Domänenspezifische Sprachen in Forschung und Lehre

Prof. Dr. Thomas Baar

11. Dezember 2019

Hochschule für Technik und Wirtschaft Berlin
Fachbereich 1 - Energie und Information

- Vortragsreihe WS 19/20 der Research School am FB 1 -

Hochschule für Technik und Wirtschaft Berlin
University of Applied Sciences

Outline

DSLs in der Lehre

Ausflug in die (Mathematische) Logik

DSLs in der Forschung

DSLs in der Lehre

Mission Statement

Domänenspezifische Sprachen (Engl.: *Domain-Specific Languages (DSLs)*) können helfen, die Programmierausbildung zu entschlacken!

Mission Statement

Domänenspezifische Sprachen (Engl.: *Domain-Specific Languages (DSLs)*) können helfen, die Programmierausbildung zu entschlacken!

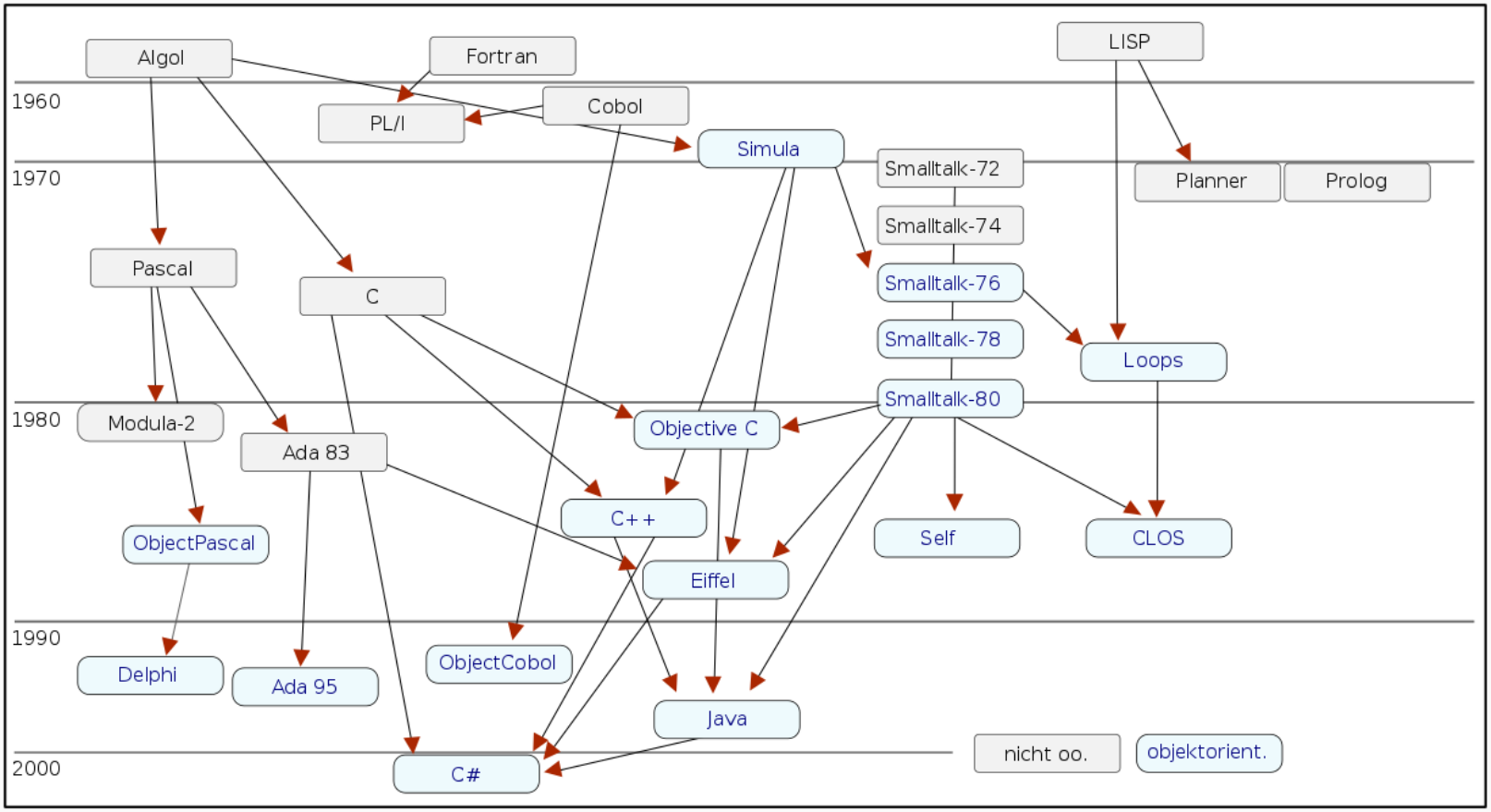
- Was sind (herkömmliche) Programmiersprachen?

Mission Statement

Domänenspezifische Sprachen (Engl.: *Domain-Specific Languages (DSLs)*) können helfen, die Programmierausbildung zu entschlacken!

- Was sind (herkömmliche) Programmiersprachen?
- Was sind DSLs?

Geschichte der Programmiersprachen



Quelle: Nepomuk Frädrich - `http://de.wikipedia.org`

Programmieren in Java

DemoTalk.java

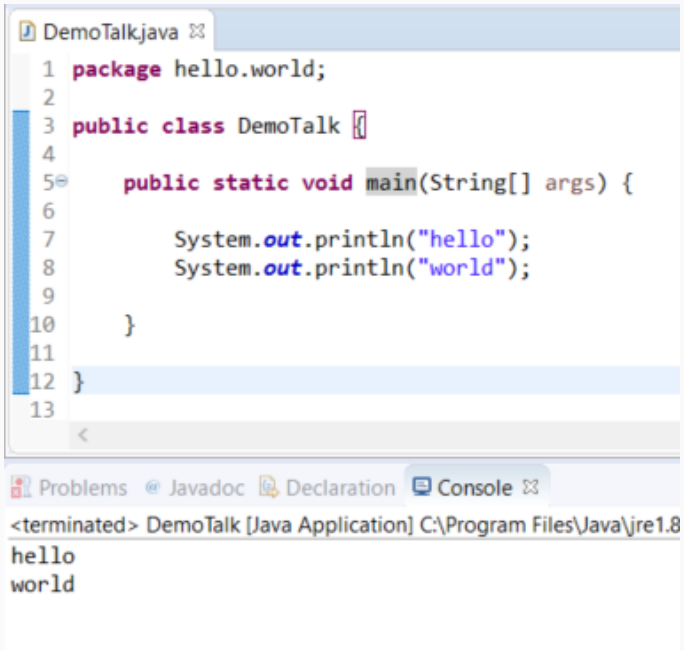
```
1  package hello.world;
2
3  public class DemoTalk {
4
5      public static void main(String[] args) {
6
7          System.out.println("hello");
8          System.out.println("world");
9
10     }
11
12 }
13
```

Problems Javadoc Declaration Console

<terminated> DemoTalk [Java Application] C:\Program Files\Java\jre1.8

```
hello
world
```

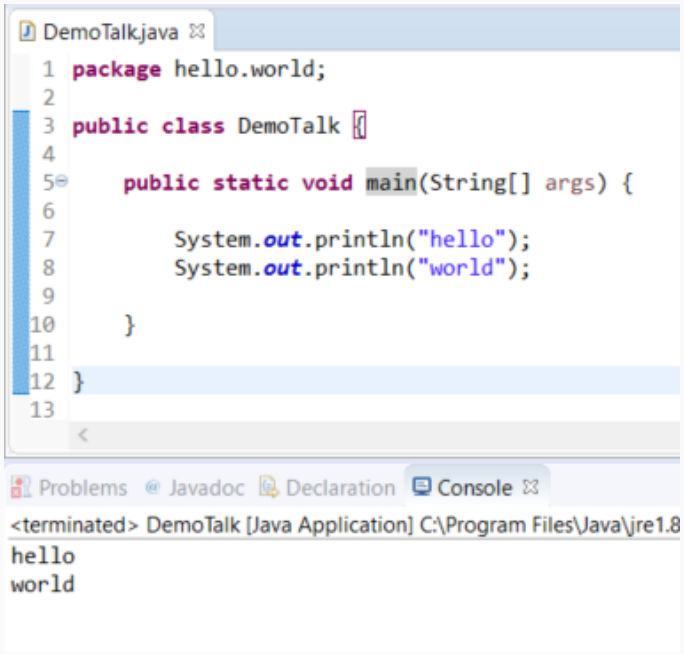
Programmieren in Java



Vorteile:

- industriell eingesetzte Sprache
- große Bibliothek
- IDE wie in der Praxis

Programmieren in Java



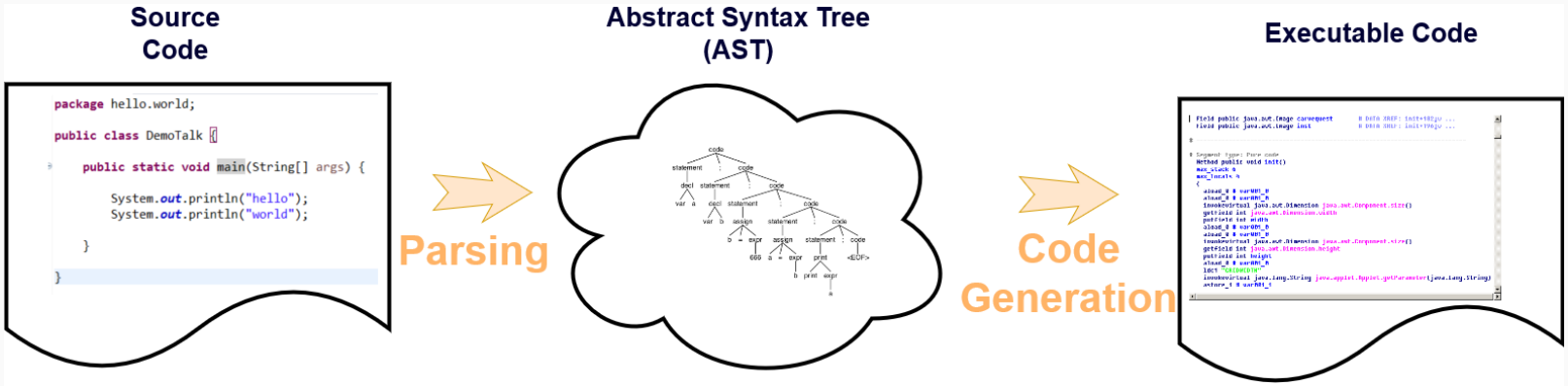
Vorteile:

- industriell eingesetzte Sprache
- große Bibliothek
- IDE wie in der Praxis

Nachteile:

- vielen Klammern
- Overhead von Klassendefinitionen

Prozess Programmiersprachen



Quelle: Eigene Darstellung

LV Algorithmen, Datenstrukturen und Komplexität

Ziele:

- Schärfung des algorithmischen Denkens
- Einführung in grundlegende Datenstrukturen
- Laufzeitanalysen für Algorithmen

Ziele:

- Schärfung des algorithmischen Denkens
- Einführung in grundlegende Datenstrukturen
- Laufzeitanalysen für Algorithmen

Braucht man für diese Ziele den Overhead von C oder Java?

DSL `pseudo2`

- Entwickelt an der HTW Berlin
- Angelegt an Python
- Kann über Webbrowser benutzt werden:
  `http://141.45.91.69/#/`

```
1  print "hello"
2  print "world"
```

Beispiel 1: Sortierung von Skat-Karten

Algorithmische Idee (Insertion Sort)**:**

Für $i = 2$ bis *AnzahlDerKarten* führe aus:

1. Nimm Karte an Stelle i
2. Füge Karte an richtige Stelle im Bereich $1 \ldots i$ ein

Beispiel 1: Sortierung von Skat-Karten

Algorithmische Idee (Insertion Sort)**:**

Für $i = 2$ bis *AnzahlDerKarten* führe aus:

1. Nimm Karte an Stelle i
2. Füge Karte an richtige Stelle im Bereich $1 \ldots i$ ein

Danach sind alle Karten sortiert :-)

Beispiel 1: Sortierung von Skat-Karten

```
1
2  var Karten=[4,2,7,5,1]
3
4  print "Unsortiert: " + Karten
5
6  insertionSort(Karten)
7
8  print "Sortiert: " + Karten
9
10
11 func insertionSort( A[1..n] )
12     for i = 2 to n
13         var x = A[i] // take ith element
14         var j = i-1
15         while j>0 && A[j]>x
16             A[j+1] = A[j]
17             j = j-1
18         A[j+1]=x
19
```

Beispiel 2: Addition zweier Zahlen mit +1 und -1

Problemstellung

Gegeben: Zwei Kartenstapel A und B

Gesucht: Wie viele Karten hat der übereinandergelegte Kartenstapel?

Beispiel 2: Addition zweier Zahlen mit +1 und -1

Problemstellung

Gegeben: Zwei Kartenstapel A und B

Gesucht: Wie viele Karten hat der übereinandergelegte Kartenstapel?

Algorithmische Idee:

1. Nimm eine Karte von B und lege sie auf A (erhöhe die Anzahl der Karten von A um 1)
2. Wiederhole 1. Schritt bis Stapel B leer ist

Beispiel 2: Addition zweier Zahlen mit +1 und -1

Implementierung in `pseudo2`

```
1  var a_orig = 5 // choose integer value
2  var b_orig = 7 // choose integer value
3
4  if b_orig < 0 || (b_orig mod 1) != 0
5      throw("b_orig must be non-negative integer")
6
7  var a = a_orig
8  var b = b_orig
9
10 // add a and b by counting up/down
11 while b != 0
12     a = a + 1
13     b = b - 1
14
15 print("The sum of " + a_orig
16 + " and " + b_orig + " is " + a)
17
```

Beispiel 2: Addition zweier Zahlen mit +1 und -1

Implementierung in `pseudo2`

```
1  var a_orig = 5 // choose integer value
2  var b_orig = 7 // choose integer value
3
4  if b_orig < 0 || (b_orig mod 1) != 0
5      throw("b_orig must be non-negative integer")
6
7  var a = a_orig
8  var b = b_orig
9
10 // add a and b by counting up/down
11 while b != 0
12     a = a + 1
13     b = b - 1
14
15 print("The sum of " + a_orig
16 + " and " + b_orig + " is " + a)
17
```

Warum steht am Ende in Variable `a` immer die Summe von `a_orig` und `b_orig`?

Beispiel 2: Addition zweier Zahlen mit +1 und -1

Implementierung in `pseudo2`

```
1  var a_orig = 5 // choose integer value
2  var b_orig = 7 // choose integer value
3
4  if b_orig < 0 || (b_orig mod 1) != 0
5      throw("b_orig must be non-negative integer")
6
7  var a = a_orig
8  var b = b_orig
9
10 // add a and b by counting up/down
11 while b != 0
12     a = a + 1
13     b = b - 1
14
15 print("The sum of " + a_orig
16 + " and " + b_orig + " is " + a)
17
```

Warum steht am Ende in Variable `a` immer die Summe von `a_orig` und `b_orig`?

Korrektheit des Algorithmus ?!

Ausflug in die (Mathematische) Logik

Homework

A sheik shows Ali three leather bags. One of them is filled with gold, the other two with stones. If Ali finds out, which one is filled with gold, this bag will become his. The bags are labeled with statements as follows:

- Bag 1: *The gold is not in Bag 3.*
- Bag 2: *The gold is not in this bag.*
- Bag 3: *The gold is in this bag.*

Example: Three bags, one with gold

Homework

A sheik shows Ali three leather bags. One of them is filled with gold, the other two with stones. If Ali finds out, which one is filled with gold, this bag will become his. The bags are labeled with statements as follows:

- Bag 1: *The gold is not in Bag 3.*
- Bag 2: *The gold is not in this bag.*
- Bag 3: *The gold is in this bag.*

- In welchem Sack ist das Gold, wenn alle Aussagen richtig sind?

Formalization and Proof

Example: Three bags, one with gold

Homework

A sheik shows Ali three leather bags. One of them is filled with gold, the other two with stones. If Ali finds out, which one is filled with gold, this bag will become his. The bags are labeled with statements as follows:

- Bag 1: *The gold is not in Bag 3.*
- Bag 2: *The gold is not in this bag.*
- Bag 3: *The gold is in this bag.*

- In welchem Sack ist das Gold, wenn alle Aussagen richtig sind?
- In welchem Sack ist das Gold, wenn genau eine Aussage falsch ist?

Homework

A sheik shows Ali three leather bags. One of them is filled with gold, the other two with stones. If Ali finds out, which one is filled with gold, this bag will become his. The bags are labeled with statements as follows:

- Bag 1: *The gold is not in Bag 3.*
- Bag 2: *The gold is not in this bag.*
- Bag 3: *The gold is in this bag.*

- In welchem Sack ist das Gold, wenn alle Aussagen richtig sind?
- In welchem Sack ist das Gold, wenn genau eine Aussage falsch ist?
- In welchem Sack ist das Gold, wenn genau zwei Aussagen falsch sind?

Example: Three bags, one with gold

Homework

A sheik shows Ali three leather bags. One of them is filled with gold, the other two with stones. If Ali finds out, which one is filled with gold, this bag will become his. The bags are labeled with statements as follows:

- Bag 1: *The gold is not in Bag 3.*
- Bag 2: *The gold is not in this bag.*
- Bag 3: *The gold is in this bag.*

- In welchem Sack ist das Gold, wenn alle Aussagen richtig sind?
- In welchem Sack ist das Gold, wenn genau eine Aussage falsch ist?
- In welchem Sack ist das Gold, wenn genau zwei Aussagen falsch sind?
- In welchem Sack ist das Gold, wenn alle Aussagen falsch sind?

Rettung: Theorembeweiser

Formalisierung der Aussagen

Theorembeweiser sind Programme, die logische Schlussfolgerungen korrekt und präzise treffen.

Sie benötigen jedoch eine formalisierte Eingabe:

Rettung: Theorembeweiser

Formalisierung der Aussagen

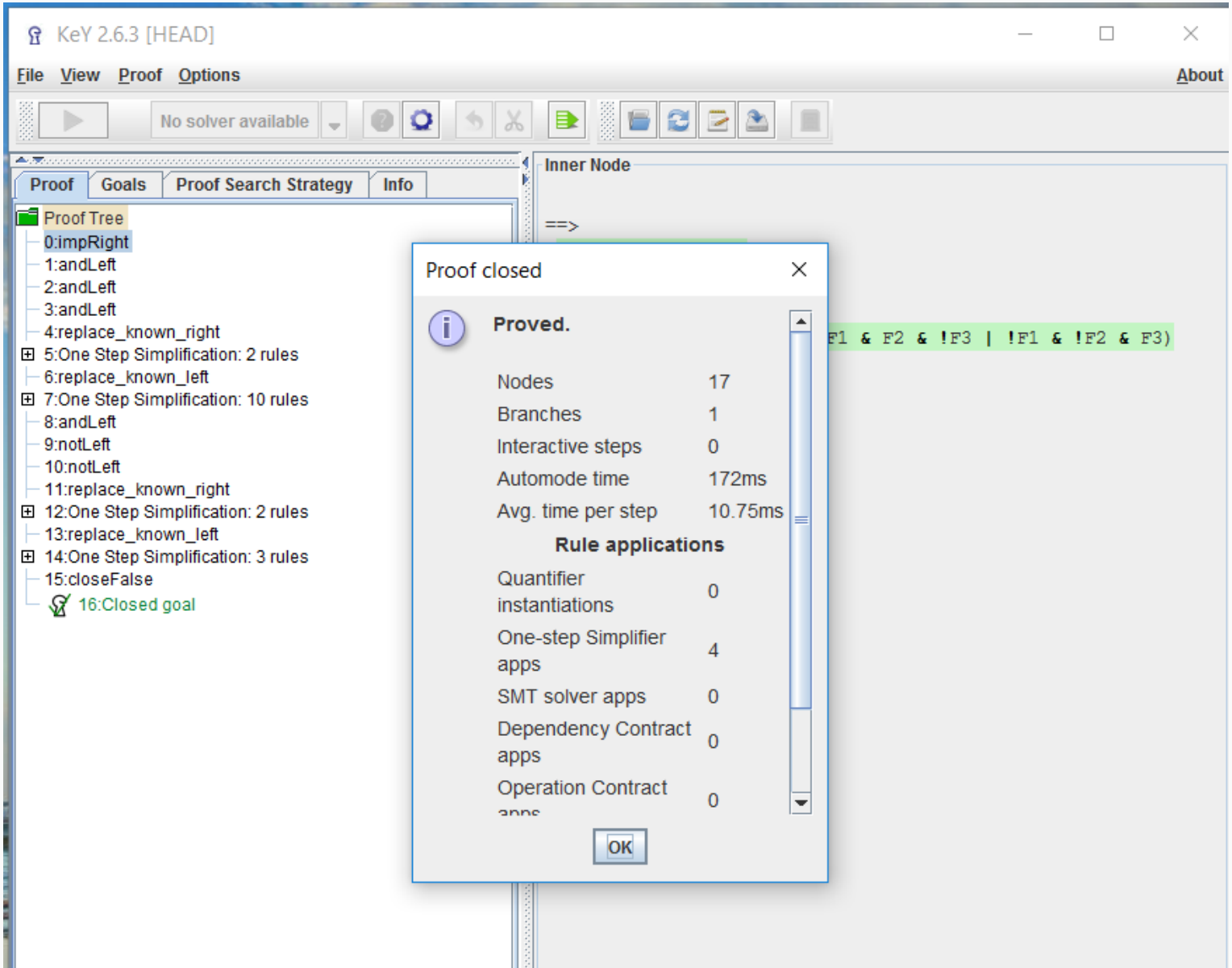
Theorembeweiser sind Programme, die logische Schlussfolgerungen korrekt und präzise treffen.

Sie benötigen jedoch eine formalisierte Eingabe:

```
\predicates{
        B1; B2; B3;  // gold is in bag 1/2/3
        F1; F2; F3;  // auxiliary variables
}
\problem{
        (F1 <-> ! B3) &
        (F2 <-> ! B2) &
        (F3 <-> B3) &
        (( F1 & !F2 & !F3) | // exactly one Fi is true
         (!F1 &  F2 & !F3) |
         (!F1 & !F2 &  F3)
        )
-> B2   // in Bag2 must be the gold
}
```

Rettung: Theorembeweiser

Automatischer Beweis



Beispiel 2: Addition zweier Zahlen mit +1 und -1

Implementierung in `pseudo2`

```
1  var a_orig = 5 // choose integer value
2  var b_orig = 7 // choose integer value
3
4  if b_orig < 0 || (b_orig mod 1) != 0
5      throw("b_orig must be non-negative integer")
6
7  var a = a_orig
8  var b = b_orig
9
10 // add a and b by counting up/down
11 while b != 0
12     a = a + 1
13     b = b - 1
14
15 print("The sum of " + a_orig
16 + " and " + b_orig + " is " + a)
17
```

Warum steht am Ende in Variable `a` immer die Summe von `a_orig` und `b_orig`?

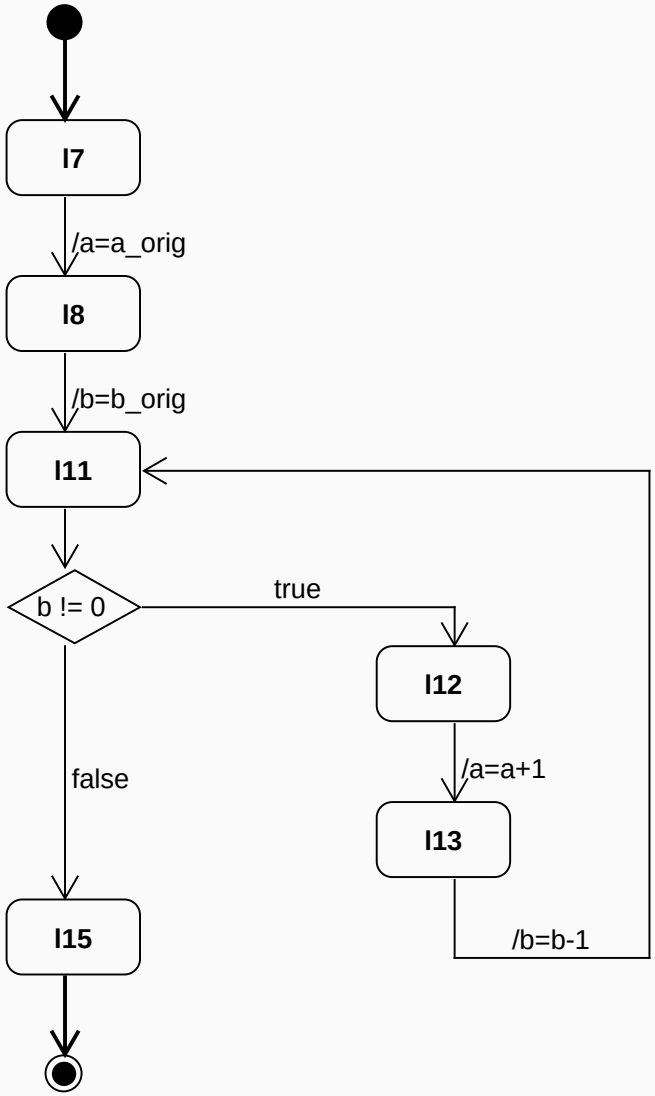
Korrektheit des Algorithmus ?!

Programm als Kontrollflussgraph (CFG)

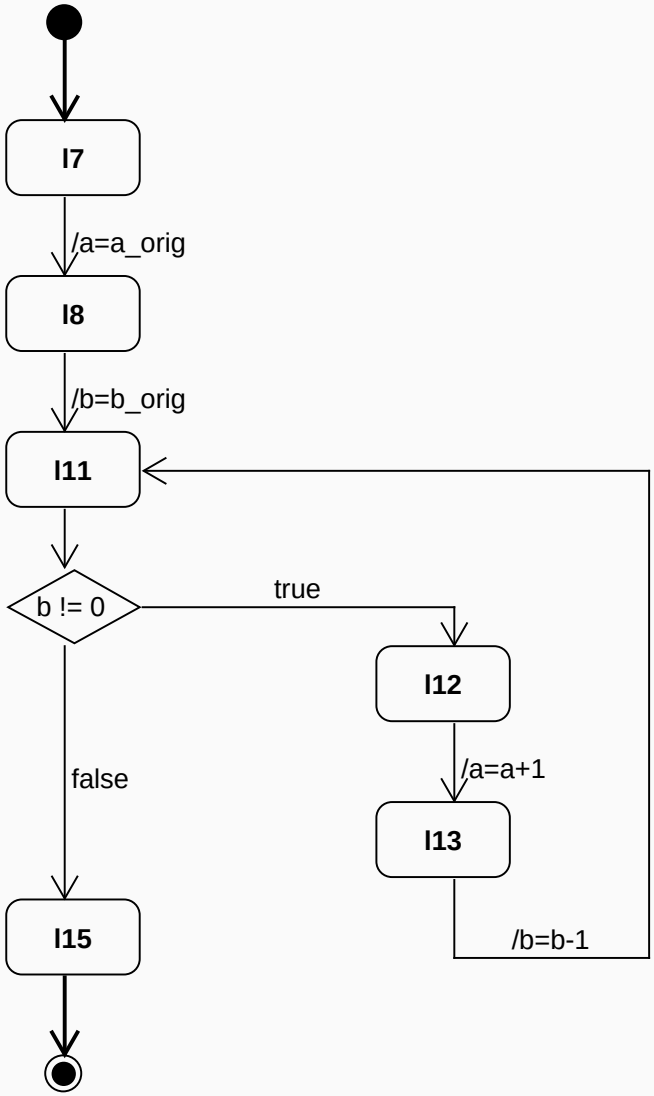
```
1  var a_orig = 5 // choose integer value
2  var b_orig = 7 // choose integer value
3
4  if b_orig < 0 || (b_orig mod 1) != 0
5      throw("b_orig must be non-negative integer")
6
7  var a = a_orig
8  var b = b_orig
9
10 // add a and b by counting up/down
11 while b != 0
12     a = a + 1
13     b = b - 1
14
15 print("The sum of " + a_orig
16 + " and " + b_orig + " is " + a)
17
```

Programm als Kontrollflussgraph (CFG)

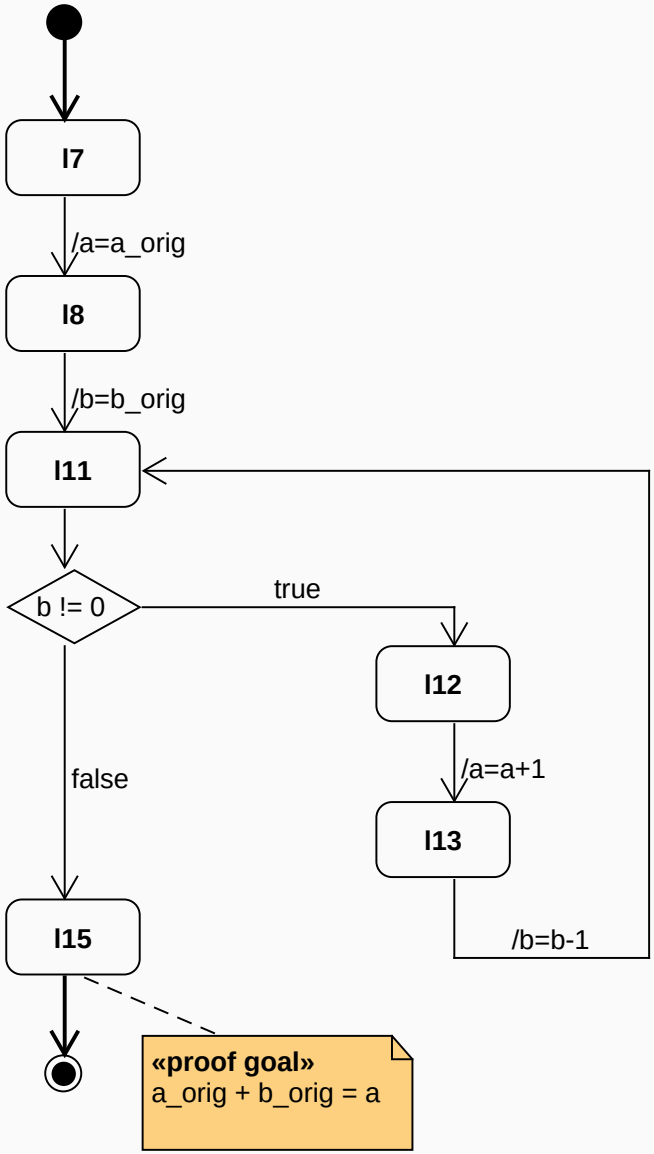
```
1  var a_orig = 5 // choose integer value
2  var b_orig = 7 // choose integer value
3
4  if b_orig < 0 || (b_orig mod 1) != 0
5      throw("b_orig must be non-negative integer")
6
7  var a = a_orig
8  var b = b_orig
9
10 // add a and b by counting up/down
11 while b != 0
12     a = a + 1
13     b = b - 1
14
15 print("The sum of " + a_orig
16 + " and " + b_orig + " is " + a)
17
```



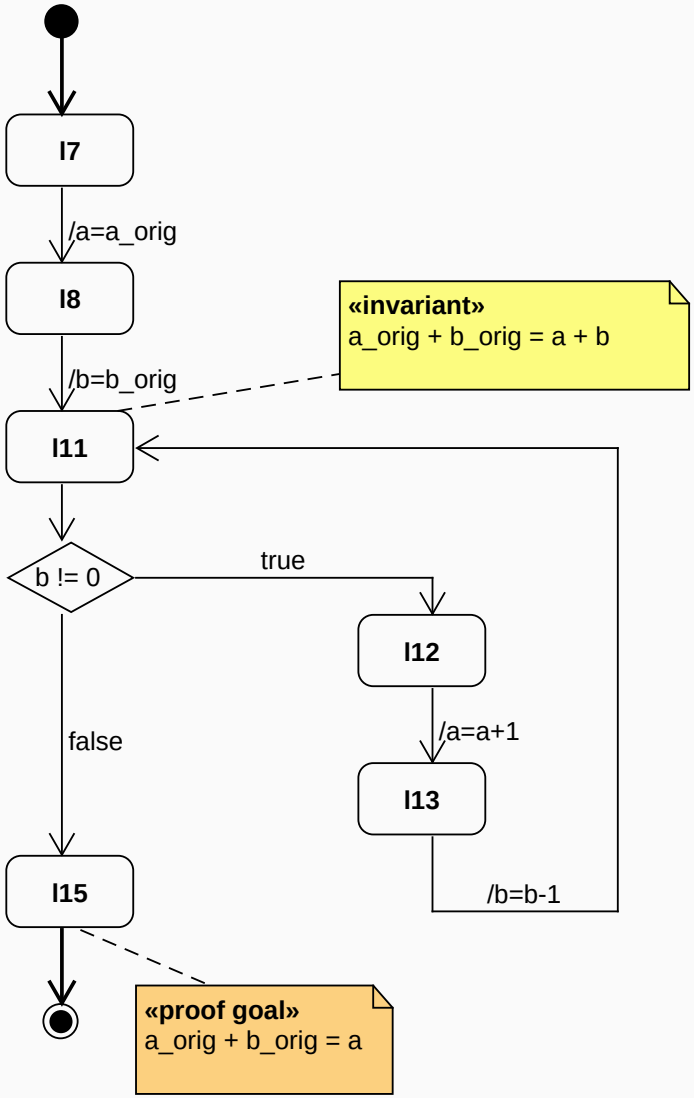
Analyse Kontrollflussgraph (CFG)



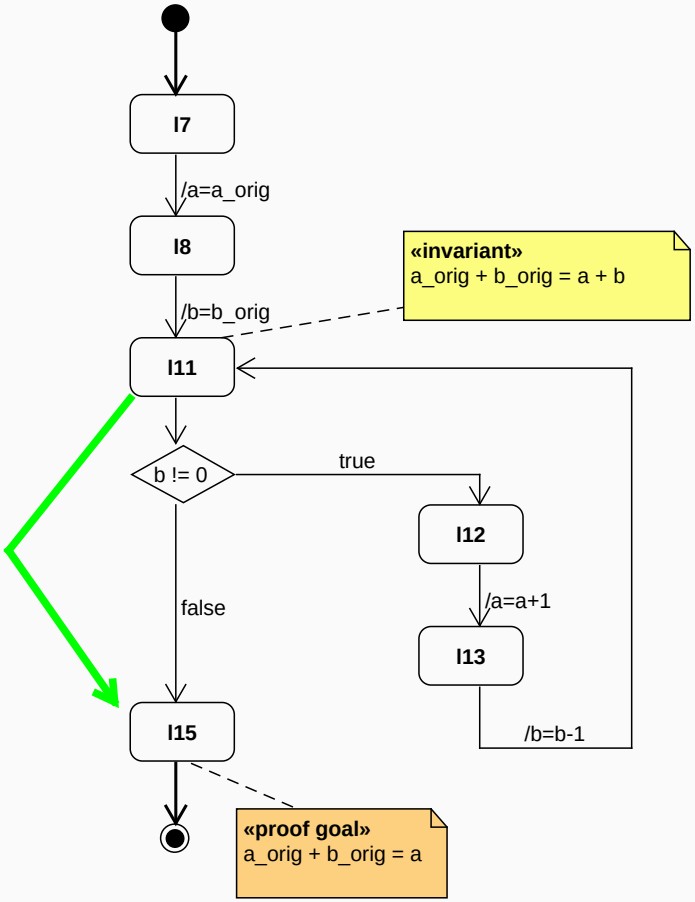
Analyse Kontrollflussgraph (CFG)



Analyse Kontrollflussgraph (CFG)



Analyse Kontrollflussgraph (CFG)

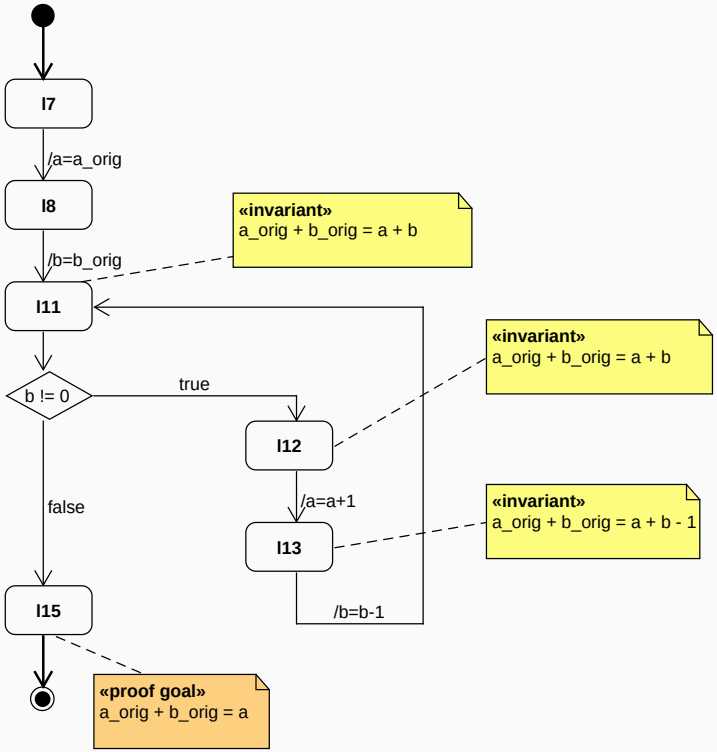


Beweisverpflichtung:

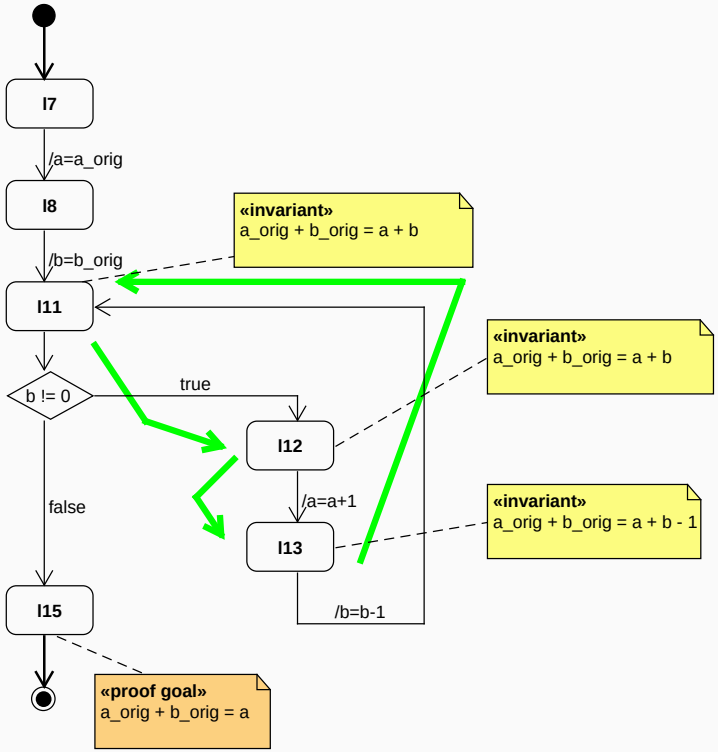
$a\_orig + b\_orig = a_{pre} + b_{pre} \wedge \neg(b_{pre} != 0) \wedge a_{post} = a_{pre} \wedge b_{post} = b_{pre}$

$\rightarrow a\_orig + b\_orig = a_{post}$

Analyse Kontrollflussgraph (CFG)



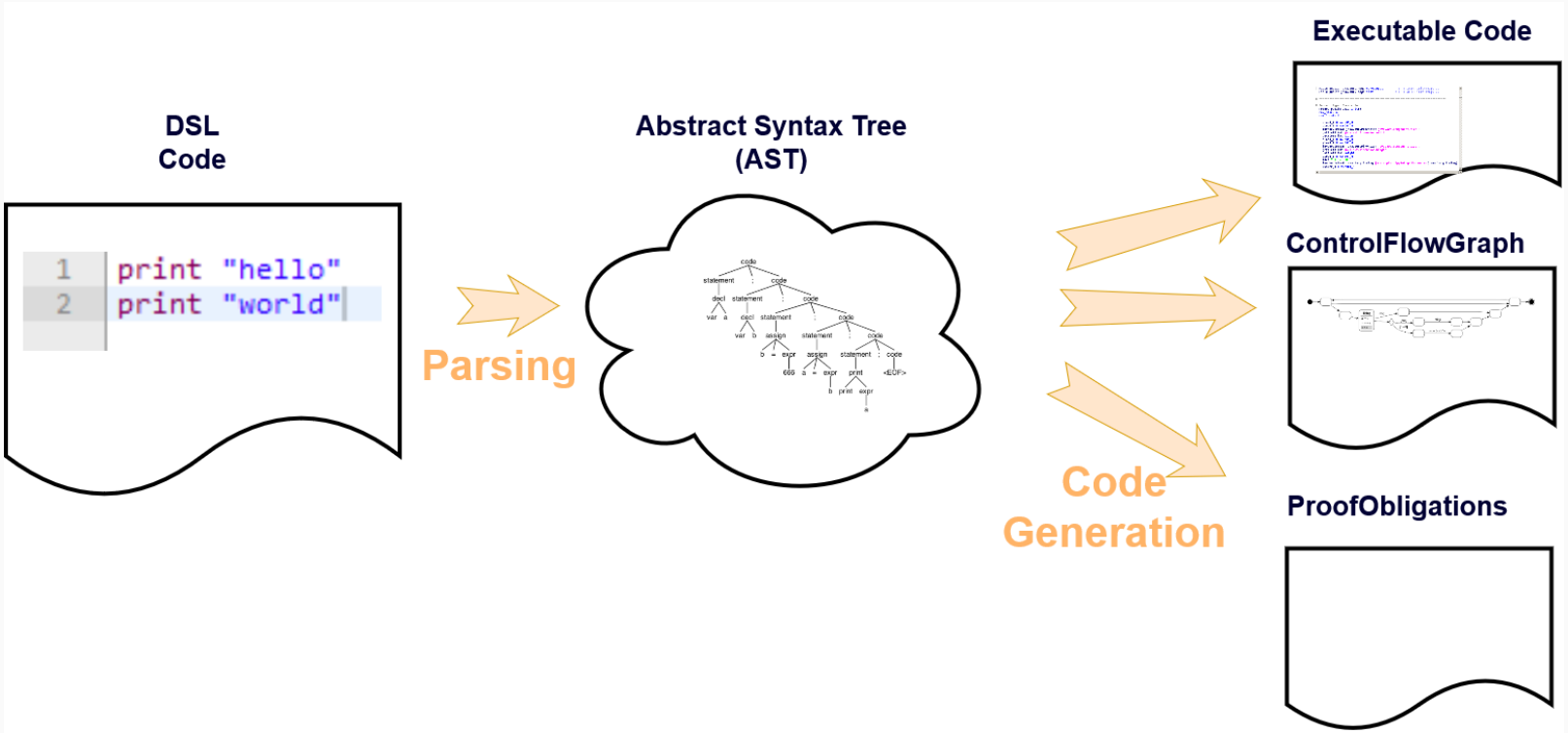
Analyse Kontrollflussgraph (CFG)



Beweisverpflichtungen:

. . .

Prozess DSLs



Quelle: Eigene Darstellung

Zusammenfassung Lehre

- Benutzen der vergleichsweise einfachen Programmiersprache `pseudo2`

Zusammenfassung Lehre

- Benutzen der vergleichsweise einfachen Programmiersprache `pseudo2`
- Automatische Überführung in CFG
  - *Denken in Zuständen*

Zusammenfassung Lehre

- Benutzen der vergleichsweise einfachen Programmiersprache `pseudo2`
- Automatische Überführung in CFG
  - *Denken in Zuständen*
- Thematisierung des Themas Korrektheit
  - *Logisches Denken in Zuständen*

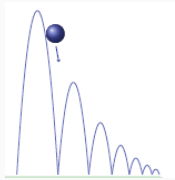
Zusammenfassung Lehre

- Benutzen der vergleichsweise einfachen Programmiersprache `pseudo2`
- Automatische Überführung in CFG
  - *Denken in Zuständen*
- Thematisierung des Themas Korrektheit
  - *Logisches Denken in Zuständen*

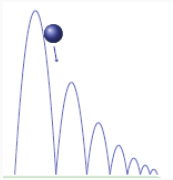
Eigene Ideen/Werkzeuge lassen sich prinzipiell umsetzen

DSLs in der Forschung

1. Beispiel: Bouncing Ball

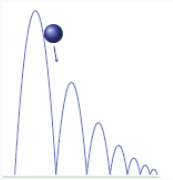


1. Beispiel: Bouncing Ball

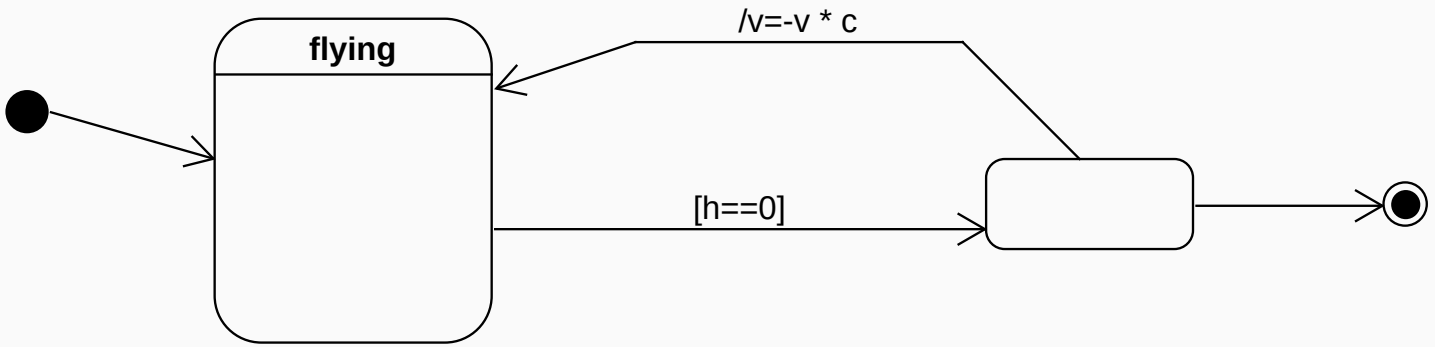


Beschreibung des Verhaltens mit Graph/Automaten:

1. Beispiel: Bouncing Ball



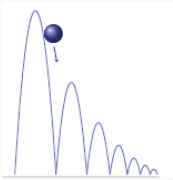
Beschreibung des Verhaltens mit Graph/Automaten:



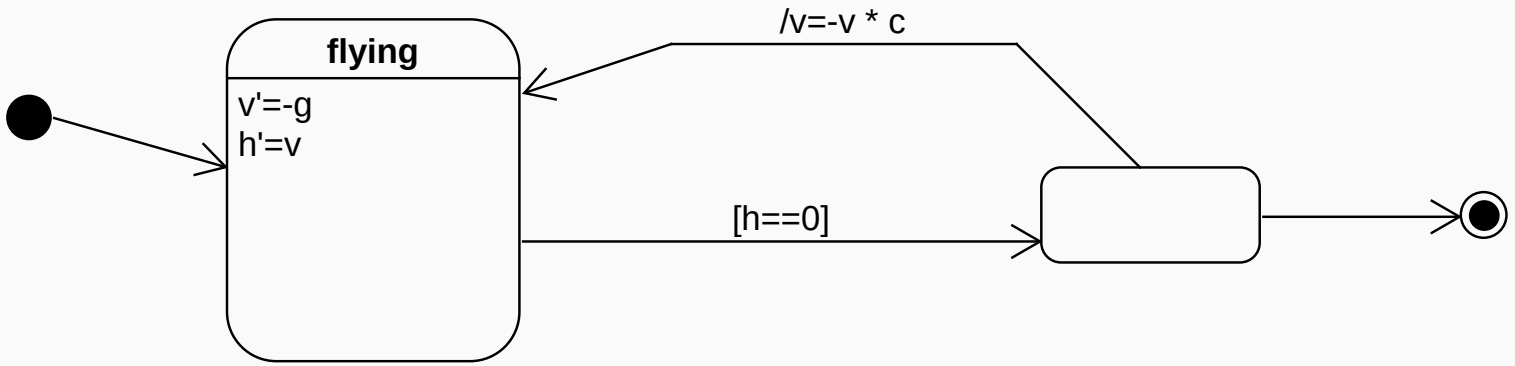
$$v(t) = v_0 + -g * t$$

$$h(t) = h_0 + \int_{j=0}^{t} v(j)$$

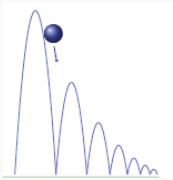
1. Beispiel: Bouncing Ball



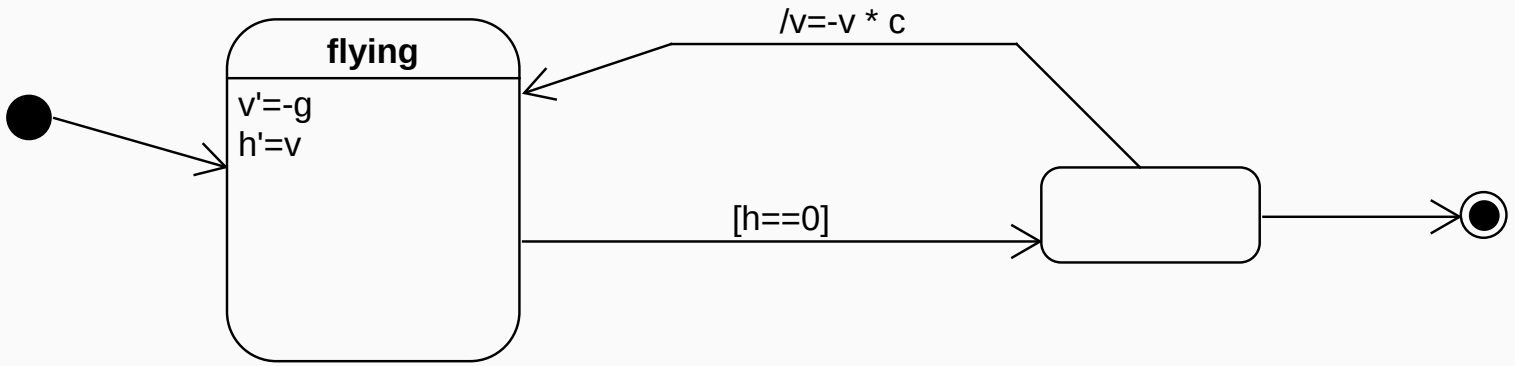
Als Hybrider Automat:



1. Beispiel: Bouncing Ball



Als Hybrider Automat:



Korrektheitseigenschaft: Es gilt immer dass $h \leq H$

1. Beispiel: Bouncing Ball

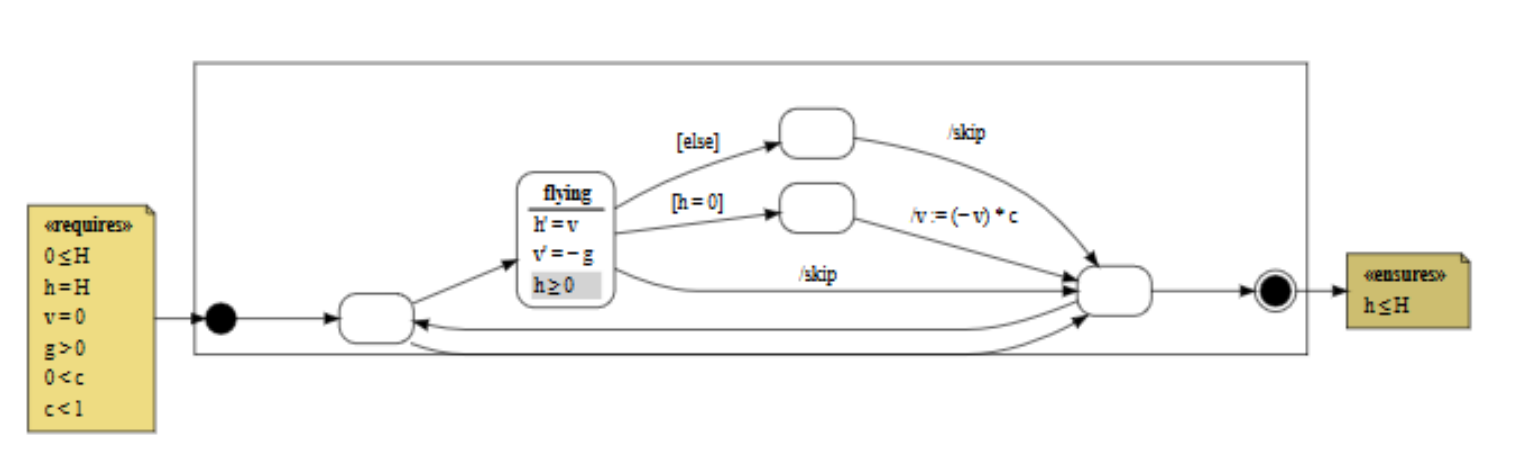
Umsetzung in DSL `vercps`

```
1  project bouncingBall
2
3  constants
4  H  // initial value of h
5  g  // gravity of earth
6  c // damping constant
7
8  evolutions
9  h  // position
10
11 evolution_assignables
12 v  // velocity
13
14 evstate flying
15     h' = v
16     v' = -g
17 domain
18     h >= 0
19
20
21
22 program
23     {flying;
24     // TODO: Not really invariant modelled here?!
25     skip ++
26     if h=0 then v:=(-v)*c else skip endif
27     }*
28 requires
29     0<= H
30     h=H
31     v=0
32     g > 0
33     0 < c
34     c < 1
35 ensures
36     h <= H
```

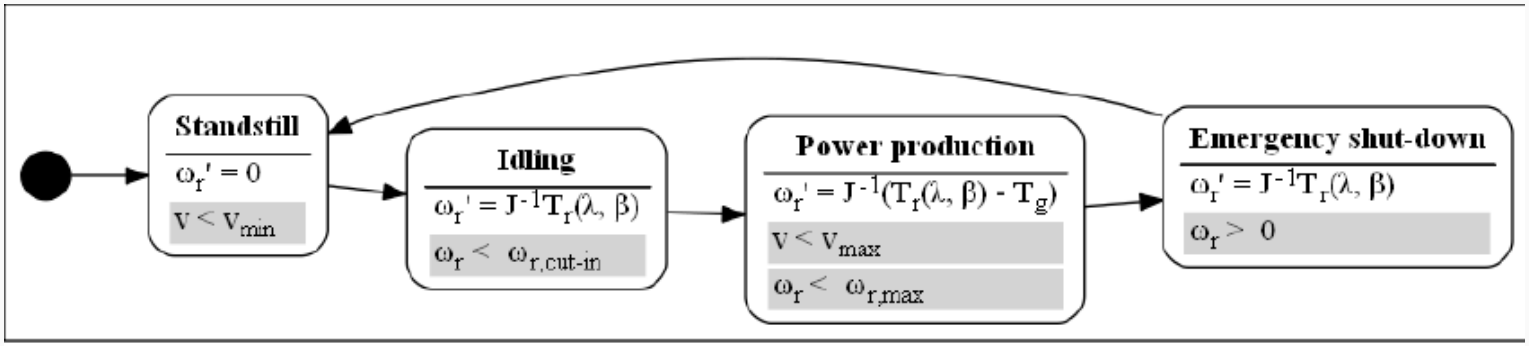
1. Beispiel: Bouncing Ball

Umsetzung in DSL `vercps`

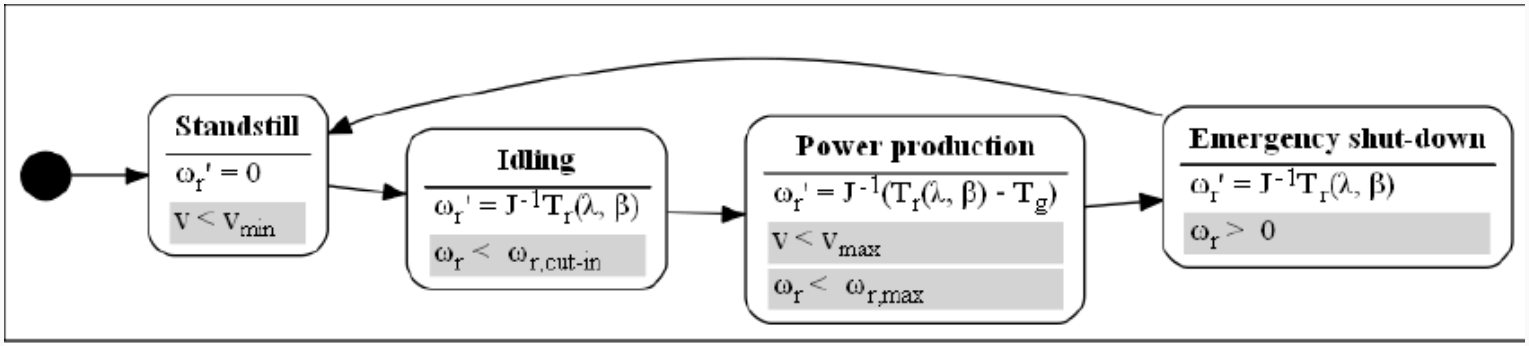
Automatische Generierung des Kontrollflussgraphen:



2. Beispiel: Steuerung von Windkraftanlagen



2. Beispiel: Steuerung von Windkraftanlagen

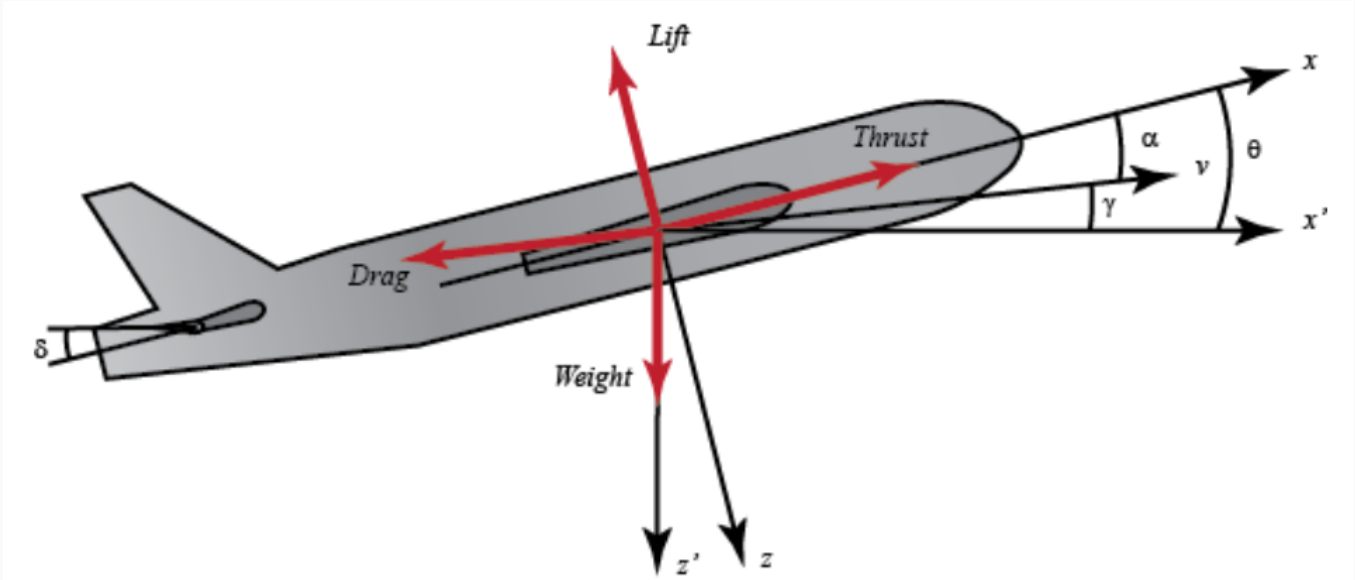


Korrektheitseigenschaft: ???

3. Beispiel: Boeing 737 MAX Abstürze

3. Beispiel: Boeing 737 MAX Abstürze

Mathematisches Modell



Source: `http://ctms.engin.umich.edu/CTMS/index.php?example=AircraftPitch`

Legende:

- α - angle of attack
- θ - pitch angle
- δ - elevator deflection angle
- γ - flight path angle

3. Beispiel: Boeing 737 MAX Abstürze

Mathematisches Modell

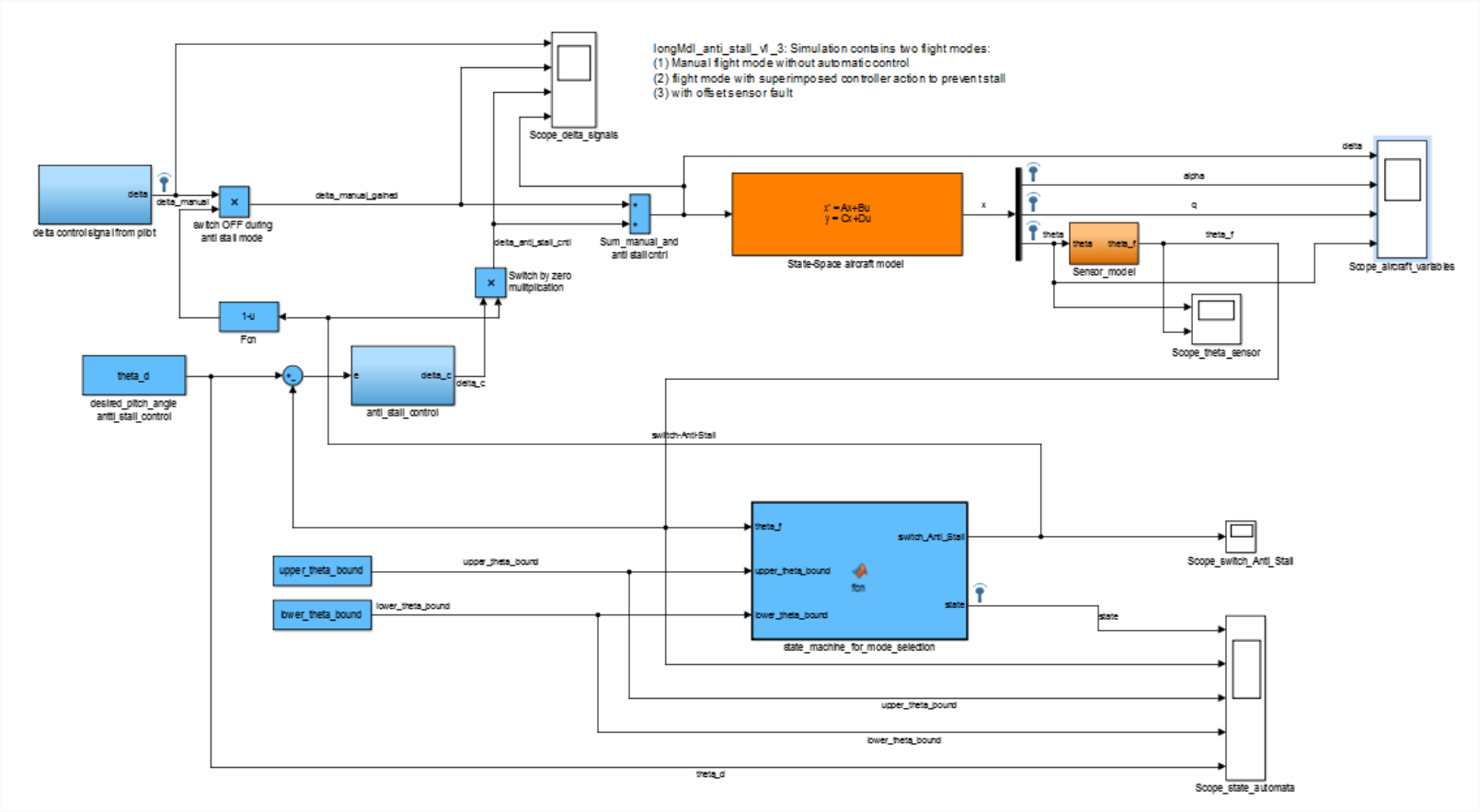
Zustandsvektor:

$$x = \begin{pmatrix} \alpha \\ q \\ \theta \end{pmatrix}$$

$$\begin{aligned} \dot{\alpha} &= -0.313\,\alpha & +56.7\,q & +0.232\,\delta \\ \dot{q} &= -0.0139\,\alpha & -0.426\,q & +0.0203\,\delta \\ \dot{\theta} &= & 56.7\,q & \end{aligned}$$

3. Beispiel: Boeing 737 MAX Abstürze

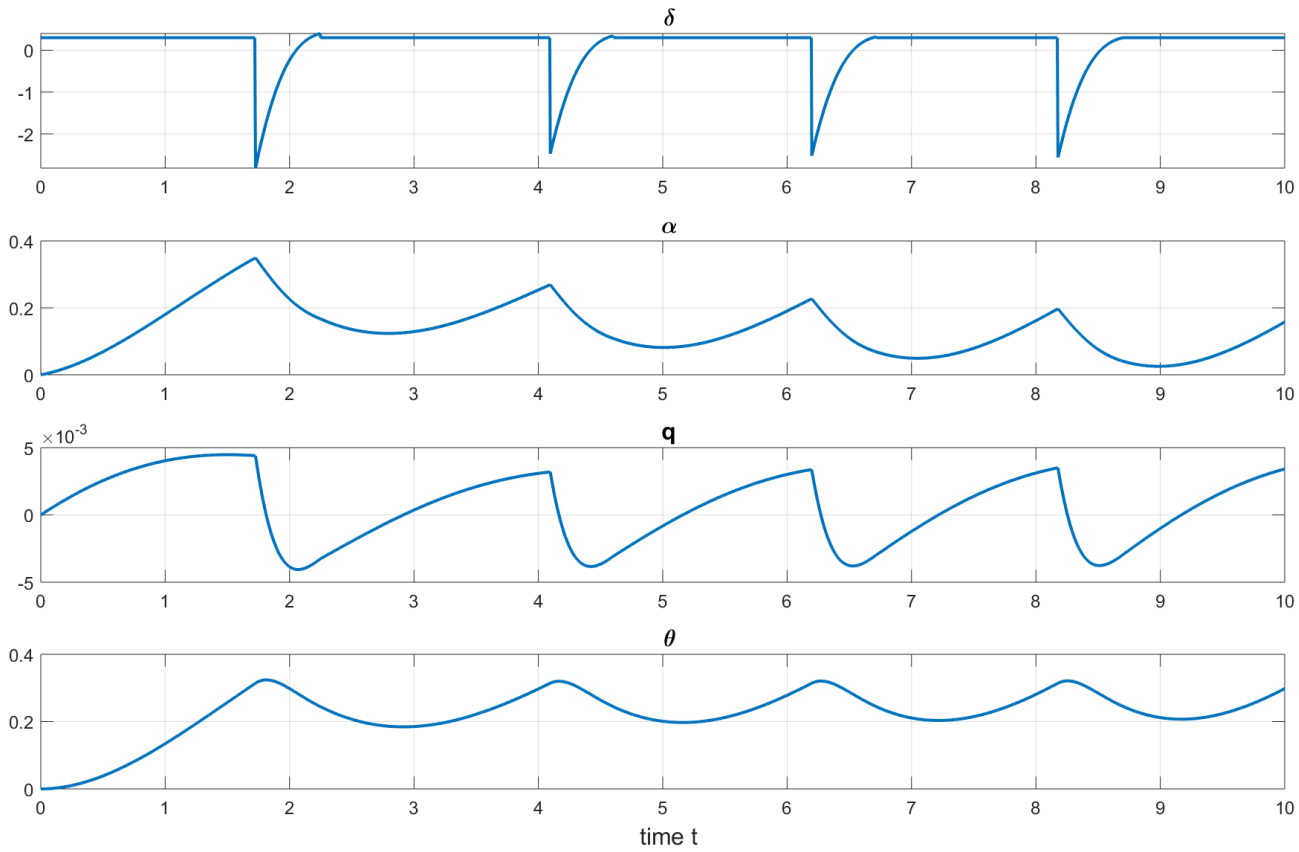
MATLAB/Simulink



3. Beispiel: Boeing 737 MAX Abstürze

MATLAB/Simulink Simulation

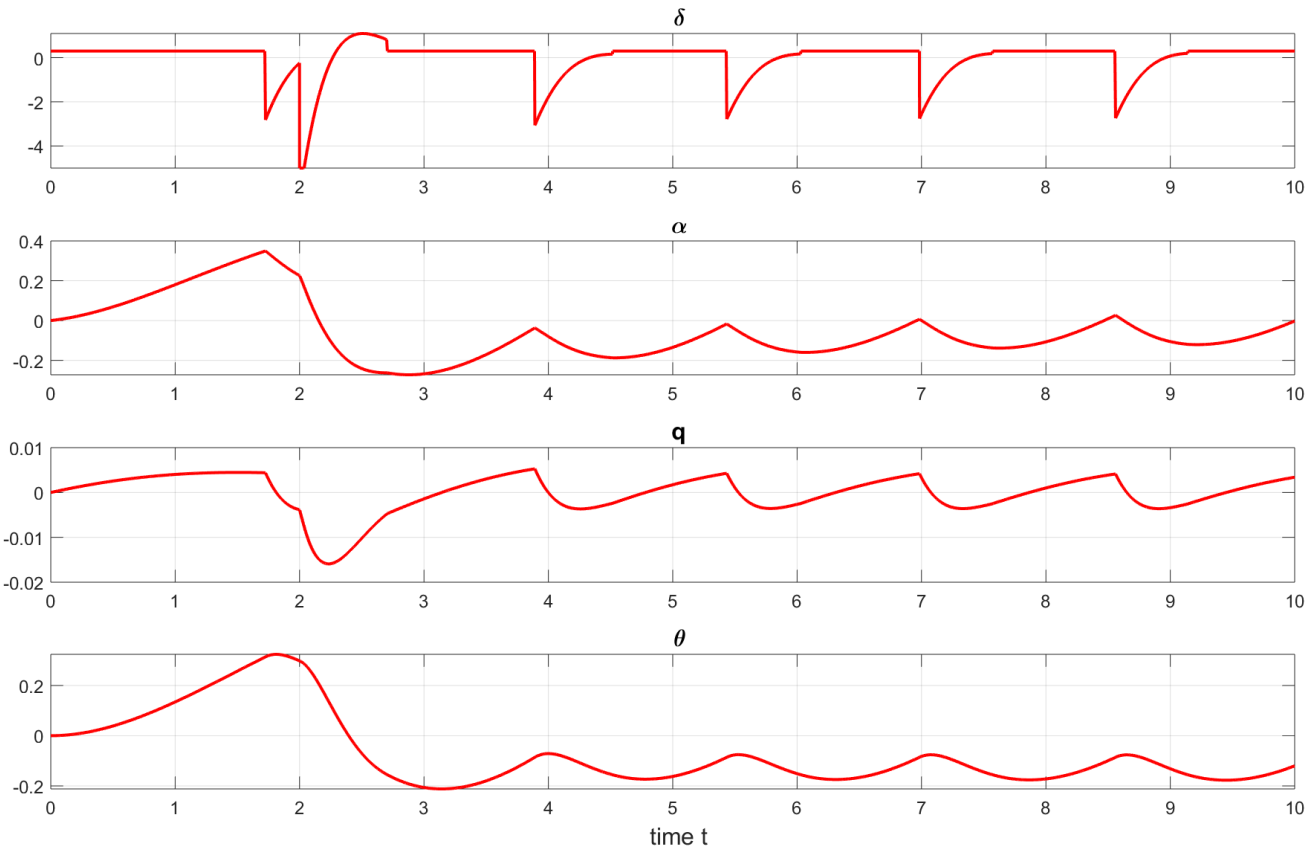
1. **Fall:** θ wird korrekt von Sensoren gemessen :-)



3. Beispiel: Boeing 737 MAX Abstürze

MATLAB/Simulink Simulation

2. Fall: θ wird inkorrekt von Sensoren gemessen :-(



Überführung in vercps

$$
\begin{aligned}
\text{PROG} = \\
&\{ \\
&\quad \mathit{if} \quad \theta > \mathit{upper\_theta\_bound} \;\; //\mathit{adjust\ the\ mode} \\
&\quad \mathit{then} \quad \mathit{stallCtrl\_mode} := 1 \\
&\quad \mathit{else} \quad \mathit{if}\ \theta < \mathit{lower\_theta\_bound} \\
&\qquad\qquad \mathit{then}\ \mathit{stallCtrl\_mode} := 0 \\
&\qquad\qquad \mathit{else}\ \mathit{skip} \\
&\qquad\qquad \mathit{endif} \\
&\quad \mathit{endif}; \\
&\quad \mathit{delta\_stall} := (\mathit{theta}_d - \theta) * k_p * \mathit{stallCtrl\_mode}; \\
&\quad \mathit{delta} := \mathit{delta\_stall} + \mathit{delta\_manual} * (1 - \mathit{stallCtrl\_mode}); \\
&\quad t := 0; \\
&\quad \{ \quad \alpha' = -0.313 * \alpha + 56.7 * q + 0.232 * \delta, \\
&\qquad\qquad q' = -0.0139 * \alpha + -0.426 * q + 0.0203 * \delta, \\
&\qquad\qquad \theta' = 56.7 * q, \\
&\qquad\qquad t' = 1 \\
&\qquad\qquad \&\ t <= \mathit{ep} \\
&\quad \} \\
&\}*
\end{aligned}
$$

Beweisziel

$$\theta > 0 \land dm = 0.3 \land kp = 5 \land tub = 0.3142 \land tlb = 0.2443 \land$$
$$td = 0.2094 \land eps = 0.0001$$

$$\rightarrow$$

$$[PROG]\theta > 0$$

Zusammenfassung Forschung

- DSLs können für verschiedene, technische Domänen entwickelt werden

Zusammenfassung Forschung

- DSLs können für verschiedene, technische Domänen entwickelt werden
- Unsere Sprache `vercps` codiert Hybride Automaten
  - Hybride Automaten sind insbesondere für *Cyber-Physikalische Systeme* nützlich

Zusammenfassung Forschung

- DSLs können für verschiedene, technische Domänen entwickelt werden
- Unsere Sprache `vercps` codiert Hybride Automaten
  - Hybride Automaten sind insbesondere für *Cyber-Physikalische Systeme* nützlich
- DSL-Tool ist guter Assistent für Domänen-Experten
  - Zwingt den Nutzer, den Sachverhalt formal auszuformulieren

Nächste Ziele

- Weiterentwicklung von `vercps` hin zu MATLAB/SIMULINK

Nächste Ziele

- Weiterentwicklung von `vercps` hin zu MATLAB/SIMULINK
- Vereinfachung der Bedienung bei der Verifikation

Nächste Ziele

- Weiterentwicklung von `vercps` hin zu MATLAB/SIMULINK
- Vereinfachung der Bedienung bei der Verifikation
- Visualisierung von Systemabläufen

Vielen Dank!