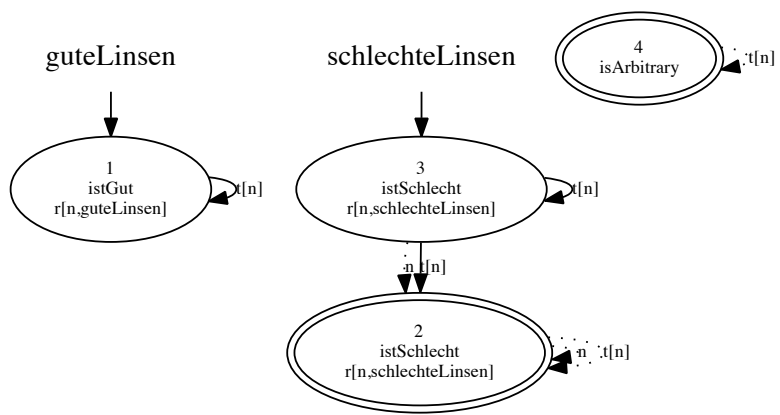


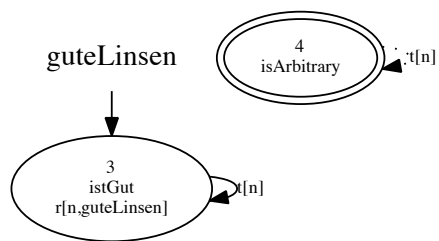


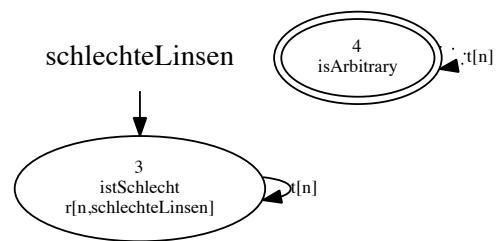
Program Location
exitB

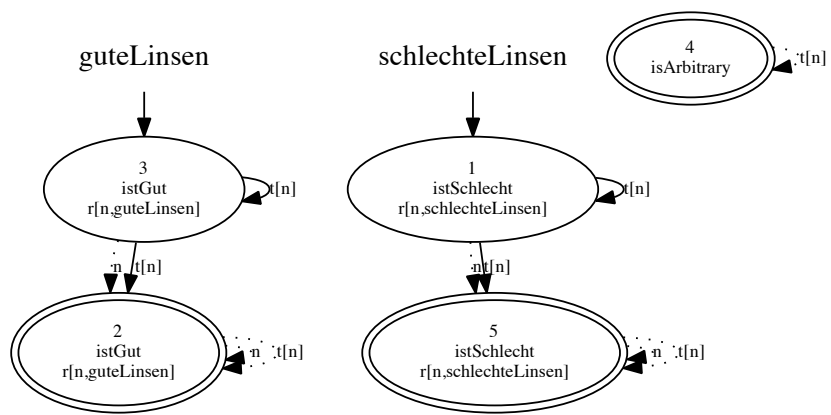


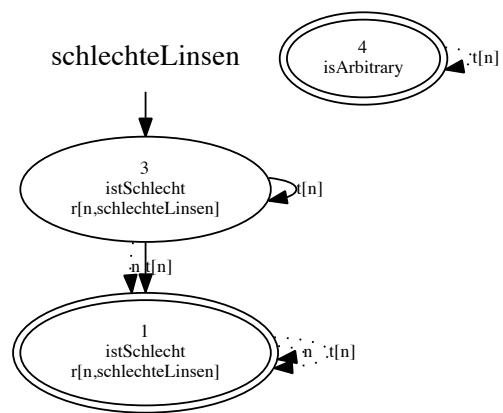


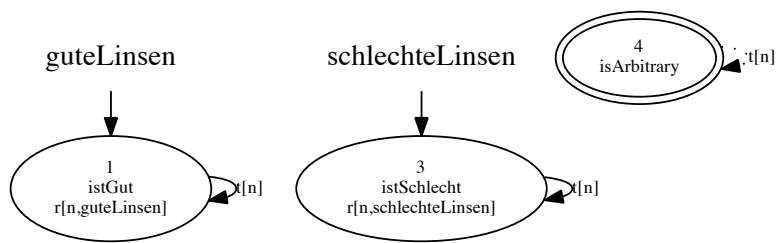












Program Location error