Abschätzung der Wertebereiche von zeitdiskreten, dynamischen Systemen

Diplomarbeit

vorgelegt von

Dr.-Ing. Ulrich Eisemann aus Schwäbisch Hall

Matrikel-Nr.: 4311124

Betreuer: Prof. Dr. W. Hochstättler, Lehrstuhl für Diskrete Mathematik der FernUniversität in Hagen

Inhaltsverzeichnis

Kapitel 1

Einleitung

Das Thema der Diplomarbeit ist die robuste Abschätzung von Wertebereichsgrenzen für diskrete, dynamische Systeme, wie sie in Form von Steuerungs- und Regelungsfunktionen in der Automobilindustrie typischerweise entwickelt werden. Der Terminus robust impliziert hierbei, dass die ermittelten Minimal- und Maximalwerte aller im System vorhandenen Größen die im tatsächlichen Einsatz angenommenen Wertebereiche mit absoluter Gewissheit überdecken müssen. Es handelt sich also um eine konservative Abschätzung, wobei das Unterschätzen von Maxima bzw. das Überschätzen von Minima in keinster Weise tolerabel ist.

Die untersuchten diskreten, dynamischen Systeme sind ausschließlich durch Blockschaltbilder in der grafischen Modellierungsumgebung Simulink®/TargetLink® beschrieben und modellieren Steuerungs- und Regelungsfunktionen. Diese Vorgehensweise wird im Allgemeinen als modellbasiertes Design bezeichnet und hat sich in der Automobilindustrie mittlerweile fest etabliert. Ein wichtiger Vorteil dieser Vorgehensweise besteht darin, dass aus diesen grafisch modellierten Systemen unmittelbar Code generiert werden kann, der anschließend direkt in automotiven Steuergeräten eingesetzt wird. Aus der automatischen Code-Generierung motiviert sich auch die Wertebereichsabschätzung als eigentliches Thema der Diplomarbeit. Viele Steuerungs- und Regelungsfunktionen, die initial in Fließkomma-Arithmetik entworfen wurden, müssen zur Implementierung, bzw. zur Umsetzung in Software in Festkomma-Algorithmen überführt werden. Robust abgeschätzte Minimal- und Maximalwerte aller Systemgrößen sind hierzu von essentieller Bedeutung, da nur durch deren Kenntnis sichergestellt werden kann, dass im Betrieb keine Fehler durch Über- oder Unterläufe auftreten. Eng abgeschätzte Grenzen, wie sie trotz Robustheit naturgemäß wünschenswert sind, sorgen gleichzeitig dafür, dass eine optimale Genauigkeit bei allen Berechnungen erzielt wird. Desweiteren könnten die Minimal- und Maximalwerte auch direkt dazu genutzt werden, um das Design der Regelungs- und Steuerungsfunktion auf mögliche Laufzeitfehler wie beispielsweise Divisionen durch Null zu überprüfen oder als Basis für ausgefeiltere Optimierungen in Code-Generatoren dienen.

Die im Rahmen der Diplomarbeit eingesetzten Verfahren zur Bestimmung von Wertebereichsgrenzen beruhen im Wesentlichen auf der Anwendung von Intervallarithmetischen Methoden, wie sie schon seit mehreren Jahrzehnten bekannt sind und durch den Einsatz von leistungsfähigen Rechnern in letzter Zeit einen Aufschwung erfahren haben. Methoden der Intervallarithmetik haben die wichtige Eigenschaft, dass sie den Wertebereich von Funktionen vollständig überdecken, wodurch sich die Forderung der Robustheit erfüllen lässt. Im Fall von dynamischen, zustandsbehafteten Systemen ist zudem essentiell, dass

die Abschätzung der Wertebereichsgrenzen für alle möglichen Zeitpunkte Gültigkeit haben muss, was die Lösung von Fixpunktgleichungen für Mengenfunktionen erfordert, die in dieser Diplomarbeit behandelt werden.

Die Arbeit ist folgendermaßen gegliedert:
In Kapitel 2 wird der Charakter und die Semantik der graphisch modellierten, zeitdiskreten Systeme beschrieben, die im Rahmen der Diplomarbeit behandelt werden. Gleichfalls wird das Anwendungsszenario der Diplomarbeit im Zusammenhang mit dem Designprozess von Steuerungs- und Regelungsfunktionen genauer beleuchtet, durch welches die Diplomarbeit motiviert wurde. Die Bereitstellung der Intervallarithmetischen Hilfsmittel zur Wertebereichsabschätzung erfolgt in Kapitel 3, die dann in Kapitel 4 zur Berechnung von Wertebereichsgrenzen für modellbasierte Designs angewandt werden. Im Rahmen der Diplomarbeit ist ferner eine prototypische Realisierung der vorgeschlagenen Algorithmik entwickelt worden, die ebenfalls in Kapitel 4 beschrieben ist. In Kapitel 5 wird anschließend anhand des Prototypen die Anwendbarkeit der entwickelten Methodik auf modellbasierte Designs untersucht, was sowohl anhand von idealisierten als auch realen Modellen aus der Praxis geschieht. Dabei wird ein Fazit hinsichtlich der Leistungsfähigkeit bzw. den in der Praxis auftretenden Limitierungen gezogen und die Algorithmik insgesamt bewertet. Die Arbeit schließt mit einer Zusammenfassung und einem kurzen Ausblick in Kapitel 6.

Kapitel 2

Modellbasierte Designs als zeitdiskrete, dynamische Systeme

In diesem Kapitel wird zunächst in Abschnitt 2.1 auf das übergeordnete Szenario zur Anwendung der Diplomarbeit eingegangen, nämlich die modellbasierte Entwicklung von Steuerungs- und Regelungsfunktionen und deren Implementierung durch automatische Code-Generierung. Eine genaue Motivierung der Diplomarbeit erfolgt anschließend in Abschnitt 2.2, in dem aufgezeigt wird, dass eine robuste Abschätzung von Wertebereichsgrenzen essentiell ist, um einen Fließkomma-Algorithmus in Festkomma-Code umzusetzen, was genau der Anwendungsfall der Diplomarbeit ist. In Abschnitt 2.3 wird anschließend eine mathematische Beschreibung der modellbasierten Designs der betrachteten Steuerungs- und Regelungsfunktionen gegeben, die Semantik von Simulink/TargetLink Modellen also in eine äquivalente, mathematische Systembeschreibung übersetzt. Die präzise Formulierung der Aufgabenstellung zur Abschätzung der Wertebereiche für modellbasierte Designs erfolgt schließlich in Abschnitt 2.4.

2.1 Modellbasierte Entwicklung und automatische Code-Generierung

In den letzten Jahren hat sich die modellbasierte Entwicklung im Bereich des Designs von Steuerungs- und Regelungsfunktionen für eingebettete Systemen (d.h. Geräte mit eingebauter programmierbarer Elektronik und Software) etabliert. Der Terminus modellbasierte Entwicklung impliziert, dass die Funktionalitäten graphisch beschrieben werden, konkret in Form von Block- und Zustandsdiagrammen. Insbesondere in der Automobilindustrie ist man vielfach dazu übergegangen, die Software-Entwicklung für Steuergeräte in Kraftfahrzeugen (z.B. für Motorsteuergeräte, Getriebesteuergeräte, elektronisches Stabilitätsprogramm ESP, Anti-Blockiersystem ABS, Airbag etc.) durch grafische Modellierung in Simulink®[1]/TargetLink®[2] vorzunehmen, siehe Abbildung 2.1 und Abbildung 2.2.

[1] Simulink ist eine Entwicklungsumgebung zur grafischen Modellierung von zeitkontinuierlichen und zeitdiskreten Systemen der Firma The MathWorks.

[2] TargetLink ist ein Werkzeug zur automatischen Code-Generierung aus Simulink Modellen der Firma dSPACE.

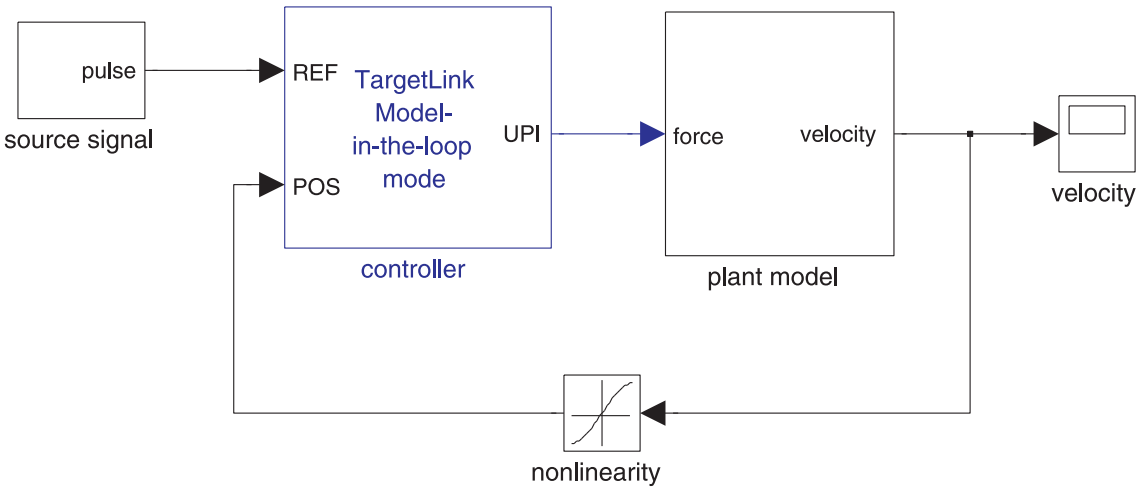


Abbildung 2.1: Oberste Hierarchieebene eines exemplarischen Simulink/TargetLink Modells. Die zu entwerfende Regelungsfunktion ist im Block *controller* gekapselt, dessen Innenleben in Abbildung 2.2 dargestellt ist. Der Block *source signal* repräsentiert ein zu Simulationszwecken vorgegebenes Stimulus-Signal. Der Block *plant model* bzw. dessen Innenleben modelliert die relevante physikalische Umgebung der Regelungsfunktion (ein sogenanntes Streckenmodell) inklusive der zu regelnden Größe(n) (in diesem Beispiel die Geschwindigkeit *velocity*). In Verbindung mit dem Block *nonlinearity*, der etwa die nichtlineare Kennlinie eines Sensors modellieren könnte, bilden die Regelungsfunktion *controller* und das Streckenmodell *plant model* einen geschlossenen Regelkreis, in dem *controller* die Größe *velocity* auf den vorgegebenen Werteverlauf des Stimulus-Signals *source signal* regeln soll. Nur die Regelungsfunktion *controller* wird später durch automatische Code-Generierung in Software umgesetzt und auf dem realen Steuergerät eines Embedded Systems ausgeführt. Die Eingangssignale *REF* und *POS* sowie das Ausgangssignal *UPI* von *controller* werden auf dem realen Steuergerät mit Sensoren bzw. Aktuatoren verbunden.

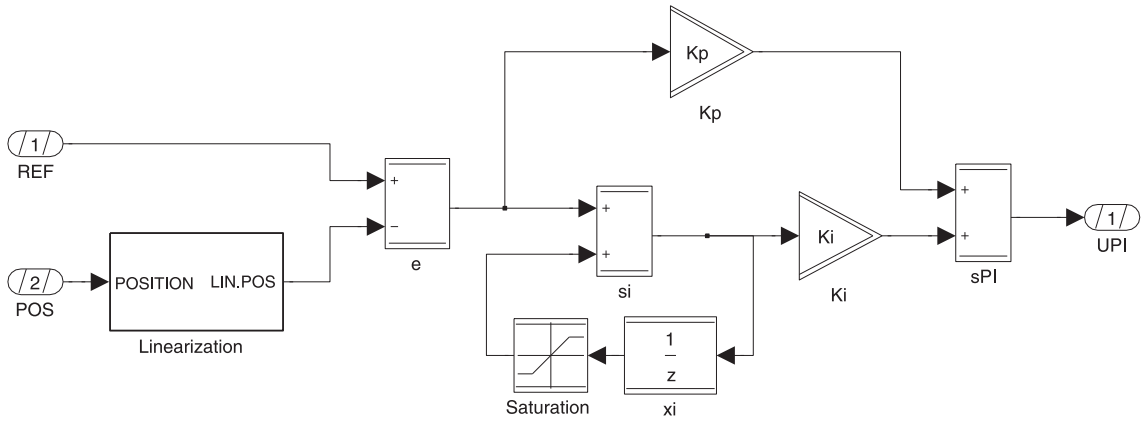


Abbildung 2.2: Das Innenleben des *controller* Blockes aus Abbildung 2.1 als Beispiel für eine einfache Regelungsfunktion in Simulink/TargetLink. Die Gesamtfunktionalität ergibt sich aus der Verschaltung von einzelnen Blöcken, die jeder für sich eine elementare Operation wie Summation, Verstärkung, Verzögerung etc. ausführen. Die grafisch modellierte Regelungsfunktion wird im Verlauf des Entwicklungsprozesses per automatischer Code-Generierung implementiert.

Diese Vorgehensweise hat nicht nur den Vorteil, dass Entwürfe für Steuerungs- und Regelungsfunktionen sehr schnell und einfach erstellt und durch Simulationen am Computer „ausprobiert“ werden können, sondern insbesondere, dass aus diesen Simulink/TargetLink Modellen auch unmittelbar die eigentliche Software für die Steuergeräte selbst generiert werden kann, was heutzutage vielfach geschieht, siehe [Bei03, Han01]. Software für Steuerungs- und Regelungsfunktionen muss also nicht mehr von Hand geschrieben werden, sondern wird direkt aus einem Simulink/TargetLink Modell, also von einer höheren Abstraktionsebene aus, automatisch erzeugt. Simulink stellt dabei die eigentliche Modellierungsumgebung dar, in der die Regelungsalgorithmen entworfen und simuliert werden, wohingegen TargetLink als Code-Generator zur Implementierung der Modelle in Form von effizientem, gut lesbarem C-Code dient.

Der Entwicklungsprozess für eine Regelungsfunktion, angefangen vom Entwurf der Algorithmik bis zur Generierung der Software, beinhaltet typischerweise die folgenden Phasen[3], siehe Abbildung 2.3:

1. Zunächst wird ein erstes Design der Algorithmik der Regelungsfunktion vorgenommen, indem einzelne Blöcke elementarer Funktionalität wie in Abbildung 2.2 zur Gesamtalgorithmik verschaltet werden. Aufgrund des grafischen Entwurfes auf Basis vordefinierter Blöcke ist dieser Designprozess sehr effizient und liefert schnell einen ersten Entwurf der Regelungsfunktion.

2. Die modellierte Algorithmik wird anschließend durch Computer-Simulationen im Hinblick auf das gewünschte Verhalten überprüft, indem die Regelungsfunktion im Zusammenspiel mit der Regelstrecke und vorgegebenen Stimulus-Signalen simuliert wird, siehe Abbildung 2.1. Die Signalverläufe sämtlicher relevanter Größen innerhalb des Modells werden hierzu während der Simulation aufgezeichnet und anschließend visualisiert, um die Funktionsfähigkeit der Regelungsfunktion beurteilen zu können, siehe Abbildung 2.4. Diese Simulation wird typischerweise in Fließkomma-Arithmetik durchgeführt, damit sich der Designer vollständig auf die Entwicklung der eigentlichen Algorithmik konzentrieren kann, ohne sich jetzt schon mit Aspekten der Festkomma-Arithmetik befassen zu müssen.

3. Die Design und Simulationsphasen aus Schritt 1. und 2. werden so lange iterativ durchlaufen, bis das graphische Design der Reglungsfunktion genau das gewünschte Simulationsverhalten an den Tag legt und die eigentliche Algorithmik der Regelungsfunktion somit fertig entwickelt ist.

4. Zur Implementierung der Regelungsfunktion mittels automatischer Code-Generierung wird nun das Modell um zusätzliche, Software-bezogene Informationen erweitert, die genau festlegen, wie der aus dem Fließkomma-Modell zu generierende Code aussehen soll. Es ist ein Charakteristikum der Software im Automobilbereich, dass eher selten Fließkomma-Code sondern überwiegend Festkomma-Code entwickelt wird[4]. Festkomma-Code zeichnet sich dadurch aus, dass für jede Größe innerhalb der Regelungsfunktion nicht nur ein Datentyp sondern auch Skalierungsinformationen bestehend aus einem Skalierungsfaktor und einem Offset hinzugefügt

[3]Hier werden nur die Schritte aufgezählt, die einen Bezug zur Diplomarbeit aufweisen. In der Praxis eines realen Entwicklungsprozesses existieren natürlich weitere Phasen wie Anforderungsanalyse, Tests etc.

[4]Dies ist teilweise durch Kostenaspekte bedingt, da Prozessoren ohne Fließkomma-Einheit preiswerter sind.

werden müssen, siehe Abbildung 2.5, auf die bei der Code-Generierung zurückgegriffen wird. Die Skalierungsinformationen beschreiben dann in Kombination mit dem Datentyp den Zusammenhang zwischen realem (physikalischem) Wert einer Größe und dessen Software-interner Repräsentation im Speicher. Der Software-Entwickler, der mit der Umsetzung der eigentlichen Algorithmik in Festkomma-Code befasst ist, muss das eigentliche algorithmische Fließkomma-Modell um diese speziellen Implementierungsinformationen erweitern, um so zu einem Implementierungsmodell zu gelangen, das für die automatische Code-Generierung geeignet ist.

5. Nach Finalisierung des Implementierungsmodells wird aus diesem anschließend der fertige Festkomma-Code für das eingebettete System generiert, wie dies in Abbildung 2.6 exemplarisch für die in Abbildung 2.2 dargestellte Regelungsfunktion geschehen ist.

Die obigen Schritte 1. bis 5. werden (neben weiteren Phasen im Entwicklungsprozess) für jeden modellbasierten Entwurf durchlaufen. Die Motivation für diese Arbeit ergab sich aus dem 4. Schritt, also der Umsetzung eines Fließkomma-Modells in Festkomma-Code, wie im nachfolgenden Abschnitt dargestellt wird.

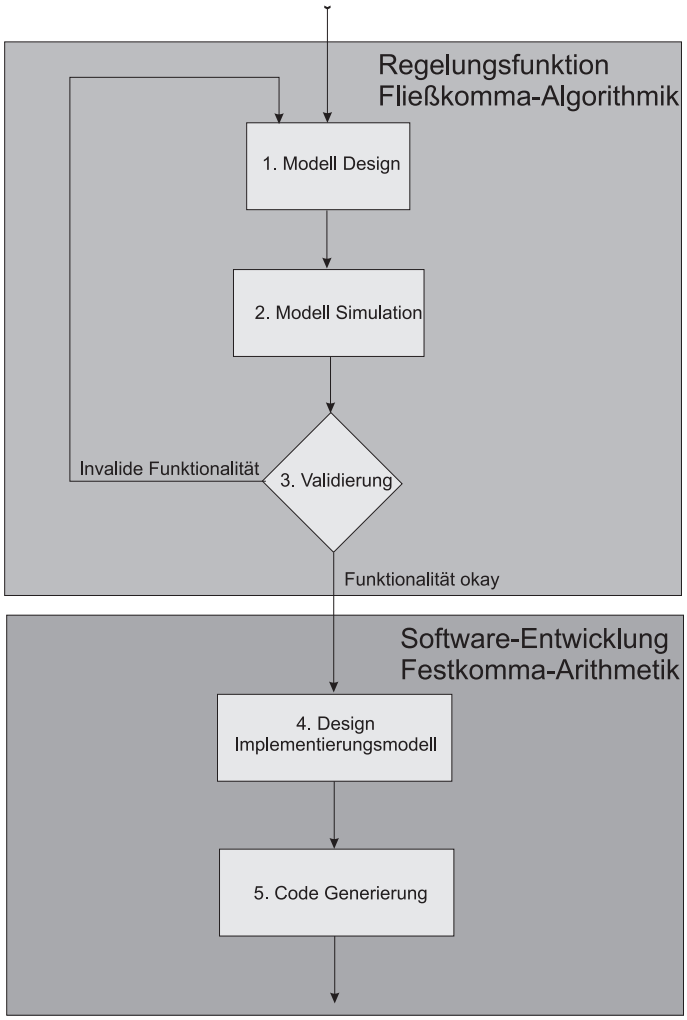


Abbildung 2.3: Übersicht über einzelne Schritte im Entwicklungsprozess vom Fließkomma-Modell zum Festkomma-Code.

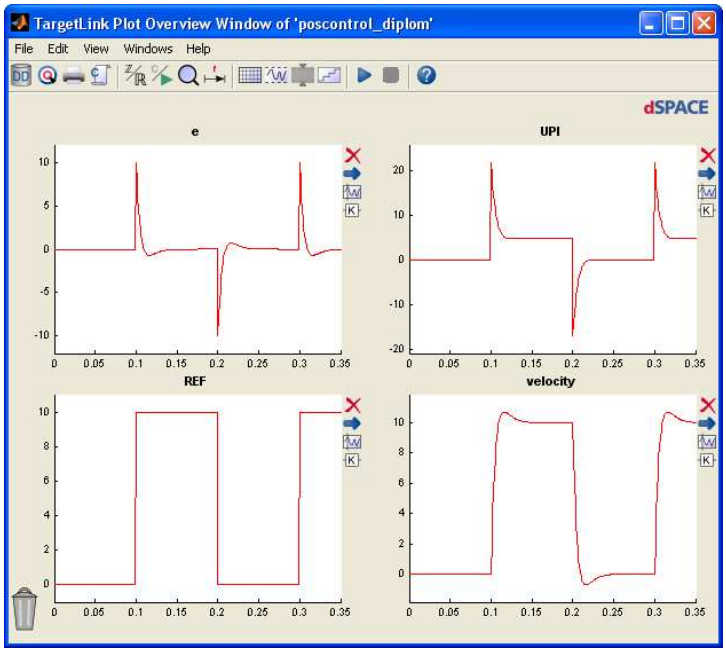


Abbildung 2.4: Während der Simulation aufgezeichnete Signale, um die Algorithmik der Regelungsfunktion aus Abbildung 2.2 validieren zu können. Betrachtet wird hier insbesondere das Stimulussignal *REF* und die dementsprechend zu regelnde Größe *velocity*.

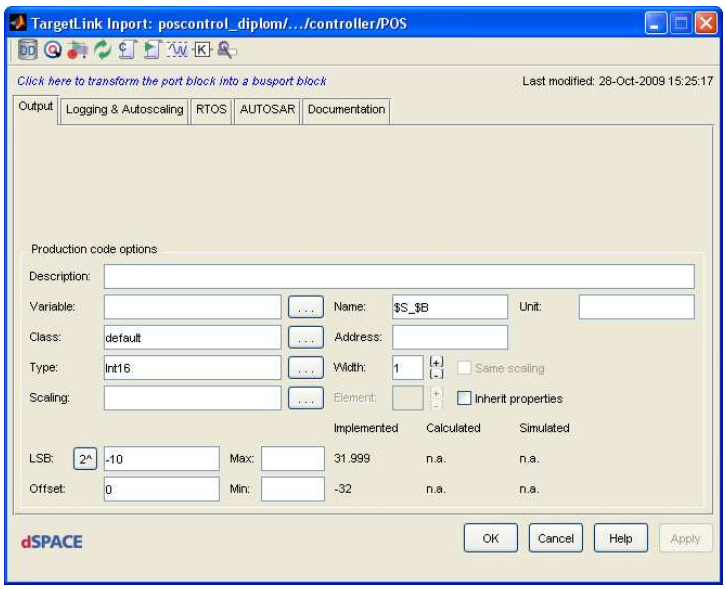


Abbildung 2.5: Hinzufügen von Implementierungsinformationen zur Generierung von Festkomma-Code aus dem Modell. Für jedes Signal wird neben einem Datentyp (hier konkret ein vorzeichenbehafteter 16 Bit Integer Wert) auch Skalierungsinformation in Form von einem Skalierungsfaktor (LSB, Least Significant Bit) und einem Offset vorgegeben. Erst wenn diese Skalierungsinformationen für jede Größe (Variable) vom Software-Entwickler spezifiziert worden sind, kann der Festkomma-Code aus dem Modell generiert werden.

```
Void controller(Void)
{
   /* SLLocal: Default storage class for local variables | Width: 16 */
   Int16 Sa1_e /* LSB: 2^-10 OFF:  0 MIN/MAX:  -32 .. 31.9990234375 */;
   Int16 Sa1_si /* LSB: 2^-9 OFF:  0 MIN/MAX:  -64 .. 63.998046875 */;

   /* SLStaticLocalInit: Default storage class for static local variables with initvalue | Width: 16
    */
   static Int16 X_Sa1_xi = 0 /* LSB: 2^-9 OFF:  0 MIN/MAX:  -64 .. 63.998046875 */;

   /* update(s) for inport controller/Linearization/POSITION */
   Sa2_POSITION = Sa1_POS;

   /* call of function: controller/Linearization */
   Sa2_Linearization();

   /* Sum: controller/e */
   Sa1_e = (Int16) (((UInt16) Sa1_REF) - ((UInt16) Sa2_LIN_POS));

   /* Sum: controller/si
      # combined # Saturation: controller/Saturation */
   Sa1_si = (Int16) (((UInt16) (Int16) (Sa1_e >> 1)) + ((UInt16) C__I16SATI16_SATb(X_Sa1_xi, 25600
    /* 50. */, -25600 /* -50. */)));

   /* Sum: controller/sPI
      # combined # Gain: controller/Kp
      # combined # Gain: controller/Ki */
   Sa1_sPI = (Int16) (((UInt16) Sa1_e) + ((UInt16) (Int16) ((((Int32) Sa1_si) * 13107) >> 16)));

   /* TargetLink outport: controller/UPI */
   Sa1_UPI = Sa1_sPI;

   /* Unit delay: controller/xi */
   X_Sa1_xi = Sa1_si;
}
```

Abbildung 2.6: Automatisch generierter Festkomma-Code in Form einer C-Funktion für die in Abbildung 2.2 dargestellte Regelungsfunktion. Typisch für die Implementierung in Festkomma-Arithmetik sind die Integer-Datentypen mit entsprechenden Typumwandlungen (Type-Casts) und die enthaltenen Umskalierungsoperationen (Multiplikationen und Bitshifts), die sich aus den spezifizierten Skalierungsinformationen (LSB, Offset) ergeben.

2.2 Festkomma-Implementierung auf Basis von abgeschätzten Wertebereichsgrenzen

Ein besonders zeitaufwendiger Arbeitsschritt im Entwicklungsprozess stellt die Umsetzung des Fließkomma-Modells in Festkomma-Code dar. Wie im vorherigen Abschnitt beschrieben, werden initial Fließkomma-Modelle entwickelt, die dann im Schritt 4. des Entwicklungsprozesses für die Generierung von Festkomma-Code zu erweitern sind. Es müssen also zunächst Skalierungsinformationen für jedes einzelne Modellsignal spezifiziert werden, bevor Festkomma-Code generiert werden kann. Hierzu ist die Kenntnis der Wertebereichsgrenzen für jedes Modellsignal unentbehrlich um sicherzustellen, dass grundsätzlich keine Über- oder Unterläufe durch Zahlenbereichsüberschreitungen auftreten können. Die Abschätzung der Wertebereichsgrenzen muss daher unter allen Umständen robust bzw. konservativ sein. Andererseits sollen die ermittelten Wertebereichsgrenzen möglichst eng sein um eine optimale Genauigkeit zu erzielen, wie in den nachfolgenden Abschnitten dargestellt werden wird.

Bemerkung 1. *Auch wenn diese Arbeit durch die vereinfachte Festkomma-Implementierung für modellbasierte Designs motiviert wurde, sind natürlich weitere Anwendungsfälle für die robuste Abschätzung von Wertebereichsgrenzen vorstellbar. So können Wertebereichsgrenzen beispielsweise dazu genutzt werden, um Modelle automatisiert auf potenzielle Laufzeitfehler zu überprüfen. Analog zu geschriebenem Code ist es natürlich auch bei modellbasierten Designs möglich, dass durch Design-Fehler Divisionen durch Null,*

Quadratwurzeln aus negativen, reellen Zahlen etc. auftreten. Robust abgeschätzte Wertebereichsgrenzen könnten dann in vielen Fällen eingesetzt werden, um den Nachweis der Laufzeitfehlerfreiheit zu erbringen, für eine Division etwa durch den Nachweis, dass die Wertebereichsgrenzen des Divisors die Null nicht enthalten. Desweitern könnten Wertebereichsgrenzen natürlich auch verwendet werden, um die Optimierungsroutinen des Code-Generators selbst zu verfeinern und damit effizienteren Code zu generieren.

2.2.1 Physikalische Werte und ihre Festkomma-Darstellungen

Üblicherweise wird zur Festkomma-Repräsentation von Signalen eine einfache, affine Transformation eingesetzt, um den realen, physikalischen Wert x_{phys} einer Größe mit dessen Festkomma-(Integer) Repräsentation x_{int} im Code in Beziehung zu setzen, was nach der folgenden Vorschrift geschieht:

$$x_{phys} = LSB_x \cdot x_{int} + Offset_x \tag{2.1}$$

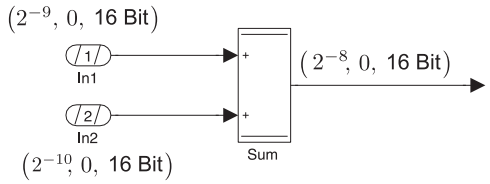
Hierbei bildet das Tupel (LSB_x, $Offset_x$, Datentyp(x_{int})) die Skalierungsinformation, die für jedes im Modell auftretende Signal x_{phys} zu spezifizieren ist, siehe Abbildung 2.5. Für den Datentyp von x_{int}, Datentyp(x_{int}), wird ein 8, 16 oder 32 Bit breiter Integer-Datentyp spezifiziert, was abhängig von der geforderten Genauigkeit der Berechnungen wie auch der Registerbreite des eingesetzten Prozessors abhängig gemacht wird. Die Integer-Repräsentation x_{int} ist der Wert des Signals im Speicher, wenn dieser als einfache Binärzahl interpretiert wird. Die Integer-Repräsentation x_{int} stellt dann die im generierten Code existente Variable dar, mit der gerechnet wird. Die Werte von x_{phys} sind hingegen die realen physikalischen Werte, wie sie sich in der Simulation entsprechend Abbildung 2.4 ergeben und die durch die im Rahme der Diplomarbeit entwickelte Algorithmik robust abgeschätzt werden sollen.

Beispiel 2. *Für die Skalierung des physikalischen Signals x_{phys} seien die Skalierungsdaten (LSB_x, $Offset_x$, Datentyp(x_{int})) = (2^{-8}, 0, Signed 16 Bit) gewählt. Dann besteht zwischen x_{phys} und x_{int} nach Gleichung 2.1 der Zusammenhang $x_{phys} = 2^{-8} \cdot x_{int}$ und für den 16-Bit Integer Wert $x_{int} = 0000000011111111_2 = 255_{10}$ ergibt sich ein physikalischer Wert $x_{phys} = 2^{-8} \cdot x_{int} = 2^{-8} \cdot 255 = {}^{255}/_{256} \approx 0.996$. Zweierpotenzen wie 2^{-8} als LSB-Werte haben den Vorteil, dass Umskalierungsoperationen relativ effizient als Bitshifts implementiert werden können. Ein Offset von 0 führt zudem dazu, dass die affine Transformation besonders einfach und damit effizient ist.*

2.2.2 Operationen auf Festkomma-Zahlen

Die Gesetzmäßigkeiten der Arithmetik für Festkomma-Zahlen ergeben sich direkt aus dem Zusammenhang zwischen physikalischen Größen und reinen Integer-Werten entsprechend Gl. 2.1. Die in einem Modell auftretenden Block-Operationen wie Summationen, Multiplikationen etc. werden naturgemäß so ausgeführt, dass sie die gewünschten Operationen für die realen, physikalischen Werte x_{phys} korrekt nachbilden. Im Festkomma-Code müssen diese Operationen dann auf Basis der Integer-Repräsentationen x_{int}, also der Variablen im Code, realisiert werden, wozu der Code-Generator jeweils Gl. 2.1 heranzieht. Der Anwender muss also lediglich die Skalierungsinformationen für alle involvierten Signale spezifizieren, woraufhin der Code-Generator automatisch alle erforderlichen Code-Pattern für die Festkomma-Implementierung aller Operationen generiert.

Beispiel 3. *In Abbildung 2.7 ist exemplarisch dargestellt, wie sich die Summation zweier Signale mit vorgegebenen Skalierungen in die zugehörige Festkomma-Operation für die Integer-Repräsentationen übersetzt, die vom Code-Generator basierend auf den Skalierungsinformationen generiert wird. Der Code beinhaltet Umskalierungsoperationen, um die unterschiedlichen Skalierungen der jeweiligen Signale zu berücksichtigen.*




```
/* Sum: controller/Sum */
Sum = (Int16) (((Int16)  (In1 / ((Int16)  2))) + ((Int16) (In2 / ((Int16)  4))));
```

Abbildung 2.7: Beispiel einer Modelloperation und der zugehörigen Festkomma-Implementierung. Berechnet werden soll die Summation $\text{In1}_{phys} + \text{In2}_{phys} = \text{Sum}_{phys}$. Für die Signale sind jeweils Skalierungsinformationen in der Form $(LSB_x,\ Offset_x,\ \text{Datentyp}(x_{int}))$ vorgegeben. Dann ergibt sich als entsprechende Festkomma-Implementierung das angegebene Code-Fragment im unteren Teil des Bildes. Die Variablen Sum, In1, In2 im generierten Code sind die entsprechenden Integer-Repräsentationen der physikalischen Größen, also $\text{Sum} = \text{Sum}_{int}$, $\text{In1} = \text{In1}_{int}$ und $\text{In2} = \text{In2}_{int}$.

2.2.3 Wertebereichsgrenzen als Schlüssel zur Spezifikation von Skalierungsinformationen

Die Kenntnis von Wertebereichsgrenzen für jedes Signal innerhalb des Modells, genauer, die Kenntnis der minimalen und maximalen physikalischen Werte, die ein Modell-Signal x_{phys} annehmen kann, ist essentielle Voraussetzung zur Spezifikation von Skalierungsdaten $(LSB_x,\ Offset_x,\ \text{Datentyp}(x_{int}))$ für x_{phys}, wobei letztere wiederum zur Generierung von Festkomma-Code zwingend erforderlich sind. Nach Gl. 2.1 resultiert bei gegebenen Skalierungsdaten $(LSB_x,\ Offset_x,\ \text{Datentyp}(x_{int}))$ ein maximaler physikalischer Wertebereich $[x_{phys,min},\ x_{phys,max}]$, der durch die Integer-Repräsentation im Code abgedeckt werden kann:

$$[x_{phys,min},\ x_{phys,max}] = LSB_x \cdot [x_{int,min},\ x_{int,max}] + Offset_x \tag{2.2}$$

Hier stellen $[x_{int,min},\ x_{int,max}]$ die ganzzahligen Grenzen des Wertebereiches für Integer-Datentypen im Rechner dar, etwa der Bereich $0\,..\,2^8 - 1$ für einen vorzeichenlosen 8 Bit Wert oder der Bereich $-2^{15}\,..\,2^{15} - 1$ für einen vorzeichenbehafteten 16 Bit Wert, der im Zweierkomplement dargestellt wird. Falls nicht sichergestellt ist, dass die sich aus Gl. 2.2 ergebenden physikalischen Grenzen den im realen Betrieb angenommenen Wertebereich überdecken, so können bei der Ausführung des generierten Codes schwerwiegende Fehler durch Wertebereichsüberschreitungen (Überläufe/Unterläufe) auftreten, die zu völlig falschem Verhalten führen und unter Umständen katastrophale Auswirkungen haben können. Ist umgekehrt sichergestellt, dass das reale physikalische Signal x_{phys} grundsätzlich

innerhalb der durch Gl. 2.2 bestimmten Grenzen $[x_{phys,min},\ x_{phys,max}]$ liegt, so ist sichergestellt, dass der generierte Code prinzipiell frei von Überläufen/Unterläufen ist.

Im Entwicklungsprozess ist die Vorgehensweise zur Generierung von Festkomma-Code naturgemäß folgendermaßen:

Der Software-Entwickler ist im Entwicklungsschritt 4. damit befasst, das Fließkomma-Modell in ein Festkomma-Implementierungsmodell umzuwandeln. Hierzu muss er zunächst robuste, physikalische Wertebereichsgrenzen $[x_{phys,min},\ x_{phys,max}]$ aller Signale ermitteln, wozu die im Rahmen dieser Diplomarbeit entwickelte Algorithmik dient. Basierend auf diesen Grenzen kann der Software-Entwickler dann existierende Fähigkeiten von TargetLink nutzen, um die Skalierungsdaten (LSB_x, $Offset_x$, Datentyp(x_{int})) zu spezifizieren bzw. von TargetLink bestimmen zu lassen. TargetLink greift dabei auf die ermittelten Grenzen $[x_{phys,min},\ x_{phys,max}]$ zu und bestimmt die Skalierungsdaten so, dass der physikalische Wertebereich durch die gewählte Skalierung überdeckt wird. In den meisten Fällen wird der Datentyp(x_{int}) vom Anwender vorgegeben (oftmals als 16 Bit Integer Wert), $Offset_x$ auf null gesetzt und der LSB_x-Wert als dann noch einziger Freiheitgrad von TargetLink so bestimmt, dass der geforderte Wertebereich $[x_{phys,min},\ x_{phys,max}]$ gerade überdeckt wird. In jedem Fall stellt TargetLink bei korrekt ermittelten Wertebereichsgrenzen sicher, dass grundsätzlich keine Überläufe/Unterläufe im realen Betrieb des generierten Codes auftreten können.

2.3 Mathematische Beschreibung der betrachteten Modellbasierten Designs

In diesem Abschnitt soll die Semantik modellbasierter Designs in Simulink/TargetLink genauer beschrieben werden. Dazu wird insbesondere dargestellt, wie sich ein modellbasiertes Design in eine Abfolge mathematischer Operationen für die entsprechenden Blöcke und der zwischen ihnen ausgetauschten Signale übersetzt. Prinzipiell stellt die Simulink/TargetLink Modellierungsumgebung eine domänenspezifische Sprache zur Beschreibung von dynamischen Systemen dar, die zwar nicht ausschließlich aber insbesondere zur Entwicklung von Steuerungs- und Regelungsfunktionen Anwendung findet. Der Unterabschnitt 2.3.1 beschreibt dazu zunächst die prinzipielle Semantik individueller TargetLink Blöcke[5], wohingegen Unterabschnitt 2.3.2 das sich aus der Verschaltung der Blöcke ergebende Gesamtsystem betrachtet. Insgesamt wird hier nur auf die für diese Diplomarbeit relevanten Aspekte von Simulink/TargetLink eingegangen. Für weiterführende Informationen, insbesondere auch zur Beschreibung von allgemeineren mathematischen Systemen als den hier betrachteten, sei auf die offizielle Simulink Dokumentation [SL09] verwiesen[6].

2.3.1 Die Semantik einzelner TargetLink Blöcke

Jeder Block eines Simulink/TargetLink Modells realisiert im weitesten Sinne "elementare" mathematische Operationen, beispielsweise eine einfache Summation oder Multiplikation

[5] Wir bezeichnen alle Blöcke in einem Modell im Folgenden der Einfachheit halber grundsätzlich als TargetLink Blöcke.

[6] Man beachte, dass Simulink keine formale sondern nur eine semi-formale Beschreibung der modellierten Systeme darstellt.

der Eingänge des Blockes und stellt das Resultat dieser Operation am Ausgang bzw. den Ausgängen des Blockes zur Verfügung. Ein Block kann jedoch auch eine komplexere Funktionalität wie beispielsweise eine tabellenbasierte Abbildung realisieren.

Im Allgemeinsten Fall besitzt ein Block B die folgenden Block-spezifischen Variablen:

- eine Zahl von Blockeingängen, die im Eingangsvektor $x \in X \subseteq \mathbb{R}^{n_x}$ des Blockes zusammengefasst sind. Bei der Ausführung des Blockes entstammen diese Werte den Ausgangswerten der Vorgängerblöcke, wie sie sich aufgrund der Verschaltung der Blöcke ergeben.

- eine Zahl von Blockausgängen, die im Ausgangsvektor $y \in Y \subseteq \mathbb{R}^{n_y}$ des Blockes zusammengefasst sind. Nach der Ausführung des Blockes stehen die Resultate der vom Block ausgeführten Operationen an den Ausgängen zur Verfügung.

- eine Zahl von Zustandsvariablen, die im Zustandsvektor $z \in Z \subseteq \mathbb{R}^{n_z}$ des Blockes zusammengefasst sind. Zustände existieren für solche Blöcke, bei denen der Wert der Ausgangsvariablen nicht nur von den aktuellen Eingangsvariablen abhängt, sondern auch von früheren Zeitpunkten. Die Zustände geben dem Block also ein "Gedächtnis" und machen das TargetLink Modell damit zu einem dynamischen System. Für den Zustandsvektor muss ein Initialwert z_0 vorgegeben werden, um das mathematische Verhalten des Blockes eindeutig zu determinieren.

- eine Zahl von Parametern, die im Parametervektor $p \in P \subseteq \mathbb{R}^{n_p}$ des Blockes zusammengefasst sind und vom Anwender vorgegeben werden. Parameter sind Variablen, die während der immer wiederkehrenden Ausführung des Blockes überwiegend konstant sind, jedoch zu einzelnen (unbestimmten) Zeitpunkten vom Anwender explizit manuell abgeändert werden können. Sie sind also letztlich nicht wirklich konstant. Typischerweise wird für jeden Parameter ein Intervall vorgegeben, innerhalb dessen der Anwender den Wert des Parameters frei verstellen kann. Kalibrier-Parameter sind im automotiven Kontext weit verbreitet und dienen dazu, das "Finetuning" einer Steuerungs- und Regelungsfunktion durch Verstellen der Parameter vorzunehmen.

Naturgemäß weist ein Block B im Allgemeinen nicht alle Arten von Variablen auf. So besitzt ein Summationsblock beispielsweise ausschließlich Eingänge und Ausgänge, jedoch keine Zustandsvariablen oder Parameter. Ein Verzögerungsblock hingegen weist neben einer Eingangs- und Ausgangsvariablen auch eine Zustandsvariable auf, mit deren Hilfe die Verzögerung realisiert wird, siehe Abbildung 2.8 bzw. die nachfolgenden Beispiele.

Im Allgemeinen realisiert jeder Block B zwei mathematische Operationen g und h, wobei g die Berechnung des Ausgangsvektors y vornimmt (im folgenden als "Output-Funktion" bezeichnet), h hingegen den neuen Werte des Zustandsvektors z_{neu} berechnet (im folgenden als "Update-Funktion" bezeichnet):

$$\begin{array}{rlcll} g: & (X \times Z \times P) & \rightarrow & Y & \\ & (x,\, z,\, p) & \rightarrow & y & = \; g\,(x,\, z,\, p) \end{array} \tag{2.3}$$

$$\begin{array}{rlcll} h: & (X \times Z \times P) & \rightarrow & Z & \\ & (x,\, z,\, p) & \rightarrow & z_{neu} & = \; h\,(x,\, z,\, p) \end{array} \tag{2.4}$$

Die Simulink Semantik definiert dabei, dass die Output-Funktion g vor der Update-Funktion h ausgeführt wird, d.h. zunächst wird der Ausgang auf Basis der aktuellen Zustandsvariablen, Eingänge und Parameter berechnet und erst dann werden die Zustandsvariablen aktualisiert. Naturgemäß existieren nur für solche Blöcke B Output- bzw. Update-Funktionen, für die es überhaupt Ausgangs- bzw. Zustandsvariablen gibt.

Bemerkung 4. *Eine für die Verschaltung bzw. Auswertungsreihenfolge von Blöcken relevante Eigenschaft ist, ob die Output-Funktion g des Blockes wirklich eine explizite Abhängigkeit von einzelnen Eingangsvariablen hat (in der Simulink Terminologie als "Direct Feedthrough" bezeichnet) oder nicht. Im letzteren Fall gilt $y = g(z, p)$ und es besteht kein direkter Durchgriff von Eingängen auf den Ausgang. Darauf wird im Unterabschnitt 2.3.2 im Hinblick auf die Verschaltung von Blöcken genauer eingegangen.*

Beispiel 5. *Ein einfacher Summationsblock B_{Sum} summiert die einlaufenden Eingänge des Blockes und stellt an seinem Ausgang die Summe zur Verfügung. Der Block besitzt keine Parameter- oder Zustandsvariablen und demnach auch keine Update-Funktion, sondern lediglich eine Output-Funktion g_{Sum} in der folgenden Form, wobei g_{Sum} eine einfache Addition darstellt.*

$$\begin{array}{rccccc} g_{Sum}: & X & \rightarrow & Y & & \\ & x & \rightarrow & y & = & g_{Sum}(x) \end{array} \tag{2.5}$$

Damit hat jeder Eingang von B_{Sum} direkten Durchgriff auf den Ausgang.

Im nachfolgenden Beispiel eines Verzögerungsblockes wird der Charakter von TargetLink Modellen als dynamischen Systemen deutlich.

Beispiel 6. *Ein Verzögerungblock B_z ("Unit Delay") dient dazu, den Wert des Eingangssignals nicht unmittelbar am Ausgang des Blockes erscheinen zu lassen, sondern erst bei der nächsten Ausführung des Blockes B_z. Der Wert der Eingangsvariablen wird dazu in der Zustandsvariablen "zwischengespeichert" und erscheint bei der nächsten Ausführung des Blockes am Ausgang. Demnach besitzt der Block neben Eingangsvektor x und Ausgangsvektor y einen Zustandsvektor z sowie die folgenden Output- und Update-Funktion:*

$$\begin{array}{rccccc} g_z: & Z & \rightarrow & Y & & \\ & z & \rightarrow & y & = & z \\ h_z: & X & \rightarrow & Z & & \\ & x & \rightarrow & z_{neu} & = & x \end{array} \tag{2.6}$$

Damit hat der Verzögerungsblock kein "Direct-Feedthrough" vom Eingang auf den Ausgang, weshalb er sehr häufig zur Modellierung von sogenannten Rückkopplungschleifen genutzt wird, wie in den Beispielen in Kapitel 5 noch deutlich werden wird.

Weitere Beispiele für unterschiedliche Blockarten sind in Abbildung 2.8 aufgeführt. Insgesamt wird damit die Funktionalität eines TargetLink Blockes folgendermaßen definiert:

Definition 7. *Ein formaler TargetLink Block ist ein Tupel (g, h) bestehend aus einer Output-Funktion g entsprechend Gl. 2.3 und einer Update-Funktion h entsprechend Gl. 2.4. Für vorgegebene Eingangsvariablen, Parameter- und Zustandsvariablen werden mit Hilfe von g bzw. h die Ausgangsvariablen bzw. neuen Zustandsvariablen berechnet.*

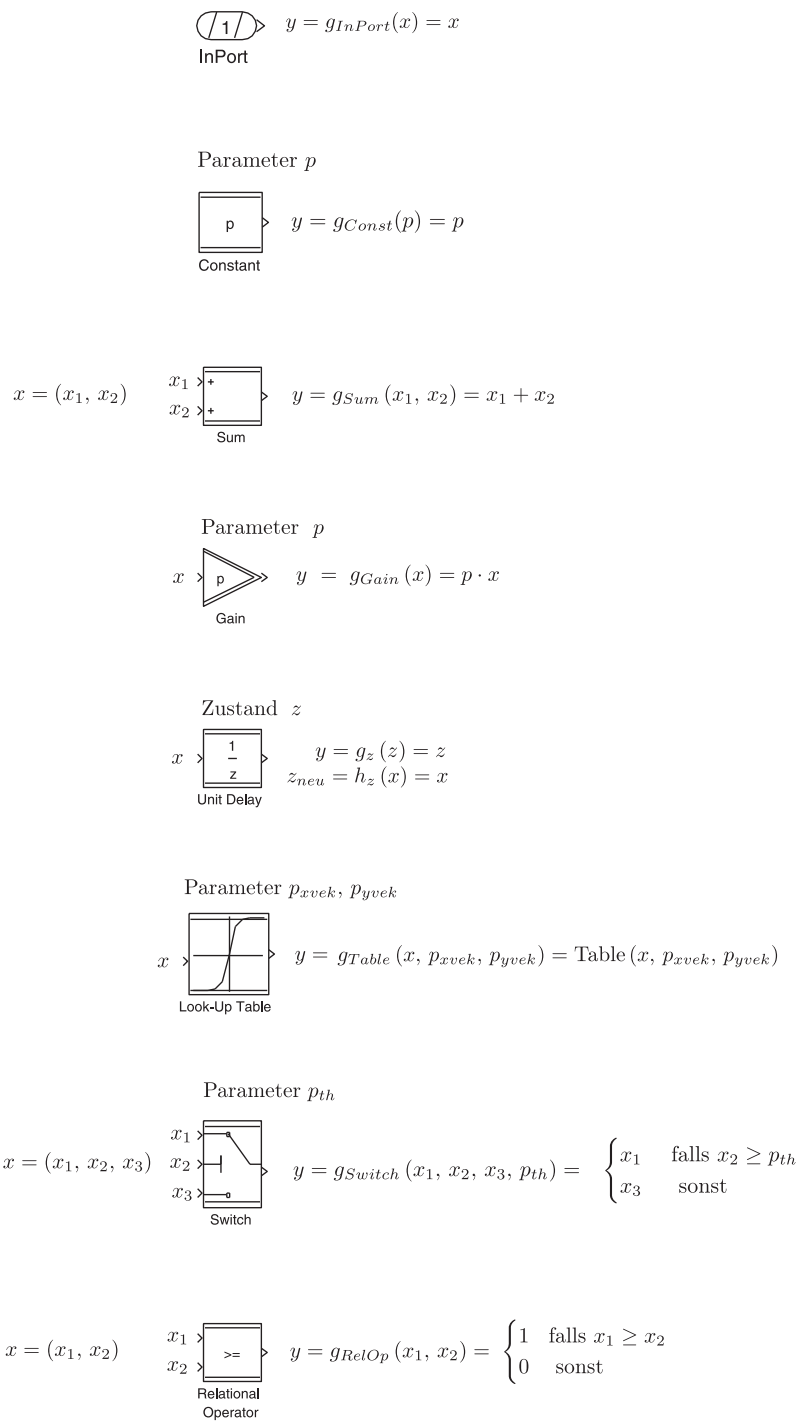


Abbildung 2.8: Beispiele für einzelne TargetLink Blöcke mit ihren Blockvariablen und zugehörigen Output- und Update-Funktionen g bzw. h: Das graphische Icon jedes Blockes zeigt die Art des Blockes an, von denen hier einige aufgelistet sind. Jeder Block innerhalb eines Modells weist zudem einen Blocknamen auf, der jeweils unter dem Block-Icon visualisiert ist und der den Block eindeutig identifiziert. Die unterschiedlichen Arten von dargestellten Blöcken sind (von oben nach unten): Port-Block *InPort* zur Aufteilung der Gesamtfunktionalität auf mehrere Subsysteme, Konstanten-Block *Constant,* Summations-Block *Sum*, Verstärkungs-Block *Gain*, Verzögerungs-Block *Unit Delay*, Tabellen-Block *Look-up Table*, Schalter-Block *Switch* und Vergleichs-Block *Relational Operator*. Teilweise sind die obigen Blöcke noch hinsichtlich ihrer Funktionalität vom Anwender konfigurierbar, etwa um eine andere als die dargestellte Vergleichsoperation im *Relational Operator* Block durchzuführen oder unterschiedliche Kriterien für das "Durchschalten" von Eingängen beim *Switch*-Block zu realisieren.

Bemerkung 8. *Es sei auch darauf hingewiesen, dass Simulink/TargetLink Modelle nicht nur rein mathematische Operationen sondern beispielsweise auch elementare Kontrollfluss-Strukturen ("If then Else") unterstützen, wie man sie von allen gängigen imperativen Programmiersprachen her kennt. Zustandsautomaten hingegen werden nicht direkt mit Hilfe von TargetLink-Blöcken beschrieben. Stattdessen gibt es eine in Simulink integrierte Entwicklungsplattform namens Stateflow, die zur graphischen Modellierung von Zustandsautomaten dient. Diese wird im Rahmen der Diplomarbeit nicht behandelt.*

2.3.2 Die Semantik eines Simulink/TargetLink Subsystems

In diesem Abschnitt soll nun ein komplettes Simulink/TargetLink Subsystem betrachtet werden, wie es exemplarisch in Abbildung 2.9 dargestellt ist und welches durch die im Rahmen der Diplomarbeit entwickelte Algorithmik analysiert werden soll. Der Analyse-Fokus für die robuste Abschätzung von Wertebereichsgrenzen bezieht sich auf alle in einem solchen Subsystem auftretenden Blöcke bzw. Variablen. Das Subsystem selbst beinhaltet eine Zahl von miteinander verschalteten TargetLink Blöcken, deren Semantik jeweils durch Definition 7 beschrieben wird, und die im Laufe einer Simulation nacheinander abgearbeitet werden. Ferner kann ein Subsystem in weitere Subsysteme unterteilt werden, um Hierarchien im Modell abzubilden, was in Abbildung 2.9 angedeutet ist und in der Praxis intensiv genutzt wird. Die zur Unterteilung eingeführten Subsysteme müssen naturgemäß in eine Wertebereichsanalyse des übergeordneten Subsystems einbezogen werden. Für die Eingangsblöcke (Inport-Blöcke) des Analyse-Subsystems gibt der Anwender Wertebereichsgrenzen vor, wohingegen diese für alle weiteren Variablen innerhalb des Subsystems von der entwickelten Algorithmik bestimmt werden sollen. Die in der Praxis eingesetzten, übergeordneten Subsysteme reichen hinsichtlich Ihrer Größe von kleinen Systemen mit bis zu 10 Eingangs- und Ausgangsvariablen und maximal 100 Blöcken bis hin zu großen Systemen mit einer 2-3 stelligen Zahl von Eingangs- und Ausgangssignalen und Tausenden von Blöcken.

Hinsichtlich der Semantik des Gesamtsystems ist zu beachten, dass dieses in einen konkreten zeitlichen Kontext gesetzt wird (in der generierten Software entspricht dies einem Software-Task), der die Ausführung des gesamten Systems verursacht. TargetLink unterstützt nur die Code-Generierung aus zeitdiskreten, dynamischen Systemen, d.h. die einzelnen Blöcke B_i des betrachteten Systems werden allesamt nacheinander zu definierten Zeitschritten k ausgeführt. In diesem zeitdiskreten Kontext stellen sich die Output- und Update-Funktionen der einzelnen Blöcke aus Gl. 2.3 und Gl. 2.4 folgendermaßen dar:

$$\begin{array}{rlcl} g_i: & (X_i \times Z_i \times P_i) & \rightarrow & Y_i \\ & (x(k),\, z(k),\, p(k)) & \rightarrow & y(k) \;=\; g_i\,(x(k),\, z(k),\, p(k)) \end{array} \tag{2.7}$$

$$\begin{array}{rlcl} h_i: & (X_i \times Z_i \times P_i) & \rightarrow & Z_i \\ & (x(k),\, z(k),\, p(k)) & \rightarrow & z(k+1) \;=\; h_i\,(x(k),\, z(k),\, p(k)) \end{array} \tag{2.8}$$

In jedem Zeitschritt k berechnet die Output-Funktion g_i jedes Blockes den aktuellen Wert des Ausgangsvektors $y(k)$ basierend auf dem aktuellen Eingangsvektor $x(k)$, Zustandsvektor $z(k)$ und Parametervektor $p(k)$ des Blockes. Die Update-Funktion h_i jedes Blockes ermittelt hingegen den Wert des Zustandsvektors z, den dieser im nächsten Zeitschritt $k+1$ annimmt, also $z(k+1)$.

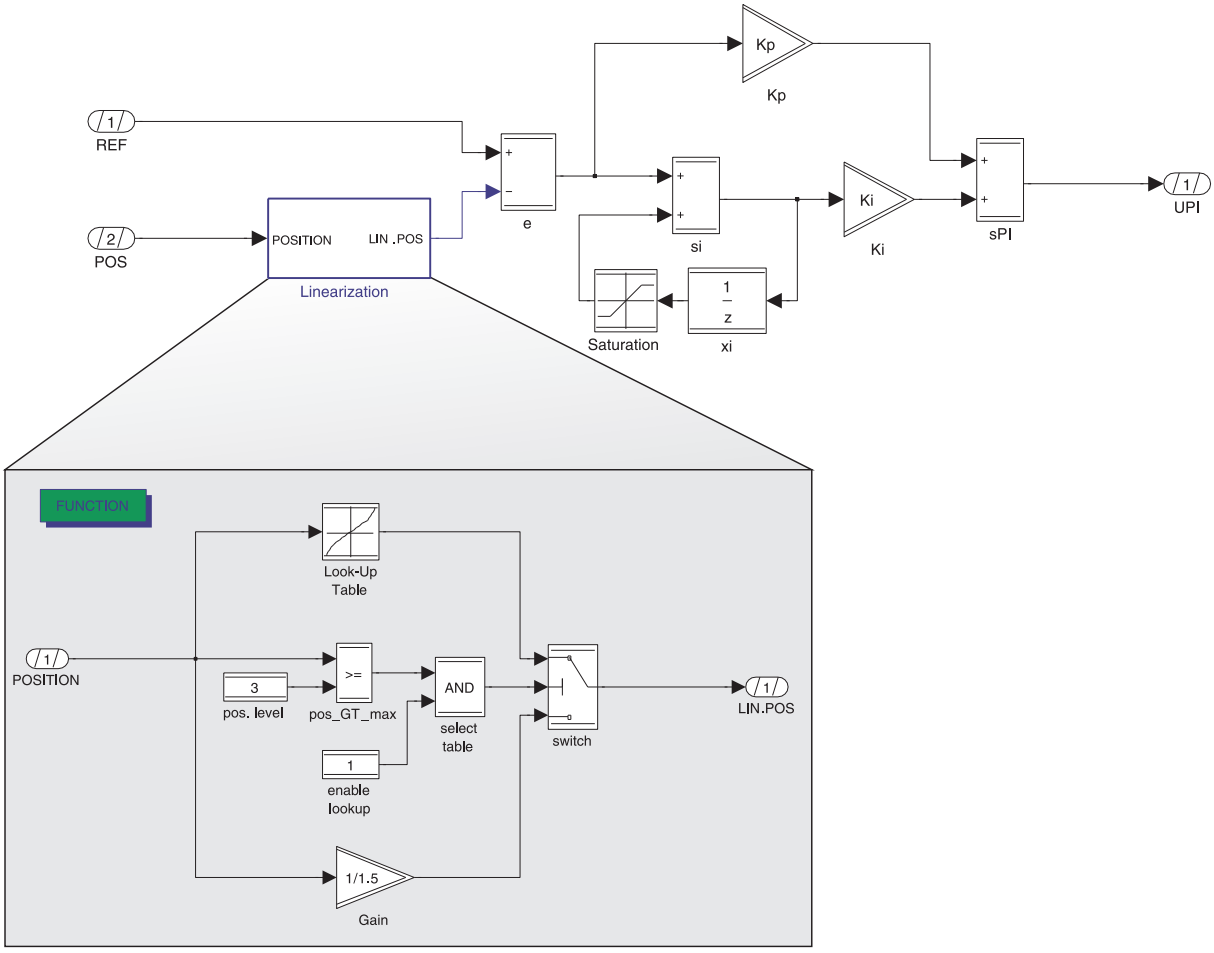


Abbildung 2.9: Ein einfaches Subsystem, wie es durch die im Rahmen der Diplomarbeit entwickelte Algorithmik analysiert werden soll. Das Subsystem weist zwei Eingangsblöcke namens *REF* und *POS* auf (sogenannte TargetLink Inports, siehe Abbildung 2.8 oben), für die Wertebereichsgrenzen vom Anwender vorgegeben werden müssen. Zur besseren Strukturierung kann die Funktionalität in weitere Subsysteme unterteilt werden, wie dies durch Einfügen des Subsystem *Linearization* inklusive der darin enthaltenen Funktionalität erfolgt ist. Die Blöcke *Position* und *Lin.Pos* innerhalb des *Linearization* Subsystems sind implizit mit den Ein- und Ausgängen gleichen Namens des *Linearization* Blockes in der übergeordneten Hierarchie verknüpft, wodurch der Signalfluss zwischen unterschiedlichen Hierarchieebenen definiert ist. Die TargetLink Blöcke sind untereinander entsprechend der benötigten Funktionalität verschaltet. Dies legt den Signalfluss von den Ausgängen eines Blockes zu seinen Nachverfolgern fest, nämlich in Richtung der Linienpfeile. Die Ausgangsvariable eines Blockes stellt damit gleichzeitig die Eingangsvariable seiner Nachfolgerblöcke dar.

Bemerkung 9. *Parameter werden ebenfalls mit einer expliziten Zeitabhängigkeit $p = p(k)$ modelliert und nicht als konstant angesehen, da die Parameterwerte zu undefinierten Zeitpunkten vom Anwender geändert werden können, wie bereits erwähnt. Damit werden Parameter in Bezug auf ihre zeitliche Abhängigkeit wie Eingangsvariablen $x(k)$ eines Blockes behandelt. Man beachte aber, dass Parameter rein blockspezifisch sind und durch ein externes Ereignis verändert werden.*

Die Verschaltung der einzelnen Blöcke zum Gesamtsystem, siehe Abbildung 2.9, definiert nun den Signalfluss zwischen den Blöcken. Wann immer eine Linie vom Ausgang eines Blockes B_i auf den Eingang eines nachfolgenden Blockes B_j führt, impliziert dies, dass dessen Eingangsvariable x_j den Wert der Ausgangsvariablen y_i des Blockes B_i zugewiesen bekommt. Jede Verbindung von Block B_i zu einem Nachfolger B_j hat also eine Zuweisung $x_j = y_i$ nach erfolgter Berechnung der Output-Funktion von Block B_i zur Folge. Ferner wird durch die Verschaltung der Blöcke bzw. den Signalfluss auch die Abarbeitungsreihenfolge der einzelnen Blöcke (bis auf irrelevante Unterschiede) determiniert. Die TargetLink Semantik besagt, dass die Output-Funktion g_j eines Block B_j in jedem Zeitschritt k erst dann ausgeführt werden kann, wenn alle laut Gl. 2.7 relevanten Größen bereits bekannt sind. Hat die Ausgangsvariable also eine tatsächliche Abhängigkeit von einer Eingangsvariablen ("Direct Feedthrough") so muss die Output-Funktion des entsprechenden Vorgänger-Blockes im aktuellen Zeitschritt k bereits berechnet worden sein. Hierdurch ist die Abarbeitungsreihenfolge (B_1, B_2, B_3, ..., B_n) aller Blöcke bis auf funktional-irrelevante Unterschiede vorgegeben.

Bemerkung 10. *Enthält das modellierte System Rückkopplungsschleifen, wie dies beispielsweise in Abbildung 2.9 für die Blockabfolge xI, Saturation, sI der Fall ist, so muss es mindestens einen Block in der Schleife geben, der kein "Direct Feedthrough" aufweist. Die Ausführungsreihenfolge der Blöcke beginnt dann bei diesem Block, da für diesen die Eingangssignale zur Berechnung der Output-Funktion g aufgrund des fehlenden "Direct Feedthroughs" noch nicht bekannt sein müssen. Im Beispiel in Abbildung 2.9 weist der xI Unit Delay Block kein "Direct Feedthrough" auf, siehe Beispiel 6.*

Im Folgenden soll nun noch die prinzipielle mathematische Beschreibung des aus der Verschaltung einer Vielzahl von Blöcken bestehenden Gesamtsystems angegeben werden. Dazu seien die Ausgangsvariablen, Parameter- und Zustandsvariablen aller Blöcke jeweils zu einem Blockausgangsvektor y_{sys}, Parametervektor p_{sys} und Zustandsvektor z_{sys} mit dem kartesischen Produkt der jeweiligen Definitionsbereiche als neuem Urbildbereich X_{sys}, Z_{sys}, bzw. P_{sys}, zusammengefasst. Die Eingangsvariablen der Inports des betrachteten Subsystems (in Abbildung 2.9 sind dies die Eingangsvariablen der Blöcke *REF* und *POS*) bilden den Eingangsvektor x_{sys} des Gesamtsystems (alle anderen individuellen Blockeingänge ergeben sich aus den Ausgangsgrößen anderer Blöcke). Auch hier ist der Urbildbereich X_{sys} wieder gegeben durch das kartesische Produkt der einzelnen Definitionsbereiche. Fass man nun die Output- und Update Funktionen g_i und h_i aller Blöcke zur vektorwertigen Gesamtfunktion g_{sys} bzw. h_{sys} zusammen und ersetzt jeweils iterativ die Eingangsvariablen durch die Ausgangsvariablen der Vorgängerblöcke so erhält man die prinzipielle Struktur des Gesamtsystems in der folgenden Form:

$$\begin{array}{rlcl} g_{sys}: & (X_{sys} \times Z_{sys} \times P_{sys}) & \to & Y_{sys} \\ & (x_{sys}(k),\, z_{sys}(k),\, p_{sys}(k)) & \to & y_{sys}(k) \;=\; g_{sys}\,(x_{sys}(k),\, z_{sys}(k),\, p_{sys}(k)) \end{array} \tag{2.9}$$

$$\begin{array}{rlcl} h_{sys}: & (X_{sys} \times Z_{sys} \times P_{sys}) & \to & Z_{sys} \\ & (x_{sys},\, z_{sys},\, p_{sys}) & \to & z_{sys}(k+1) \;=\; h_{sys}\,(x_{sys}(k),\, z_{sys}(k),\, p_{sys}(k)) \end{array} \tag{2.10}$$

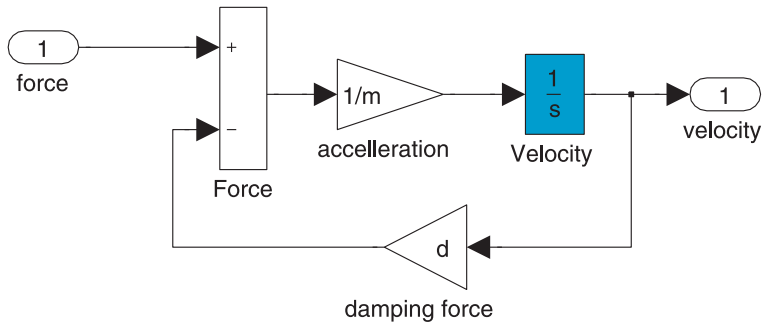


Abbildung 2.10: Zeitkontinuierliches System in Simulink mit Integrator Block (Block "Velocity", farbig markiert). Solche zeitkontinuierlichen Systeme werden oftmals für die Modellierung von Streckenmodellen, also der physikalischen Umgebung eines Reglers eingesetzt, nicht jedoch zur Modellierung des Reglers selbst. Der Regler muss als zeitdiskretes System in TargetLink modelliert werden, um daraus Code generieren zu können.

Durch Gl. 2.9 werden die Ausgangsvariablen aller Blöcke im Subsystembereich auf Basis der Eingangsvariablen des Subsystems, der aktuellen Zustandsvariablen und der Parameter berechnet. Gl. 2.10 beschreibt hingegen die Werte aller Zustandsvariablen des Systems im nächsten Zeitschritt $k+1$ in Abhängigkeit der aktuellen Eingangsvariablen, Zustände und Parameter. Die Struktur der Output- und Update-Funktion g_{sys} und h_{sys} des Gesamtsystems ergibt sich im Wesentlichen durch Verkettung der Output- und Update-Funktionen der Einzelblöcke entsprechend deren Verschaltung.

Definition 11. *Ein formales TargetLink System sei ein Tupel* (g_{sys}, h_{sys}) *bestehend aus einer System-Output-Funktion entsprechend Gl. 2.9 und einer System-Update-Funktion entsprechend Gl. 2.10. Für vorgegebene Eingangsvariablen, Parameter- und Zustandsvariablen werden mit Hilfe von* g_{sys} *bzw.* h_{sys} *die Ausgangsvariablen aller Blöcke bzw. die neuen Zustandsvariablen aller Blöcke berechnet.*

Bemerkung 12. *In Simulink können weit allgemeinere Systeme als die durch die Gleichungen 2.9 und 2.10 beschriebenen modelliert und simuliert werden, etwa zeitkontinuierliche Systeme, siehe Abbildung 2.10, hybride (d.h. gemischt zeitkontinuierliche und zeitdiskrete) System und solche, die in Form von impliziten transzendenten (Differential)-Gleichungen gegeben sind. Alle diese allgemeineren Systeme werden jedoch für den Einsatz in Steuerungs- und Regelungsfunktionen nicht eingesetzt, da sie beispielsweise nicht echtzeitfähig sind, zur Code-Generierung ohnehin diskretisiert werden müssen oder nicht die Anforderungen in sicherheitskritischen Anwendungen erfüllen.*

Die Simulink/TargetLink Semantik bezüglich der Ausführung des zeitdiskreten, dynamischen Systems stellt sich nun insgesamt folgendermaßen dar, siehe auch Abbildung 2.11:

- **Modellinitialisierung:** In einem initialen Schritt zu Beginn der eigentlichen Simulation bzw. Code-Generierung wird das Modell auf Modellierungsfehler überprüft, die Abarbeitungsreihenfolge der Blöcke ermittelt und anschließend das Modell in eine geeignete Datenstruktur zur Simulation bzw. Code-Generierung übersetzt. Insbesondere werden die Parameter p_{sys} des Gesamtsystems sowie die Initialwerte des Zustandsvektors $z_{sys,0}$ auf ihre initialen Werte gesetzt.

- **Berechnung der Output-Funktionen:** In jedem Zeitschritt k werden nun der Reihe nach die Output-Funktionen g_i der einzelnen Blöcke B_i ausgeführt, mithin also die Output-Funktion g_{sys} des Gesamtsystems entsprechend Gl. 2.9.

- **Berechnung der Update-Funktionen:** In jedem Zeitschritt k werden nun der Reihe nach die Update-Funktionen h_i der einzelnen Blöcke B_i ausgeführt, mithin also die Update-Funktion h_{sys} des Gesamtsystems entsprechend Gl. 2.10. Die Berechnungen beginnen jedoch erst, nachdem die Output-Funktion g_{sys} des Gesamtsystems vollständig berechnet wurde, siehe Abbildung 2.11.

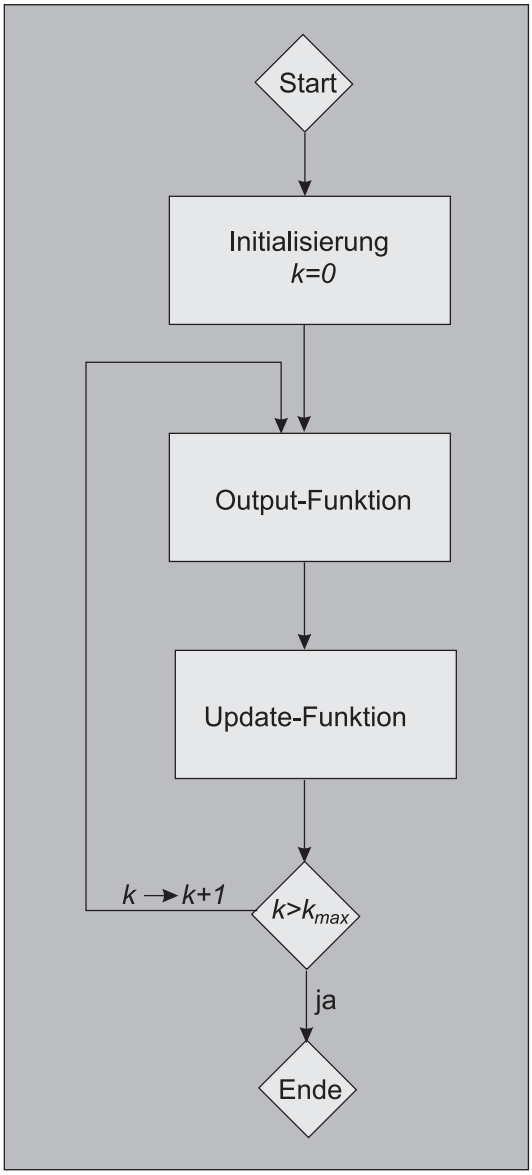


Abbildung 2.11: Ablaufdiagramm der Simulation in Simulink für zeitdiskrete Systeme. Während der Modellinitialisierung werden Parameterwerte und Zustandsvariablen auf ihre initialen Werte gesetzt. Anschließend erfolgt in jedem Zeitschritt k zunächst die Berechnung der Output-Funktion und anschließend der Update-Funktion des Systems. Für den aus einem solchen System generierten Code gilt Analoges, wobei für den auf einem Steuergerät ausgeführten Code naturgemäß $k_{max} = \infty$ gilt.

2.4 Präzise Formulierung der Aufgabenstellung zur Abschätzung von Wertebereichsgrenzen

Im Folgenden soll die Aufgabenstellung zur robusten Abschätzung der Wertebereiche in TargetLink Modellen exakt zusammengefasst und der Kontext zur Anwendung hergestellt werden. Wie bereits zuvor angeführt, ist der Analysefokus für die Wertebereichsabschätzung ein Subsystem, welches entsprechend Definition 11 bzw. anhand der Gleichungen 2.9 und 2.10 beschrieben ist. Um die Wertebereichsanalyse durchführen zu können, muss der Anwender nun zusätzlich folgende Daten spezifizieren:

- Für jede Komponente x_i des System-Eingangsvektors x_{sys} sei ein Intervall $[x_i^u, x_i^o]$ oder ein ganzzahliger Wertebereich $x_i^u \mathinner{..} x_i^o$ gegeben, so dass $x_i(k) \in [x_i^u, x_i^o]$ bzw. $x_i(k) \in \{x_i^u, ..., x_i^o\}$ für alle Zeitpunkte $k \in \mathbb{N}_0$ erfüllt ist. Es wird also von konkreten Eingangssignalen abstrahiert und lediglich gefordert, dass das Eingangssignal innerhalb des spezifizierten Bereiches liegt. Es muss ferner von einer Unabhängigkeit der Werte unterschiedlicher Komponenten von x_{sys} als auch der Werte einer Komponente zu unterschiedlichen Zeitpunkten ausgegangen werden, d.h. der Definitionsbereich ist das kartesische Produkt der einzelnen Intervalle. Es wird also konservativ abgeschätzt, um die geforderte Robustheit der Wertebereichsgrenzen zu erhalten. Ganzzahlige Wertebereiche $x_i(k) \in \{x_i^u, ..., x_i^o\}$ werden zugelassen, weil einzelne Signale diskreter Natur sein können, beispielsweise Bool'sche Variablen.

- Für jede Komponente p_i des Parametervektor p_{sys} sei ein Intervall $[p_i^u, p_i^o]$ oder ein ganzzahliger Wertebereich $p_i^u \mathinner{..} p_i^o$ gegeben, so dass $p_i(k) \in [p_i^u, p_i^o]$ bzw. $p_i(k) \in \{p_i^u, ..., p_i^o\}$ für alle Zeitpunkte $k \in \mathbb{N}_0$ gilt. In diesem Sinne werden Parameter also genau wie Eingangsvariablen behandelt. Da der Wert eines Parameters vom Anwender zu beliebigen Zeitpunkten verstellt werden kann muss auch hier von einer Unabhängigkeit der einzelnen Komponenten von p_{sys} als auch der Werte einer Komponente zu unterschiedlichen Zeitpunkten ausgegangen werden, was erneut zum kartesischen Produkt der Intervalle als Definitionsbereich führt.

- Für jede Komponente z_i des Vektors der Zustandsvariablen z_{sys} sei ein Intervall $\left[z_{i,0}^u, z_{i,0}^o\right]$ oder ein ganzzahliger Wertebereich $z_{i,0}^u \mathinner{..} z_{i,0}^o$ für den Initialwert $z_{i,0} = z_i(0)$ vorgegeben, also $z_{i,0} \in \left[z_{i,0}^u, z_{i,0}^o\right]$ bzw. $z_{i,0} \in \left\{z_{i,0}^u, ..., z_{i,0}^o\right\}$. Auch hier wird vom kartesischen Produkt der Intervalle als Definitionsbereich für den Initialwert des Zustandsvektors z_{sys} ausgegangen.

Bemerkung 13. *Der Spezialfall, dass etwa für einen Parameter p_i kein echtes Intervall sondern nur ein einzelner Wert vorgegeben wird, ist naturgemäß durch die obige Festlegung in Form eines Intervalls mit identischer oberer und unterer Grenze ebenfalls erfasst.*

In Abbildung 2.12 ist exemplarisch für das bereits bekannte Subsystem dargestellt, für welche Variablen Wertebereichsgrenzen vom Anwender vorgegeben werden müssen. Diese Spezifikationen werden für die einzelnen Blöcke direkt im Modell mit Hilfe von Blockdialogen vorgenommen, siehe Abbildung 2.13. Die im Rahmen dieser Diplomarbeit entwickelte Algorithmik greift dann auf die Modellbeschreibung inklusive der spezifizierten Extremalwerte zu und dient zur Ermittlung der folgenden Wertebereichsgrenzen:

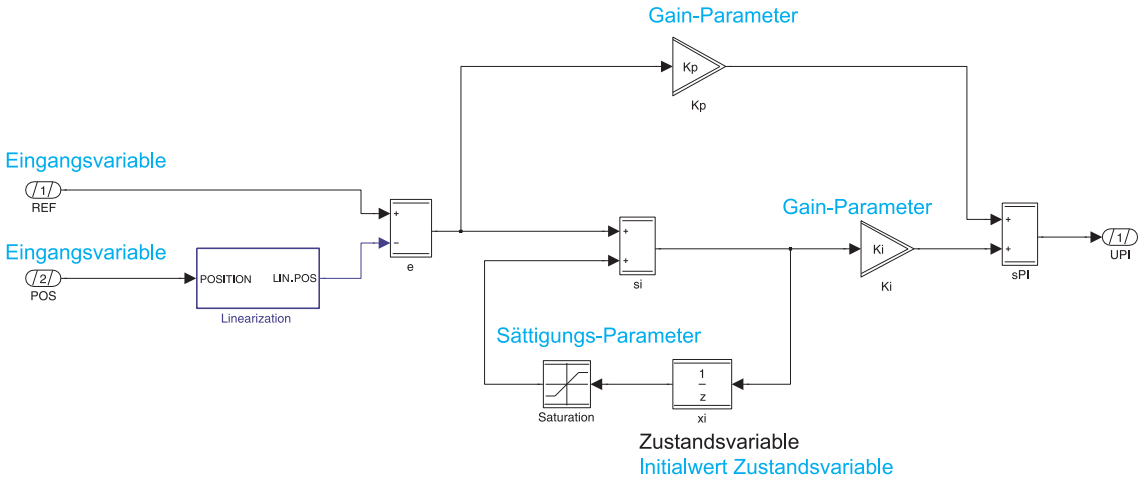


Abbildung 2.12: Die relevanten Variablen für die Abschätzung von Wertebereichsgrenzen des betrachteten Subsystems: Minimal- und Maximalwerte für die blau markierten Variablen müssen vom Anwender vorgegeben werden. Wertebereichsgrenzen für schwarz markierte Variablen und alle Blockausgangsvariablen, die nicht gesondert angegeben sind, werden von der vorgeschlagenen Algorithmik ermittelt.

- Für jede Komponente y_i des Ausgangsvektors y_{sys}, der wie beschrieben aus den Ausgangsvariablen aller Blöcke besteht, wird ein Intervall $[y_i^u,\, y_i^o]$ ermittelt, so dass $y_i(k) \in [y_i^u,\, y_i^o]$ für alle Zeitpunkte $k \in \mathbb{N}_0$ gilt.
- Für jede Komponente z_i des Zustandsvektors z_{sys} wird ein Intervall $[z_i^u,\, z_i^o]$ ermittelt, so dass $z_i(k) \in [z_i^u,\, z_i^o]$ für alle Zeitpunkte $k \in \mathbb{N}_0$ gilt.

Die obigen Ausführungen seien nun nochmal in der nachfolgenden Aufgabenstellung zusammengefasst.

Aufgabenstellung 14. *Gegeben sei ein TargetLink System entsprechend Definition 11 bzw. Gl. 2.9 und Gl. 2.10. Für die Komponenten des Eingangsvektors x_{sys}, Parametervektor p_{sys} und Initialwert des Zustandsvektors $z_{sys}(0)$ seien jeweils Wertebereichsgrenzen entsprechend Tabelle 2.1 vorgegeben, wobei sich der Gesamtdefinitionsbereich über alle Komponenten und Zeitpunkte k jeweils als kartesisches Produkt der einzelnen Definitionsbereiche ergebe. Dann besteht die Aufgabenstellung darin, robuste Wertebereichsgrenzen für die Komponenten des Ausgangsvektors y_{sys} und Zustandsvektors z_{sys} für alle Zeitpunkte $k \in \mathbb{N}_0$ entsprechend Tabelle 2.2 abzuleiten, wenn in jedem Zeitschritt $k \in \mathbb{N}_0$ zunächst die Output-Funktion 2.9 und dann die Update-Funktion 2.10 des Systems ausgeführt wird.*

Nach erfolgreich durchgeführter Abschätzung der Wertebereichsgrenzen hat der Anwender dann entsprechend Abschnitt 2.2 alle erforderlichen Informationen zur Hand, um das Festkomma-Implementierungsmodell des eigentlichen Algorithmus zu finalisieren und durch automatische Code-Generierung zu implementieren.

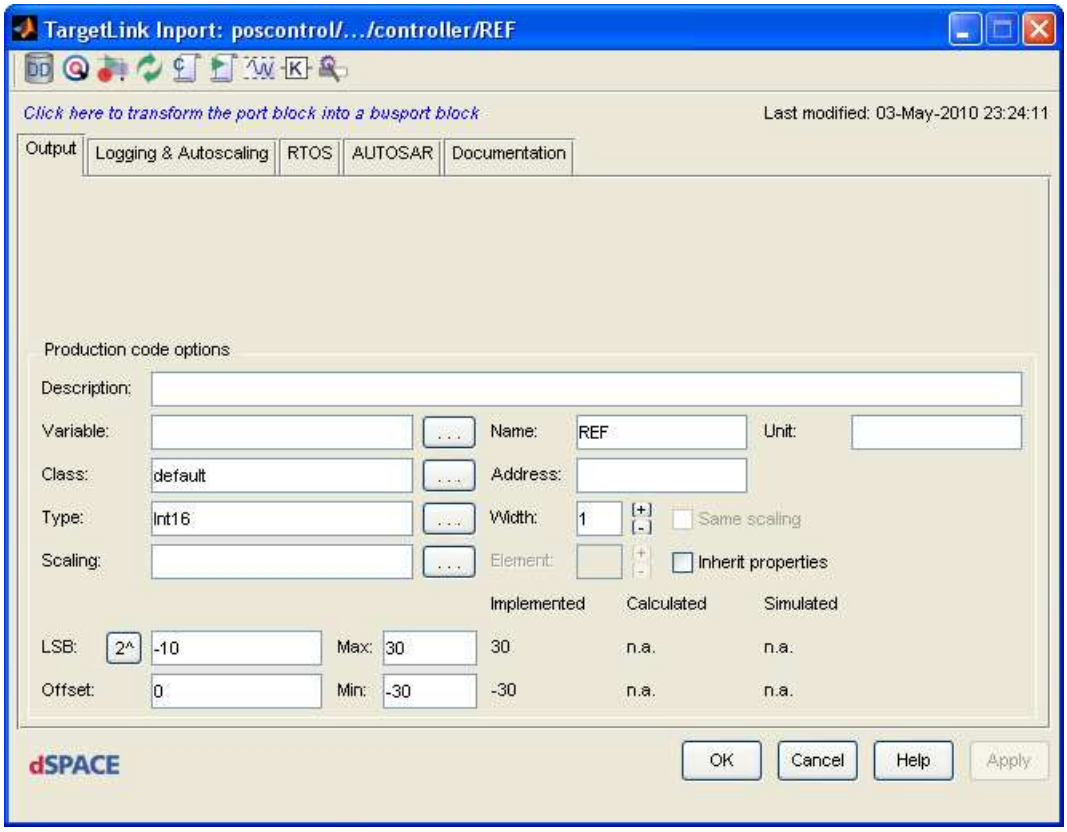


Abbildung 2.13: Vorgabe von Minimal- und Maximalwerten für die Eingangsvariable *REF* des betrachteten Subsystems aus Abbildung 2.12. Analoge Vorgaben muss der Anwender für alle Parameter und Initialwerte der Zustandsvariablen machen.

Variablen	Vorgaben des Anwenders
System-Eingangsvariablen x_i	$x_i(k) \in [x_i^u, x_i^o]$ oder $x_i(k) \in \{x_i^u, ..., x_i^o\}$
Parameter p_i	$p_i \in [p_i^u, p_i^o]$oder $p_i \in \{p_i^u, ..., p_i^o\}$
Initialwert Zustandsvariablen $z_i(0)$	$z_i(0) \in [z_i^u(0), z_i^o(0)]$oder $z_i(0) \in \{z_i^u(0), ..., z_i^o(0)\}$

Tabelle 2.1: Erforderliche Vorgaben zur Anwendung der Algorithmik: Für alle System-Eingangsvariablen x_i, Parameter p_i und Initialwerte der Zustandsvariablen $z_i(0)$ müssen vom Anwender Wertebereichsgrenzen vorgegeben werden, die wie in Abbildung 2.13 dargestellt, spezifiziert werden.

Variablen	Zu berechnende Wertebereichsgrenzen
Zustandsvariablen z_i	$z_i(k) \in [z_i^u, z_i^o]$
Blockausgangsvariablen y_i	$y_i(k) \in [y_i^u, y_i^o]$

Tabelle 2.2: Zu ermittelnde Wertebereichsgrenzen: Für alle Zustandsvariablen z_i und Blockausgangsvariablen y_i innerhalb des betrachteten Subsystems muss die vorgeschlagene Algorithmik Wertebereichsgrenzen in Form von Minimal- und Maximalgrenzen ableiten, die für alle $k \in \mathbb{N}_0$ Gültigkeit haben.

Kapitel 3

Grundlegende Hilfsmittel

In diesem Kapitel werden die wichtigsten Hilfsmittel zur Abschätzung der Wertebereiche für modellbasierte Designs bereitgestellt, wie sie im nachfolgenden Kapitel 4 zur Anwendung kommen. Grundlegend ist hierbei der Abschnitt 3.1 über einschließende Funktionen, für die der nachfolgende Abschnitt 3.2 über Intervall-Arithmetik eine konkrete Realisierung beschreibt. Der letzte Abschnitt 3.3 stellt dann noch einige Begriffe aus dem Bereich der Graphentheorie bereit, wie sie in Kapitel 4 bei der Beschreibung der eigentlichen Algorithmik Anwendung finden.

Bemerkung 15. *Die Darstellung in den Abschnitten 3.1 und 3.2 ist zu substanziellen Teilen aus den Quellen [Jau01], [Mo79] und [Mo09] entnommen. In vielen Fällen verwenden wir eingedeutschte Begriffe der Originalliteratur, wobei die englischen Begriffe ebenfalls angegeben sind.*

3.1 Wrapper und Einschließende Funktionen

Wie in Kapitel 4 gezeigt werden wird, besteht eine der Hauptaufgaben zur robusten Abschätzung von Extremalwerten aller Größen eines TargetLink Systems darin, das Bild $f(D)$ einer Funktion $f: D \subseteq \overline{\mathbb{R}}^{\mathrm{n}} \to f(D) \subseteq \overline{\mathbb{R}}^{\mathrm{m}}$ ($\overline{\mathbb{R}}^{\mathrm{n}}$ bezeichnet hierbei wie üblich $\mathbb{R}^{\mathrm{n}}$ mit Abschluss) durch eine Obermenge abzuschätzen, also eine Menge $\hat{B}$ zu identifizieren, mit $f(D) \subseteq \hat{B}$, wobei $\hat{B}$ im Sinne der Mengeninklusion naturgemäß möglichst klein sein soll. Die Vorgehensweise dazu ist, einerseits die Definitionsmenge D, die prinzipiell von sehr allgemeiner Gestalt sein kann, durch ein System von Mengen mit relativ einfacher Struktur zu überdecken und dann auf diesen Mengen eine Funktion $[f]$ zu definieren, die das Bild von f überdeckt, siehe Abbildung 3.1. Dies führt auf die Begriffe des Wrapper-Systems und der einschließenden Funktionen ("Enclosure Function" oder "Inclusion" Function), die im Wesentlichen aus [Jau01] entnommen sind.

Bemerkung 16. *Im Folgenden werden wir für eine Funktion $f: D \subseteq \overline{\mathbb{R}}^n \to \overline{\mathbb{R}}^m$ und eine gegebene Menge $D_i \subseteq \overline{\mathbb{R}}^n$ die abkürzende Schreibweise $f(D_i) := \{f(x) \mid x \in D_i \cap D\}$ verwenden. Gemeint ist also das Bild von f unter D_i wobei Punkte x mit $x \notin D$ unberücksichtigt bleiben, da f auf diesen Punkten nicht definiert ist. Falls $D_i \cap D = \emptyset$ gilt, wenn es also keine Überschneidung zwischen D_i und D gibt, so gilt $f(D_i) = \emptyset$.*

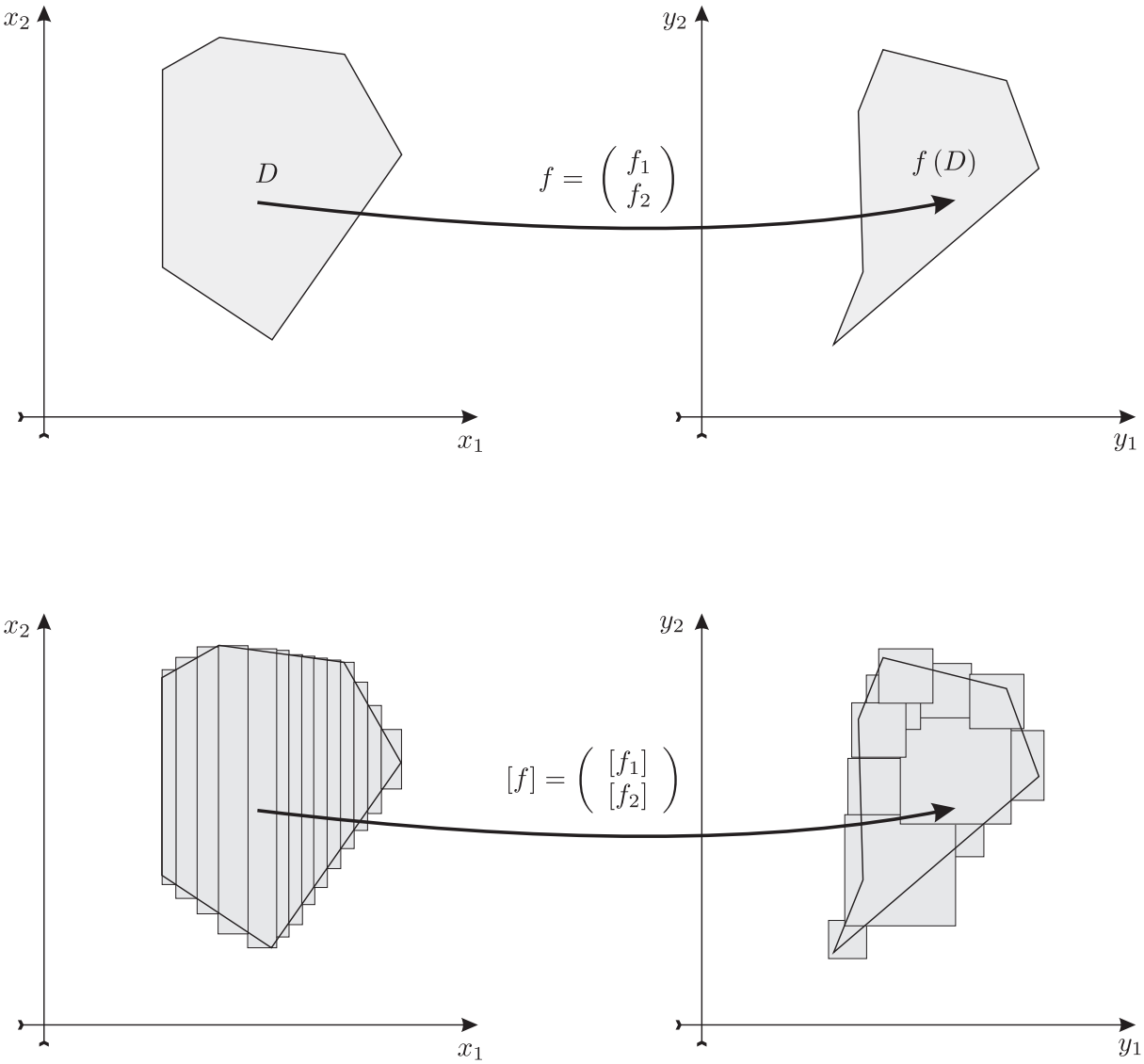


Abbildung 3.1: Oben: Allgemeine Abbildung f von einem Definitionsbereich D mit im Allgemeinen komplexer Berandung in den Zielbereich $f(D)$. Das Bild $f(D)$ weist im Allgemeinen naturgemäß eine komplexe Struktur auf. **Unten:** Überdeckung von D durch eine Menge von Wrapper-Mengen $\bigcup_i D_i \supseteq D$ einfacher Struktur sowie Definition einer einschließenden Funktion $[f]$ auf der Wrapper-Menge mit $[f](D_i) \supseteq f(D_i)$. Als Konsequenz der Überdeckung gilt dann $\bigcup_i [f](D_i) \supseteq f(D)$. Die eingezeichneten Wrapper-Mengen haben nur exemplarischen Charakter.

Definition 17. *Ein Mengensystem* $\overline{IR}^n = \left\{D_i\,(i \in I) \mid D_i \subseteq \overline{\mathbb{R}}^n\right\}$ *von Teilmengen von* $\overline{\mathbb{R}}^n$ *heißt ein Wrapper-System für* $\overline{\mathbb{R}}^n$*, falls folgende Eigenschaften erfüllt sind:*

a) $\{x\} \in \overline{IR}^n$ *für alle* $x \in \overline{\mathbb{R}}^n$ *(Singletons sind im Wrapper-System enthalten)*

b) $\overline{\mathbb{R}}^n \in \overline{IR}^n$ *und* $\emptyset \in \overline{IR}^n$

c) $D_i \cap D_j \in \overline{IR}^n$

Wie oben bereits dargestellt, werden wir Wrapper-Systeme verwenden, um beliebige Teilmengen des $\overline{\mathbb{R}}^n$ zu überdecken. Neben dem System von Intervallen und Boxen, die im nachfolgenden Abschnitt als eine spezielle Art von Wrapper-Systemen beschrieben werden, sind auch kompliziertere Wrapper-Systeme für $\overline{\mathbb{R}}^n$ denkbar und werden teilweise auch genutzt, wie zum Beispiel alle konvexen Teilmengen von $\overline{\mathbb{R}}^n$. Entscheidend ist naturgemäß, dass die Elemente des Wrapper-Systems strukturell einfacher sind als beliebige Teilmengen des $\overline{\mathbb{R}}^n$. Man beachte insbesondere, dass sich aufgrund der Definition 17 jede beliebige Menge $D \subseteq \overline{\mathbb{R}}^n$ durch Wrapper überdecken lässt, wenn auch nicht notwendigerweise "beliebig genau". Im Folgenden sollen nun Elemente eines Wrapper-Systems herausgegriffen werden, welche zur Überdeckung einer beliebigen Menge $D \subseteq \overline{\mathbb{R}}^n$ genutzt werden.

Definition 18. *Gegeben sei eine Menge* $D \subseteq \overline{\mathbb{R}}^n$ *und ein Wrapper-System* $\overline{IR}^n$ *von* $\overline{\mathbb{R}}^n$*. Dann heißt eine Teilmenge* $\breve{\mathfrak{D}} = \left\{D_i \mid D_i \in \overline{IR}^n (i \in I)\right\} \subseteq \overline{IR}^n$ *des Wrapper-Systems eine Überdeckung von* D*, falls* $D \subseteq \bigcup_i D_i \;\; (i \in I)$ *gilt.*

Damit ist die Basis zur Überdeckung des (möglicherweise komplex strukturierten) Definitionsbereiches einer Funktion durch einfachere Mengen gegeben. Die Überdeckung des Bildbereiches wird jetzt mit Hilfe von sogenannten einschließenden Funktionen realisiert, wie die folgende Definition zeigt.

Definition 19. *Gegeben sei eine Funktion* $f : D \subseteq \overline{\mathbb{R}}^n \to \overline{\mathbb{R}}^m$ *und Wrapper-Systeme* $\overline{IR}^n = \{D_i\,(i \in I)\}$ *von* $\overline{\mathbb{R}}^n$*. Dann nennen wir eine Funktion* $[f] : \overline{IR}^n \to \overline{IR}^m$ *also eine auf dem Wrapper-System erklärte Mengenfunktion eine einschließende Funktion ("Inclusion function", "Enclosure Function") zu* f *falls gilt:*

$$f\,(D_i) \subseteq [f]\,(D_i) \;\; \forall D_i \in \overline{IR}^n \tag{3.1}$$

Die Funktion $[f]$ *heißt ferner inklusions-monoton ("inclusion monotonic") falls aus* $D_i \subseteq D_j$ *grundsätzlich* $[f]\,(D_i) \subseteq [f]\,(D_j)$ *folgt und zwar* $\forall D_i \in \overline{IR}^n$, $\forall D_j \in \overline{IR}^n$.

Dabei ist $f\,(D_i)$ als $f\,(D_i) := \{f\,(x) \mid x \in D_i \cap D\}$ im Sinne von Bemerkung 16 zu interpretieren. $[f]\,(D_i)$ ist hingegen einfach der Funktionswert der Mengenfunktion $[f]$ an der "Stelle" D_i und selbst ein Element von $\overline{IR}^m$. Das Bild von f auf der Wrapper-Menge D_i wird also durch den Funktionswert der einschließenden Funktion vollständig überdeckt.

Durch einschließende Funktionen im Sinne von Definition 19 ist in Verbinduung mit Wrappern die Basis geschaffen worden, um das Bild $f\,(D)$ einer unter Umständen komplizierten Funktion auf einem gegebenenfalls kompliziert strukturierten Definitionsbereich D konservativ, d.h. robust abzuschätzen, und hierfür die unter Umständen einfacher strukturierten Funktionen $[f]$ und einfacher strukturierten Mengen D_i zu nutzen. Dies wird in dem nachfolgenden, sehr einfachen Satz über die Überdeckung des Bildes $f\,(D)$ von Funktionen deutlich werden.

Satz 20. *Betrachtet werde die Funktion $f : D \subseteq \overline{\mathbb{R}}^n \to \overline{\mathbb{R}}^m$ und Wrapper-Systeme $\overline{IR}^n$ von $\overline{\mathbb{R}}^n$ bzw. $\overline{IR}^m$ von $\overline{\mathbb{R}}^m$. Für den Definitionsbereich D sei eine Überdeckung durch Wrapper D_i gegeben, es gelte also $D \subseteq \bigcup_i D_i$. Ferner sei $[f]$ eine einschließende Funktion zu f. Dann lässt sich das Bild $f(D)$ durch die Überdeckungsmengen D_i und die einschließende Funktion $[f]$ abschätzen, d.h. es gilt $f(D) \subseteq \bigcup_i [f](D_i)$.*

Beweis. Vorausgesetzt wird, dass die ursprüngliche Definitionsmenge D von den Wrapper-Mengen D_i überdeckt wird, also $D \subseteq \bigcup_i D_i$ gilt. Damit ergibt sich nun

$$\begin{array}{rcll} f(D) & = & \{f(x) \mid x \in D\} & = \; f(\bigcup_i D_i) \\ & = & \{f(x) \mid x \in \bigcup_i D_i \;\wedge f(x) \text{ ist wohldefiniert}\} & = \; \bigcup_i f(D_i) \\ & \subseteq & \bigcup_i [f](D_i) & \end{array}$$

Damit ist der Erhalt der Überdeckungseigenschaft beim Rechnen mit Wrappern sichergestellt. Man beachte, dass f nicht in jedem Punkt der einzelnen Wrapper-Mengen D_i definiert zu sein braucht und diese Forderung im Allgemeinen auch nicht erfüllt ist. Unter Umständen, wenn etwa $D \cap D_i = \emptyset$ gilt, kann $f(D_i) = \emptyset$ gelten. Ein solches Element des Wrapper-Systems könnte dann jedoch auch selbstverständlich aus der Überdeckung von D entfernt werden, da es nichts beiträgt. □

In der Praxis besteht naturgemäß das Problem, dass zu einer komplex strukturierten Funktion f nicht ad-hoc eine einschließende Funktion $[f]$ bekannt ist. Häufig ist es jedoch so, dass eine kompliziertere Funktion f aus einfacheren Funktionen zusammengesetzt ist. Das ist etwa für Simulink/TargetLink Modelle, wo einzelne Blöcke mit elementarer Funktionalität zu einer komplexen Gesamtfunktionalität verschaltet sind, naturgemäß der Fall. Wir geben daher zunächst in der nachfolgenden Definition dieser Klasse von zusammengesetzten Funktionen einen Namen.

Definition 21. *Eine Funktion f, die sich aus der Verschaltung von endlich vielen, elementaren TargetLink Blockfunktionen zusammensetzt, werde im Folgenden als faktorisierte Funktion bezeichnet. Die Abbildungsvorschriften von f hat dann folgende, rekursive Gestalt*

$$f(x_1, ..., x_n) = f_{el}(f_{fa,1}(x_1, ..., x_n), ..., f_{fa,m}(x_1, ..., x_n))$$

wobei f_{el} die Funktionalität eines TargetLink Blockes darstellt und $f_{fa,i}$ selbst wieder faktorisierte Funktionen oder elementare TargetLink Blockfunktionen darstellen.

Die Abbildungsvorschrift einer faktorisierten Funktion wird also rekursiv mit Hilfe von Unterausdrücken definiert, wobei die Komposition bzw. das rekursive Einsetzen als Funktionsargument zum Tragen kommt. Im nachfolgenden Satz 22 wird nun noch festgehalten, dass zu einer faktorisierten Funktion f eine einschließende Funktion $[f]$ sehr einfach auf Basis von einschließenden Funktionen der Subausdrücke bzw. der elementaren TargetLink Blockfunktionen, aus denen f besteht, konstruiert werden kann und so auch die Inklusions-Monotonie vererbt wird.

Satz 22. *Sei f eine faktorisierte Funktion im Sinne von Definition 21, d.h. darstellbar als endliche Kombination von elementaren Funktionen f_i, für die jeweils eine einschließende Funktion $[f_i]$ existiere. Dann kann zu f eine einschließende Funktion $[f]$ konstruiert werden, indem in der formelmäßigen Zerlegung von f die elementaren Funktionen f_i durch die entsprechenden einschließenden Funktionen $[f_i]$ ersetzt werden. Sind zudem alle $[f_i]$ inklusions-monoton, so gilt dies auch für die entsprechend konstruierte Funktion.*

Beweis. Wir nehmen an, dass eine rekursive Darstellung von f in der Form

$$f(x_1, ..., x_n) = f_{el}(f_{fa,1}(x_1, ..., x_n), ..., f_{fa,m}(x_1, ..., x_n))$$

mit faktorisierten Funktionen $f_{fa,i}$ und einer elementaren Funktion f_{el} gegeben ist und unterstellen induktiv, dass es zu $f_{fa,i}$ bzw. f_{el} einschließende Funktionen $[f_{fa,i}]$ bzw. $[f_{el}]$ gibt. Dann folgt daraus

$$\begin{aligned} f(D_i) &= f_{el}(f_{fa,1}(D_i), ..., f_{fa,m}(D_i)) \\ &\subseteq [f_{el}]([f_{fa,1}](D_i), ..., [f_{fa,m}](D_i)) \end{aligned}$$

Folglich ist $[f_{el}]([f_{fa,1}], .., [f_{fa,m}])$ eine einschließende Funktion zu $f = f_{el}(f_{fa,1}, .., f_{fa,m})$, woraus sich bei rekursiver Anwendung der Inhalt des Satzes ergibt. Analog wird für die Inklusions-Monotonie argumentiert. □

Wir werden in Abschnitt 4.3.1 sehen, dass für die elementaren Funktionen von TargetLink Blöcken in der Tat sinnvolle, einschließende Funktionen konstruiert werden können, woraus sich dann mit Hilfe von Satz 22 eine einschließende Funktion für das gesamte TargetLink System ergibt.

3.2 Intervall-Analysis

Im vorherigen Abschnitt wurde die Überdeckung von Teilmengen des $\overline{\mathbb{R}}^n$ bzw. des Bildbereiches von Funktionen durch Wrapper-Systeme und einschließende Funktionen in großer Allgemeinheit dargestellt. Die naheliegendste und einfachste Art der Realisierung für Wrapper und einschließende Funktionen sind Intervalle bzw. Boxen als Wrapper-Systeme und sogenannte Intervallerweiterungen als einschließende Funktionen. Dies sind die essentiellen Hilfsmittel, wie sie zur robusten Abschätzung von Wertebereichsgrenzen für TargetLink-Systeme in Kapitel 4 eingesetzt werden.

3.2.1 Intervalle und Boxen

Wir geben nun zuerst eine Definition für das Wrapper-System der Intervalle bzw. Boxen als einer besonders einfachen Realisierungsmöglichkeit für Überdeckungen.

Definition 23. ***Intervalle und Boxen***

a) Unter einem Intervall $[x_1]$ verstehen wir eine zusammenhängende Teilmenge von $\mathbb{R}$, konkret die Menge $[x_1] = [\underline{x_1}, \overline{x_1}] = \left\{x \in \overline{\mathbb{R}} \mid \underline{x_1} \leq x \leq \overline{x_1}\right\}$ mit unterer Grenze $\underline{x} = lb([x])$ und oberer Grenze $\overline{x} = ub([x])$, siehe Abbildung 3.2 links. Unter der Breite $w([x])$ des Intervalls sei wie üblich die reelle Zahl $w([x]) = (\overline{x} - \underline{x})$ verstanden.

b) Wir bezeichnen die Menge aller reellen Intervalle $[x] = [\underline{x}, \overline{x}]$ mit $\underline{x} \in \overline{\mathbb{R}}$ und $\overline{x} \in \overline{\mathbb{R}}$ mit dem Symbol $\overline{IR}$. Offensichtlich stellt $\overline{IR}$ ein Wrapper-System für $\overline{\mathbb{R}}$ im Sinne von Definition 17 dar, weshalb $\overline{IR}$ als Symbol verwandt wird.

c) Für n gegebene Intervalle $[x_1] \in \overline{IR}$, $[x_2] \in \overline{IR}$ bis $[x_n] \in \overline{IR}$ bezeichnen wir das kartesische Produkt $[\mathbf{x}] = [x_1] \times [x_2] ... \times [x_n]$ als eine Box. Diese stellt das mehrdimensionale Analogon zu Intervallen dar, siehe Abbildung 3.2 rechts.

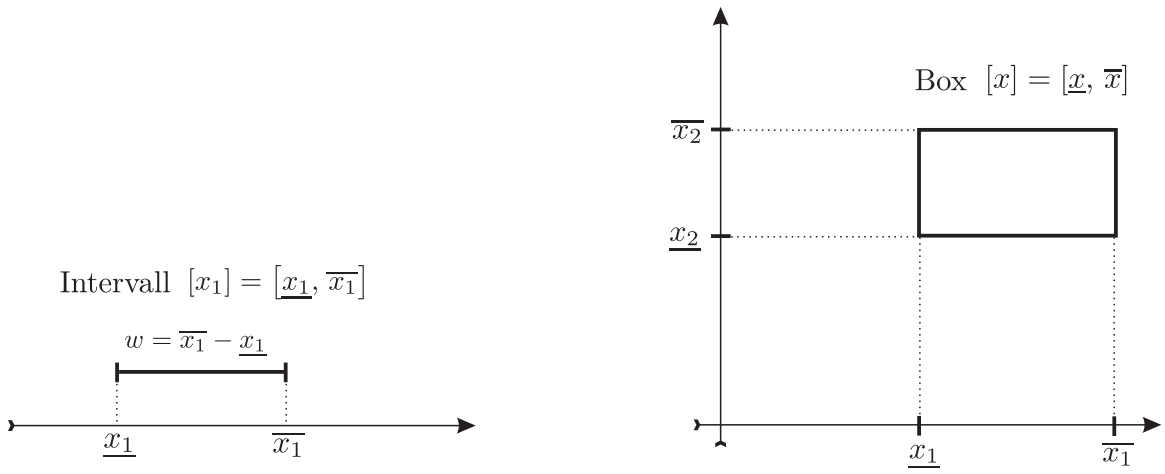


Abbildung 3.2: Links: Intervall $[x_1]$ als zusammenhängende Teilmenge der reellen Zahlen. **Rechts:** Eine Box als kartesisches Produkt von Intervallen. Da Boxen das multidimensionale Analogon zu Intervallen darstellen, wird dieselbe Nomenklatur $[x]$ für Boxen und Intervalle verwendet.

d) Die Menge aller Boxen werde im Folgenden als $\overline{IR}^n$ *bezeichnet. Dabei sei* $\overline{IR}^1 = \overline{IR}$. *Offensichtlich stellt* $\overline{IR}^n$ *ein Wrapper-System für* $\overline{\mathbb{R}}^n$ *im Sinne von Definition 17 dar, weshalb dasselbe Symbol* $\overline{IR}^n$ *verwandt wird.*

e) Jede individuelle reelle Zahl $x \in \mathbb{R}$ *wird als Intervall* $[x, x] \in \overline{IR}$ *der Breite* $w([x, x]) = 0$ *aufgefasst. Analoges gilt für die Elemente von* $\mathbb{R}^n$, *die als Boxen mit dem Volumen 0 angesehen werden. Durch diese naheliegende Festsetzung lassen sich insbesondere die nachfolgenden arithmetischen Operationen und Funktionen auf gemischten Ausdrücken für reellwertige Zahlen und Intervalle bzw. reellwertige Vektoren und Boxen erklären.*

Mit Definition 23 wird eine spezielle Form eines Wrapper-Systems als Konkretisierung von Definition 17 vorgeschlagen, nämlich die Nutzung von Intervallen bzw. Boxen zur Überdeckung von beliebigen Teilmengen von $\overline{\mathbb{R}}$ bzw. $\overline{\mathbb{R}}^n$. Diese Art der Wrapper haben den Vorteil, dass sie strukturell besonders einfach sind und dass sich auf ihnen damit auch sehr einfach einschließende Funktionen erklären lassen, was im nachfolgenden Abschnitt dargestellt werden soll.

3.2.2 Intervallfunktionen und Intervallerweiterungen

In diesem Abschnitt sollen Intervallfunktionen ("Interval Functions") und die sogenannten Intervallerweiterungen ("Interval Extensions") zu gegebenen reellwertigen Funktionen diskutiert werden. Erstere sind Mengenfunktionen, die als Bild grundsätzlich wieder Intervalle, also Elemente aus $\overline{IR}$ liefern. Intervallfunktionen haben im Vergleich zu allgemeinen Mengenfunktionen den Vorteil, dass sie sich sehr einfach beschreiben lassen, nämlich lediglich auf Basis von Intervallgrenzen. Zur konkreten Konstruktion einer Intervallfunktion zu einer vorgegebenen reellwertigen Funktion f wird die nachfolgende Definition 24 sowie das Konstruktionsprinzip 25 hinzugezogen.

Definition 24. *Generell bezeichnen wir eine Mengenfunktion* $[f] : \overline{IR}^n \to \overline{IR}$ *also eine Funktion, die auf Intervallen/Boxen operiert und als Bilder Intervalle liefert, als Intervallfunktion. Zusätzlich heißt eine Intervallfunktion* $[f] : \overline{IR}^n \to \overline{IR}$ *eine Intervallerwei-*

terung zur reellwertigen Funktion $f : D \subseteq \overline{\mathbb{R}}^n \to \overline{\mathbb{R}}$, falls $\{f(x)\} = [f]([x, x])$ für alle $x \in D$ gilt, wenn also die Funktionswerte auf allen Boxen des Voluments $w([x, x]) = 0$ übereinstimmen.

Prinzip 25. *Konstruktionsprinzip für Intervallerweiterungen*

Die Intervallerweiterung $[f] : \overline{IR}^n \to \overline{IR}$ zu einer vorgegebenen Funktion $f : D \subseteq \overline{\mathbb{R}}^n \to \overline{\mathbb{R}}$ wird im Allgemeinen so festgelegt, dass

$$[f]([x]) = [y] \supseteq f([x])$$

gilt und $[y]$ im Sinne der Mengeninklusion minimal ist, dass also für alle $[\tilde{y}] \supseteq f([x])$ immer auch $[\tilde{y}] \supseteq [y]$ gilt.

Bemerkung 26. *Wir betrachten hier nur reellwertige Funktionen, also Abbildungen $f : D \subseteq \overline{\mathbb{R}}^n \to \overline{\mathbb{R}}$ wohingegen die Ausführungen im vorherigen Abschnitt den allgemeineren Fall von Abbildungen in den $\overline{\mathbb{R}}^m$ adressiert haben, siehe Definition 19. Der Grund ist naturgemäß, dass wir solche Funktionen in ihre einzelnen Komponentenfunktionen zerlegen können. In Abschnitt 3.2.7 wird darauf eingegangen, wie der allgemeinere Fall des Bildbereiches $\overline{\mathbb{R}}^m$ durch Intervallerweiterungen erfasst wird und was hierbei zu beachten ist.*

Im Allgemeinen ist eine Intervallerweiterung durch Definition 24 natürlich noch nicht eindeutig festgelegt, schließlich wird dort nur der Wert auf degenerierten Intervallen vorgegeben. Erst durch Heranziehung des Konstruktionsprinzips 25 ergeben sich konkrete Intervallerweiterungen, welche die folgenden Eigenschaften aufweisen:

Satz 27. Eigenschaften von Intervallerweiterungen

a) Intervallerweiterungen zu elementaren Funktionen, die entsprechend der obigen Konstruktionsvorschrift erzeugt werden. sind inklusions-monoton. Faktorisierte Funktionen aus diesen elementaren Funktionen sind gleichfalls inklusions-monoton.

b) Intervallerweiterungen sind einschließende Funktionen im Sinne von Definition 19.

Beweis. Zu a) Die Inklusions-Monotonie für elementare Funktionen ergibt sich direkt aus dem Konstruktionsprinzip, das bei fehlender Inklusions-Monotonie offensichtlich verletzt würde. Die Inklusions-Monotonie für faktorisierte Funktionen folgt dann direkt aus Satz 22. Zu b) Aufgrund der Definition der Intervallerweiterung gilt $\{f(x)\} = [f]([x, x])$ für alle $x \in [y]$ und woraus sich dann mit der Inklusions-Monotonie von $[f]$ die Einschließungs-Eigenschaft $f([y]) \subseteq [f]([y])$ für alle $[y] \in \overline{IR}^n$ ergibt. □

3.2.3 Intervallerweiterungen für Elementare Operationen und Funktionen

Nachfolgend sollen nun für einige elementare Operationen und Funktionen Intervallerweiterungen angegeben werden, auf die wir teilweise in Abschnitt 4.3.1 zurückkommen werden.

Definition 28. *Für eine arithmetische Operation* $\otimes : \mathbb{R}\times\mathbb{R} \to \mathbb{R}$*, wobei* $\otimes$ *die gewöhnliche Summation, Subtraktion oder Multiplikation auf den reellen Zahlen repräsentiert, definieren wir die folgenden Intervallerweiterungen* $[\otimes] : IR\times IR \to IR$*, wobei* $[x] = [\underline{x}, \overline{x}] \in \overline{IR}$ *und* $[y] = [\underline{y}, \overline{y}] \in \overline{IR}$ *gelte:*

$$\begin{array}{lcl} [x] + [y] & = & \left[\underline{x} + \underline{y},\ \overline{x} + \overline{y}\right] \\ [x] - [y] & = & \left[\underline{x} - \overline{y},\ \overline{x} - \underline{y}\right] \\ [x] \cdot [y] & = & \left[\min\left(\underline{x}\cdot\underline{y}, \underline{x}\cdot\overline{y}, \overline{x}\cdot\underline{y}, \overline{x}\cdot\overline{y}\right), \max\left(\underline{x}\cdot\underline{y}, \underline{x}\cdot\overline{y}, \overline{x}\cdot\underline{y}, \overline{x}\cdot\overline{y}\right)\right] \end{array}$$

Bemerkung 29. *Auf die Division gehen wir in Bemerkung 32 genauer ein.*

Offensichtlich liefert die Definition 28 einschließende Funktionen für die gewöhnlichen arithmetischen Operation der reellen Zahlen. Darüber hinaus überprüft man leicht, dass die getroffene Wahl optimal ist, d.h. es gibt entsprechend der obigen Forderung kein kleineres Intervall, dass ebenfalls die Einschließungseigenschaft aufweist. Für eine weitere Klasse von Funktionen ergibt sich ebenfalls eine unmittelbare Festsetzung der Intervallerweiterung.

Definition 30. *Für eine stetige, monotone Funktion* $f : \mathbb{R} \to \mathbb{R}$ *und Intervalle* $[x] = [\underline{x}, \overline{x}]$ *definieren wir die zugehörige Intervallerweiterung* $[f] : \overline{IR} \to \overline{IR}$ *entsprechend der folgenden Abbildungsvorschrift*

$$[f]([x]) = \left[\min\left(f(\underline{x}), f(\overline{x})\right), \max\left(f(\underline{x}), f(\overline{x})\right)\right]$$

Offensichtlich handelt es sich bei $[f]$ in der Tat um eine Intervallerweiterung. Darüber hinaus ist $[f]$ offensichtlich eine einschließende Funktion und es gilt sogar $f([x]) = [f]([x])$, d.h. die Bildmenge des Intervalls $[x]$ unter f ist mit $[f]([x])$ identisch. Um nun auch für komplexer strukturierte, faktorisierte Funktionen f zugehörige Intervallerweiterungen zu konstruieren, wird auf Satz 22 zurückgegriffen. So zusammengesetzte Intervallerweiterungen sind naturgemäß im Allgemeinen nicht mehr optimal, sondern liefern ein Überschätzen. Wir kommen im Abschnitt 4.3.1 noch auf die Konstruktion von weiteren elementaren Intervallerweiterungen zurück und wollen es an dieser Stelle dabei bewenden lassen.

3.2.4 Intervallerweiterungen und Lücken im Definitionsbereich

Im Folgenden soll nun noch der Umgang mit "Lücken" im Definitionsbereich der zu überdeckenden Funktionen präzisiert werden. In vielen Situationen ist die Funktion f aus Definition 24 nicht auf ganz $\mathbb{R}$ bzw. $\mathbb{R}^n$ definiert sondern nur auf einer Teilmenge D wie beispielsweise die reellwertige Quadratwurzelfunktion $\mathrm{sqrt} : \mathbb{R}_0^+ \to \mathbb{R}$. Im Zuge der Überdeckung durch Wrapper und Intervallerweiterungen tritt dann gelegentlich die Situation ein, dass die Wrapper-Mengen auch solche Bereiche überdecken, auf denen f nicht mehr definiert ist. In Analogie zu Definition 24 bzw. Definition 19 wird die Intervallerweiterung dann auch hier so festgelegt, dass $\{f(x)\} = [f]([x, x])$ bzw. $f([y]) \subseteq [f]([y])$ eingehalten wird, wobei Lücken im Definitionsbereich einfach ignoriert werden, weil auf ihnen keine Bilder von f existieren. Man beachte hierzu auch erneut Bemerkung 16 im Hinblick auf die Interpretation des Ausdruckes $f([y]) = \{f(x) \mid x \in [y] \cap D\}$. Dementsprechend nimmt die Intervallerweiterung $[\mathrm{sqrt}] : IR \to IR$ beispielsweise die Werte $[\mathrm{sqrt}]([-3, 4]) = [0, 2]$ und $[\mathrm{sqrt}]([-4, -3]) = \emptyset$ an (Man beachte, dass per Definition $\emptyset \in \overline{IR}$ gilt).

3.2.5 Verallgemeinerung des Intervallbegriffs

Im Kontext von TargetLink Modellen treten nicht selten unstetige Funktionen auf (siehe z.B. den *Relational Operator* Block oder den *Switch* Block aus Abbildung 2.8), die jedoch stückweise stetig sind. In solchen Fällen kann die Abschätzung des Bildbereiches durch ein einziges Intervall teilweise sehr grob werden. Zudem werden vom Anwender gelegentlich diskrete Wertebereiche für Eingangsvariablen und Parameter vorgegeben, die selbst keine Intervalle im Sinne von Definition 23 sind, siehe Tabelle 2.1. Wir nehmen daher noch eine gewisse Verallgemeinerung des Intervallbegriffs vor und lassen gelegentlich auch sogenannte unstetige Intervalle ("Discontinuous Intervals", siehe [Hy92]) entsprechend der nachfolgenden Definition zu. Wir werden im Folgenden die Unterscheidung zwischen einem "unstetigen" Intervall $[\breve{x}]$ und einem "gewöhnlichen" Intervall $[x]$ der Einfachheit halber jedoch oft weglassen und grundsätzlich $[x]$ schreiben.

Definition 31. *Ein unstetiges Intervall, abgekürzt* $[\breve{x}]$*, sei eine endliche Vereinigung von gewöhnlichen Intervallen* $[x_i]$ *im Sinne von Definition 23, also* $[\breve{x}] = \bigcup_i [x_i]$*. Die Definition von Intervallerweiterungen auf einem solchen unstetigen Intervall*$[\breve{x}]$ *erfolgt dann separat für die einzelnen Intervalle* $[x_i]$ *im Sinne von Definition 24.*

Bemerkung 32. *Mit "unstetigen" Intervallen kann nun auch eine Intervallerweiterung für die elementare Division auf* $\mathbb{R}$ *für solche Definitionsbereiche angegeben werden, die im Divisor die 0 einschließen und die nicht das "Trivialresultat"* $[-\infty, \infty]$ *liefern. Dementsprechend ergibt sich z.B.* $1/[-2, 2] = [-\infty, -1/2] \cup [1/2, \infty]$*.*

3.2.6 Das Dependency-Problem und die Reduktion des Überschätzens

Bis jetzt wurde in diesem Abschnitt im Wesentlichen dargestellt, dass Intervallerweiterungen immer so konstruiert werden, dass sie einschließende Funktionen zu einer Klasse von reellwertigen Funktionen darstellen und auf Intervallen der Breite 0 mit diesen übereinstimmen. Es wurde auch darauf hingewiesen, dass Intervallerweiterungen für elementare Funktionen und Operationen so gewählt werden, dass sie kleinstmögliche Intervalle als Funktionswerte besitzen. Im Folgenden soll nun der Aspekt des Überschätzens bei Intervallerweiterungen genauer betrachtet werden, d.h. die Tatsache, dass Intervallerweiterungen bei faktorisierten Funktionen oftmals sehr "konservative" Intervalle liefern. Ein sehr einfaches Beispiel zur Demonstration des "Überschätzens" bei Intervallerweiterungen ist in Abbildung 3.3 dargestellt. Das triviale TargetLink System realisiert die Abbildungsvorschrift $f(x) = x - x$, die in reeller Arithmetik naturgemäß zu 0 vereinfacht bzw. ausgewertet wird. Betrachtet man jedoch die zugehörige natürliche Intervallerweiterung $[f]([x]) = [x] - [x]$ so ergibt sich ein drastisches Überschätzen. Dies ist eine direkte Auswirkung der Tatsache, dass bei der Intervallarithmetik mit Mengen gerechnet wird und keinerlei Beziehungen zwischen diesen bekannt sind. Dies wird im Allgemeinen als "Dependency"-Problem bezeichnet und tritt immer dann auf, wenn Variablen im formelmäßigen Ausdruck mehrfach vorkommen, wie dies beispielsweise in $f(x) = x - x$ der Fall ist. Man sieht auch, dass die Art der formelmäßigen Darstellung direkten Einfluss auf die berechneten Intervalle hat, selbst wenn dies in reeller Arithmetik nicht der Fall ist. Auf die algebraischen Eigenschaften der Intervallarithmetik soll hier jedoch nicht weiter eingegangen werden.

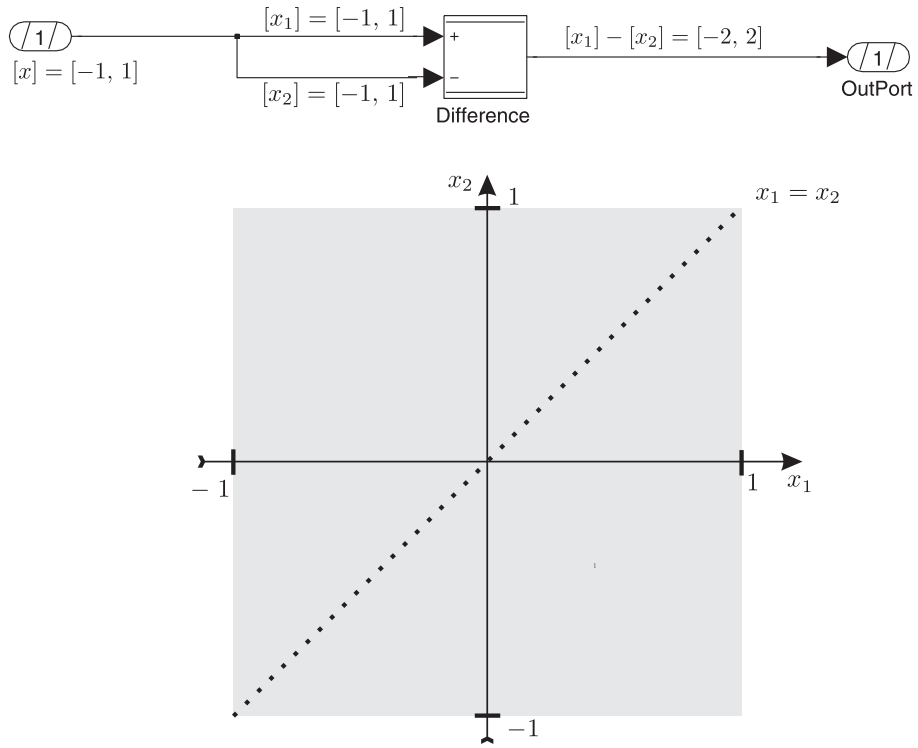


Abbildung 3.3: Oben: Einfaches Modellfragment zur Demonstration von "Dependency" Fehlern anhand des formelmäßigen Ausdruckes $f(x) = x - x$ für $[x] = [-1, 1]$. Da die Beziehung zwischen Minuend und Subtrahend der Intervallarithmetik nicht bekannt ist, wird $[x_1] - [x_2] = [-1, 1] - [-1, 1] = [-2, 2]$ berechnet. Das Intervall enthält aufgrund der Überdeckungseigenschaft der Intervallfunktion naturgemäß die tatsächliche Bildmenge $f([-1, 1]) = \{0\}$, jedoch kommt es zu drastischem Überschätzen. **Unten:** Man kann das Überschätzen durch die Intervallarithmetik als Konsequenz der Tatsache interpretieren, dass der angenommene Definitionsbereich für die Intervall-Subtraktion der Bereich $[x_1] \times [x_2] = [-1, 1] \times [-1, 1]$ ist, wohingegen der tatsächliche Definitionsbereich nur dessen Schnittmenge mit der Winkelhalbierenden $x_1 = x_2$ ist.

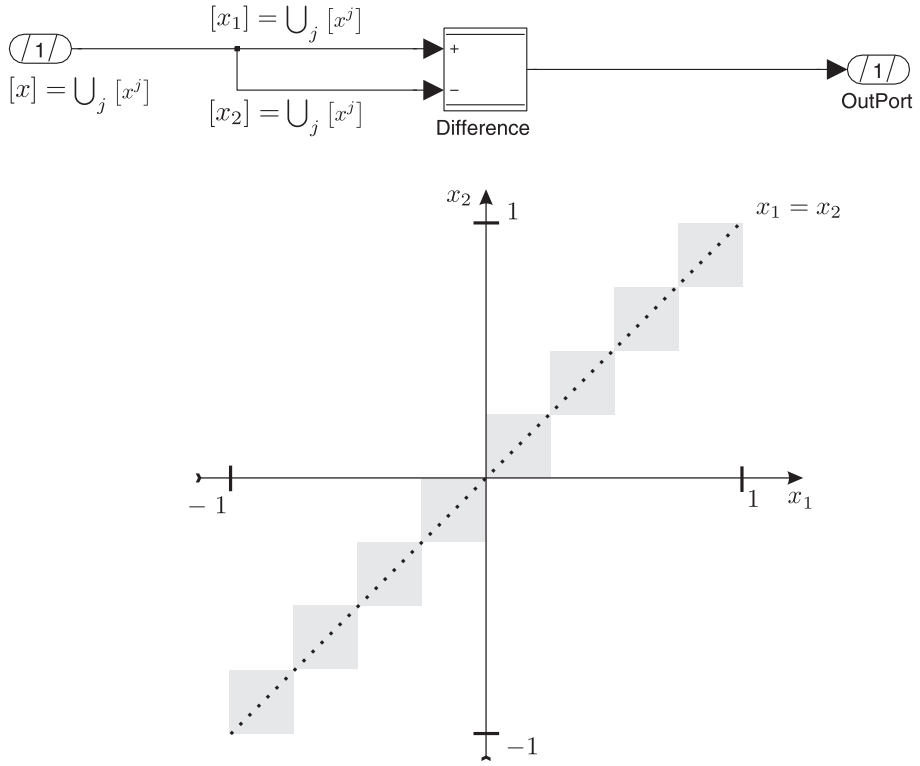


Abbildung 3.4: Oben: Identisches Modellfragment wie in Abbildung 3.3, wobei jetzt jedoch der Definitionsbereich $[x] = [-1, 1] = \bigcup_i [x_1]$ durch mehrere Intervalle $[x_i]$ reduzierter Breite $w([x_1]) = 0.25$ überdeckt wird und Satz 20 zur Berechnung der Überdeckung von $f([x])$ herangezogen wird. Dies führt zur Reduzierung der Dependency Fehler und es ergibt sich eine verbesserte Abschätzung $f([-1, 1]) \subseteq [x_i] - [x_i] = [-0.25, 0.25]$. **Unten:** Darstellung des Definitionsbereiches für die Zerlegung von $[x]$. Offensichtlich nähert sich der durch die Intervalle überdeckte Definitionsbereich dem tatsächlichen an, weshalb die Fehler $\Delta = [x_i] - [x_i]$ reduziert und bei Verfeinerung beliebig nahe an die 0 gedrückt werden können.

In Abbildung 3.4 ist skizziert, wie "Dependency"-Fehler reduziert werden können, was für eine große Klasse von Funktionen zu beliebig geringem Überschätzen durch Intervallarithmetik führt. Der Gedanke besteht darin, den Definitionsbereich durch Intervalle bzw. Boxen $[x_i]$ reduzierter Breite $w([x_i])$ zu unterteilen und dann unter Nutzung von Satz 20 die Berechnung für jedes Intervall $[x_i]$ separat vorzunehmen. Für das Beispiel in Abbildung 3.4 würde für $w([x_i]) \to 0$ auch das Überschätzen beliebig reduziert. Diese Eigenschaft bzw. die Bedingungen unter denen sich dies grundsätzlich erzielen lässt, sollen in dem folgenden Satz festgehalten werden.

Satz 33. *Gegeben sei eine inklusions-monotone Intervallerweiterung* $[f] : \overline{IR}^n \to \overline{IR}$ *zu einer Funktion* $f : D \subseteq \overline{\mathbb{R}}^n \to \overline{\mathbb{R}}$. *Ferner sei* $[f]$ *Lipschitz-stetig, d.h. es gebe ein* $L \in \mathbb{R}$ *mit* $w([f]([x])) \leq L \cdot w([x])$. *Sei ferner* $[f]_N([x]) = \bigcup_{J_i} [f]([x_{J_i}])$ *eine sogenannte Verfeinerung von* $f([x])$, *wobei* $[x] = \bigcup_{J_i} [x_{J_i}]$ *eine Überdeckung von* $[x]$ *sei mit* $w([x_{J_i}]) = {}^{w([x])}/_N$ *(die Box* $[x]$ *werde dabei in jeder Dimension in* N *äquidistante Teile zerlegt). Dann gilt*

$$[f]_N([x]) := \bigcup_{J_i} [f]([x_{J_i}]) = f([x]) + [e_N]$$

wobei $[e_N]$ *ein Intervall mit der Breite* $w([e_N]) \leq const \cdot \frac{w([x])}{N}$ *ist.*

Bemerkung 34. *Die Breite des "Restintervalls" $[e_N]$, welches den "Unterschied" zwischen $f([x])$ und dessen verfeinerter Überdeckung $\bigcup_{J_i}[f]([x_{J_i}])$ beschreibt, ist also für hinreichend großes N, d.h. genügend feine Unterteilung von $[x]$ in einzelne Boxen $[x_{J_i}]$ beliebig klein zu kriegen.*

Beweis. Da $[f]$ laut Voraussetzung eine inklusions-monotone Intervallerweiterung zu f ist, gilt die Überdeckungseigenschaft $[f]([x_{J_i}]) \supseteq f([x_{J_i}])$ für alle Boxen $[x_{J_i}]$. Dann lässt sich $[f]([x_{J_i}])$ darstellen als $f([x_{J_i}]) = f([x_{J_i}]) + e_{N,J_i}$und für die Breite des "Restintervalls" $[e_{N,J_i}]$ in der Box $[x_{J_i}]$ gilt die Abschätzung

$$w([e_{N,J_i}]) = w([f]([x_{J_i}])) - w(f([x_{J_i}])) \leq w([f]([x_{J_i}])) \leq L \cdot w([x_{J_i}]) = L \cdot \frac{w([x])}{N}$$

Da diese Abschätzung für alle Boxen $[x_{J_i}]$ zutreffend ist und diese $[x]$ überdecken ergibt sich damit die Behauptung. □

Bemerkung 35. *Die Verfeinerung der Intervallüberdeckung ist eine generelle Methode, um "Dependency-Fehler" zu reduzieren. Für das in Abbildung 3.3 skizzierte Problem lassen sich naturgemäß andere Methoden finden, die den Fehler sogar vollständig auf 0 drücken, wie etwa die Anwendung der sogenannten affinen Arithmetik, siehe [St94], als Verallgemeinerung der Intervall-Arithmetik.*

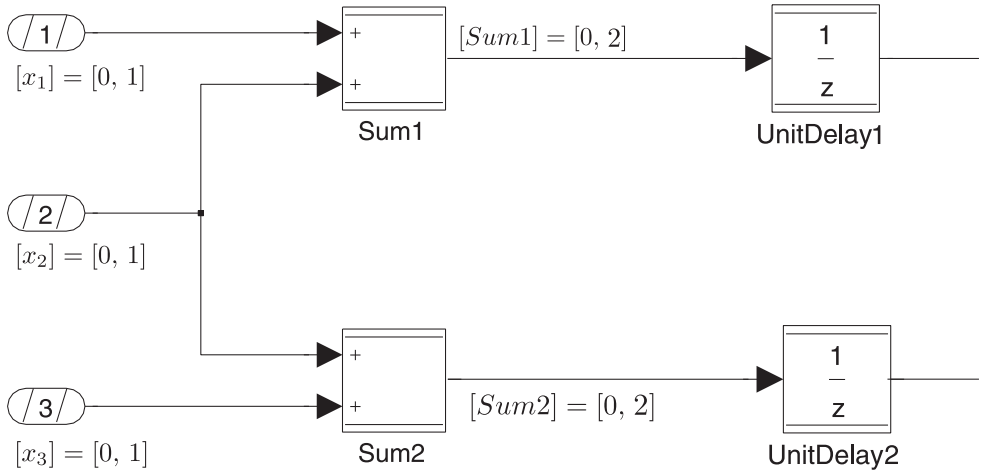
3.2.7 Das Wrapping-Problem und die Reduktion des Überschätzens

Ein weiterer wichtiger Aspekt, der für das Abschätzen der Wertebereiche der Zustandsvariablen in Abschnitt 4.3.4.3 von Bedeutung sein wird, soll hier noch kurz angerissen werden. Wie bereits bemerkt, beschränkten sich die Abschätzungen durch Intervallerweiterungen in diesem Abschnitt auf Abbildungen der Form $[f] : \overline{IR}^n \to \overline{IR}$, der Bildbereich ist also durch Intervalle aber nicht durch Boxen beliebiger Dimension gegeben. Die Update-Funktion h_{sys} eines TargetLink Systems entsprechend Gleichung 2.10 ist aber im allgemeinen Fall eine Abbildung in den $\overline{\mathbb{R}}^{\mathrm{m}}$.

$$\begin{array}{rlcl} h_{sys}: & (X_{sys} \times Z_{sys} \times P_{sys}) & \to & Z_{sys} \\ & (x_{sys},\, z_{sys},\, p_{sys}) & \to & z_{sys}(k+1) \;=\; h_{sys}\,(x_{sys}(k),\, z_{sys}(k),\, p_{sys}(k)) \end{array}$$

Das kartesische Produkt von Intervallerweiterungen der einzelnen Komponentenfunktionen $h_{sys,i}$ ist naturgemäß eine einschließende Funktion der Update-Funktion h_{sys} im Sinne von Definition 19, d.h. es erfüllt die geforderte Überdeckungseigenschaft. Es ist im Allgemeinen jedoch auch eine Abschätzung, die sehr konservativ ist, weil das Bild der Update-Funktion in der Regel nicht unmittelbar als kartesisches Produkt darstellbar ist. Das Problem ist systematischer Natur und entsteht durch das genutzte Wrapper-System in Form von Intervallen bzw. Boxen, weshalb es im Allgemeinen auch als Wrapping-Effekt bezeichnet wird. Das einfache Beispiel in Abbildung 3.5 demonstriert das Problem.

Bemerkung 36. *Naturgemäß tritt der Wrapping-Effekt auch für die Output-Funktion g_{sys} eines TargetLink Systems auf. Dies hat auf die abgeschätzten Wertebereichsgrenzen für die einzelnen Ausgangsvariablen y_i jedoch keinen Einfluss, weil die Ausgangsvariablen im Unterschied zu den Zustandsvariablen nicht für Berechnungen in nachfolgenden Zeitschritten verwendet werden, siehe Gl. 2.9 und Gl. 2.10.*



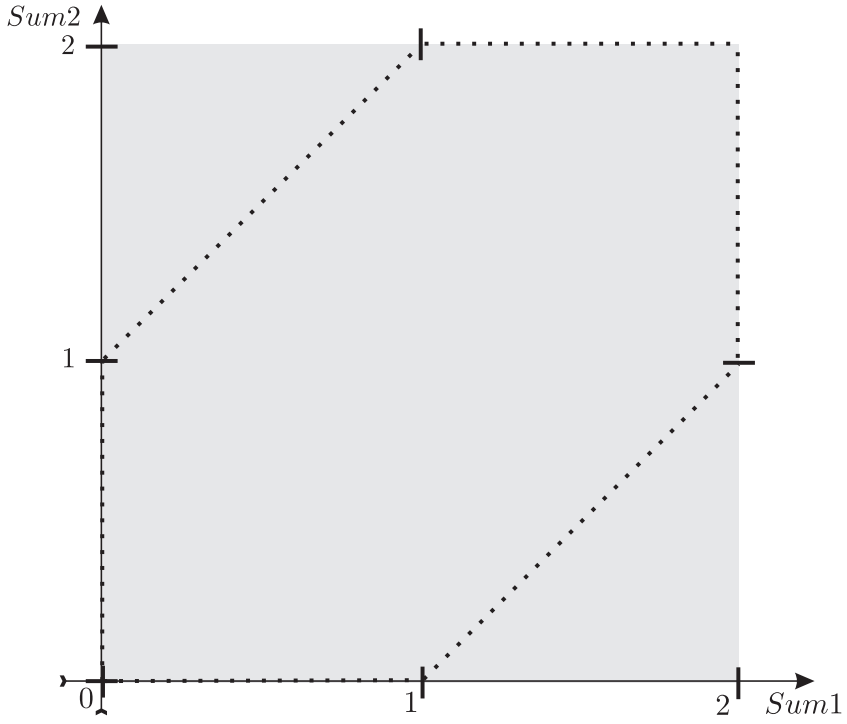
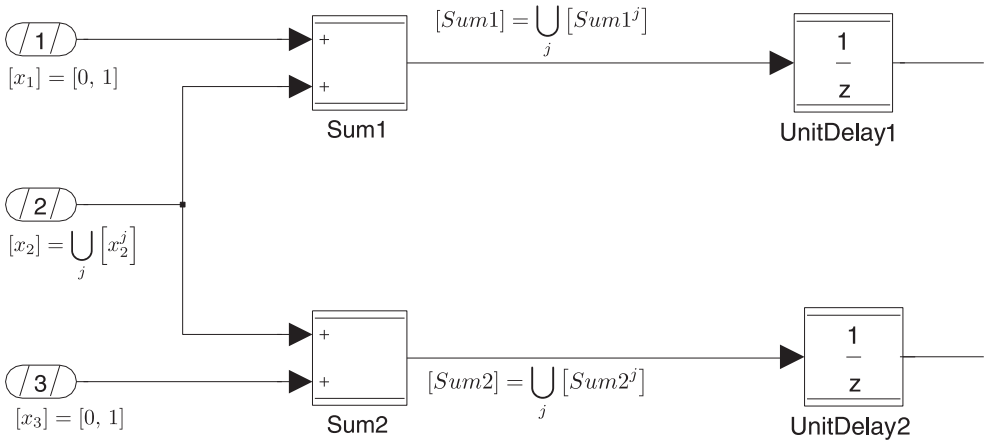


Abbildung 3.5: Oben: Einfaches Modellfragment zur Demonstration des Überschätzens durch "Wrapping". Die beiden Blöcke *UnitDelay1* und *UnitDelay2* weisen jeweils eine Zustandsvariable auf, deren Wert sich jeweils durch die Eingangssignale $Sum1$ bzw. $Sum2$ zum vorherigen Zeitpunkt ergibt (siehe Abbildung 2.8 wegen der Funktionalität des Unit-Delay Blockes). Da die Blöcke *UnitDelay1* und *UnitDelay2* mit dem Eingang x_2 eine gemeinsame Variablenabhängigkeit aufweisen, ist der Zustandsraum $\{(z_1, z_2) \mid z_1 \text{ aus } UnitDelay1,\ z_2 \text{ aus } UnitDelay2\}$ nicht als kartesisches Produkt darstellbar. **Unten:** Dem angenommenen Zustandsraum $[Sum1] \times [Sum2] = [0, 2] \times [0, 2]$ bei Bildung des kartesischen Produktes (grau eingezeichnet) ist der gestrichelt eingezeichnete, tatsächliche Zustandsraum gegenübergestellt. Letzterer ist durch die Verquickung über die Eingangsvariable x_2 prinzipiell nicht Box-förmig, wodurch Fehler durch Wrapping entstehen.

Zur Reduzierung des Überschätzens durch den Wrapping-Effekt wird genau so vorgegangen, wie zur Reduzierung des Dependency-Problems, nämlich durch Überdeckung des Definitionsbereiches durch Intervalle bzw. Boxen reduzierter Breite w_i und Nutzung von Satz 20. Folglich kann der tatsächliche Zustandsraum weitaus besser approximiert werden, wie dies in Abbildung 3.6 exemplarisch dargestellt ist.



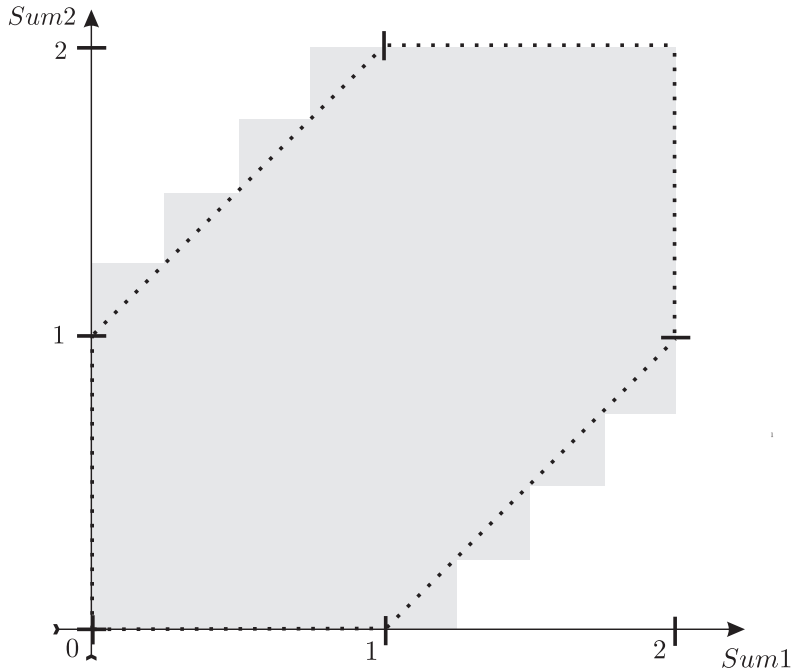


Abbildung 3.6: Oben: Identisches Modellfragment wie in Abbildung 3.5, jetzt jedoch mit einer Überdeckung des Definitionsbereiches für die Eingangsvariable x_2 durch Intervalle $\left[x_2^j\right]$ verminderter Breite. **Unten:** Darstellung des tatsächlichen (gestrichelt) vs. des bei Intervallverfeinerung ermittelten Zustandsraumes (grau eingefärbt). Offensichtlich ergibt sich gegenüber Abbildung 3.5 eine deutliche Reduzierung des Überschätzens.

3.3 Elementare Terminologie aus der Graphentheorie

Für die Abschätzung der Wertebereichsgrenzen von TargetLink Systemen wird in Abschnitt 4.3.2 eine Graph-Struktur zur eleganten Modellierung eingesetzt. Dies ist primär dadurch begründet, dass ein TargetLink System mit seinen Blöcken und gerichteten Verbindungslinien eine Repräsentation als Graph geradezu verlangt. Infolgedessen soll hier kurz auf einige grundlegenden Elemente aus der Graphentheorie eingegangen werden.

Definition 37. ***Gerichteter Graph, Ecken und Kanten, Pfade in Graphen***

- Ein gerichteter Graph G sei definiert als das geordnete Paar $G = (E, K)$ einer Eckenmenge $E = \{e_i \mid i \in I\}$ und Kantenmenge $K \subseteq \{(e_i, e_j) \mid e_i, \in E,\ e_j \in E\}$. Die Elemente $e_i \in E$ heißen Ecken, die Elemente von K heißen Kanten. Jede Kante $k = (e_i, e_j)$ kann folglich als eine "gerichtete Verbindung" von der Ecke e_i zur Ecke e_j interpretiert werden wodurch sich eine Visualisierung von Graphen analog zum Beispiel in Abbildung 3.7 anbietet. Wir kürzen im Folgenden jede Kante durch die Schreibweise e_ie_j mit Startpunkt e_i und Endpunkte e_j ab.
- Ein Weg $P(e_1, e_n)$ mit Anfangspunkt e_1 und Endpunkt e_n ist ein n-Tupel von Ecken $(e_1, e_2, ..., e_n)$ des Graphen, wobei jeweils $e_ie_{i+1} \in K$ gilt, es also jeweils eine Kante von e_i nach e_{i+1} gibt. Folglich kann der Graph von $e_1$bis zu e_n durchlaufen werden.

Im Kontext von TargetLink Modellen bzw. der Algorithmik zur Abschäzung der Wertebereichsgrenzen entsprechend Kapitel 4 ist darüber hinaus eine spezielle Art von Graphen relevant, die in der folgenden Definition erfasst werden sollen.

Definition 38. ***Endlicher, schlingenfreier azyklischer Graph ohne Mehrfachkanten***

Ein gerichteter, schlingenfreier azyklischer Graph ohne Mehrfachkanten ist ein Graph entsprechend Definition 37, der zudem die folgenden Eigenschaften aufweist:

- Der Graph ist endlich, d. h. die Ecken- und Kantenmengen E und K sind endlich.
- Der Graph ist schlingenfrei, d. h. es gibt keine Kanten e_ie_i, also Kanten, bei denen Start- und Endpunkt identisch sind.
- Der Graph ist azyklisch, d. h. es gibt keine geschlossenen Wege $P(e_i, e_i)$, also Wege mit identischem Start- und Endpunkt.
- Der Graph ist mehrfachkantenfrei, d.h. es gibt maximal eine Kante e_ie_j von der Ecke e_i zur Ecke e_j. Dies ist auch schon bereits durch Definition 37 ausgeschlossen.

In Abbildung 3.8 ist ein Beispiel eines Graphen dargestellt, der sowohl Schlingen, Zyklen und Mehrfachkanten aufweist. Wir werden in Abschnitt 4.3.2 sehen, wie TargetLink Systeme zur Abschätzung der Wertebereichsgrenzen vorteilhaft als gerichtete, azyklische Graphen repräsentiert werden.

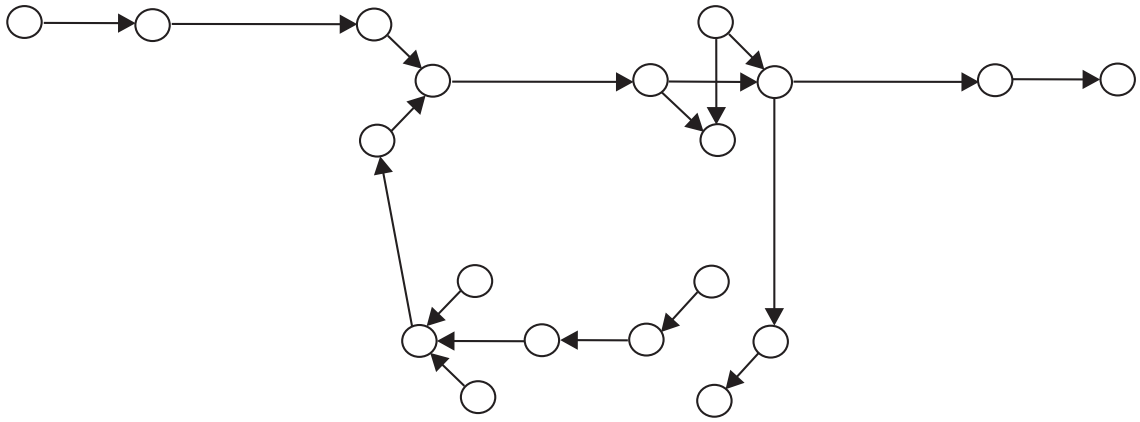


Abbildung 3.7: Beispiel eines einfachen Graphen, wie er sich anhand der in Abschnitt 4.3.2 beschriebenen Vorgehensweise für TargetLink Systeme ergibt. Der Graph ist schlingenfrei, azyklisch und ohne Mehrfachkanten.

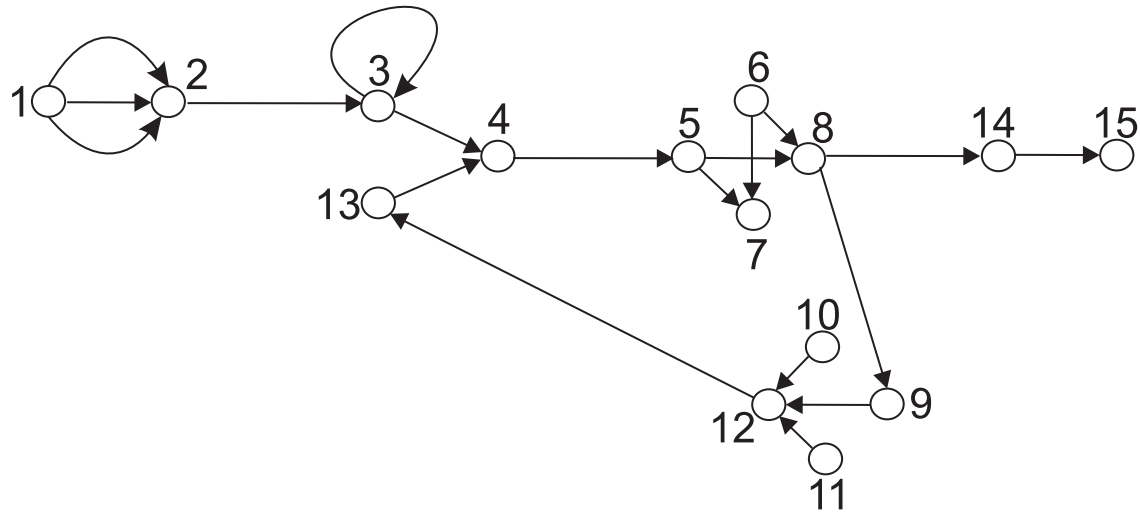


Abbildung 3.8: Beispiel eines zyklischen Graphen, der zudem eine Schlinge und Mehrfachkanten enthält. Die Ecke 3 ist sowohl Start- als auch Endpunkt der selben Kante, was als Schlinge bezeichnet wird. Die Ecke 1 ist Startpunkt dreier Kanten, die Ecke 2 zum Ziel haben, sogenannte Mehrfachkanten. Die Ecken 4, 5, 8, 9, 12, 13, 4 bilden einen geschlossenen Weg, was den Graphen zyklisch macht.

Kapitel 4

Algorithmik zur Abschätzung von Wertebereichsgrenzen für modellbasierte Designs

In diesem Kapitel werden nun die in Kapitel 3 beschriebenen Hilfsmittel angewandt, um robuste Wertebereichsgrenzen für TargetLink Systeme entsprechend der in Abschnitt 2.4 formulierten Aufgabenstellung abzuleiten. Dazu wird zunächst in Abschnitt 4.1 die prinzipielle Vorgehensweise beschrieben und anhand eines sehr einfachen Beispiels in Abschnitt 4.2 die Problematik näher erläutert. In Abschnitt 4.3 erfolgt dann die detaillierte Darstellung der einzelnen Schritte der entwickelten Algorithmik, die im Folgenden genau wie der entwickelte Prototyp als TalInt (TArgetLink Intervall INTerpreter) bezeichnet werden soll. Der letzte Abschnitt dieses Kapitels enthält dann noch einige übergeordnete Anmerkungen und Erklärungen zur Implementierung des TalInt Prototypen in seiner existierenden Form, der im nächsten Kapitel auf reale Modelle angewendet und bewertet werden wird.

4.1 Die Fixpunkt-Gleichung für den Zustandsraum des TargetLink Systems

Die prinzipielle Vorgehensweise zur Lösung der Aufgabenstellung 14 besteht darin, die kumulierten Bildmengen der Output- und Update-Funktionen des betrachteten TargetLink Systems in jedem Zeitschritt k durch eine Obermenge $[y^{\breve{}}_{sys}](k)$ für den Blockausgangsvektor und $[z^{\breve{}}_{sys}](k)$ für den Zustandsvektor abzuschätzen. Hierzu werden die Methoden der Intervall-Analysis aus dem vorherigen Kapitel eingesetzt.

Bemerkung 39. *Wir wollen hier durch die Schreibweise $[y^{\breve{}}_{sys}](k)$ bzw. $[z^{\breve{}}_{sys}](k)$ mit dem Vereinigungssymbol andeuten, dass es sich bei diesen Mengen um Vereinigungen von Boxen bzw. Intervallen handelt, nicht notwendigerweise um eine einzelne Box, bzw. ein einzelnes Intervall (siehe auch die Ausführungen in Abschnitt 3.2.5 zu "Discontinuous Intervals").*

Ausgangspunkt der Überlegungen sind naturgemäß die Output- und Update-Funktion des TargetLink Systems

$$\begin{array}{rcl} y_{sys}(k) & = & g_{sys}\left(x_{sys}(k),\, z_{sys}(k),\, p_{sys}(k)\right) \\ z_{sys}(k+1) & = & h_{sys}\left(x_{sys}(k),\, z_{sys}(k),\, p_{sys}(k)\right) \end{array} \tag{4.1}$$

Da die Wertebereichsmenge für den Eingangsvektor x_{sys} entsprechend Tabelle 2.1 für alle Zeiten konstant ist, ergibt sich als Menge der Eingangswerte eine Box $[x_{sys}] = \left[\underline{x_{sys}}, \overline{x_{sys}}\right]$. Analoges gilt für den Parametervektor p_{sys} des Systems, dessen Wertemenge durch die Box $[p_{sys}] = \left[\underline{p_{sys}}, \overline{p_{sys}}\right]$ beschrieben wird. Vorgegeben ist ferner die Menge der möglichen Werte des Zustandsvektors für $k = 0$ als Box $Z_{sys,0} = [z_{sys,0}]$.

Bemerkung 40. *Wir werden im Folgenden Mengen wie $Z_{sys,0}$, $Z_{sys,1}$ (siehe unten) mit Großbuchstaben abkürzen und so hervorheben, dass es sich im Allgemeinen nicht um Intervalle bzw. Boxen oder eine endliche Vereinigung derselben handelt, sondern prinzipiell um Mengen beliebiger Gestalt.*

Entscheidend an den System-Gleichungen 4.1 ist primär die Rekursionsgleichung für den Zustandsvektor. Betrachtet wird nun explizit das Bild der kumulierten Wertemenge des Zustandsvektors unter der Update-Funktion für jeden Zeitschritt $k = 0, 1, 2, ...$ für das sich iterativ folgendes ergibt

$$\begin{array}{lcl} Z_{sys,1} & = & h_{sys}\left([x_{sys}]\,,\, Z_{sys,0},\, [p_{sys}]\right) \cup Z_{sys,0} \\ Z_{sys,2} & = & h_{sys}\left([x_{sys}]\,,\, Z_{sys,1},\, [p_{sys}]\right) \cup Z_{sys,1} \\ \vdots & \vdots & \vdots \\ Z_{sys,k+1} & = & h_{sys}\left([x_{sys}]\,,\, Z_{sys,k}\, [p_{sys}]\right) \cup Z_{sys,k} \end{array} \tag{4.2}$$

wobei dann in jedem Zeitschritt k der Wertebereich des kumulierten Ausgangsvektors durch die Update-Funktion gegeben ist.

$$Y_{sys,k} \;=\; g_{sys}\left([x_{sys}]\,,\, Z_{sys,k},\, [p_{sys}]\right) \tag{4.3}$$

Zu beachten ist dabei, dass die Mengen $Z_{sys,k}$ und $Y_{sys,k}$ im Allgemeinen keine Intervalle sind, sondern beliebig berandete Gebiete darstellen. Der Grund warum in Gl. 4.2 die kumulierten Zustandswerte betrachtet werden liegt darin begründet, dass die Gesamtheit aller angenommenen Werte zu allen Zeitschritten k für die robuste Abschätzung der Wertebereichsgrenzen die maßgebliche Größe ist. Zudem werden wir später noch sehen, dass die Bildung der Menge der kumulierten Zustandsvektoren auf eine sehr einfache und angenehme Fixpunkt-Gleichung führt.

Als Resultat der Iterationen in Gl. 4.2 erhält man eine aufsteigende Kette $Z_{sys,k} \subseteq Z_{sys,k+1}$ für den Zustandsvektor und analog für den Blockausgangsvektor $Y_{sys,k} \subseteq Y_{sys,k+1}$. Wir werden im Folgenden einen sehr einfachen Satz formulieren, der zeigt, wie die kumulierten Mengen des Zustands- und Ausgangsvektors mit Hilfe von Intervallerweiterungen $[g_{sys}]$ und $[h_{sys}]$ überdeckt werden können.

Satz 41. *Gegeben seien die System-Gleichungen für die Output- und Update Funktion des betrachteten Systems entsprechend 4.1 sowie die Anwendervorgaben für $[x_{sys}]$, $[p_{sys}]$ und $[z_{sys,0}]$ nach Tabelle 2.1. Es sei ferner Widening$([\breve{z}_k])$ eine Mengenoperation mit Widening$([\breve{z}_k]) \supseteq [\breve{z}_k]$. Existieren zur Output-Funktion g_{sys} und Update-Funktion h_{sys} einschließende Funktionen $[g_{sys}]$ und $[h_{sys}]$ und werden iterativ die folgenden Größen berechnet*

$$\begin{array}{lcl} [\breve{z}_{sys,1}] & = & \mathit{Widening}\left([h_{sys}]\left([x_{sys}]\,,\, [z_{sys,0}]\,,\, [p_{sys}]\right)\right) \cup [z_{sys,0}] \\ [\breve{z}_{sys,2}] & = & \mathit{Widening}\left([h_{sys}]\left([x_{sys}]\,,\, [\breve{z}_{sys,1}]\,,\, [p_{sys}]\right)\right) \cup [\breve{z}_{sys,1}] \\ \vdots & \vdots & \vdots \\ [\breve{z}_{sys,k+1}] & = & \mathit{Widening}\left([h_{sys}]\left([x_{sys}]\,,\, [\breve{z}_{sys,k}]\,,\, [p_{sys}]\right)\right) \cup [\breve{z}_{sys,k}] \end{array}$$

bzw. jeweils

$$[y_{\breve{sys},k}] = [g_{sys}]\left([x_{sys}]\,,\,[z_{\breve{sys},k}]\,,\,[p_{sys}]\right)$$

dann gilt in jedem Zeitschritt

$$Z_{sys,k+1} \subseteq [z_{\breve{sys},k+1}] \quad \textit{und} \quad Y_{sys,k} \subseteq [y_{\breve{sys},k}]\,,$$

die kumulierten Wertebereiche für Zustandsvektor und Blockausgangsvektor werden also in jedem Zeitschritt überdeckt.

Bemerkung 42. *Die erwähnte Widening-Operation wird selektiv angewandt (oftmals ist es einfach die Identität) und dient zum "künstlichen" Ausdehnen des Zustandsraumes, um nach endlich vielen Iterationen k einen Fixpunkt von $[h_{sys}]$ zu ermitteln, wie er für den nachfolgenden, zentralen Satz erforderlich ist. In Abschnitt 4.2 wird die Widening-Operation anhand eines einfachen Beispiels demonstriert. Der Terminus Widening ist in Anlehnung an die gleichnamige Operation im Kontext der Abstrakten Interpretation von Programmen gewählt, siehe z. B. [Cou77, Cou92], zu der eine gewisse inhaltliche Nähe besteht.*

Beweis. Wegen der vorausgesetzten Überdeckungseigenschaft von $[h_{sys}]$ gegenüber h_{sys} bzw. $[g_{sys}]$ gegenüber g_{sys} sowie der Widening Eigenschaft ergibt sich der geforderte Zusammenhang trivialerweise nach k-facher iterativer Anwendung der vorausgesetzten Überdeckungseigenschaft. □

Entsprechend der Aufgabenstellung 14 ist es das Ziel, die Wertebereiche für Zustandsvariablen und Blockausgangsvariablen für alle $k \in \mathbb{N}_0$ robust abzuschätzen und nicht nur bis zu einem festen Index $k \leq k_0$. Der folgende einfache, aber zugleich wichtigste Satz der Diplomarbeit zeigt, wie dies möglich ist.

Satz 43. *Es seien die Bedingungen des Satzes 41 gegeben und die Iterationsvorschriften zur Berechnung von $[z_{\breve{sys},k+1}]$ und $[y_{\breve{sys},k}]$ werden dementsprechend in der folgenden Art und Weise gebildet*

$$\begin{aligned} [y_{\breve{sys},k}] &= [g_{sys}]\left([x_{sys}]\,,\,[z_{\breve{sys},k}]\,,\,[p_{sys}]\right) \\ [z_{\breve{sys},k+1}] &= \textit{Widening}\left([h_{sys}]\left([\breve{X}_{sys}],\,[z_{\breve{sys},k}]\,,\,[p_{sys}]\right)\right) \cup [z_{\breve{sys},k}] \end{aligned} \tag{4.4}$$

Existiert dann ein Index k_{final} mit

$$\left[z_{sys,\breve{k}_{final}+1}\right] = \left[z_{sys,\breve{k}_{final}}\right] \tag{4.5}$$

so gilt für alle $k \in \mathbb{N}_0$:

$$Z_{sys,k} \subseteq \left[z_{sys,\breve{k}_{final}}\right] \quad \textit{und} \quad Y_{sys,k} \subseteq \left[y_{sys,\breve{k}_{final}}\right]$$

es ist also sichergestellt, dass die zum Zeitpunkt k_{final} ermittelten Wertebereichsgrenzen für alle $k \in \mathbb{N}_0$ Gültigkeit besitzen, wodurch die Aufgabenstellung 14 gelöst wäre.

Beweis. Mit Hilfe von Satz 41 ist die Überdeckungseigenschaft $Z_{sys,k} \subseteq [z_{\breve{sys},k}]$ für alle k sichergestellt und wegen $\left[z_{sys,\breve{k}_{final}+1}\right] = \left[z_{sys,\breve{k}_{final}}\right]$ ergibt sich die Behauptung aus $\left[z_{sys,\breve{k}_{final}}\right] \supseteq [z_{\breve{sys},k}]$ für alle $k \in \mathbb{N}_0$. □

Mit dem einfachen Satz 43 ist somit eine Möglichkeit gegeben, um explizit nachzuweisen, dass Wertebereichsgrenzen grundsätzlich, d.h. für jeden beliebigen Zeitpunkt eingehalten werden. Dazu muss lediglich ein Fixpunkt im Zustandsraum unter der Iterationsvorschrift Gl. 4.4 gefunden werden, den man durch Iteration in Kombination mit der Widening-Operation zu finden hofft, siehe Abbildung 4.1. Die konkrete Ausgestaltung z.B. der Widening-Operation bzw. die zu ergreifenden Maßnahmen, um einerseits Überschätzen durch Dependency- bzw. Wrapping-Effekte zu reduzieren gleichzeitig aber den Rechenaufwand nicht ins grenzenlose zu steigern, werden im Abschnitt 4.3 behandelt.

Bemerkung 44. *Hier wie auch in den nachfolgenden Kapiteln wird oft von einem Fixpunkt gesprochen. Konkret ist natürlich eine Menge entsprechend der Bedingung (4.5) gemeint. Da die Überdeckungsfunktionen jedoch Mengenfunktionen sind, ist der Ausdruck Fixpunkt auch hier ganz passend.*

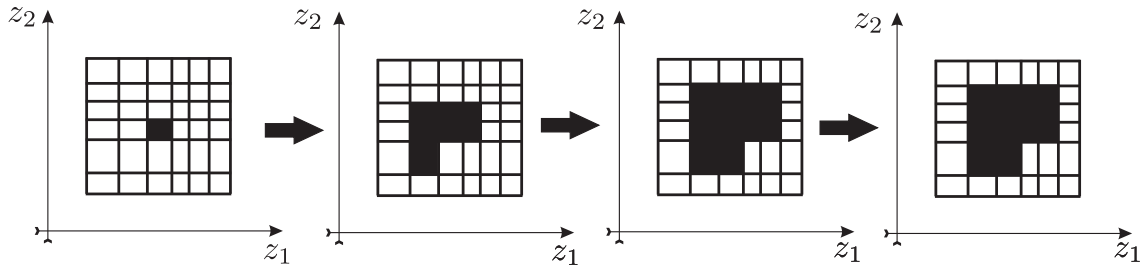


Abbildung 4.1: Visualisierung der Vorgänge bei der Fixpunkt Iteration entsprechend Gl. 4.4 für einen angenommenen zweidimensionalen, diskretisierten Zustandsraum. Im letzten Iterationsschritt ist die Existenz des Fixpunktes bewiesen.

4.2 Beispiel zur Ermittlung der Wertebereichsgrenzen eines einfachen linearen Filters

Ein einfaches Beispiel soll nun im Folgenden dazu dienen, die Abschätzung von Wertebereichsgrenzen für TargetLink Systeme und potenzielle Probleme dabei besser zu veranschaulichen. Dazu sei ein einfaches lineares Filter 1.Ordnung vorgegeben, wie man es häufig in realen TargetLink Modellen findet, siehe Abbildung 4.2 oben und unten. Das Beispiel hat den Vorteil, dass die analytische Bestimmung von Wertebereichsgrenzen hier sehr einfach möglich ist und man auch eine Lösung mittels Intervallarithmetik quasi manuell durchführen kann. In Abschnitt 5.1.4 wird das Beispiel dann noch mal mit der entwickelten TalInt-Algorithmik analysiert.

Das System besitzt eine Zustandsvariable durch den *UnitDelay* Block, die über eine Rückkopplungsschleife auf den Eingang zurückgeführt wird. Die einzelnen Blöcke des Systems sind durch die Output-Funktionen

$$\begin{aligned} UnitDelay(k) &= z(k) \\ Gain(k) &= GainFactor \cdot UnitDelay(k) \\ Sum(k) &= In(k) + Gain(k) \end{aligned} \tag{4.6}$$

sowie die Update-Funktion des *UnitDelay* Blocks

$$z(k+1) \quad = \quad Sum(k) \tag{4.7}$$

beschrieben. Zusätzlich seien noch Vorgaben für den Initialwert $z\,(0) = 0$ und die Eingangswerte $In\,(k) \in [-1,\,1]$ gemacht, um die Systembeschreibung zu komplettieren. Von besonderem Interesse ist die Update-Funktion des *UnitDelay* Blocks, für die sich aus Gl. 4.6 und Gl. 4.7 die folgende lineare Differenzengleichung bei gegebenem Initialwert und Eingangssignal $In(k)$ ergibt:

$$\begin{array}{lcl} z(k+1) & = & In(k) + GainFactor \cdot z(k) \\ z(0) & = & 0 \\ In(k) & \in & [-1,\,1] \end{array} \tag{4.8}$$

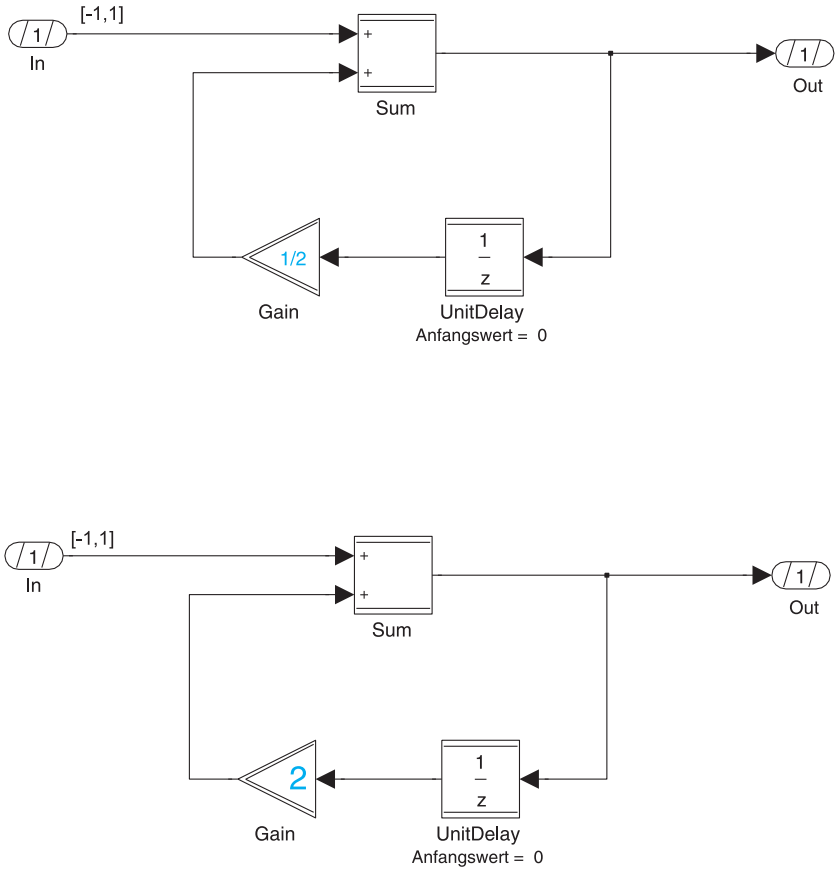


Abbildung 4.2: Oben: Einfache IIR (Infinite Impuls Response) Filter-Funktionalität, die als TargetLink Modell vorliegt. Der eingestellte Gain-Faktor von 1/2 (blau markiert) sorgt dafür, dass Wertebereichsgrenzen existieren bzw. die Signale innerhalb des Modells nicht über alle Grenzen wachsen. **Unten:** Gleiche Filterstruktur wie in der obigen Abbildung, jetzt jedoch mit einem Gain-Faktor von 2 (blau markiert), der dazu führt, dass die Signale im System nicht größenbeschränkt sind. Ein solches Filter bezeichnet man typischerweise als instabil und ist damit für die Anwendung in Steuerungs- und Regelungsfunktionen außer für ganz spezielle Anwendungsszenarien nicht geeignet.

Bei gegebenem, positiven $GainFactor = 1/2$ (siehe Abbildung 4.2 oben) lassen sich exakte obere und untere Grenzen für alle $k \in \mathbb{N}_0$ sehr einfach analytisch bestimmen,

k	$k=0$	$k=1$	$k=2$	$k=3$	$k=4$	$k=5$ *(Widening)*	$k=6$
$[z](k)$	$[0,\,0]$	$[-1,\,1]$	$[-\frac{3}{2},\,\frac{3}{2}]$	$[-\frac{7}{4},\,\frac{7}{4}]$	$[-\frac{15}{8},\,\frac{15}{8}]$	$[-\frac{34}{16},\,\frac{34}{16}]$	$[-\frac{66}{32},\,\frac{66}{32}] \subseteq [-\frac{34}{16},\,\frac{34}{16}]$

Tabelle 4.1: Fixpunkt Iterationen für die Zustandsgröße $[z](k)$. Die ersten 4 Iterationen wurden anhand von Gl. 4.11 durchgeführt. Im Zeitschritt $k = 5$ wurde nach der Berechnung ein Widening-Schritt zur Ausdehnung des Zustandsraumes vorgenommen. Im Zeitschritt $k = 6$ liegt dann ein Fixpunkt für die kumulierten Wertebereiche vor und die Berechnung terminiert mit einem leichten Überschätzen des tatsächlichen Wertebereiches.

denn $\sup\{z(k) \mid k \in \mathbb{N}_0\}$ und $\inf\{z(k) \mid k \in \mathbb{N}_0\}$ werden für ein konstantes Eingangssignal $In(k) = 1$ bzw. $In(k) = -1$ angenommen so dass sich mittels der geometrischen Reihe unmittelbar die folgenden Abschätzungen ergeben:

$$\begin{aligned} z(k) &\leq \lim_{k\to\infty} \sum_{i=0}^{k} GainFactor^i &&= \lim_{k\to\infty} \sum_{i=0}^{k} \left(\tfrac{1}{2}\right)^i \\ &= \lim_{k\to\infty} \frac{1-\left(\frac{1}{2}\right)^k}{1-\frac{1}{2}} &&= 2 \end{aligned} \tag{4.9}$$

und

$$\begin{aligned} z(k) &\geq (-1)\cdot \lim_{k\to\infty} \sum_{i=0}^{k} GainFactor^i &&= (-1)\cdot \lim_{k\to\infty} \sum_{i=0}^{k} \left(\tfrac{1}{2}\right)^i \\ &= (-1)\cdot \lim_{k\to\infty} \frac{1-\left(\frac{1}{2}\right)^k}{1-\frac{1}{2}} &&= -2 \end{aligned} \tag{4.10}$$

insgesamt also $-2 \leq z\,(k) \leq 2$. Man liest auch unmittelbar ab, dass -2 bzw. 2 die engsten, möglichen Grenzen für die Zustandsvariable z darstellen.

Eine Bestimmung von Wertebereichsgrenzen mittels Intervall-Arithmetik kann man beispielsweise dadurch erzielen, dass man Gleichung 4.8 einer natürlichen Intervallerweiterung unterzieht, was dann zur folgenden Rekursionsgleichung führt:

$$[z](k+1) = [In] + GainFactor \cdot [z](k) \tag{4.11}$$

Aufgrund der Tatsache, dass es mit $z(k)$ lediglich eine Zustandsgröße im System gibt, tritt kein Überschätzen durch den Wrapping-Effekt auf. Ferner gibt es auch keinen Dependency-Effekt, weil in der rechten Seite von Gl. 4.11 jede Größe nur einmal auftritt. Daher liefert die Intervall-Arithmetik in ihrer einfachsten Form, d.h. ohne Verfeinerungen der einzelnen Intervalle, für jeden Zeitschritt k exakte Grenzen ohne Überschätzen. In Tabelle 4.1 sind 6 Iterationsschritte aufgelistet, wobei zum Zeitpunkt $k = 5$ explizit ein Widening-Schritt eingebaut wurde, um den Wertebereich vom errechneten Wert $\left[-\frac{30}{16},\,\frac{30}{16}\right]$ auf den Bereich $\left[-\frac{34}{16},\,\frac{34}{16}\right]$ zu vergrößern (die konkreten Werte sind hier willkürlich gewählt). Im nachfolgenden Schritt $k = 6$ liegt der errechnete Wertebereich dann bei $\left[-\frac{66}{32},\,\frac{66}{32}\right] \subseteq \left[-\frac{34}{16},\,\frac{34}{16}\right]$, d.h. innerhalb des im Schritt $k = 5$ angenommen Wertebereiches. Für die kumulierten Wertebereiche über alle Zeitschritte k hinweg bedeutet dies, dass zum Zeitpunkt $k = 6$ ein Fixpunkt detektiert wird und die Berechnung mit dem Ergebnis $z(k) \in \left[-\frac{34}{16},\,\frac{34}{16}\right]$ abgeschlossen ist. Offensichtlich wird durch das Widening im Allgemeinen ein Überschätzen des tatsächlichen Wertebereiches verursacht, die Widening-Operation ist aber notwendig, um nach endlich vielen Iterationen einen Fixpunkt detektieren zu können. Ansonsten hätten sich die abgeschätzten Wertebereiche dem Bereich $[-2,\,2]$ nur asymptotisch genähert.

Bemerkung 45. *Aus Gl. 4.9 bzw. Gl. 4.10 geht auch unmittelbar hervor, dass für einen* $GainFactor > 1$ *wie z.B. in Abbildung 4.2 unten kein (reellwertiges) Supremum oder*

Infimum existiert, weil $z(k)$ für $k \in \mathbb{N}_0$ offensichtlich über alle Grenzen wächst/sinkt. Dementsprechend wird die Algorithmik nie einen Fixpunkt finden und nach endlich vielen Iterationen abbrechen. Ein solches Design ist als instabil und damit als fehlerhaft zu interpretieren und muss vom Designer abgeändert werden. Dies ist auch der Grund, warum man für praktische TargetLink Modelle hoffen kann, einen Fixpunkt zu finden. Steuerungs- und Regelungsfunktionen müssen Anforderungen hinsichtlich Stabilität erfüllen und sind nicht mit beliebigen physikalischen Systemen zu vergleichen, wo es solche Punkte im Allgemeinen natürlich nicht geben muss.

Diese einfache Beispielrechnung mit "manueller" Intervallarithmetik diente primär zur Veranschaulichung. Die grundsätzlichen Ansätze finden sich jedoch auch in der nachfolgend beschriebenen Algorithmik von TalInt wieder.

4.3 Die Algorithmik von TalInt

In diesem, sehr umfangreichen Abschnitt mit diversen Untersektionen, soll nun die eigentliche Algorithmik von TalInt zur Lösung der Fixpunkt-Gleichung aus Satz 43 und damit zur robusten Abschätzung von Wertebereichsgrenzen entsprechend Aufgabenstellung 14 genauer erläutert werden. Die Basis dafür ist naturgemäß Satz 43 selbst, wobei die konkreten Berechnungen von TalInt zur Anwendung des Satzes einige Feinheiten bzw. Spezifika aufweisen, die in den nachfolgenden Unterabschnitten genauer dargestellt werden.

Prinzipiell führt TalInt zur Lösung der Fixpunkt-Gleichung aus Satz 43 die folgende Iterationsvorschrift aus, die auch in Abbildung 4.3 verdeutlicht ist:

1. In einem initialen Schritt zur Iteration $k = 0$ werden alle Komponenten des Parametervektors p_{sys}, der Initialwert des Zustandsvektors $z_{sys}(0)$ sowie die Eingangsvariablen x_{sys} mit ihren vom Anwender vorgegebenen Wertemengen initialisiert.

2. In jedem Iterationsschritt k werden nun zuerst Intervall-Erweiterungen der Output-Funktionen $[g_i]$ der einzelnen Blöcke B_i in der durch Simulink-vorgegebenen Berechnungsreihenfolge abgearbeitet und damit insgesamt eine Intervall-Erweiterung der einzelnen Komponenten der System-Output-Funktion $[g_{sys}]$ nach Satz 22 berechnet. Als Resultat ergibt sich mit $[\breve{y}_{sys,j}](k)$ eine Intervallüberdeckung der tatsächlichen Bildmenge der einzelnen Blockausgänge des Systems unter der System-Output Funktion g_{sys}.

3. In jedem Iterationsschritt k werden anschließend Intervall-Erweiterungen der Update-Funktionen $[h_i]$ der einzelnen Blöcke B_i in der durch Simulink-vorgegebenen Berechnungsreihenfolge abgearbeitet und damit insgesamt eine Intervall-Erweiterung der einzelnen Komponenten der System-Update-Funktion $[h_{sys,j}]$ nach Satz 22 berechnet. Als Resultat ergibt sich mit $[z_{\breve{sys},j}](k+1)$ eine Intervallüberdeckung der tatsächlichen Bildmenge der einzelnen Komponenten des Zustandsvektors $z_{sys,j}(k+1)$ des Systems unter der System-Update Funktion h_{sys}. Bei der Berechnung der einzelnen Komponenten $[z_{\breve{sys},j}](k+1)$ wird grundsätzlich von Intervallverfeinerungen Gebrauch gemacht, um die Überdeckung der tatsächlichen Wertemenge $Z_{sys}(k+1)$ nicht allzu grob durch das kartesische Produkt von $Hull([z_{\breve{sys},j}])(k+1)$ zu überdecken und damit systematisches Überschätzen durch den Wrapping-Effekt in Kauf

zu nehmen. Stattdessen wird $[z\breve{}_{sys}]\,(k+1)$ auf Basis von Intervallverfeinerungen im nächsten Schritt konstruiert.

4. Als Abschluss jeder Iteration k muss der Zustandsraum möglichst exakt, das heißt ohne allzu großes Überschätzen durch den Wrapping-Effekt ermittelt werden, was auf Basis der Intervallverfeinerungen für die einzelnen Komponenten $[z\breve{}_{sys,j}]\,(k+1)$ geschieht, siehe Abschnitt 3.2.7. $[z\breve{}_{sys}]\,(k+1)$ wird dabei nicht grob aus dem kartesischen Produkt der Hülle $Hull\,([z\breve{}_{sys,j}])\,(k+1)$ der einzelnen Komponenten des Zustandsvektors zusammengesetzt, sondern aus den kartesischen Produkten der Intervallverfeinerungen der einzelnen Komponenten $[z\breve{}_{sys,j}]\,(k+1)$. Hierdurch kann der Wrapping-Effekt bei hinreichender Intervallverfeinerung drastisch reduziert werden. Final wird beim Übergang von $k+1 \rightarrow k$ die Kumulation der Überdeckungen $[z\breve{}_{sys}]\,(k+1) \cup [z\breve{}_{sys}]\,(k)$ des Zustandsraumes vorgenommen, wie dies zur Anwendung von Satz 41 erforderlich ist. Gleichzeitig wird ermittelt, ob $[z\breve{}_{sys}]\,(k+1) = [z\breve{}_{sys}]\,(k)$ gilt, ob der Zustandsraum also überhaupt noch angewachsen ist oder bereits ein Fixpunkt erreicht wurde.

5. Der Algorithmus wird nach $k = k_{final}$ Iterationen beendet, sofern entweder in Schritt 4. detektiert wurde, dass $[z\breve{}_{sys}]\,(k_{final}) = [z\breve{}_{sys}]\,(k_{final}-1)$ gilt, dass also der gesuchte Fixpunkt gefunden wurde, oder dass die vom Benutzer vorgegebene Zahl von $k_{final} = NIter_{max}$ maximalen Iterationen überschritten wurde. Im letzteren Fall erfolgt die Terminierung ohne allgemeingültige Wertebereichsgrenzen

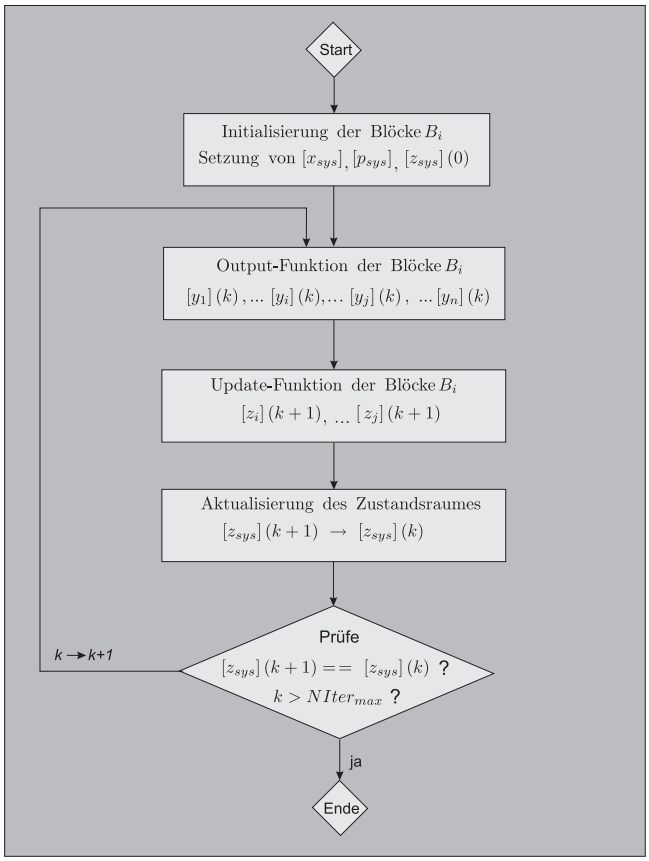


Abbildung 4.3: Die im Rahmen einer TalInt Ausführung durchgeführte Iteration zur Lösung der Fixpunkt-Gleichung in Satz 43.

Bemerkung 46. *Vergleicht man diese, in TalInt angewandte Vorgehensweise mit Satz 43 zur Lösung der Fixpunkt-Gleichung, so fällt der hier separat aufgeführte 4. Schritt auf, der die explizite Konstruktion des Zustandsraumes* $[z^{\breve{}}_{sys}]\,(k+1)$ *zum Ziel hat, wohingegen sich dieser in Satz 43 direkt aus einer allgemeinen Überdeckungsfunktion* $[h_{sys}]$ *ergibt. Der Grund hierfür ist, dass die in TalInt eingesetzten Intervallfunktionen nur reellwertige Funktionen ohne systematisches Überschätzen durch Wrapping überdecken können, jedoch keine allgemeinen vektorwertigen Funktionen, wie es für den gesamten Zustandsraum notwendig wäre. Die Abhilfe gegen Wrapping-Fehler in TalInt besteht analog zu Abschnitt 3.2.7 darin, Intervallverfeinerungen zu nutzen, um so den Zustandsraum aus vielen einzelnen Boxen zusammenzusetzen. Dieser Berechnungsschritt wird in 4. vorgenommen und ist aufgrund des Aufwandes separat aufgeführt.*

In den nachfolgenden Unterabschnitten werden die einzelnen Berechnungsschritte mit ihren TalInt-eigenen Besonderheiten genauer dargestellt:

- Im Abschnitt 4.3.1 wird dargelegt, dass es für die einzelnen TargetLink Blöcke überhaupt Intervallerweiterungen $[g_i]$ bzw. $[h_i]$ als Basis der obigen Iterationen von TalInt gibt, was allerdings nach den Ausführungen in Kapitel 2 und 3 praktisch nur noch eine Formalität darstellt.

- Im Abschnitt 4.3.2 wird vorgestellt, wie ein TargetLink System in TalInt in einen Graphen umgesetzt wird, der sich gut für Analysen und die eigentlichen Berechnungen während der obigen Iteration eignet.

- In Abschnitt 4.3.3 wird die Graphstruktur des TargetLink Systems für Signalanalysen verwendet, die mehrere Ziele haben: Zum einen wird versucht, den Zustandsraum des Gesamtsystems in mehrere Unterräume niedrigerer Dimensionalität zu unterteilen, ohne Überschätzen durch den Wrapping-Effekt in Kauf zu nehmen. Hierdurch vereinfachen sich die Berechnungen im obigen 4. Schritt ganz erheblich. Ferner werden Analysen durchgeführt, um die Widening-Operation aus Satz 43 geeignet zu definieren und so die Lösung der Fixpunkt-Gleichung in möglichst vielen Fällen zu erzwingen. Die Graphstruktur dient schließlich auch dazu, sogenannte Bereichsvariablen in TalInt festzulegen. Dies sind genau die Variablen innerhalb eines TargetLink Systems, für die Intervallverfeinerungen zur Reduzierung des Überschätzens durch Dependency- und Wrapping-Effekt vorgenommen werden. Gleichzeitig wird versucht, den Rechenaufwand durch eine selektive Auswahl möglichst weniger Bereichsvariablen so klein wie möglich zu halten. Neben Bereichsvariablen definieren wir die TalInt-Datenstruktur der sogenannten Bereichsintervalle, welche die Basis für sämtliche nachfolgenden Berechnungen bilden.

- Im Abschnitt 4.3.4 wird schließlich dargestellt, wie auf Basis der Datenstruktur der Bereichsintervalle die Intervallerweiterungen $[g_i]$ und $[h_i]$ für die einzelnen Blöcke B_i des TargetLink Systems entsprechend Schritt 2. und Schritt 3. berechnet werden und wie die Konstruktion des Zustandsraumes im 4. Schritt erfolgt.

- Abschnitt 4.3.5 fasst schließlich die Bedingungen zusammen unter denen TalInt erfolgreich/erfolglos terminiert, was den 5. und gleichzeitig letzten Schritt im obigen Iterationsschema von TalInt darstellt.

4.3.1 Überdeckungsfunktionen für TargetLink Systeme

In diesem Unterabschnitt soll exemplarisch demonstriert werden, dass für die Output- und Update Funktionen g_{sys} und h_{sys} des in Anhang A spezifizierten Sprachsubsets von TargetLink einschließende Funktionen existieren, wie sie zur Anwendung von Satz 43 erforderlich sind. Es soll ferner demonstriert werden, dass die Überdeckungsfunktionen auf Intervallen bzw. Boxen optimal sind, d.h. gar nicht "kleiner" gewählt werden können.

Satz 47. *Für jeden, der in Anhang A aufgelisteten TargetLink Blöcke existieren Überdeckungsfunktionen $[g]$ und $[h]$, welche die Output-Funktion g bzw. Update-Funktion h überdecken. Darüber hinaus sind die Überdeckungsfunktionen $[g]$ und $[h]$ auf Intervallen/Boxen als Definitionsbereichen optimal, d.h. es gilt sogar $g([X]) = [g]([X])$ und $h([X]) = [h]([X])$.*

Beweis. Wir greifen exemplarisch einige Blöcke unterschiedlichen Charakters heraus und geben die zugehörigen Überdeckungsfunktionen bzw. Intervallerweiterungen an. Der Nachweis der Überdeckungseigenschaft bzw. auch der jeweiligen Optimalität ist daraus dann offensichtlich.

a) Für den Sum- und Produkt-Block, welche die elementaren arithmetische Operationen $+, -, \cdot, /$, realisieren, werden unmittelbar die in Gl. 28 vorgegebenen analogen Operationen für Intervalle als Überdeckungsfunktionen realisiert. Die Überdeckungsfunktionen sind auf dem kartesischen Produkt zweier Intervalle als Definitionsmenge optimal.

b) Für die unären, parameterfreien Operationen des TargetLink *Math*-Blockes und *Trigonometric* Blockes wie exp, cos, sqrt etc. gelten die in Definition 30 bzw. Definition 31 vorgeschlagenen Überdeckungsfunktionen, die optimal auf Intervallen als Definitionsmengen sind.

c) Sättigungsblock: Der Sättigungsblock, der dazu dient das Eingangssignal auf eine obere bzw. untere Grenze zu sättigen, weist keine Zustände und somit keine Update-Funktion auf, sondern lediglich einen Eingang und je einen Parameter p_o bzw. p_u für die obere und untere Sättigungsgrenze. Die Output-Funktion des Blockes ist gegeben durch

$$\begin{array}{lll} g:\ X \times P_o \times P_u & \to & \mathbb{R} \\ (x,\, p_o,\, p_u) & \to & g(x,\, p_o,\, p_u) = \begin{cases} p_o & \text{falls } x \geq p_o \\ x & \text{falls } p_u < x < p_o \\ p_u & \text{falls } x \leq p_u \end{cases} \end{array}$$

Die naheliegende, minimale Überdeckungsfunktion auf dem kartesischen Produkt der Intervalle hierzu ist dann

$$\begin{array}{lll} [g] : \overline{IR} \times \overline{IR} \times \overline{IR} & \to & \overline{IR} \\ \left([\underline{x},\, \overline{x}]\,,\ [\underline{p_o},\, \overline{p_o}]\,,\ [\underline{p_u},\, \overline{p_u}]\right), & \to & [g]\left([\underline{x},\, \overline{x}]\,,\ [\underline{p_o},\, \overline{p_o}]\,,\ [\underline{p_u},\, \overline{p_u}]\right) \\ & = & \left[\min\left(\max\left(\underline{x},\, \underline{p_u}\right),\, \overline{p}_o\right),\, \max\left(\min\left(\overline{x},\, \overline{p}_o\right),\, \underline{p_u}\right)\right] \end{array}$$

d) *MinMax*-Block: Der Min-Max-Block dient dazu, das Minimum bzw. Maximum eine von n Eingängen $x_1,\, x_2 \ldots x_n$ zu bilden, die Output-Funktion lautet also $g\,(x_1.\, x_2,\, \ldots x_n) = \max\,(x_1.\, x_2,\, \ldots x_n)$ für die Maximum-Bildung und analog für die Berechnung des Minimums. Als minimale Überdeckungsfunktion auf der Definitionsmenge des kartesischen Produktes ergibt sich dann

$$\begin{array}{lll} [g] : \overline{IR} \times \overline{IR} \ldots \times \overline{IR} & \to & \overline{IR} \\ g\,([x_1]\,,\, [x_2]\,,\, \ldots [x_n]) & = & \left[\max\left(\underline{x_1},\, \underline{x_2},\, \ldots \underline{x_n}\right),\, \max\left(\overline{x_1},\, \overline{x_2},\, \ldots,\, \overline{x_n}\right)\right] \end{array}$$

e) *Unit Delay* Bock: Der Unit-Delay Block beinhaltet neben der Ausgangsvariable auch eine Zustandsvariable, so dass eine Überdeckungsfunktion sowohl für die Output- als auch die Update-Funktion bereitgestellt werden muss, die aufgrund der Einfachheit der Output- und Update-Funktion jedoch quasi trivial sind und sich folgendermaßen ergeben:

$$\begin{array}{lcl} [g] : \overline{IR} \times \overline{IR} & \rightarrow & \overline{IR} \\ [g]\,([z]\,,\,[x]\,) & = & [z] \\ [h] : \overline{IR} \times \overline{IR} & \rightarrow & \overline{IR} \\ [h]\,([z]\,,\,[x]\,) & = & [x] \end{array}$$

□

Satz 48. *Für jedes TargetLink System, welches die im Anhang A aufgelisteten Sprachkonstrukte und Blöcke beinhaltet, existiert eine Überdeckungsfunktion* $[g_{sys}]$ *und* $[h_{sys}]$.

Beweis. Die Tatsache folgt direkt aus Satz 47 in Verbindung mit Definition 21 und Satz 22. Für die einzelnen TargetLink Blöcke bzw. elementaren Funktionen existieren nach Satz 47 Überdeckungsfunktionen. Die Verschaltung der TargetLink Blöcke zu einem TargetLink System stellt nach Definition 21 eine faktorisierte Funktion dar, welche aus einzelnen TargetLink Funktionen zusammengesetzt ist. Für die faktorisierte Funktion, d.h. das TargetLink System existieren die Überdeckungsfunktionen dann nach Satz 22. □

4.3.2 Repräsentation des TargetLink Systems als Graph

In diesem Abschnitt soll die Repräsentation des TargetLink Systems in Form eines Graphen beschrieben werden, wie er zur Bestimmung der Wertebereichsgrenzen eingesetzt wird. Auf Basis dieses Graphen werden einerseits initiale Analysen durchgeführt, etwa um die Variablen des Systems zu ermitteln, die für eine effiziente Berechnung mit Intervallverfeinerungen besonders relevant sind. Gleichzeitig ist der Graph jedoch auch für die eigentlichen, an die Analysen anschließenden Berechnungsschritte sehr hilfreich.

Die im Folgenden vorgeschlagene Graphstruktur ist eine sehr naheliegende Repräsentation des betrachteten TargetLink Systems. Für jede Eingangsvariable, Ausgangsvariable, Parametervariable und Zustandsvariable für den Zeitpunkt k und $k+1$ eines Blockes B_i wird jeweils eine Ecke im Graph angelegt. Die zu B_i gehörigen Ecken des Graphen werden anschließend entsprechend der Signalabhängigkeiten der Output- und Update-Funktion g_i und h_i des Blockes durch Kanten verbunden, und zwar in folgender Art und Weise:

- Für jede Komponente y_j des Ausgangsvektors $y(k) = g_i\,(x(k),\, z(k),\, p(k))$ werden anhand der Output-Funktion des Blockes B_i Kanten mit Endpunkt y_j von den Eingangsvariablen x, Zustandsvariablen z und Parametervariablen p eingefügt, für die eine faktische Abhängigkeit besteht.

- Für jede Komponente z_j des Vektors der Zustandsvariablen im nächsten Zeitschritt, also $z(k{+}1) = h_i\,(x(k),\, z(k),\, p(k))$ werden anhand der Update-Funktion des Blockes B_i Kanten mit Endpunkt $z_j(k+1)$ von den Eingangsvariablen x, Zustandsvariablen z und Parametervariablen p eingefügt, für die eine faktische Abhängigkeit besteht.

Bemerkung 49. *Man beachte, dass es unterschiedliche Ecken für eine Zustandsvariable z_j zum Zeitpunkt k einerseits und zum Zeitpunkt $k+1$ andererseits gibt bzw. geben muss. Erstere fungiert in den Output- und Update-Funktionen als unabhängige Veränderliche. Letztere ist hingegen eine abhängige Veränderliche der Update-Funktion. Um die beiden Arten verbal besser unterscheidbar zu machen, sei die Zustandsvariable $z_j(k)$ im aktuellen Zeitschritt als Zustandsvariable, die Variable $z_j(k+1)$ jedoch als die zugehörige Zustandsupdatevariable bezeichnet.*

In Abbildung 4.4 ist exemplarisch für einige Blöcke dargestellt, wie sich anhand der tatsächlich existenten Blockvariablen und der Output- und Update-Funktionen des Blockes der Subgraph für den Block ergibt. Die Ecken des gesamten Graphen ergeben sich dann aus der Gesamtheit der Ecken der individuellen Blöcke. Die Kanten zwischen den einzelnen Blöcken werden exakt entsprechend der Kanten im ursprünglichen TargetLink Modell eingefügt, d.h. es werden die Ausgangsvariablen eines Blockes B mit den Eingangsvariablen der Nachfolger-Blöcke B_j verbunden. Die Gesamtheit der Kanten des Graphen ergibt sich also aus den Kanten des TargetLink Systems ergänzt um die Block-internen Kanten, welche den Signalfluss innerhalb des Blockes repräsentieren.

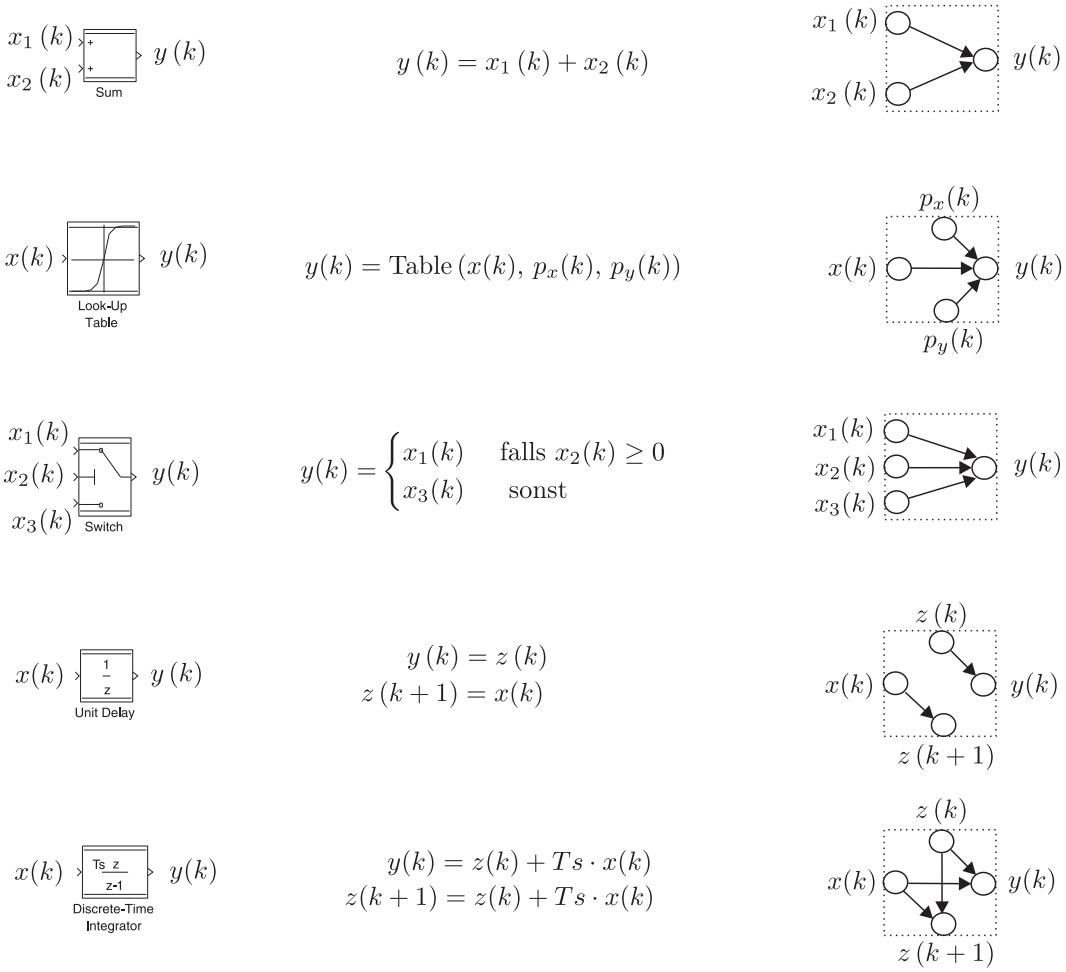


Abbildung 4.4: Repräsentation der Subgraphen für einzelne Blöcke: Für jede Blockvariable, sei es eine Eingangsvariable, Ausgangsvariable, Zustandsvariable, Zustandsupdatevariable oder Parametervariable wird eine Ecke im Subgraphen für den Block angelegt. Anschließend werden die Block-internen Kanten entsprechend der Output- und Update Funktion g und h des Blockes mit den Endpunkten $y(k)$ bzw. $z(k+1)$ gezogen.

Darüber hinaus ist zu beachten, dass TargetLink Systeme durch das Einfügen von Subsystemen hierarchisch strukturiert werden können. Subsysteme werden im Graphen quasi "flachgeklopft", d. h. die Hierarchien werden aufgelöst und der Inhalt des Subsystems wird in den Graphen eingefügt, wobei die Inport- und Outport Blöcke des Subsystems passend verbunden werden, siehe Abbildung 4.5. Der Vorteil dieser Vorgehensweise besteht darin, dass Signalflussanalysen dann sehr einfach auf dem Graphen durchgeführt werden können, statt spezielle Erweiterungen für die hierarchischen Subsysteme zu erfordern.

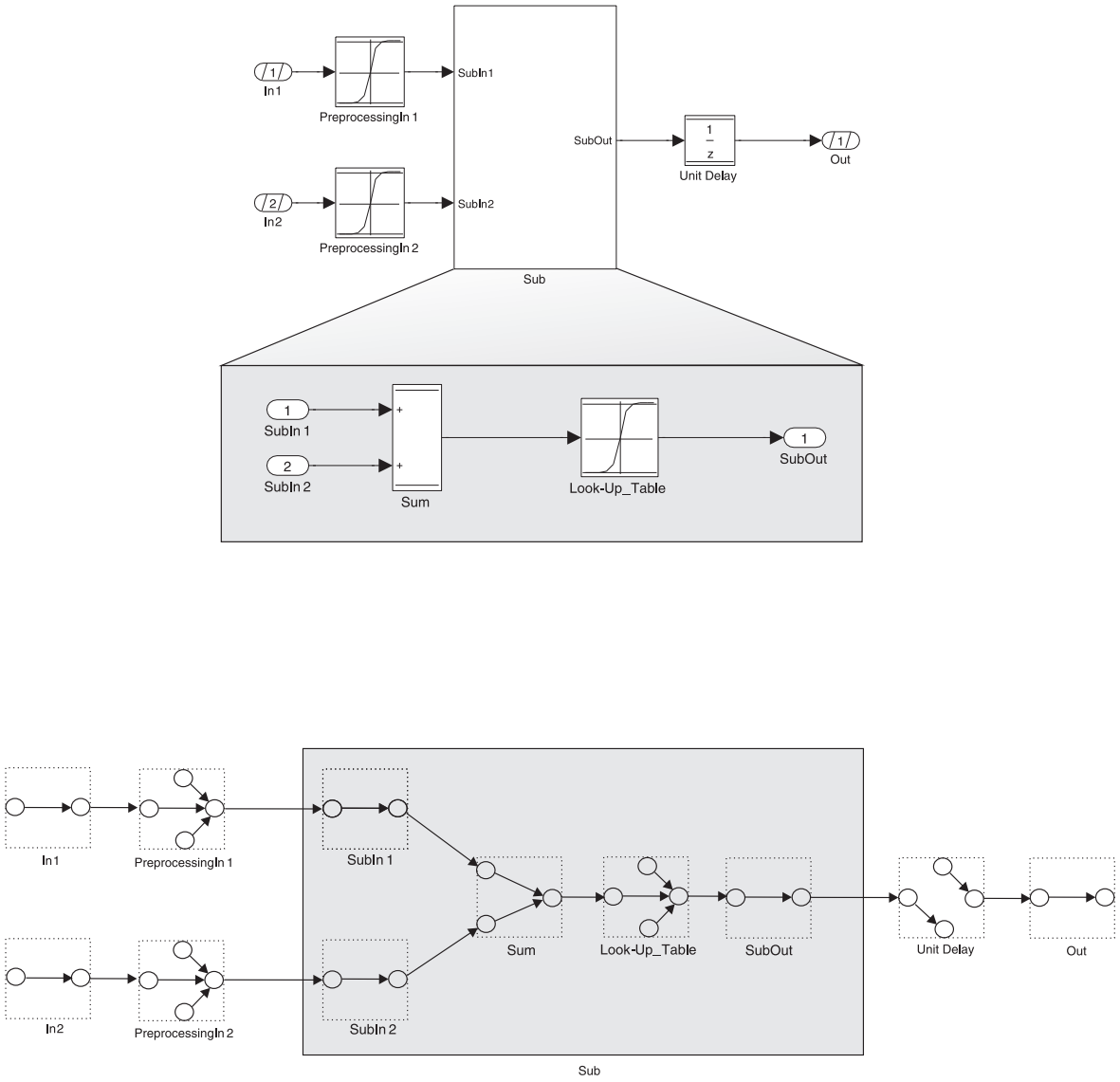


Abbildung 4.5: Graphische Repräsentation eines kompletten TargetLink Systems mit einem darin befindlichen Subsystem zur hierarchischen Strukturierung des Modells. **Oben:** Das eigentliche TargetLink System mit einzelnen Blöcken und einem darin befindlichen Subsystem *Sub*, dessen Inhalt darunter grau hinterlegt ist. Das Subsystem beinhaltet die Blöcke *SubIn1*, *SubIn2*, *Sum*, *Look-up Table* und *SubOut*. **Unten:** Graph für das TargetLink System. Das Subsystem *Sub* wird "flachgeklopft", d.h. der Inhalt des Subsystems wird direkt in den Graphen eingefügt. Der Block des Subsystems *Sub* hat keine direkte Repräsentation innerhalb des Graphen und ist nur der Anschauung halber ausgegraut eingezeichnet.

Insgesamt wird damit das TargetLink System zu einem endlichen, azyklischen Graphen ohne Schlingen oder Mehrfachkanten, den wir im Folgenden als G_{TL} bezeichnen wollen. G_{TL} repräsentiert sowohl die Output-Funktion g_{sys} als auch die Update-Funktion h_{sys} des gesamten TargetLink Systems. Ecken für Parametervariablen, Zustandsvariablen und Eingangsvariablen der Ports auf oberster Ebene repräsentieren die unabhängigen Veränderlichen in den Systemgleichungen g_{sys} und h_{sys} und weisen dementsprechend keine einlaufenden Kanten auf. Die Ecken der Ausgangsvariablen aller Blöcke sowie der Zustandsupdate-Variablen stellen hingegen die abhängigen Veränderlichen der Systemgleichungen dar. Ecken für Eingangsvariablen im Innern des Systems dienen einfach der Verbindung der Blöcke untereinander und haben keine direkte Wirkung.

Für spätere Ausführungen werden die folgenden Schreibweisen verwendet:

- Für die Menge aller Ecken eines Graphen G verwenden wir im Folgenden die Bezeichnung $\mathrm{Nodes}\,(G)$.
- $\mathrm{Node}\,(z_i\,(k))$ bezeichnet die Ecke des Graphen, die der Zustandsvariablen z_i im aktuellen Zeitschritt zugeordnet ist, also $z_i\,(k)$. Dagegen bezeichnet $\mathrm{Node}\,(z_i\,(k+1))$ die Ecke des Graphen, welche der Zustandsupdate-Variablen zu z_i zugeordnet ist, also $z_i\,(k+1)$.
- $\mathrm{Succ}\,(e)$ bezeichne die Menge aller Ecken eines Graphen G, welche Nachfolger der Ecke e sind, also die Menge $\mathrm{Succ}\,(e) = \{e_j \in \mathrm{Nodes}\,(G) \mid \exists\,\mathrm{Weg\ P}\,(e,\, e_j,)\}$.
- $\mathrm{Predec}\,(e)$ bezeichne die Menge aller Ecken eines Graphen G, welche Vorgänger der Ecke e sind, also $\mathrm{Predec}\,(e) = \{e_j \in \mathrm{Nodes}\,(G) \mid \exists\,\mathrm{Weg\ P}\,(e_j,\, e)\}$

4.3.3 Graph-basierte Datenanalyse der Systemvariablen

Die im letzten Abschnitt entwickelte Graphstruktur G_{TL} des TargetLink Systems wird nun in diesem Abschnitt für die folgenden Analysen verwendet:

- die Struktur des Zustandsvektors z_{sys} des TargetLink Systems soll im Hinblick auf eine mögliche Aufteilung in Unterräume reduzierter Dimensionalität untersucht werden, um so die Lösung der Fixpunkt-Gleichung Gl. 4.4 für den Zustandsraum zu vereinfachen. Die Überlegungen hierzu finden sich in Abschnitt 4.3.3.1.
- Die einzelnen Komponenten des Zustandsvektors z_{sys} sollen im Hinblick auf ihre Präsenz in Rückkopplungsschleifen untersucht werden. Dies ist zur Definition der Widening-Operation in der Fixpunkt-Gleichung Gl. 4.4 hilfreich, um so die Konvergenz gegen einen Fixpunkt zu erzwingen. Die Überlegungen hierzu finden sich in Abschnitt 4.3.3.2.
- Die einzelnen Ecken des Graphen werden dahingehend untersucht, wo genau Intervallverfeinerungen zur Reduzierung des Überschätzens durch Dependency- und Wrapping-Effekt bei der Lösung von Gl. 4.4 erforderlich sind und wo dies zwecks Reduzierung des Rechenaufwandes unterbleiben kann. Dies führt auf den Begriff der Bereichsvariablen und der Bereichsintervalle, die in Abschnitt 4.3.3.3 eingeführt werden und die die Basis für die späteren Berechnungen der Intervallerweiterungen darstellen. In Abschnitt 4.3.3.4 wird dazu der TalInt eigene Mechanismus vorgestellt, um festzulegen, welche Variablen innerhalb des TargetLink Systems Bereichsvariablen bilden.

4.3.3.1 Zerlegung des Zustandsraumes in kartesische Produkte zur Dimensionsreduktion

Wie in Abschnitt 3.2.7 dargelegt wurde, ist der gesamte Zustandsraum $Z_{sys}(k)$, d.h., die Menge der möglichen Werte des Zustandsvektors $z_{sys}(k)$ zum Zeitpunkt k aufgrund des Wrapping-Effektes im Allgemeinen nicht das kartesische Produkt der einzelnen Komponenten des Zustandsvektors $z_{sys,i}(k)$, sondern nur eine Teilmenge davon. Als Maßnahme zur Reduzierung des Überschätzens durch den Wrapping-Effekt dient genau wie bei der Reduzierung von Dependency-Fehlern die Überdeckung des Zustandsraumes durch Boxen $[z_i]$ hinreichend kleinen Volumens $w([z_i])$, was nach Definition eine Unterteilung in jeder Dimension erfordert. Daraus wird deutlich, dass eine solche Vorgehensweise ohne genauere Analyse des wirklichen Zustandsraumes bei einem hochdimensionalen Zustandsvektor zu einer sehr großen Zahl von Boxen führt und daher rechenaufwändig ist. Dieser Aufwand ist jedoch im Allgemeinen nicht erforderlich, weil innerhalb eines großen, realen Target-Link Systems naturgemäß viele Komponenten des Zustandsvektors z.B. $z_{sys,i}$ und $z_{sys,j}$ existieren, für die entweder wirklich

$$\{(z_{sys,i}(k),\, z_{sys,j}(k))\} \;=\; Z_{sys,i}(k) \times Z_{sys,j}(k) \;\text{ für alle}\, k \in \mathbb{N}_0$$

gilt oder die Annahme eines kartesischen Produktes zu keinerlei Überschätzen führt. Generell soll im Folgenden der Zustandsraum Z_{sys} geeignet in das kartesische Produkt von Subräumen $Z_{sys,i}$ zerlegt werden, die keinerlei Überschätzen durch den Wrapping-Effekt hervorruft. Dies bedeutet, dass nur noch innerhalb der Subräume $\{(z_{sys,i_1}(k),\, ...,\, z_{sys,i_n}(k))\}$ dem Wrapping-Effekt durch eine Verfeinerung durch Boxen reduzierter Breite entgegengewirkt werden muss. Zur Aufteilung des Zustandsraumes in Subräume wird der folgende, einfache (und gleichzeitig auch etwas grobe) Ansatz anhand der Graph-Repräsentation G_{TL} des TargetLink Systems gemacht:

1. Für jede Zustandsvariable z_j des Systems wird eine zusätzliche Verbindungskante $k_i = (\text{Node}(z_i(k+1)), \text{Node}(z_i(k)))$ eingefügt, also eine gerichtete Verbindung vom Knoten von $z_i(k+1)$ nach $z_i(k)$. Das Resultat dieser Operationen ist die neue Graphstruktur $\hat{G_{TL}}$, siehe Abbildung 4.6.

2. Für jedes Paar von Zustandsvariablen z_i und z_j in $\hat{G}_{TL}$ wird geprüft, ob für deren Nachfolger $\text{Succ}(\text{Node}(z_i(k))) \cap \text{Succ}(\text{Node}(z_j(k))) \neq \emptyset$ gilt. Ist dies der Fall, so gehören z_i und z_j zwangsläufig zum selben Subraum des Zustandsvektors Z_{sys}.

3. Der Zustandsvektor Z_{sys} des Gesamtsystems muss nun so in einzelne Subräume minimaler Größe partitioniert werden, dass in $\hat{G}_{TL}$ für beliebige, aus unterschiedlichen Subräumen stammende z_m und z_n gilt:

$$\text{Succ}(\text{Node}(z_m(k))) \cap \text{Succ}(\text{Node}(z_n(k))) = \emptyset$$

Anschaulich interpretiert bedeutet diese Vorgehensweise, dass nur solche Zustände z_m und z_n in unterschiedlichen Subräumen liegen können, falls deren Werte nie in einem der nachfolgenden Blöcke verknüpft werden. Dies muss nicht nur für den derzeitigen Zeitschritt k sondern auch für alle nachfolgenden Zeitschritte sichergestellt sein. Letzteres ist der Grund, weshalb die zusätzlichen Kanten eingefügt werden, die quasi die "Aktualisierung" eines Zustandes von $z(k+1)$ nach $z(k)$ nach jedem Berechnungsschritt repräsentieren, siehe Abbildung 4.6.

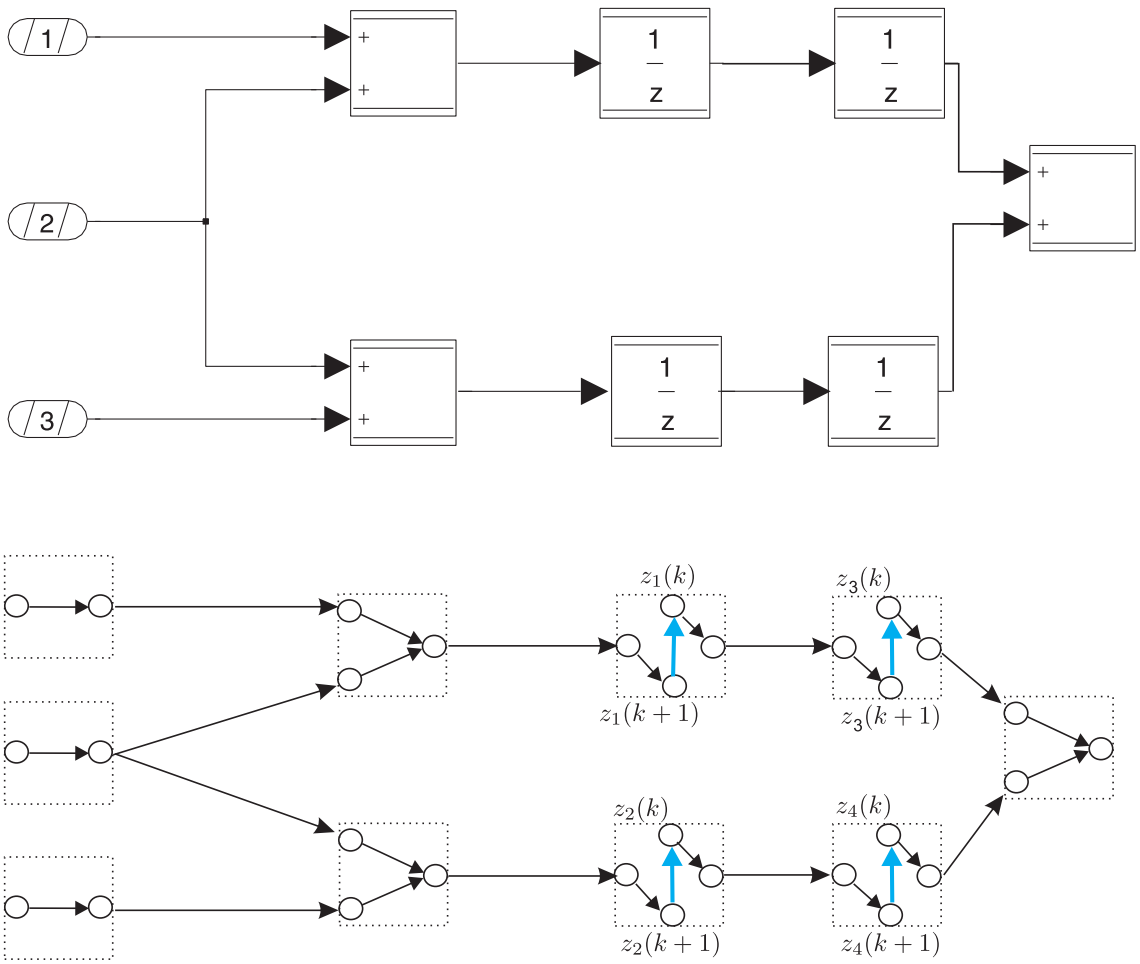


Abbildung 4.6: Oben: Modellfragment zur Demonstration der Abhängigkeit von Zustandsvariablen (hier in den Unit Delay Blöcken), die sich allesamt im selben Subraum befinden, um Überschätzen durch den Wrapping-Effekt zu vermeiden. **Unten:** Durchführung der Analyse der Abhängigkeiten der Zustandsvariablen. Das Einfügen von zusätzlichen Kanten in den Graphen trägt der Tatsache Rechnung, dass Verknüpfungen von Werten von Zustandsvariablen sich gegebenenfalls erst zu späteren Zeitpunkten einstellen.

Bemerkung 50. *Erkennbar ist die obige Vorgehensweise zur Partitionierung des Zustandsraumes recht grob. Sie stellt aber sicher, dass systematisches Überschätzen durch den Wrapping-Effekt aufgrund einer unterstellten Unabhängigkeit von einzelnen Komponenten des Zustandsvektors unterbleibt. Die "Grobheit" der Analyse wirkt sich jedoch gegebenenfalls in erhöhtem Rechenaufwand bei der Berechnung des Zustandsraumes aus, weil dieser gegebenenfalls in mehr Subräume niederer Dimensionalität hätte zerlegt werden können.*

In der gegenwärtigen Version des TalInt-Prototypen ist eine gegenüber dem in diesem Abschnitt beschriebenen Vorgehen leicht modifizierte Variante zur Unterteilung des Zustandsraumes in Unterräume implementiert, die ein gewisses Überschätzen durch den Wrapping-Effekt zugunsten vereinfachter Zustandsräume in Kauf nimmt. In Abschnitt 5.2 werden generell einige Anmerkungen zur Vereinfachung des Zustandsraumes zu Lasten der Genauigkeit gemacht.

4.3.3.2 Identifikation von Zustandsvariablen innerhalb von Rückkopplungsschleifen

Wie anhand des Filter-Beispiels in Abbildung 4.2 schon aufgezeigt wurde, stellen Zustandsvariablen innerhalb von Rückkopplungsschleifen dahingehend eine besondere Herausforderung dar, weil diese im Allgemeinen dazu führen, dass nach endlich vielen Iterationen ohne Widening kein Fixpunkt erreicht wird. Gibt es hingegen keinerlei Rückkopplungsschleifen, so wird grundsätzlich nach maximal $\dim(z_{sys})$ Schritten ein Fixpunkt erreich und die Algorithmik terminiert. Daher ist es wesentlich zu identifizieren, ob es im TargetLink System Rückkopplungsschleifen gibt und welche Komponenten des Zustandsvektors z_{sys} in diesen enthalten sind. Genau auf diese Komponenten des Zustandsvektors wird im Rahmen der Fixpunkt-Iteration eine Widening-Operation angewandt um die Konvergenz in einen Fixpunkt zu erzwingen. Die Analyse wird mit Hilfe der folgenden, elementaren Algorithmik vorgenommen:

1. Ausgehend von der Graphstruktur G_{TL} des TargetLink Systems wird für jede Zustandsvariable $z_{sys,i}$ eine Kante $k_i = (\text{Node}(z_{sys,i}(k+1)), \text{Node}(z_{sys,i}(k)))$ eingefügt, also eine gerichtete Verbindung vom Knoten von $z_{sys,i}(k+1)$ nach $z_{sys,i}(k)$. Das Resultat dieser Operationen ist die neue Graphstruktur $\hat{G}_{TL}$, siehe Abbildung 4.7.

2. Für jede Komponente $z_{sys,i}$ des Zustandsvektors wird geprüft, ob es in der neuen Graphstruktur $\hat{G}_{TL}$ einen Pfad von Node$((z_{sys,i}(k))$ nach $\text{Node}(z_{sys,i}(k+1))$ gibt. Wir fassen diese Komponenten $z_{sys,i}$ des Zustandsvektors in der Menge $Z_{sys,Feedback}$ zusammen.

Abbildung 4.7 zeigt die Vorgehensweise anhand des bekannten Trivialbeispiels. In Abschnitt 4.3.4.2 wird auf die Menge $Z_{sys,Feedback}$ zurückgegriffen, um die Lösung der Fixpunkt-Gleichung zu erzwingen. Sofern $Z_{sys,Feedback} = \emptyset$ gilt, tritt die Konvergenz nach endlich vielen Schritten ohne Widening-Operation automatisch ein.

4.3.3.3 Bereichsvariablen und Bereichsintervalle

Wie bereits zuvor angemerkt wurde, weisen TargetLink Modelle nicht selten eine zwei- oder sogar dreistellige Zahl von Eingangs- und Ausgangsvariablen sowie nicht selten Hunderte von Blöcken auf, die im Rahmen einer Analyse mit TalInt berücksichtigt werden müssen. Für eine praxistaugliche Lösung zur Abschätzung von Wertebereichsgrenzen in TargetLink Modellen ist es daher essentiell, dass der Rechenaufwand für TalInt in Grenzen gehalten wird. Unter diesem Aspekt besonders kritisch zu sehen, ist eine direkte Anwendung von Satz 33, der als Maßnahme zur Reduzierung des Überschätzens durch Dependency- und Wrapping-Effekt Intervallverfeinerungen aller unabhängiger Veränderlicher vorsieht. In Bezug auf ein TargetLink System würde dies bedeuten, dass eine Intervallverfeinerung für sämtliche Eingangs-, Parameter- und Zustandsvariablen des TargetLink Systems vorzunehmen wäre, was bereits bei TargetLink Systemen mittlerer Größe schnell in eine "kombinatorische Explosion" ausarten kann. Glücklicherweise ist diese Vorgehensweise für typische TargetLink Modelle auch keineswegs erforderlich, wie anhand von Abbildung 4.8 verdeutlicht wird. Die Eingangsvariablen $In1$ bis $In4$ des Modellfragments wie auch die Ausgangssignale der nachfolgenden Blöcke werden jeweils nur ein

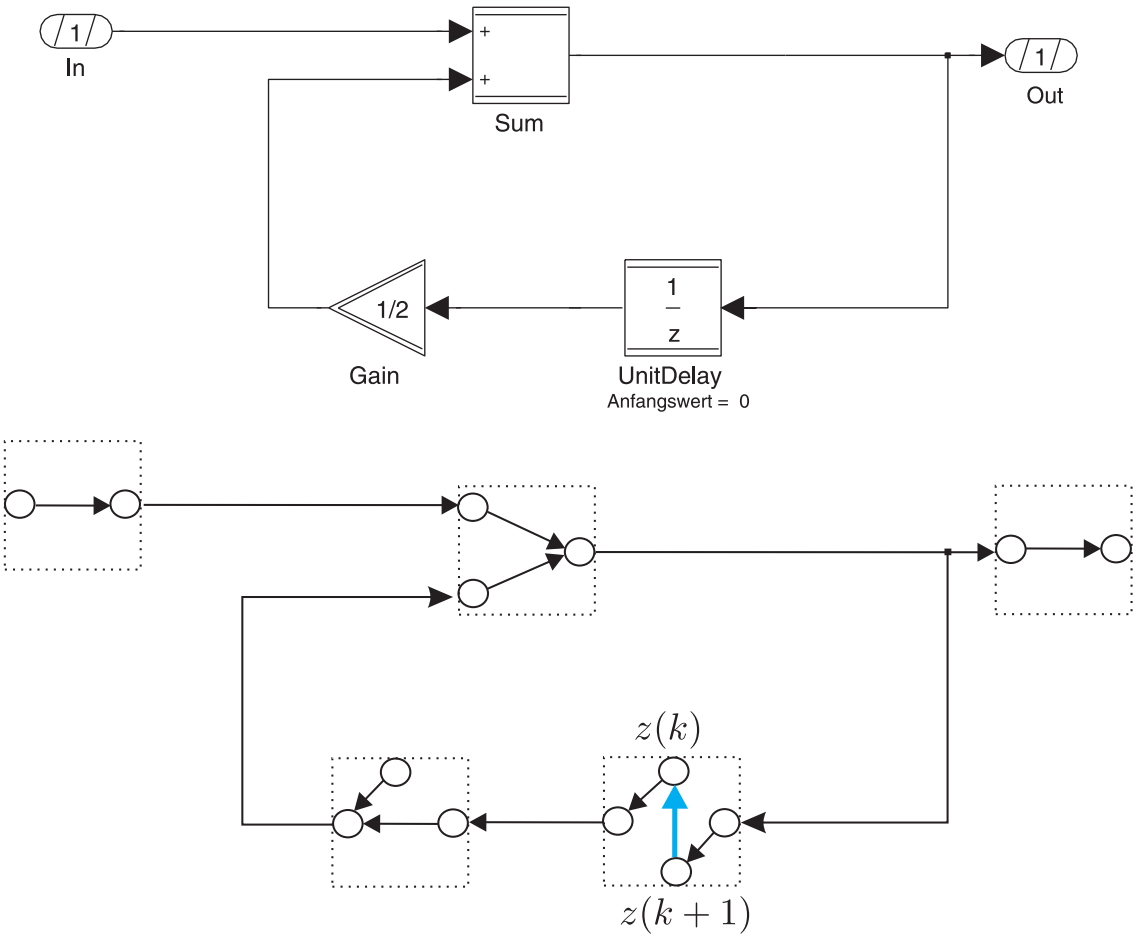


Abbildung 4.7: Oben: Modellfragment eines einfachen IIR-Filters analog zu Abbildung 4.2. **Unten:** Zugehöriger Modellgraph $\hat{G}_{TL}$. Zur Identifikation von Zuständen, die sich innerhalb von Rückkopplungsschleifen befinden, werden für jede Zustandsvariable, zusätzliche Kanten von jeder $z_{sys,i}\,(k+1)$ Ecke zur zugehörigen $z_{sys,i}\,(k)$ Ecke eingefügt (die blau eingezeichnete Kante).

einziges Mal "verwendet" (*outdeg* der Ausgänge ist jeweils 1), was dazu führt, dass es keinerlei Abhängigkeiten zwischen den Systemgrößen gibt (keine Dependency-Fehler). Da die Intervallerweiterungen für die verwendeten TargetLink Blöcke auf Intervallen/Boxen als Definitionsbereiche keinerlei Überschätzen verursachen, siehe Satz 47 und die Definitionsbereiche aufgrund der Unabhängigkeit der internen Zwischenvariablen jeweils Boxen/Intervalle sind, ergibt sich iterativ, dass der Wertebereich des Ausgangssignals des *Multiplication* Blockes auch ohne Intervallverfeinerungen exakt ermittelt wird. Folglich müssen z.B. für die Eingangsvariablen in den Blöcken $In1$ bis $In4$ keine Intervallverfeinerungen vorgenommen werden. Letztere können hingegen selektiv auf Zwischengrößen angewandt werden, wo sie einen Effekt haben.

Die in TalInt gewählte Vorgehensweise zur Reduktion des Überschätzens bei gleichzeitiger Beachtung der Rechenaufwände besteht nun darin, Intervallverfeinerungen sehr selektiv auf Basis heuristischer Kriterien auf einzelne Variablen eines TargetLink Systems anzuwenden statt auf die Gesamtheit aller unabhängigen Veränderlichen. Satz 33, in dem Letzteres zur Reduktion des Überschätzens vorgeschlagen wird, ist in seiner Generalität

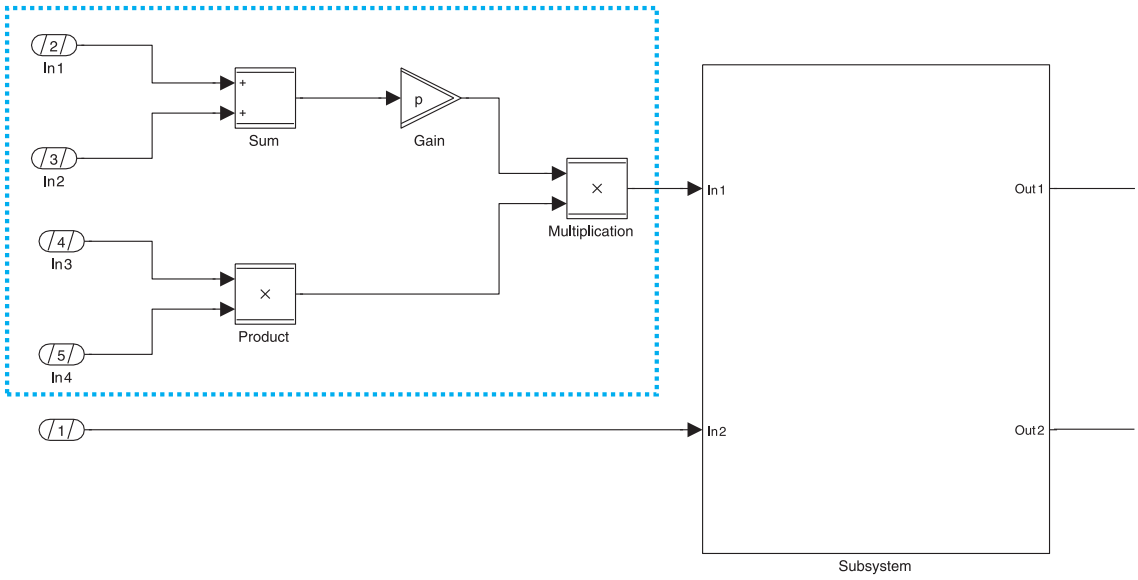


Abbildung 4.8: Blau umrandetes Fragment eines TargetLink Systems mit einer Subfunktion, für die keinerlei Überschätzen durch Dependency- bzw. Wrapping-Effekte auftritt.
.

auch automatisch sehr konservativ und führt bei Modellen aus der Praxis sehr schnell zu einem explosionsartigen Anstieg der Rechenzeit.

Variablen eines TargetLink Systems, für die explizit eine Intervallverfeinerung durch TalInt durchgeführt wird, sollen im Folgenden als Bereichsvariablen bezeichnet werden. Dementsprechend wird für Bereichsvariablen der mögliche Wertebereich in einzelne Intervalle reduzierter Breite unterteilt, wie dies in Abbildung 4.9 links für die Bereichsvariable $bvar_i$ dargestellt ist. Die einzelnen Bereiche $\left[bvar_i^j\right]$ werden durch einen einfachen Index j identifiziert und unterteilen den derzeitigen Wertebereich der Bereichsvariablen in eine Reihe von Intervallen $\left[bvar_i^j\right]$ reduzierter Breite. Da Bereichsvariablen nicht notwendigerweise unabhängige Veränderliche eines TargetLink Systems sein müssen, sondern gegebenenfalls auch Zustandsvariablen, Zustandsupdatevariablen oder normale Blockausgänge sein können, ist die Wertebereichsskala einer Bereichsvariablen $bvar_i$ im Allgemeinen nicht konstant sondern wächst mit den einzelnen Iterationsschritten. Dazu wird die Skala im Zuge der Iterationen immer dann erweitert, sofern die neu berechneten Werte der Variablen $bvar_i$ außerhalb der derzeitigen Skala liegen. In den Abbildungen 4.13, 4.14 und 4.15 bzw. im Anhang B.2.2 ist dargestellt, wie die Skala einer Bereichsvariablen initialisiert bzw. erweitert wird, was durch TalInt-Parameter gesteuert wird, siehe Anhang B.2.2.

Da naturgemäß diverse Bereichsvariablen in einem TargetLink System existieren und diese über die unterschiedlichen Blockoperationen miteinander verknüpft werden, sind die verfeinerten Bereiche, auf denen die Intervall-Arithmetik ausgeführt wird, im Allgemeinen Boxen die sich aus den kartesischen Produkten der verfeinerten Intervalle der einzelnen Bereichsvariablen $bvar_{i_1}$ bis $bvar_{i_n}$ ergeben:

$$Box_{i_1,i_2..i_n}^{j_1,j_2,..j_n} = \left[bvar_{i_1}^{j_1}\right] \times \left[bvar_{i_2}^{j_2}\right] ..... \times \left[bvar_{i_n}^{j_n}\right] \tag{4.12}$$

Die Boxen werden im gewissen Sinne von den Bereichsvariablen i_1, i_2 bis i_n aufgespannt und die Kenntnis, um welche Bereichsvariablen es sich dabei handelt ist notwendig, um

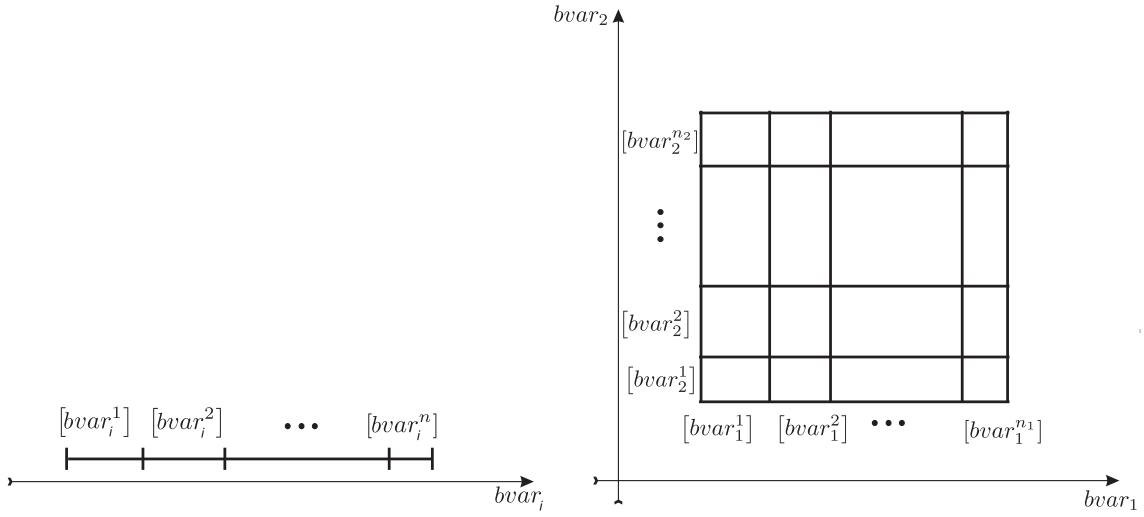


Abbildung 4.9: Links: Der gesamte Wertebereich einer Bereichsvariablen $bvar_i$ wird durch ein System von Intervallen $\left[bvar_i^j\right]$ mit reduzierter Breite $w\left(\left[bvar_i^j\right]\right)$ überdeckt, um das Überschätzen durch Dependency und/oder Wrapping-Effekte zu reduzieren. Nur für Bereichsvariablen wird eine Verfeinerung vorgenommen **Rechts:** Insgesamt ergibt sich damit für die Verfeinerungen der einzelnen Variablen ein Gitter (hier nur dargestellt für 2 Bereichsvariablen $bvar_1$ und $bvar_2$) für die einzelnen Bereiche, wobei die Systemfunktionen g_{sys} und h_{sys} dann für jeden Bereich, d.h. jede Box separat ausgewertet wird.

Intervall-Arithmetik auf den Boxen betreiben zu können. Für den Fall zweier Bereichsvariablen bildet die Gesamtheit aller möglichen Boxen ein Gitter, wie dies in Abbildung 4.9 rechts dargestellt ist. Intervallarithmetik wird jetzt auf den einzelnen Boxen, die mit Ausnahme der Randpunkte allesamt disjunkt sind, getrennt betrieben. Hierdurch wird die gewünschte Intervall- bzw. Boxverfeinerung erreicht, um das Überschätzen durch den Dependency- bzw. Wrapping Effekt zu reduzieren. Vergleicht man dieses Vorgehen mit Satz 33, so erkennt man dass der Gedanke der Verfeinerung naturgemäß derselbe ist, dass die Verfeinerungen in TalInt jedoch selektiv auf ausgewählte Variablen angewandt werden, die nicht notwendigerweise unabhängige Veränderliche sind und deren Wertebereichsskalen daher im Allgemeinen adaptiert werden müssen.

Eine Box Box_k (k als Index der einfacheren Schreibweise wegen, denn typischerweise wird in einem Block mit einer Vielzahl solcher Boxen gerechnet) wie sie durch Gl. 4.12 als kartesisches Produkt der einzelnen Verfeinerungen $\left[bvar_i^j\right]$ von Bereichsvariablen gegeben ist, soll in Verbindung mit dem zugehörigen Intervallwert $[x_k]$, den eine Variable x des TargetLink Systems auf dieser Box Box_k annimmt, durch einen besonderen Namen ausgezeichnet sein. Wir nennen ein solches geordnetes Paar (Box_k, $[x_k]$) im Folgenden ein sogenanntes Bereichsintervall, weil es den Intervall-Wert einer Variablen auf einem Bereich (der Box Box_k) beschreibt. Solche Bereichsintervalle sind die Datenstrukturen, die bei der Berechnung der Intervall-Output- und Intervall-Update-Funktion $[g_i]$ und $[h_i]$ später verwendet werden. Konkret werden bei der Berechnung eines kompletten Iterationsschrittes in TalInt im Wesentlichen Berechnungen auf solchen Bereichsintervallen (Box_k, $[x_k]$) vorgenommen. Naturgemäß muss zur korrekten Berechnung mit Bereichsintervallen jeweils bekannt sein, welche Bereichsvariablen die Box jeweils aufspannen.

In Abbildung 4.10 ist exemplarisch der Umgang mit Bereichsintervallen dargestellt. Die

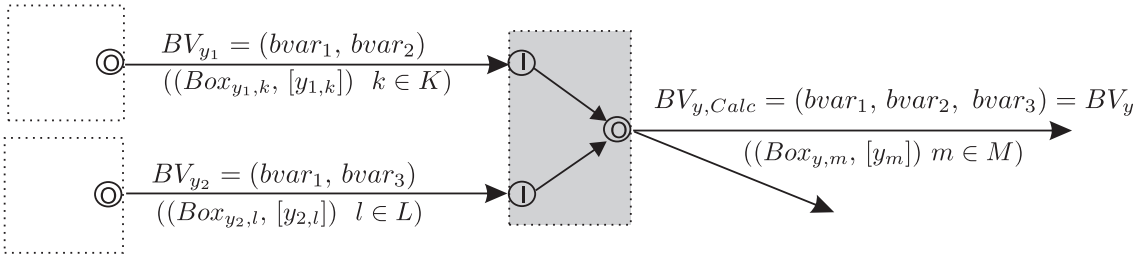


Abbildung 4.10: Umgang mit Bereichsintervallen $(Box_k,\ [x_k])$ bei Operationen. Abhängig von den Bereichsvariablen, die die einzelnen Boxen aufspannen ergeben sich im Allgemeinen neue Boxen für die Bereichsintervalle am Ausgang

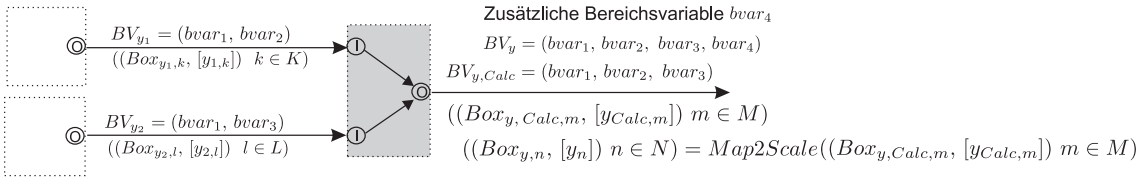


Abbildung 4.11: Umgang mit Bereichsintervallen $(Box_k,\ [x_k])$ bei Operationen. Im Unterschied zu Abbildung 4.10 ist die Ausgangsvariable des Blockes hier selbst eine Bereichsvariable. Folglich werden erst analog zu Abbildung 4.10 die Bereichsintervalle $((Box_{y,\,Calc,m},\ [y_{Calc,m}])\ m \in M)$ auf Basis der Eingänge berechnet und diese Bereichsintervalle dann per Map2Scale-Abbildung (siehe Abbildung 4.12) auf die Skala von var_4 abgebildet. Als Resultat ergeben sich die Bereichsintervalle $((Box_{y,n},\ [y_n])\ n \in N)$

Boxen der Bereichsintervalle am Ausgang werden von der Gesamtheit aller Bereichsintervalle am Eingang aufgespannt. In Abbildung 4.11 ist zudem der Fall dargestellt, dass die Ausgangsvariable des Blockes selbst wieder eine Bereichsvariable ist. In diesem Fall wird zunächst analog zu 4.10 verfahren und die sich ergebenden Bereichsintervalle dann mittels einer als $Map2Scale$ bezeichneten Transformation auf die Skala der Bereichsvariable am Ausgang abgebildet. Die Wirkung von $Map2Scale$ ist in Abbildung 4.12 verdeutlicht.

Falls der zu berechnende Knoten eines Systems eine eigene Bereichsvariable aufweist, so muss zur Abbildung auf dessen Skala eine Operation durchgeführt werden, die im Folgenden Map2Scale heißen soll und deren Funktionsweise in Abbildung 4.12 dargestellt ist. Der Ausgangspunkt ist eine abgeschlossene Berechnung der Bereichsintervalle eines Output oder eines Zustands-Update-Knotens.

4.3.3.4 Festlegung der Bereichsvariablen

Nachdem im vorherigen Abschnitt dargestellt wurde, dass Intervall- bzw. Boxverfeinerungen ausschließlich für ausgewählte Bereichsvariablen vorgenommen werden, nicht jedoch für die Gesamtheit aller Eingangsvariablen, Parameter und Zustandsvariablen soll in diesem Abschnitt beschrieben werden, wie TalInt festlegt, welche Variablen innerhalb eines TargetLink Systems zu Bereichsvariablen gemacht werden. Die Menge der Bereichsvariablen des gesamten TargetLink Systems wird im Folgenden als BV bezeichnet.

Um die Berechnung des Zustandsraums in Abschnitt 4.18 unter Vermeidung von Wrapping-

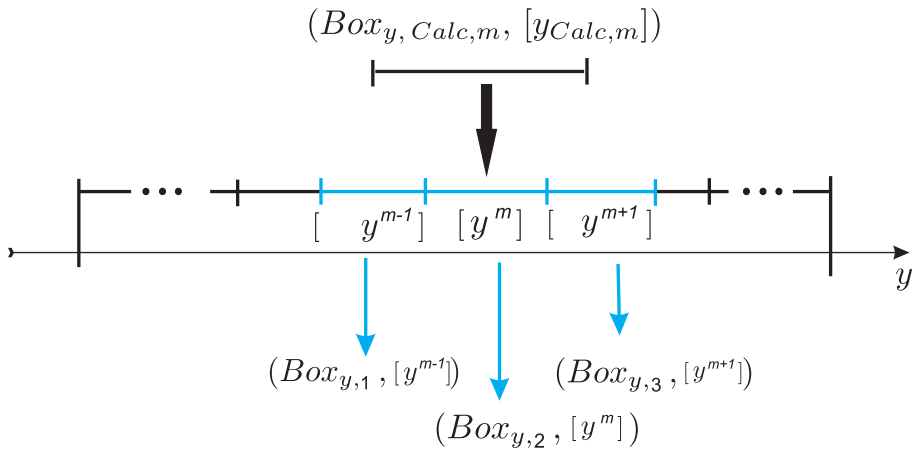


Abbildung 4.12: Realisierung der Map2Scale-Abbildung für ein vorgegebenes Bereichsintervall auf die Skala von Bereichsvariable var_i unter der Voraussetzung dass die Skalenerweiterung für var_i bereits vorgenommen wurde.

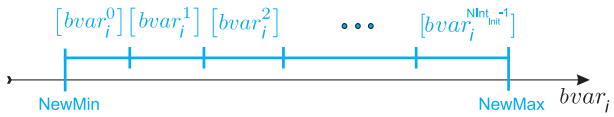


Abbildung 4.13: Initialisierung der Skala einer Bereichsvariablen $bvar_i$ auf den Wertebereich $[NewMin,\ NewMax]$, der in $NInt_{Init}$ äquidistante Intervalle unterteilt wird. Diese Intervalle stellen die Skalenbereiche $bvvar_i^j$ dar.

Fehlern zu vereinfachen bzw. zu ermöglichen und einfach prüfen zu können, ob ein Fixpunkt des Zustandsraums erreicht ist, sollte letzterer diskretisiert sein. Daher wird in TalInt (derzeit) die Vorgabe gemacht, dass alle Zustands- und Zustandsupdate-Variablen des TargetLink Systems grundsätzlich Bereichsvariablen darstellen, somit also $z_j(k) \in BV$ und $z_j\,(k+1) \in BV$ gilt ($z_j\,(k)$ bezeichne hierbei eine beliebige Zustands-Variable und $z_j\,(k+1)$ die zugehörige Zustandsupdate-Variable.) Man beachte, dass es für beide Variablen separate Ecken in G_{TL} gibt und auch geben muss, da sie während einer Iteration naturgemäß unterschiedliche Werte annehmen. $z_j\,(k+1)$ und $z_j\,(k)$ weisen in TalInt jedoch dieselbe Skala auf (siehe Abbildung 4.9 links), um prüfen zu können, ob ein Fixpunkt vorliegt und weil beim Übergang vom Iterationsschritt $k \to k+1$ ja die eine Größe durch die andere aktualisiert wird.

Ob abgesehen von Zustands- und Zustandsupdate-Variablen noch weitere Größen eines TargetLink Systems zu Bereichsvariablen gemacht und somit verfeinert bzw. diskretisiert werden, wird anhand einer Analyse des Graphen G_{TL} entschieden. Die Motivation hierbei ist ausschließlich, das Überschätzen durch Dependency- und Wrapping Effekte zu reduzieren. Hierzu wird die folgende, heuristische Vorgehensweise in TalInt gewählt:

1. Zunächst werden in einem ersten Schritt die System-Variablen ermittelt, die überhaupt ein Überschätzen verursachen können und zwar auf folgende Art und Weise:

   (a) Zunächst wird der Graph G_{TL} erweitert, indem für jedes Paar von Zustandsvariablen $(z_{i_1},\ z_{i_2})$, die sich nach Abschnitt 4.3.3.1 im selben Zustands-Subraum

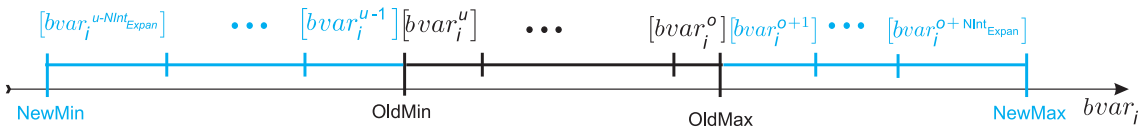


Abbildung 4.14: Erweiterung der Skala einer Bereichsvariablen $bvar_i$ nach oben und/oder unten ohne Widening. Sofern $NewMax$ bzw. $NewMin$ außerhalb der bisherigen Skala liegen, wird die Skala erweitert. Bei Überschreiten nach oben ist der neue Maximalwert der Skala durch den Wert $NewMax$ gegeben. Bei Unterschreiten nach unten ist der neue Minimalwert der Skala durch den Wert $NewMin$ gegeben. Die Erweiterungen der Skala sind blau eingefärbt.

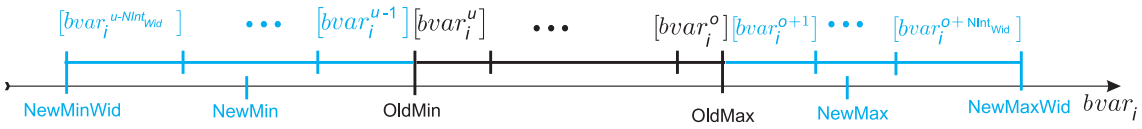


Abbildung 4.15: Erweiterung der Skala einer Bereichsvariablen $bvar_i$ nach oben und/oder unten mit Widening. Sofern $NewMax$ bzw. $NewMin$ außerhalb der bisherigen Skala liegen, wird die Skala erweitert und zwar über die Werte $NewMax$ bzw. $NewMin$ hinaus. Bei Überschreiten nach oben ist der neue Maximalwert der Skala durch den Wert $NewMaxWid$ gegeben. Bei Unterschreiten nach unten ist der neue Minimalwert der Skala durch den Wert $NewMinWid$ gegeben. Die Erweiterungen der Skala sind blau eingefärbt.

Z_{sub_i} befinden, eine neue Ecke eingefügt und diese in der in Abbildung 4.16 dargestellten Weise durch zwei zusätzliche Kanten mit den zugehörigen Zustands-Update-Variablen der Zustände verbunden. Das Resultat ist der modifizierte Graph $\tilde{G}_{TL}$.

(b) Das Überschätzen wird für TargetLink Blöcke ursächlich durch die Ecken e verursacht, deren Fan-Out mindestens zwei beträgt und unter deren direkten Nachfolgern es mindestens zwei Ecken e_{n_1} und e_{n_2} gibt, die selbst wieder einen gemeinsamen (nicht notwendigerweise direkten) Nachfolger in $\tilde{G}_{TL}$ haben, also $Succ(e_{n_1}) \cap Succ(e_{n_1}) \neq \emptyset$. Anschaulich bedeutet, dies, dass das Ausgangssignal der Ecke e verzweigt und später in einer weiteren Ecke wieder zusammengeführt wird, wo es dann zu Überschätzen durch den Dependency-Effekte kommt. Genau diese Ecken e werden markiert.

2. Im zweiten Schritt wird geprüft, ob die markierten Ecken e aus 1. bereits durch Ihre Vorgängersignale hinreichend fein aufgelöst sind oder ob die Ausgangsvariable der markierten Ecke e eine Intervallverfeinerung erfahren und somit zu einer zusätzlichen Bereichsvariable werden soll. Dies geschieht in der folgenden Art und Weise:

   (a) Die einzelnen Blöcke des TargetLink Systems werden in ihrer durch Simulink definierten Abarbeitungsreihenfolge aufgerufen, so dass für alle Variablen Signale der Reihe nach durch den Graphen durchpropagiert wird, ob diese eine etwaige Intervallverfeinerung ihrer Vorgänger "erben" also selbst noch hinreichend fein aufgelöst sind, oder nicht.

(b) Eine in 1. markierte Ecke e wird zu einer neuen Bereichsvariablen, falls sie überhaupt keinen Eingang hat (dies trifft für den TargetLink Constant-Block zu) oder mindestens einer der Eingänge keine Intervallverfeinerung aufweist. Nachdem die zu e gehörige Variable zu einer Bereichsvariablen geworden ist, können nachfolgende Ecken von dieser Intervallverfeinerung profitieren und diese "erben".

Die Vorgehensweise ist in Abbildung 4.17 veranschaulicht.

Bemerkung 51. *Natürlich handelt es sich bei der obigen Vorgehensweise im Wesentlichen um eine bloße Heuristik. Andere Ansätze sind natürlich denkbar. Man mache sich aber klar, dass es hier nur eine Strategie geht, das Überschätzen mit "begrenztem" rechentechnischen Aufwand möglichst gering zu halten. Die Überdeckungseigenschaft der entwickelten Intervallfunktionen ist davon unberührt.*

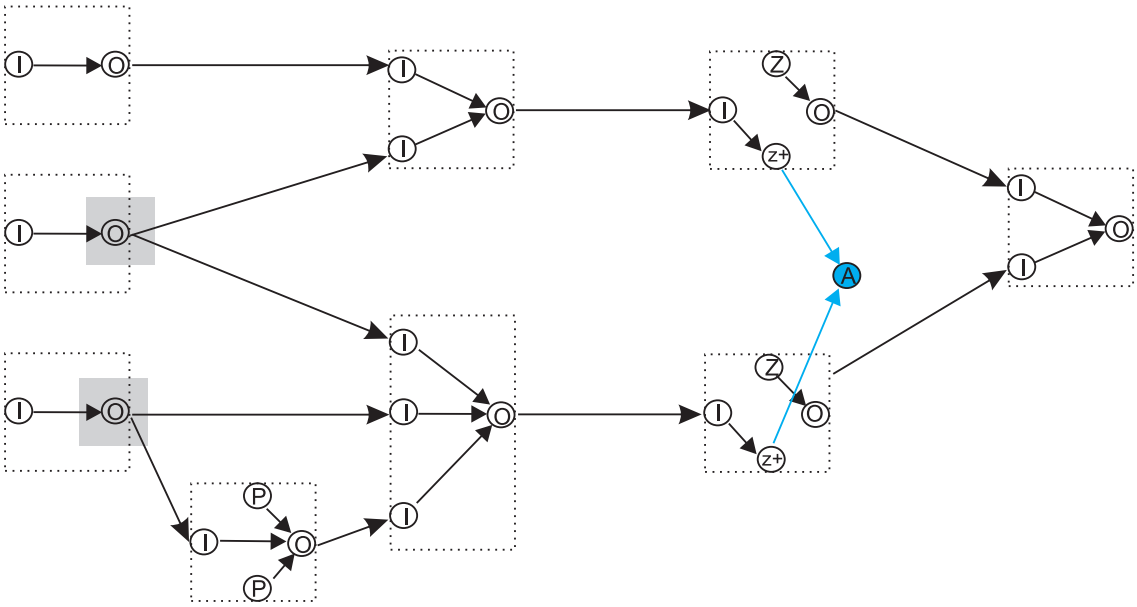


Abbildung 4.16: Graph $\tilde{G}_{TL}$ der aus dem regulären TargetLink Graphen G_{TL} durch das Einfügen von zusätzlichen Ecken und Kanten für jedes Zustandspaar innerhalb desselben Subraums Z_{sub_i} hervorgeht. Die einzelnen Blöcke sind gestrichelt eingezeichnet und die Ecken des Graphen nach ihrer Art bezeichnet. Ecken mit enthaltenem I bezeichnen die Input-Ecken eines Blockes, Ecken mit enthaltenem O repräsentieren Output-Ecken, Ecken mit enthaltenem P repräsentieren Parameter-Ecken, Ecken mit enthaltenem Z repräsentieren Zustands-Ecken, Ecken mit enthaltenem z+ repräsentieren Zustandsupdate-Ecken. Da die beiden Zustands-Ecken im selben Subraum Z_{sub_i} liegen, muss der Wrapping-Effekt zur Berechnung des Zustandsraumes ohne Überschätzen berücksichtigt werden. Durch die zusätzlich eingefügten Ecken und Kanten kann die Abhängigkeitsanalyse in 1.(b) unter Berücksichtigung der Zustände durchgeführt werden.

Bemerkung 52. *Punkt 1.(a) trägt der Tatsache Rechnung, dass Ecken e, welche mehrere Zustandsupdate-Variablen aus dem selben Subraum als Nachfolger haben, dafür sorgen, dass es Überschätzen durch den Wrapping-Effekt zwischen den Zustandsvariablen gibt, was durch Intervallverfeinerungen der zu e gehörenden Variable reduziert werden kann.*

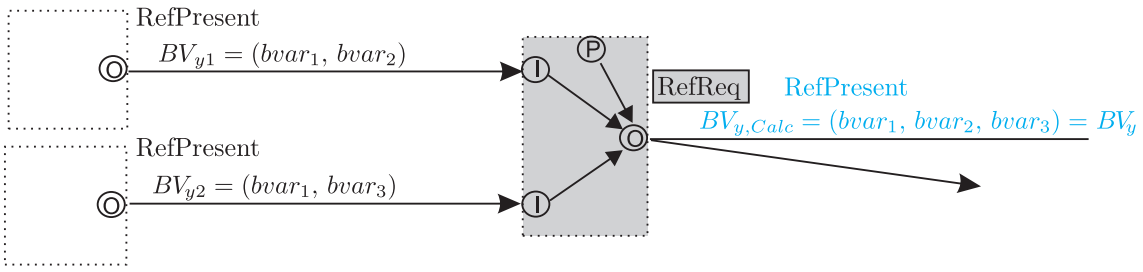


Abbildung 4.17: Heuristische Vorgehensweise zur Festlegung, welche Blockvariablen als Bereichsvariablen eine Intervallverfeinerung erfahren sollen, um das Überschätzen der Intervall-Arithmetik zu verringern. RefPresent und RefRequired sind Flags, die angeben, ob eine Verfeinerung (Refinement) vorhanden bzw. benötigt wird.

Das Einfügen der Ecken und Katen sorgt dann dafür, dass der in 1.(b) eingesetzte Mechanismus zur Detektion von Abhängigkeiten auch auf die Zustandsvariablen wirkt. In 2.(b) fokussiert sich TalInt hinsichtlich der "Vererbung" von Intervallverfeinerungen auf Eingangsvariablen, weil Zustandsvariablen ohnehin Bereichsvariablen sind. Parameter-Variablen in TargetLink Blöcken haben darüber hinaus die Eigenschaft, dass sie Intervallverfeinerungen von Eingängen und Zuständen erhalten und somit ignoriert werden.

4.3.4 Berechnung von Block-Output- und Block-Update Funktionen sowie des Zustandsraums

In diesem Abschnitt wird schließlich dargestellt, wie auf Basis von Bereichsintervallen die Intervallerweiterungen der Block-Output- und Block-Update Funktionen $[g_i]$ und $[h_i]$ berechnet werden. Die Berechnung des Zustandsraumes geschieht weitestgehend analog, wobei auch hier Bereichsintervalle wieder die entscheidende Rolle spielen.

4.3.4.1 Berechnung der Intervall-Output-Funktion einzelner Blöcke

Die Berechnung der Intervall-Output Funktion $[g]$ eines jeden Blockes B läuft in den folgenden Schritten ab:

- Zuerst wird anhand der funktionalen Abhängigkeiten in $y = g\,(x,\, z,\, p)$ geprüft, von welchen Größen die Ausgangsvariable y wirklich abhängt und von welchen Bereichsvariablen $bvar_i$ die Box-förmigen Bereiche für x, z und p aufgespannt werden, siehe Abbildung 4.11. Die Vereinigung aller dieser unterschiedlichen Bereichsvariablen $(bvar_1,\, bvar_2,\, ...,\, bvar_n)$ spannt dann die Boxen $Box_{y,Calc,i}$ $(i \in I)$ der Bereichsintervalle zur Berechnung von y auf.

- Für jede mögliche dieser Boxen $Box_{y,Calc,i}$ $(i \in I)$, mithin also jedes mögliche Tupel $\left(bvar_1^{j_1},\, bvar_2^{j_2},\, ...,\, bvar_n^{j_n}\right)$ wird der zugehörige Intervallwert der Eingänge $[x_i]$ der Zustände $[z_i]$ und Parameter $[p_i]$ ermittelt. Aufgrund der Konstruktion der Boxen von y ist die hierzu zu selektierende Box für alle Eingangsvariablen, Zustandsvariablen und Parametervariablen eindeutig bestimmt.

- Auf Basis von $[g]$ und den ermittelten Intervallwerten $[x_i]$, $[z_i]$ und $[p_i]$ wird der zugehörige Wert $[y_i]$ ermittelt, der zusammen mit $Box_{y,Calc,i}$ nun ein berechnetes Bereichsintervall bildet. Diese Schritte werden für jede vorgegebene Box $Box_{y,Calc,i}$ $(i \in I)$ wiederholt.

- Hat der Blockausgang keine eigene Bereichsvariable so bilden die Bereichsintervalle $(Box_{y,Calc,i}, [y_i]) = (Box_{y,i}, [y_i])$ $(i \in I)$ bereits das gewünschte Ergebnis für den Ausgang. Weist der Ausgang jedoch eine eigene Bereichsvariable $bvar_y$ auf, so wird die *Map2Scale*-Abbildung auf die Bereichsintervalle $(Box_{y,Calc,i}, [y_i])$ $(i \in I)$ angewandt und die Abbildung liefert daraus die "verfeinerten" Bereichsintervalle $(Box_{y,i}, [y_i])$ $(i \in I)$ für den Ausgang. Eine etwaige Skalenausdehnung für die Bereichsvariable $bvar_y$ wird ohne Widening durchgeführt, siehe Abbildung 4.14, da letzteres prinzipiell nur für Zustandsvariablen angewandt wird.

Als Resultat der Berechnung beinhaltet der Blockausgang jetzt eine aktualisierte Menge von Bereichsintervallen $(Box_{y,i}, [y_i])$ $(i \in I)$ und die Berechnung ist abgeschlossen.

Bemerkung 53. *Vereinfacht und leger kann man die obige Vorgehensweise folgendermaßen zusammenfassen: Iteriere über alle denkbaren Boxen der Ausgangsvariablen; beschaffe zu dieser Box jeweils die zugehörigen Intervall-Werte für Eingangsvariablen, Zustände und Parameter; berechnete mittels $[g]$ und den ermittelten Intervallen die Größe $[y]$ und konstruiere aus der Box und $[y]$ das gesuchte Bereichsintervall und zwar entweder direkt oder durch eine weitere Intervallverfeinerung unter Nutzung von Map2Scale.*

4.3.4.2 Berechnung der Update-Funktion einzelner Blöcke

Die Berechnung der Intervall-Update Funktion $[h]$ einer Zustandsupdate-Variablen läuft in den folgenden Schritten ab.

- Zuerst wird anhand der funktionalen Abhängigkeiten in $z(k+1) = h\,(x(k),\, z(k),\, p(k))$ geprüft, von welchen Größen $y(k+1)$ wirklich abhängt und von welchen Bereichsvariablen $bvar_i$ die Box-förmigen Bereiche für x, z und p aufgespannt werden.. Die Vereinigung aller dieser unterschiedlichen Bereichsvariablen $(bvar_1,\, bvar_2,\, ...,\, bvar_n)$ spannt dann die Boxen $Box_{z(k+1),Calc,i}$ $(i \in I)$ der Bereichsintervalle zur Berechnung von $z(k+1)$ auf.

- Für jede mögliche dieser Boxen $Box_{z(k+1),Calc,i}$ $(i \in I)$, also jedes mögliche Tupel $\left(bvar_1^{j_1},\, bvar_2^{j_2},\, ...,\, bvar_n^{j_n}\right)$ wird der zugehörige Intervallwert der Eingänge $[x_i]$ der Zustände $[z_i]$ und Parameter $[p_i]$ ermittelt. Aufgrund der Konstruktion der Boxen von $z(k+1)$ ist die hierzu zu selektierende Box für alle Eingangsvariablen, Zustandsvariablen und Parametervariablen eindeutig bestimmt.

- Auf Basis von $[h]$ und den ermittelten Intervallwerten $[x_i]$, $[z_i]$ und $[p_i]$ wird der zugehörige Wert $[z(k+1)_i]$ ermittelt, der zusammen mit $Box_{z(k+1),Calc,i}$ nun ein berechnetes Bereichsintervall bildet. Diese Schritte werden für jede vorgegebene Box $Box_{z(k+1),Calc,i}$ $(i \in I)$ wiederholt.

- Da jede Zustands-Update-Variable $z(k+1)$ nach Vorgabe in TalInt auch gleichzeitig eine Bereichsvariable ist, wird die *Map2Scale*-Abbildung auf die Bereichsintervalle

$\left(Box_{z(k+1),Calc,i}, [z(k+1)_i]\right)$ $(i \in I)$ angewandt und die Abbildung liefert daraus die "verfeinerten" Bereichsintervalle $\left(Box_{z(k+1),i}, [z(k+1)_i]\right)$ $(i \in I)$ für die Zustands-Update Variable. Ob eine etwaige Skalenausdehnung im Zuge der Berechnung von $Map2Scale$ ohne oder mit Widening ausgeführt wird, siehe 4.14 und Abbildung 4.15, hängt von der zugehörigen Zustandsvariable und der Zahl der bisherigen Iterationen k ab. Widening wird nur bei Zustandsvariablen in Rückkopplungsschleifen angewandt und auch erst dann, wenn $k > NIter_{Wid}$gilt, wobei letzteres ein vom Anwender vorzugebender Parameter ist.

Als Resultat der Berechnung beinhaltet die Zustands-Update-Variable nun eine aktualisierte Menge von Bereichsintervallen $\left(Box_{z(k+1),i}, [z(k+1)]\right)$ $(i \in I)$ und die Berechnung ist abgeschlossen.

4.3.4.3 Berechnung und Aktualisierung des Zustandsraumes als Abschluss des Iterationsschrittes

Die Berechnung des aktualisierten Zustandsraumes wird für jeden Subraum $Z_{sys,i}$ separat durchgeführt, was zulässig ist, da die Zerlegung in ein kartesisches Produkt vorher explizit vorgenommen wurde. Die einzelnen Schritte hierzu sind die Folgenden:

- Zunächst wird für jede Zustands-Update-Variable $z_i(k+1)$ innerhalb von $Z_{sys,i}$ bestimmt, welche Bereichsvariablen $bvar_i$ die Boxen ihrer Bereichsintervalle aufspannen. Die Vereinigungsmenge all dieser Bereichsvariablen ist in Abbildung 4.18 als $BV_{Z_{sys,i,calc}}$bezeichnet,
- Für jede mögliche Box Box_k, die sich anhand der Bereichsvariablen in $BV_{Z_{sys,i,calc}}$ aufspannen lässt, wird nun geprüft, ob es für jede Zustands-Update Variable im Subraum $Z_{sys,i}$ eine in Abschnitt 4.3.4.2 berechnete Box gibt, die Box_k vollständig enthält. Ist dies der Fall, so gehört die sich aus den Indizes der Zustandsvariablen der Box Box_k ergebende übergeordnete Box zum Zustandsraum. Andernfalls gehört die Box nicht dazu.

Wir zuvor schon erwähnt, bildet den Abschluss einer jeden Iteration k, d.h. nachdem alle $y_{sys}(k)$ und $z_{sys}(k+1)$ Knoten berechnet wurden, die Aktualisierung der Knoten $z_{sys}(k)$ als Vorbereitung des nächsten Iterationsschrittes, also die Berechnung des Zustandsraumes im nächsten Zeitschritt.

4.3.5 Das Terminierungsverhalten von TalInt

Die TalInt-Algorithmik terminiert grundsätzlich nach endlich vielen Iterationsschritten, wobei eine Iteration die Berechnung der System-Output-Funktion g_{sys}, der System-Update-Funktion h_{sys} und die Berechnung des Zustandsraumes $\left[\breve{Z}_{sys}\right]$ umfasst, siehe Abbildung 4.3. Es gibt dabei jedoch zwei völlig unterschiedliche Arten der Terminierung:

1. Der erfolgreiche Fall der Terminierung von TalInt liegt vor, wenn die Aktualisierung des Zustandsraumes entsprechend Abschnitt 4.3.4.3 in einem Iterationsschritt $k = k_{final}$ detektiert, dass $\left[\breve{Z}_{sys}\right](k_{final}) = \left[\breve{Z}_{sys}\right](k_{final}-1)$ gilt und der in Gl.4.4

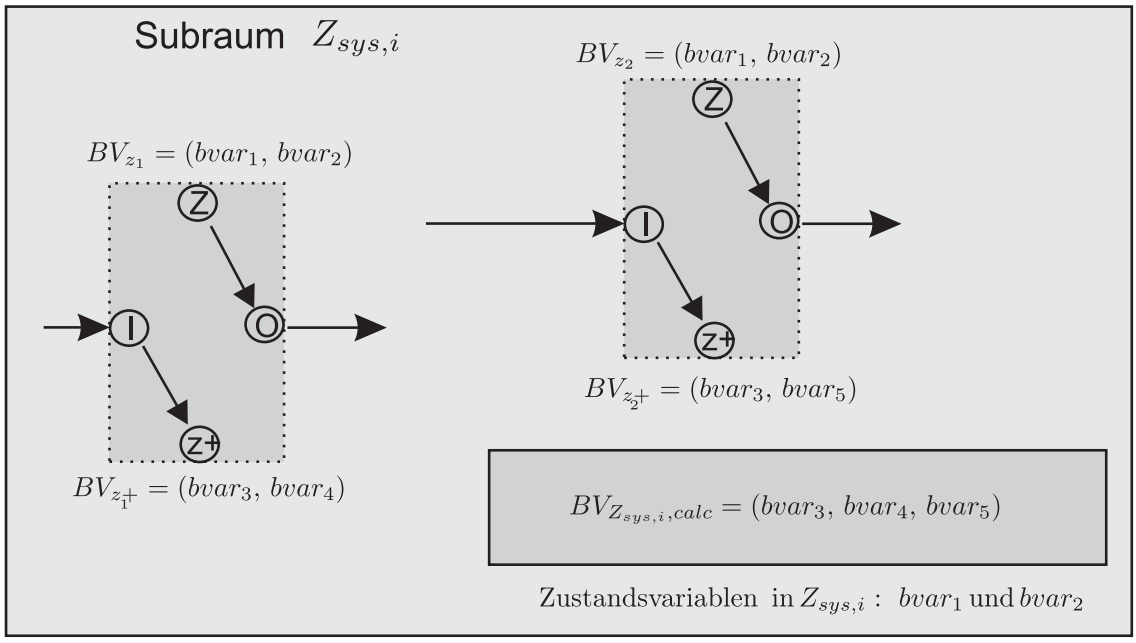


Abbildung 4.18: Zustandsraum bestehend aus zwei Zustandsvariablen. Anhand des in Abschnitt 4.3.4.3 beschriebenen Vorgehens wird die Aktualisierung des Zustandsraumes für den nächsten Zeitschritt berechnet.

geforderte Fixpunkt $\left[\breve{Z}_{sys}\right](k_{final})$ gefunden wurde. Nach Satz 43 gelten die zum Zeitpunkt $k = k_{final}$ ermittelten Wertebereichsgrenzen aller Systemgrößen somit für alle $k \in \mathbb{N}_0$ und die Aufgabenstellung 14 ist erfolgreich gelöst.

2. TalInt terminiert ohne allgemeingültige Wertebereichsgrenzen, sofern die Zahl der von TalInt durchgeführten Iterationen k die Maximalzahl $NIter_{max}$ übersteigt. $NIter_{max}$ (Number of MAXimal ITERations) ist dabei ein ganzzahliger TalInt-Parameter, welcher vom Benutzer unter der Nebenbedingung $NIter_{max} \geq 1$ vorgegeben werden muss. Wenn das TargetLink System in Folge eines "fehlerhaften" Designs prinzipiell keine Obergrenze kennt (siehe etwa das Beispiel in Abbildung 4.2 unten) und daher auch kein Fixpunkt gefunden werden kann, dann bricht die Algorithmik nach $NIter_{max}+1$ Iterationen ab. Ein weiterer Grund für den Abbruch kann darin bestehen, dass das TargetLink System zwar prinzipiell einen Fixpunkt aufweisen würde, dieser aufgrund von Überschätzen durch Dependency und/oder Wrapping-Effekte oder ungünstig vorgegebene TalInt Parameter nicht detektiert wurde. In solchen Fällen kann durch die Änderung der TalInt-Parameter bzw. durch eine feinere Unterteilung der Bereichsvariablen die Rechengenauigkeit erhöht werden, um die erforderliche Konvergenz zu erzielen.

4.4 Prototypische Umsetzung der Algorithmik

Nachdem in den vorherigen Abschnitten die eigentliche Algorithmik von TalInt besprochen wurde, soll in diesem Abschnitt die Implementierung der Algorithmen etwas detaillierter vorgestellt werden. Grundsätzlich teilt sich TalInt in die folgenden beiden Teile auf:

- Ein in MATLAB/Simulink/TargetLink eingebettetes TalInt Front-End bestehend aus MATLAB M-Skripten, welche zur Extraktion der erforderlichen Informationen aus dem eigentlichen TargetLink System dienen. Diese Informationen werden als XML System-File aus der MATLAB Umgebung exportiert.

- Ein TalInt Hauptprogramm in Form einer in C++ geschriebenen Konsolenapplikation, die basierend auf dem XML System-File und einem XML Parameter-File, welches die TalInt Algorithmen-Parameter beinhaltet, alle erforderlichen Analysen und Berechnungen entsprechend den vorhergehenden Abschnitten dieses Kapitels durchführt. In einem gewissen Sinne kann das TalInt Hauptprogramm als eine Simulationsumgebung für Intervallanalysis für eine Teilmenge der in TargetLink unterstützten Sprachkonstrukte angesehen werden. Das Resultat der Analysen und Berechnungen von TalInt, die robust abgeschätzten Wertebereichsgrenzen aller Systemgrößen, werden derzeit als reines Text-File ausgegeben. Für zukünftige Weiterentwicklungen wäre es wünschenswert, die gefundenen Resultate direkt nach TargetLink übernehmen zu können.

Die Situation ist in Abbildung 4.19 veranschaulicht[1]. Die Tatsache, dass die eigentlichen Berechnungen von TalInt nicht in MATLAB bzw. der MATLAB Skriptsprache M selbst vorgenommen werden, liegt darin begründet, dass die Rechenperformanz in MATLAB nicht mit der einer kompilierten C++ Anwendung zu vergleichen ist und der Performanz-Aspekt für die Praxistauglichkeit von TalInt von essentieller Bedeutung ist. In den nachfolgenden beiden Unterabschnitten soll sowohl auf das TalInt MATLAB Front-End als auch auf die C++ TalInt Konsolenapplikation etwas genauer eingegangen werden. Einige Hinweise zur Anwendung, insbesondere zur Bedeutung der Parameter von TalInt, finden sich im Anhang B.

4.4.1 Das TalInt MATLAB Front-End

Das MATLAB Front-End von TalInt dient dazu, alle für die Abschätzung der Wertebereichsgrenzen erforderlichen Informationen aus dem zu analysierenden Subsystem zu extrahieren. Dies ist programmiertechnisch leicht realisierbar, da sowohl MATLAB/Simulink als auch TargetLink eine leistungsfähige Programmierschnittstelle in der MATLAB Skriptsprache M anbieten, siehe [SL09][TL08], mit der sämtliche Informationen eines Simulink/TargetLink Modells ausgelesen werden können. Die mit dieser Methode ausgelesenen Informationen betreffen insbesondere

- Die hierarchische Strukturierung des zu analysierenden Systems im Hinblick auf weitere, beliebig verschachtelte Subsysteme.

- Sämtliche, innerhalb des betrachteten Subsystems enthaltenen TargetLink Blöcke und deren Blockkonfigurationen, welche die Art der vom Block vorgenommenen mathematischen Operation determinieren.

- Die Zahl der Eingangs- und Ausgangsvariablen, Zustandsvariablen und Parameter eines jeden Blockes.

[1] Eine Voraussetzung für die Anwendung des TalInt Prototypen ist offensichtlich eine existierende MATLAB/Simulink und TargetLink Installation. Anmerkungen zur Versionsabhängigkeit sind im Anhang B.1 angegeben.

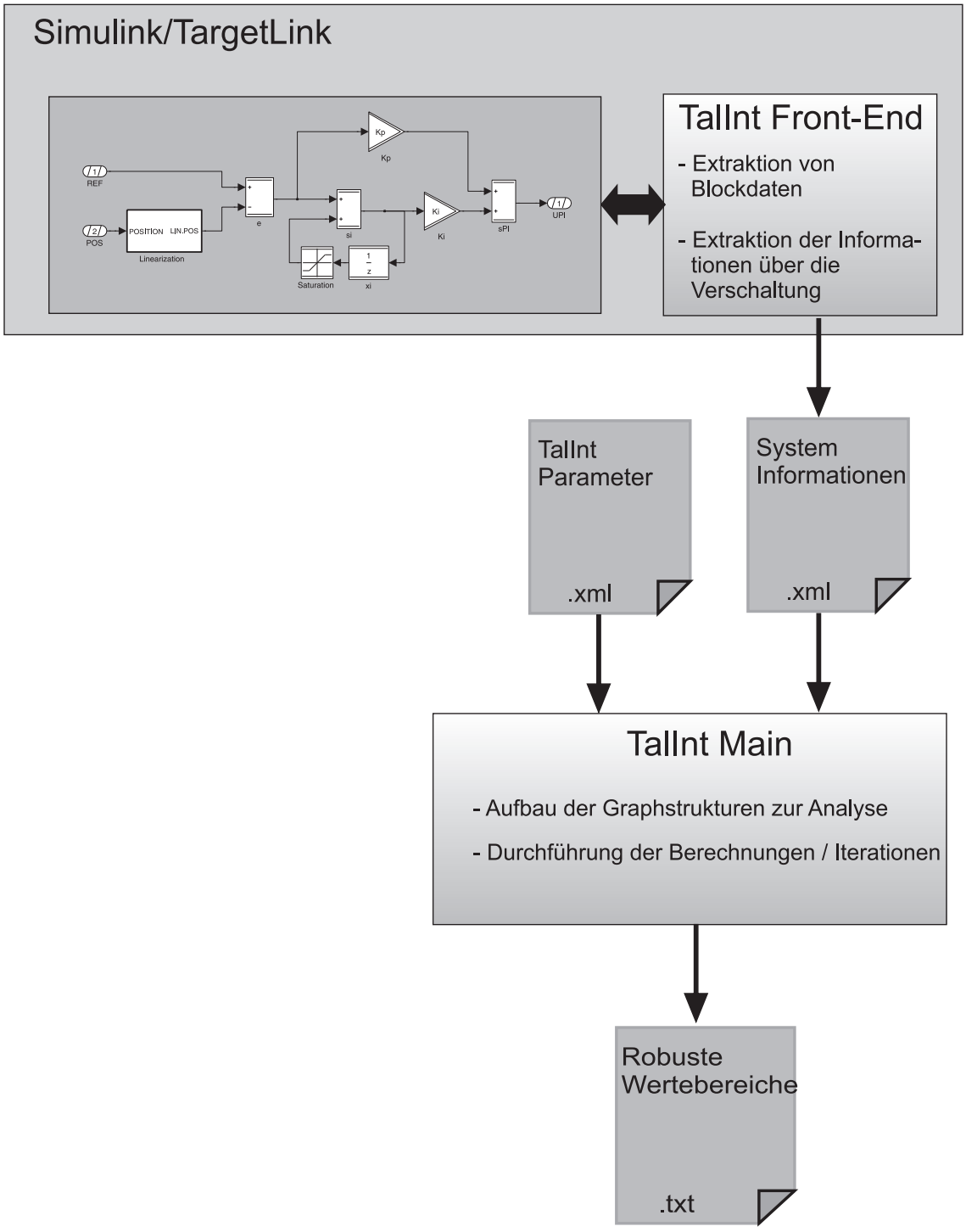


Abbildung 4.19: Grudlegende Struktur des TalInt Prototypen: Direkt innerhalb von MATLAB extrahieren die MATLAB M Skripte des Front-Ends die erforderlichen Informationen des Systems und exportieren diese als XML System-File. Die C++ Konsolenapplikation implememtiert dann die eigentlichen Analysen und Berechnungen von TalInt und exportiert die Wertebereichsgrenzen aller Systemgrößen in Form eines textuellen Output-Files.

- Die Verbindungslinien zwischen den einzelnen Blöcken im betrachteten Subsystem, die den Signalfluss zwischen den Blöcken und damit deren Verschaltung beschreiben. Simulink identifiziert hierbei die Startpunkte und Ziele der Signallinien mit Hilfe der global eindeutigen Namen der TargetLink Blöcke und der jeweiligen Port-Indizes, die festlegen, an welchem Blockeingang bzw. Ausgang die Verbindungslinien eigentlich festgemacht sind.

- Signalspezifikationen für einzelne Block-Variablen, die beispielsweise die vom Anwender vorgegebenen Minimal- und Maximalwerte für Parameter oder Eingangssignale beschreiben oder die Initialwerte für Zustandsvariablen.

Zur Ermittlung der Ausführungsreihenfolge der einzelnen Blöcke und Subsysteme wird im TalInt-Prototypen im gegenwärtigen Stadium auf die Funktionalitäten des sogenannten Simulink-Debuggers zurückgegriffen, der die Block- und Systemreihenfolgen in ein Text-File ausgibt, das anschließend vom TalInt MATLAB Front-End verarbeitet wird. Dabei werden sogenannte virtuelle Simulink-Subsysteme, welche keinerlei semantische Bedeutung haben, automatisch eliminiert bzw. "flachgeklopft". Für eine Weiterentwicklung von TalInt bietet sich naturgemäß an, diesen Schritt einzusparen und die Abarbeitungsreihenfolge auf Basis der System-Beschreibung selbst zu ermitteln, was entsprechend der Simulink Semantik im Wesentlichen auf eine topologische Sortierung anhand des Datenflusses hinausläuft.

Die obigen Informationen werden extrahiert und in Form eines XML System-Files des betrachteten Subsystems serialisiert. Abbildung 4.20 zeigt einen exemplarischen Ausschnitt aus einem solchen File. Die Serialisierung selbst erfolgt auf Basis der MATLAB M-Schnittstelle zum standardisierten DOM-Interface. Weitere Informationen für den programmiertechnischen Umgang mit XML Dateien findet sich z.B. in [MSXML60]. Konkret werden die relevanten Daten des Systems zunächst mit Hilfe der M-Anwendungsschnittstelle von Simulink und TargetLink aufgesammelt und in einer M-Struktur abgelegt, die anschließend mit Hilfe eines generischen Mechanismusses automatisch in eine DOM-Struktur umgesetzt und als solche dann in ein XML File serialisiert wird.

Das vom TalInt MATLAB Front-End erzeugte XML System-File enthält damit alle für Aufgabenstellung 14 erforderlichen Informationen, seien es die Wertebereichsgrenzen für Parameter und Eingangsvariablen, Initialwerte der Zustandsvariablen, sowie die Output- und Update-Funktionen aller Blöcke bzw. des Gesamtsystems entsprechend Gl. 2.9 und Gl. 2.10.

4.4.2 Die TalInt Main Applikation zur Berechnung von Wertebereichsgrenzen

TalInt Main stellt eine in der Sprache C++ entwickelte Konsolenapplikation dar, welche die eigentlichen Berechnungsschritte zur Lösung von Aufgabenstellung 14 vornimmt. Als Eingabedaten benötigt TalInt Main wie in Abbildung 4.19 dargestellt

- Die vom TalInt MATLAB Front-End exportierte System-Beschreibung des zu analysierenden Systems

- ein weiteres XML Parameter-File, welches die vom Anwender vorgegebenen TalInt Parameter entsprechend Abschnitt B.2.1 enthält, die entscheidenden Einfluss auf

Abbildung 4.20: Fragment eines vom TalInt MATLAB Front-End exportierten XML Files, welches das von TalInt zu analysierende TargetLink Sytem beschreibt.

Genauigkeit als auch Rechenzeit der Berechnungen von TalInt haben, siehe Abbildung B.2.

Nach Abschluss der Berechnungen exportiert TalInt dann ein einfaches Text-File, welches die ermittelten Wertebereichsgrenzen aller System-Größen enthält und somit bei Terminierung aufgrund eines erreichten Fixpunktes die Lösung von Aufgabenstellung 14 liefert. Terminiert TalInt hingegen aufgrund der Überschreitung der maximalen Anzahl an Iterationen $NIter_{max}$, siehe Abschnitt B.2.1 bzw. Abschnitt 4.3.5, so beinhaltet das exportierte Text-File lediglich Wertebereichsgrenzen, die für die erstem $NIter_{max}$ Ausführungen des Gesamtsystems Gültigkeit besitzen, jedoch nicht darüber hinaus. Abbildung 4.21 zeigt zur Anschauung ein Fragment eines von TalInt erzeugten Output-Text-Files mit den ermittelten Wertebereichsgrenzen des betrachteten Systems.

4.4.3 Allgemeine Datenstrukturen und Algorithmen

Für nahezu alle nicht-Intervall-bezogenen Datenstrukturen wie Arrays dynamischer Breite, assoziative Arrays (Dictionaries), Graphen mit Ecken und Kanten etc. wird die LEDA Bibliothek [Leda51] in der Version 5.1 verwendet. LEDA zeichnet sich insbesondere dadurch aus, dass viele Datenstrukturen mit Hilfe des C++ Template-Mechanismusses sehr flexibel konfiguriert und eingesetzt werden können.

Eine wesentliche Datenstruktur in TalInt ist naturgemäß der Graph G_{TL}, der für das betrachtete TargetLink System entsprechend den Ausführungen in Abschnitt 4.3.2 erstellt

```
Block: IIR_Filter_2ndOrder/picontroller/Subsystem/picontroller/AI_Sub/Input
Hull-Output 0:[ -1.000000,  1.000000]
End of Block Data

Block: IIR_Filter_2ndOrder/picontroller/Subsystem/picontroller/AI_Sub/Unit Delay1
Hull-Output 0:[ -3.000001,  5.000000]
Hull-State 0:[ -3.000001,  5.000000]
Hull-StateNextStep 0:[ -3.000001,  3.400000]
End of Block Data

Block: IIR_Filter_2ndOrder/picontroller/Subsystem/picontroller/AI_Sub/Gain1
Hull-Output 0:[ -1.500001,  2.500000]
End of Block Data

Block: IIR_Filter_2ndOrder/picontroller/Subsystem/picontroller/AI_Sub/Unit Delay2
Hull-Output 0:[ -5.000000,  5.000000]
Hull-State 0:[ -5.000000,  5.000000]
Hull-StateNextStep 0:[ -5.000000,  5.000000]
End of Block Data

Block: IIR_Filter_2ndOrder/picontroller/Subsystem/picontroller/AI_Sub/Gain2
Hull-Output 0:[ -2.500000,  2.500000]
End of Block Data

Block: IIR_Filter_2ndOrder/picontroller/Subsystem/picontroller/AI_Sub/Gain3
Hull-Output 0:[ -1.000001,  1.000001]
End of Block Data

Block: IIR_Filter_2ndOrder/picontroller/Subsystem/picontroller/AI_Sub/Gain5
Hull-Output 0:[ -1.000001,  1.000001]
End of Block Data

Block: IIR_Filter_2ndOrder/picontroller/Subsystem/picontroller/AI_Sub/e3
Hull-Output 0:[ -2.000001,  2.000001]
End of Block Data

Block: IIR_Filter_2ndOrder/picontroller/Subsystem/picontroller/AI_Sub/e
Hull-Output 0:[ -3.000001,  3.000001]
```

Abbildung 4.21: Textuelle Ausgabe des TalInt Prototypen mit Wertebereichsinformationen für alle Blockausgangs- und Zustandsvariablen innerhalb des betrachteten TargetLink Systems. Für zukünftige Konsolidierungen und Erweiterungen von TalInt bietet sich eine automatische Übernahme der Daten nach TargetLink an, um die Wertebereichsinformationen unmittelbar zu nutzen.

und zur Analyse des Zustandsraumes, Rückkopplungsschleifen und der Festlegung von Bereichsvariablen genutzt wird. Hierbei kommen die LEDA Klassen für Graphen, Ecken und Kanten sowie einige Graph-Algorithmen wie eine Breitensuche zum Einsatz. Gleiches gilt für die Graph-Modifikationen und Analysen wie sie in Abschnitt 4.3.3.2 und Abschnitt 4.3.4.3 beschrieben sind. Jede Ecke des Graphen ist darüber hinaus grundsätzlich mit einer weiteren Datenstruktur assoziiert, welche die gesamten Berechnungsdaten der Output- und Update-Funktionen $[g_i]$ und $[h_i]$ während der einzelnen Iterationen aufnimmt und am Ende auch die Wertebereichsgrenzen der einzelnen System-Variablen beinhaltet. Der Graph G_{TL} wird aufgebaut, sobald das aus MATLAB exportiere XML System-File eingelesen und die TargetLink Blöcke und Systeme als Datenstrukturen aufgebaut worden sind. Wie zuvor dargestellt, ist jede Variable des TargetLink Systems, also jede Ausgangsvariable, Eingangsvariable, Zustandsvariable etc. exakt mit einer Ecke des Graphen assoziiert. Die in Simulink-Modellen oft vorhandenen Hierarchien in Form von verschachtelten Subsystemen sind in G_{TL} nicht mehr sichtbar, da alle Hierarchien quasi "flachgeklopft" werden.

Wie in den Abschnitten 4.3.3 und 4.3.4 deutlich wurde, stellen die sogenannten Bereichsintervalle, also die Kombination aus einer Box (als speziellem, multidimensionalem Gebiet) und dem Intervallwert, der auf dieser Box angenommen wird, eine ganz besonders wichtige Datenstruktur für TalInt dar. Boxen, siehe Gl. 4.12, werden, dabei als eindimensionale Arrays von reinen Integer-Größen beschrieben, wobei Letztere Indizes der Intervallverfeinerungen der einzelnen Bereichsvariablen darstellen. Eine Menge von Bereichsintervallen, wie sie bei der Berechnung von Output- und Update-Funktionen $[g_i]$ und $[h_i]$ erzeugt werden, ist in TalInt dann als LEDA Dictionary-Struktur implementiert, wobei die Box als

Array von Indizes den Schlüssel zum Dictionary-Eintrag darstellt und der Intervallwert innerhalb der Box den zugehörigen Wert des Dictionary-Eintrags. Die einzelnen Einträge in Dictionaries sind in der LEDA Bibliothek zwecks schnelleren Zugriffs sortiert. Wie in Abschnitt 4.3.4 dargestellt wurde, besteht ein Großteil der Rechnung mit Bereichsintervallen darin, über alle denkbaren Boxen zu iterieren und von den unabhängigen Veränderlichen die zugehörigen, passenden Boxen mit ihren Intervallwerten zu ermitteln. In TalInt wird für jede Berechnung von $[g_i]$ und $[h_i]$ jeweils ein Iterator-Objekt erzeugt, welches die in Frage kommenden Boxen der Reihe nach durchläuft und somit die Rechnung mit Bereichsintervallen steuert.

Der Zustandsraum des TargetLink Systems bzw. dessen mögliche Subräume sind in TalInt ebenfalls als Dictionaries implementiert. Da der Zustandsraum bzw. jeder Unterraum in TalInt grundsätzlich diskretisiert ist, entsprechen die einzelnen, disjunkten Boxen des Zustandsraumes wieder eineindeutig einem Array von Indizes, welches als Schlüssel für den Dictionary-Eintrag verwendet wird. Der Wert jedes Eintrags im Dictionary ist dabei rein bool'scher Natur und gibt an, ob die zugehörige Box im diskretisierten Zustandsraum bereits angenommen wurde oder nicht. Derselbe Mechanismus wird zur Detektion genutzt, ob der Zustandsraum im Zuge einer Iteration konstant geblieben und der gesuchte Fixpunkt somit gefunden wurde.

Die unterschiedlichen Arten von TargetLink Blöcken, wie z.B. Summationsblöcke, Unit Delays etc. werden als abgeleitete Klassen einer Basisklasse implementiert, von der einige substanzielle Kernfunktionalitäten geerbt werden. Als Konsequenz daraus kostet es relativ wenig Aufwand, in TalInt einen weiteren, bisher noch nicht unterstützten Block hinzuzufügen. Im wesentlichen müssen nur die Block-spezifische XML-Einleseroutine, das Einfügen der Block-internen Kanten und natürlich die Intervallfunktionen $[g_i]$ und $[h_i]$ für den neuen Block implementiert werden.

4.4.4 Datenstrukturen und Operationen für Intervalle

Einen wichtigen Teil der Berechnungen in TalInt Main stellen naturgemäß Operationen auf Intervallen dar. Diese werden in erster Linie zur Implementierung der Intervallerweiterungen der Block-Output- und Block-Update-Funktionen $[g_i]$ und $[h_i]$ verwendet, wie sie in Abschnitt 4.3.4.1 und Abschnitt 4.3.4.2 beschrieben sind. Zur Implementierung der Intervallfunktionen in TalInt wird in hohem Maße auf die C++ Bibliothek C-XSC ("C++ Library for E**x**tended **S**cientific **C**omputing) [CXSC20] in der Version 2.0 zurückgegriffen. Die Bibliothek bietet beispielsweise

- C++ Klassen für Intervalle, Vektoren von Intervallen sowie Matrizen von Intervallen
- Konstruktoren und Konvertierungsroutinen um die obigen Strukturen aus Fließkomma-Datentypen zu erzeugen und rückzuwandeln
- Die elementaren arithmetischen Operationen Summation, Subtraktion, Multiplikation und Division für die obigen Datenstrukturen
- Trigonometrische und hyperbolische Funktionen wie sin, cos, sinh, cosh etc. auf den obigen Datenstrukturen
- Weitere unäre Operationen wie exp, abs, pow etc.

- Elementare mengentheoretische Prüfungen auf Gleichheit, Enthaltensein und ähnliche für die obigen Intervall-Datenstrukturen

Intervalle bzw. Matrizen von Intervallen werden in TalInt grundsätzlich auf Basis der C-XSC Klassen implementiert. Gleiches gilt für die Intervallerweiterungen der Output- und Update-Funktionen $[g_i]$ und $[h_i]$ einzelner Blöcke, sofern die benötigte Funktionalität bereits durch C-XSC bereitgestellt wird. Dies ist trivialerweise für Summations- und Produktblöcke, Trigonometrische Blöcke etc. der Fall. Dort wo C-XSC keine Implementierung bereitstellt, wie beispielsweise für einen TargetLink Switch Block, siehe Abbildung 2.8, wird eine eigene Implementierung von $[g_i]$ und $[h_i]$ durch Kombination von verschiedenen C-XSC Teilfunktionen und eigenem Programmcode realisiert.

Kapitel 5

Resultate bei der Anwendung der Lösungsalgorithmik

In diesem Kapitel werden die Resultate bei der Anwendung der TalInt Algorithmik auf Simulink/TargetLink Modelle dargestellt. Exemplarisch werden dazu in Abschnitt 5.1 Modellfragmente betrachtet, wie sie in umfangreichen automotiven Modellen typischerweise auftreten. Die Auswahl der zu analysierenden Modellfragmente erfolgte dabei auch mit der Zielsetzung, die Resultate von TalInt auf eine alternative Art und Weise überprüfen zu können, zumindest für einfache Spezialfälle. Desweiteren werden auch die sich ergebenden Probleme anhand von Modellfragmenten exemplarisch aufgezeigt. Der nachfolgende Abschnitt 5.2 enthält dann eine Zusammenfassung der bisherigen Erfahrungen bei der Anwendung von TalInt auf reale, umfangreichere Modelle aus der Praxis wie sie für Regelungs- und Steuerungsfunktionen in der Automobilindustrie typischerweise eingesetzt werden. Es wird eine übergeordnete Bewertung der Leistungsfähigkeit von TalInt vorgenommen, inklusive einer Liste von Vorschlägen zur weiteren Verbesserung und Verfeinerung sowohl der eigentlichen Algorithmik als auch der Implementierung zur Steigerung der Performance.

5.1 Beispiele zur Anwendung der Algorithmik auf idealisierte und reale Modellfragmente

5.1.1 Anwendung auf ein Polynomiales System ohne Zustandsgrößen

Als Spezialfall eines TargetLink Systems ohne Zustände wird TalInt auf ein einfaches Polynom zweier Veränderlicher angewandt, welches auf den rechteckigen Definitionsbereich $[-1,\,1] \times [-1,\,1]$ eingeschränkt wird:

$$\begin{array}{lll} Poly: & [-1,\,1] \times [-1,\,1] & \rightarrow \quad \mathbb{R} \\ & (x_1,\,x_2) & \rightarrow \quad y = 4 \cdot x_1^2 - 2.1 \cdot x_1^4 + 1/3 \cdot x_1^6 + x_1 \cdot x_2 - 4 \cdot x_2^2 + 4 \cdot x_2^4 \end{array} \tag{5.1}$$

Dieses Polynom stellt ein gängiges Benchmarks einer Zielfunktion für kontinuierliche, globale Optimierungsprobleme dar, welches Sattelpunkte beinhaltet und dessen Graph in

Abbildung 5.1 dargestellt ist. Das globale Minimum von *Poly* auf dem Definitionsbereich kann beispielsweise mit Hilfe des Werkzeuges GloptiPoly [Gl08] berechnet werden, das unter anderem zur Lösung von polynomialen, globalen Optimierungsproblemen dient. GloptiPoly ermittelt das globale Minimum von *Poly* zu

$$\min Poly\,(x_1,\, x_2) = -1.0316$$

Im Folgenden soll nun TalInt zur robusten Abschätzung des Wertebereichs von *Poly* auf dem gegebenen Definitionsbereich angewandt werden. Zu diesem Zweck wird *Poly* entsprechend Gl. 5.1 auf Basis von Multiplikator-, Summations- und Verstärkerblöcken in ein einfaches TargetLink System umgesetzt, welches in Abbildung 5.2 dargestellt ist. Wie in Kapitel 3 bereits deutlich wurde, ist der Grade des Überschätzens bei der Anwendung der Intervall-Arithmetik aufgrund von Dependency- und Wrapping-Effekten im Allgemeinen von der konkreten Realisierung des Modells abhängig. Die in Abbildung 5.2 gewählte Modellierung ist eine mögliche Variante, wie man sie unter Anwendung von TargetLink Modellierungsrichtlinien, siehe z.B. [TL08], zur Realisierung des funktionalen Zusammenhangs in Gl. 5.1 typischerweise wählen würde. Die beiden Veränderlichen x_1 und x_2 werden in den beiden Inports mit den Namen $x1$ und $x2$ eingespeist. Der Funktionswert $Poly\,(x_1,\, x_2)$ stellt sich am Outport *Poly* ein, dessen Ausgangssignal mit dem des Vorgängerblocks zur Summationsbildung identisch ist. Die Koeffizienten von *Poly* werden in den Gain-Blöcken eingestellt, bei denen es sich in konkreten Fall um Intervalle der Breite $w = 0$ (Singletons) handelt.

Die Modellanalyse von TalInt entsprechend Abschnitt 4.3.3 liefert nun die folgenden Resultate:

- TalInt bestimmt zunächst, dass die Eingangsvariablen x_1 und x_2 direkt als Bereichsvariablen verwendet werden, für die Intervallverfeinerungen vorgenommen werden. Dies ist notwendig, um das Überschätzen durch Dependency- und Wrapping-Effekte zu reduzieren, wie man auch direkt anhand des mehrfachen Auftretens der Variablen x_1 und x_2 im formelmäßigen Ausdruck von $Poly\,(x_1,\, x_2)$ ablesen kann.

- Weitere Bereichsvariablen existieren aufgrund der Modellstruktur bzw. des Fehlens von Blöcken mit Zustandsvariablen nicht. Somit ist der einzig relevante TalInt-Parameter die Zahl der initialen Intervallunterteilungen $NInt_{init}$, die zur Verfeinerung des Intervalls $[-1,\, 1]$ für x_1 und x_2 eingesetzt werden.

Die von TalInt ermittelten robusten Wertebereichsgrenzen für $Poly\,(x_1,\, x_2)$ auf Basis des Modells aus Abbildung 5.2 sind in Tabelle 5.1 bzw. Abbildung 5.3 für unterschiedliche Werte von $NInt_{init}$ angegeben. Andere TalInt Parameter sind ohne Auswirkung auf die Ergebnisse. Aufgrund des Dependency- und Wrapping-Effektes ergibt sich ein Überschätzen der tatsächlichen Wertebereichsgrenzen, welches sich durch Erhöhung von $NInt_{init}$ aufgrund der Lipschitz-Stetigkeit von *Poly* auf Kosten des Rechenaufwandes beliebig reduzieren lässt, siehe Satz 33. Man beachte ferner, dass aufgrund der Inklusions-Monotonie der Intervallkonstruktion die abgeschätzten Wertebereichsgrenzen bei einer Erhöhung von $NInt_{init}$ nie ungenauer werden können, dass sich jedoch selbst für $NInt_{init} = 100$ noch eine erhebliche Abweichung des berechneten Wertes $-1,29 < \min Poly\,(x_1,\, x_2) = -1.0316$ vom tatsächlichen Wert ergibt. Dies ist einerseits durch die spezielle Form von *Poly* begründet, andererseits jedoch auch durch die lediglich lineare Konvergenzgeschwindigkeit

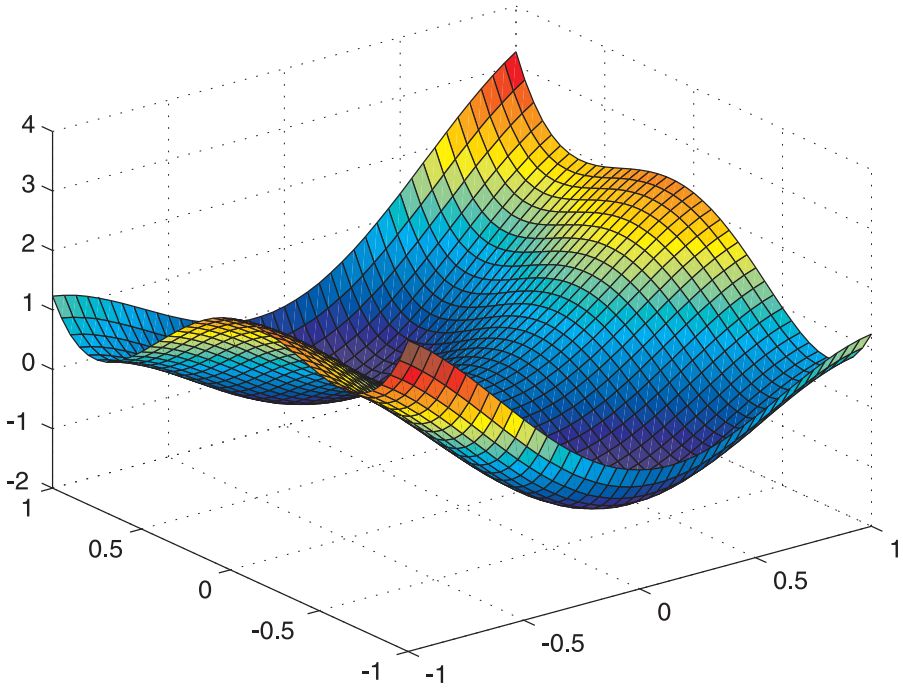


Abbildung 5.1: Graph des Polynoms *Poly* aus Gl. 5.1 über dem boxförmigen Definitionsbereich $(x_1, x_2) \in [-1, 1] \times [-1, 1]$. Das Minimum weist den Wert $\min Poly\,(x_1, x_2) = -1.0316$ auf.

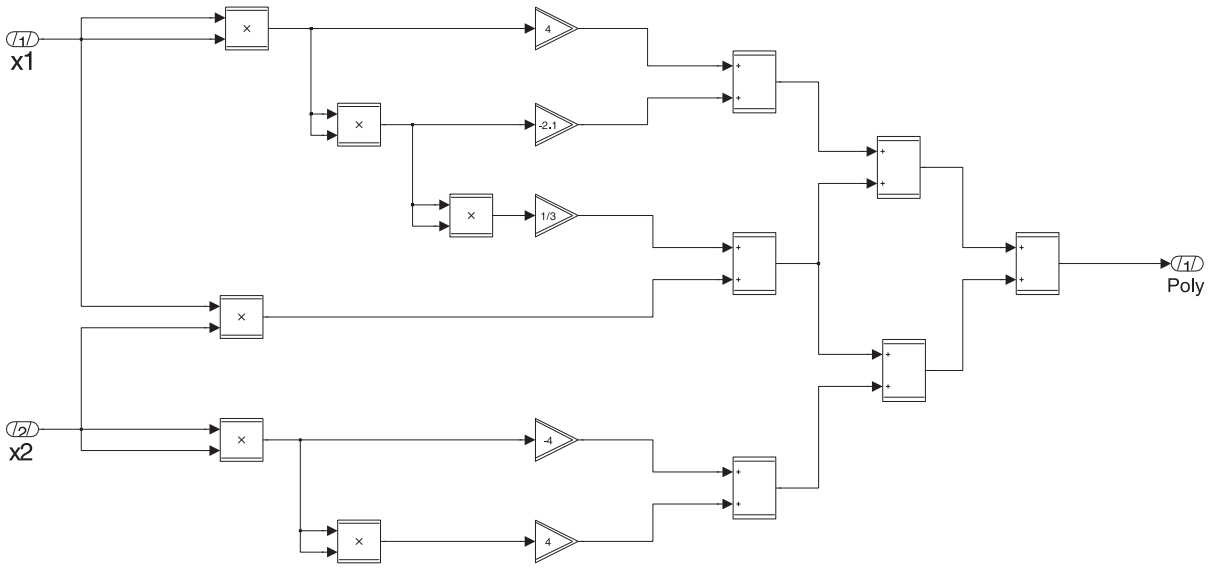


Abbildung 5.2: TargetLink System zur Implementierung des Polynoms nach Gl. 5.1 mit Hilfe von Multiplizierern, Addierern und Verstärkern. Typischerweise werden in der Praxis Modellierungsrichtlinien eingesetzt, die erfordern, dass Multiplizierer und Addierer nur mit jeweils zwei Eingängen beschaltet werden, um die Generierung von Festkomma-Code aus entsprechenden Modellen zu vereinfachen bzw. zu ermöglichen.

$NInt_{init}$	1	5	10	20	50	100
$[Poly]$	[-16.77, 16.77]	[-6.14, 8.96]	[-3.79, 7.25]	[-2.02, 6.05]	[-1.46, 5.20]	[-1.29, 4.89]

Tabelle 5.1: Mit Hilfe von TalInt ermittelte Wertebereichsgrenzen für $Poly\,(x_1,\, x_2)$ in Abhängigkeit des TalInt-Parameters $NInt_{init}$ für das Modell aus Abbildung 5.2. $NInt_{init}$ ist dabei die Zahl der Intervallverfeinerungen zur Reduktion des Überschätzens bei der Anwendung der Intervall-Analysis.

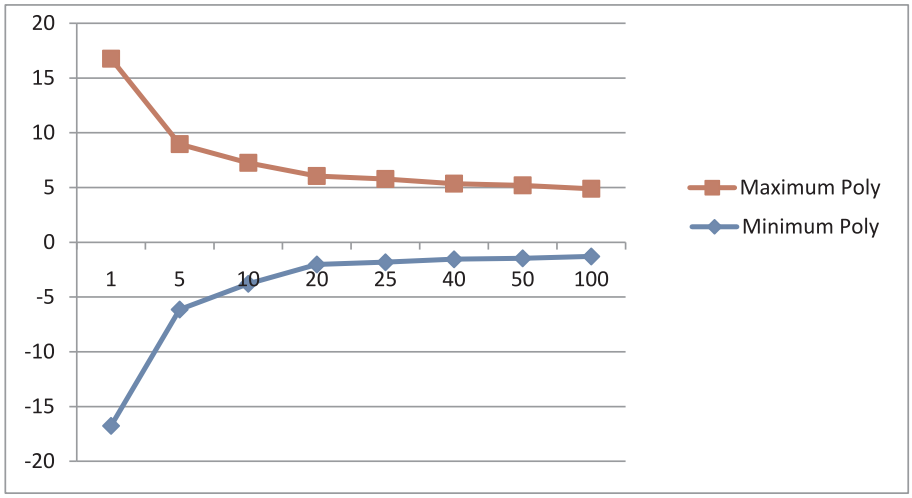


Abbildung 5.3: Visualisierung der robst abgeschätzten Wertebereichsgrenzen für $Poly$ aus Tabelle 5.1 in Abhängigkeit des TalInt Parameters $NInt_{init}$.

der Intervall-Analysis. Die Berechnungen von TalInt benötigen in der gegenwärtigen, prototypischen Implementierung auf einem handelsüblichen PC (Stand 2010) hierfür bereits einige Sekunden.

In Abbildung 5.4 ist eine alternative, leicht modifizierte Modell-Realisierung von $Poly$ mit Hilfe einer allgemeinen Potenzfunktion dargestellt, auf die TalInt analog zum vorherigen Beispiel angewandt wird. Die Resultate sind in Tabelle 5.2 bzw. Abbildung 5.5 dargestellt und zeigen ein analoges Verhalten. Auch hier ergibt sich selbst für $NInt_{init} = 100$ Unterteilungen noch eine erhebliche Abweichung des berechneten Wertes $-1,33 < \min Poly\,(x_1,\, x_2) = -1.0316$ vom tatsächlichen Wert.

Naturgemäß ist die robuste Abschätzung der Wertebereichsgrenzen für $Poly$ nicht der primäre Anwendungsfall, für den TalInt entwickelt wurde. Ein wesentlicher Unterschied besteht bereits darin, dass TalInt grundsätzlich robuste Wertebereichsgrenzen für alle im System auftretenden Variablen ermittelt, nicht nur für die Ausgangsvariable des Systems. Dies ist zur Skalierung eines TargetLink Systems auch zwingend erforderlich. Darüber hinaus ist TalInt auf automotive Modelle ausgerichtet, die im Allgemeinen keineswegs polynomial und oftmals nicht einmal stetig sind und für die dennoch Wertebereichsgrenzen ermittelt werden müssen. Darüber hinaus ist TalInt mächtig genug, auch dynamische, d.h. zustandsbehaftete Systeme zu analysieren, wie in den nachfolgenden Beispielen klar werden wird. Nichtsdestotrotz ist das polynomiale Benchmark zum Test der Funktionsfähigkeit von TalInt hilfreich und weist auch auf das Problem der niedrigen Konvergenzgeschwindigkeit der Intervall-Analysis hin.

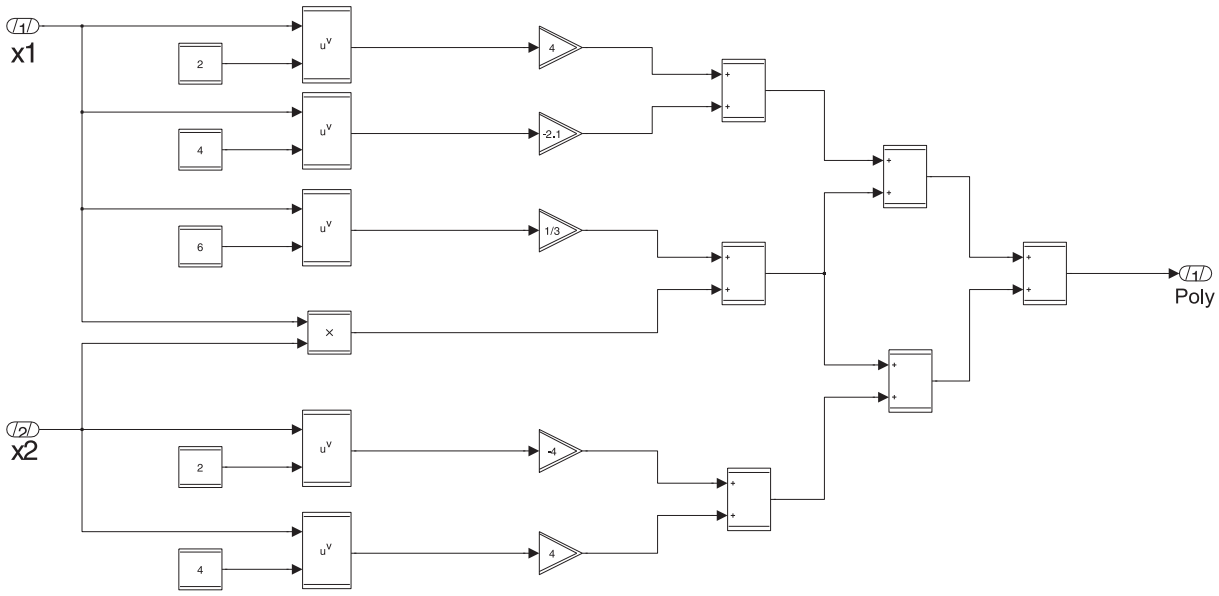


Abbildung 5.4: Alternative Realisierung eines TargetLink Modells zur Implementierung des Polynoms *Poly* mit Hilfe von Potenzfunktionen, Addierern und Verstärkern.

$NInt_{init}$	1	5	10	20	50	100
$[Poly]$	[-8.10, 10.67]	[-6.12, 8.96]	[-3.73, 7.25]	[-2.14, 6.05]	[-1.54, 5.20]	[-1.33, 4.89]

Tabelle 5.2: Mit Hilfe von TalInt ermittelte Wertebereichsgrenzen für $Poly\,(x_1,\, x_2)$ in Abhängigkeit des Parameters $NInt_{init}$ für das Modell aus Abbildung 5.4.

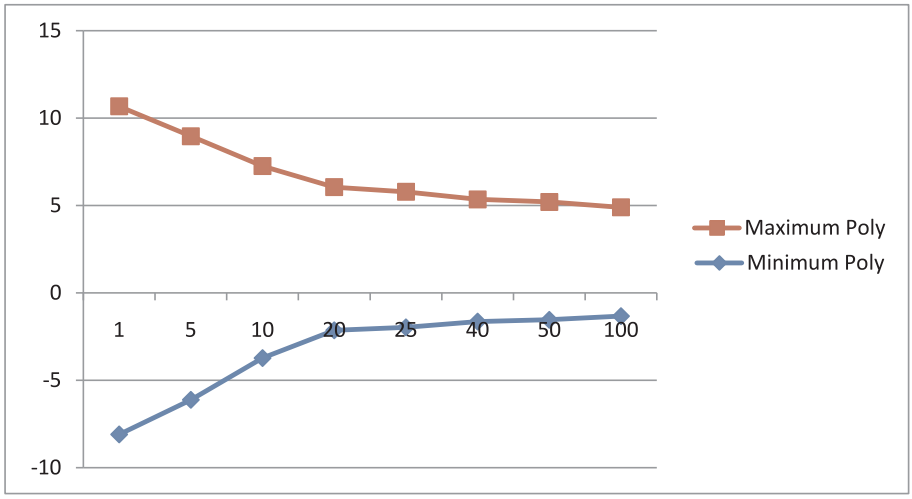


Abbildung 5.5: Visualisierung der robst abgeschätzten Wertebereichsgrenzen für *Poly* aus Tabelle 5.2 in Abhängigkeit der Zahl der Intervallunterteilungen $NInt_{init}$.

5.1.2 Anwendung auf einen allgemeinen FIR-Filter beliebiger Ordnung

Im Folgenden soll TalInt auf sogenannte Finite Impulse Response Filter (abgekürzt: FIR Filter) angewandt werden, wie man sie in vielen regelungstechnischen Anwendungen zur Signalverarbeitung findet. Ein solches lineares, zeitinvariantes FIR Filter ist durch die folgende zeitdiskrete Gleichung zwischen Eingangsvariable $x(k)$ und Ausgangsvariable $y(k)$ beschrieben:

$$y(k) = a_0 \cdot x(k) + a_1 \cdot x(k-1) + a_2 \cdot x(k-2).... + a_n \cdot x(k-n) \tag{5.2}$$

Dabei stellen a_0, a_1, $...a_n$ die vorgegebenen Koeffizienten des Filters dar, die an die gewünschte Filtercharakteristik (Tiefpassfilter, Hochpassfilter etc.) angepasst werden können und die Parameter im Sinne des TargetLink Systems sind. n ist die Ordnung des Filters. Für den Spezialfall $a_0 = a_1 = a_2... = a_n = {}^1/_{n+1}$ ergibt sich beispielsweise ein einfaches Tiefpassfilter, welches den zeitlichen Mittelwert der letzten $n+1$ Werte des Eingangssignals berechnet und diesen Wert am Ausgang zur Verfügung stellt. Abbildung 5.6 zeigt die Realisierung eines allgemeinen FIR-Filters für $n = 4$, wie man es unter Nutzung von Modellierungsrichtlinien typischerweise designen würde. Alle nachfolgend erwähnten Eigenschaften lassen sich nahtlos vom Spezialfall $n = 4$ auf den allgemeinen Fall n übertragen.

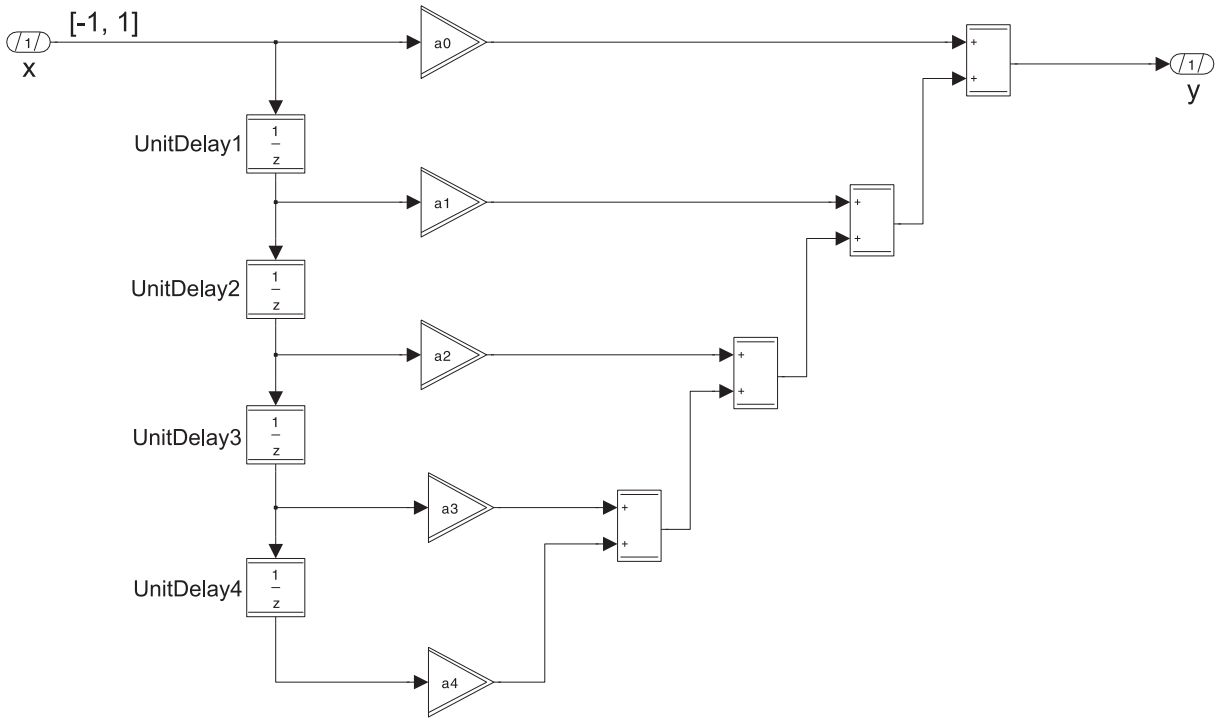


Abbildung 5.6: System zur Realisierung eines linearen, zeitinvarianten FIR Filters der Ordnung $n = 4$. Ein solches Filter ist typischerweise als Verschaltung von Addiereren, Verstärkern und Verzögerungsgliedern realisiert. Das Eingangssignal wird am Port x eingespeist und das Ausgangssignal am Port y abgegriffen. Für die Verzögerungselemente (Unit Delay Blöcke) können Anfangswerte vorgegeben werden, wobei diese in der Praxis oftmals jedoch zu 0 gesetzt werden.

Die Anwendung der TalInt Analysen entsprechend Abschnitt 4.3.3 auf die Modellstruktur aus Abbildung 5.6 ergibt nun insgesamt das folgende Bild:

- Das Modell beinhaltet vier aus den Unit-Delay Blöcken resultierende Zustandsvariablen.
- Keine Zustandsvariable liegt in einer Rückkopplungsschleife. Folglich wird grundsätzlich keine Widening-Operation in den einzelnen Iterationen durchgeführt.
- Durch den Signalverlauf von einem Unit Delay Block zum Nächsten gibt es eine gegenseitige Beeinflussung innerhalb der Zustandsvariablen. Folglich gibt es keine Subräume des Zustandsraumes, die zur Dimensionsreduktion genutzt werden könnten. In Abschnitt 5.2 werden einige generelle Vorschläge zur Dimensionsreduktion des Zustandsraumes gemacht, die ein möglicherweise eintretendes Überschätzen bewusst hinnehmen, welches sich dann auch nicht mehr beliebig reduzieren lässt.
- Außer den Zustandsvariablen bzw. den Zustandsupdatevariablen existieren keine weiteren Bereichsvariablen, da es zu keinerlei Überschätzen durch Dependency- oder Wrapping-Effekten kommen kann. Dies lässt sich bei unterschiedlichen Durchläufen von TalInt mit unterschiedlichen Parametern verifizieren.

TalInt berechnet nun für das Modell aus Abbildung 5.6 bei vorgegebenem Eingangsintervall $x \in [-1, 1]$ und zu Null gesetzten Initialwerten der Zustände die in Abbildung 5.7 und Abbildung 5.8 dargestellten Wertebereichsgrenzen für die Zustandsvariablen der Blöcke *UnitDelay3* und *UnitDelay4* über dem Index k der Iteration. Die Verläufe, die naturgemäß unabhängig von den Parametern a_0 bis a_n sind und auch keine Abhängigkeit von TalInt Parametern aufweisen, lassen sich anhand des Modells unmittelbar nachvollziehen.

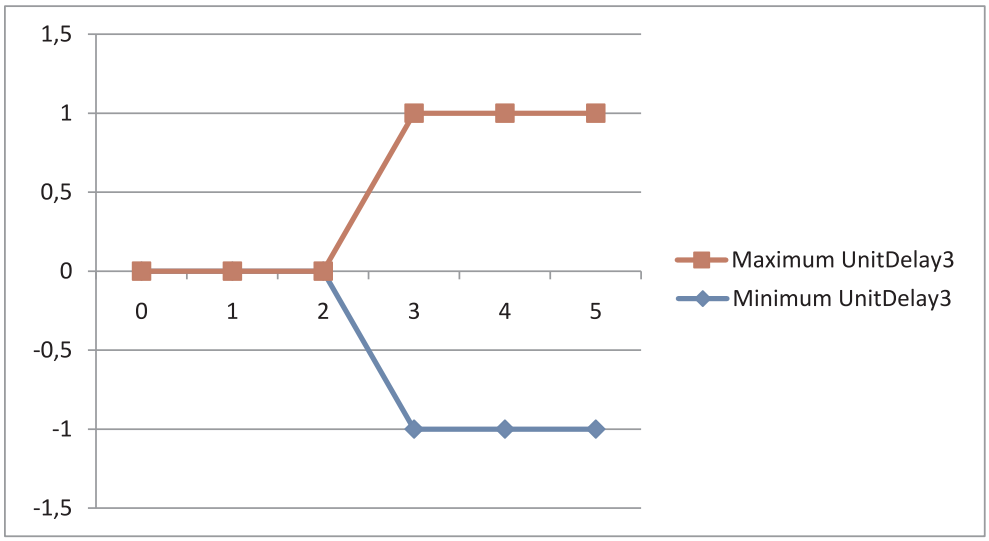


Abbildung 5.7: Wertebereichsgrenzen für die Zustandsvariable des *UnitDelay3* Blockes aus Abbildung 5.6 in Abhängigkeit der Iteration k. Man beachte, dass hier nur die Wertebereichsgrenzen aufgetragen sind, wohingegen der komplette Zustandsraum gegebenenfalls sehr fein diskretisiert sein kann (auch wenn das in diesem Beispiel zur Vermeidung von Überschätzen nicht notwendig ist).

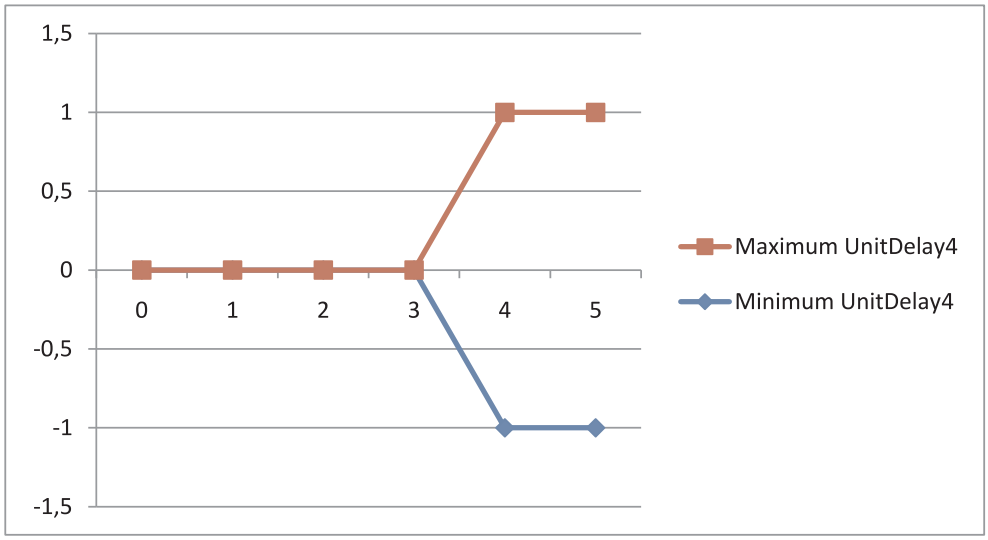


Abbildung 5.8: Wertebereichsgrenzen für die Zustandsvariable des *UnitDelay4* Blockes aus Abbildung 5.6 in Abhängigkeit der Iteration k. Ab $k = 4$ ist der Fixpunkt im Zustandsraum erreicht und die Gültigkeit der Wertebereichsgrenzen für beliebige k bewiesen.

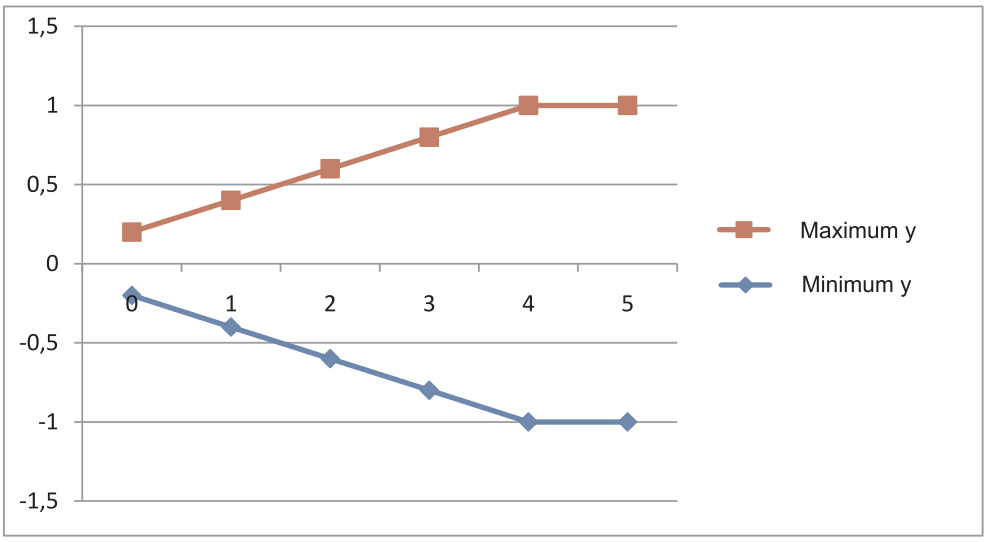


Abbildung 5.9: Wertebereichsgrenzen für die Ausgangsvariable y des FIR Filters aus Abbildung 5.6 in Abhängigkeit der Iteration k für den Spezialfall $a_0 = a_1 = ... = a_4 = 1/5$.

Nach der Iteration $k = 4$, d.h. sobald das Eingangssignal in der Kette der Verzögerungselemente bis zum Block *UnitDelay4* durchgedrungen ist, ist der Zustandsraum konstant, d.h. der gesuchte Fixpunkt der Iteration erreicht und TalInt terminiert. Analoges gilt naturgemäß für eine beliebige Zahl von verketteten Unit Delay Blöcken, d.h. für einen beliebigen Filtergrad n bei dem der Fixpunkt im Zustandsraum nach n Iterationen erreicht ist.

Für den oben erwähnten Spezialfall $a_0 = a_1 = ... = a_4 = 1/5$, d.h. die einfache zeitliche Mittellung der letzten fünf Werte des Eingangssignals x sind die zugehörigen Wertebe-

reichsgrenzen des Ausgangssignal y in Abhängigkeit der Iteration k in Abbildung 5.9 dargestellt. Für $k = 4$, d.h. zum Zeitpunkt des erreichten Fixpunktes im Zustandsraum gilt für den Wertebereich des Ausgangssignals das (abgesehen von den Rundungen) unmittelbar plausible Ergebnis $y \in [-1.000001, 1.000001]$, das folglich für alle Zeitpunkte k Gültigkeit besitzt. Selbstverständlich lässt sich die Berechnung mit TalInt für beliebige, nicht-triviale Intervalle als Wertebereichsgrenzen der Parameter a_0, a_1, ...a_n durchführen, was keinerlei Komplikationen verursacht.

5.1.3 Anwendung auf das Modell eines Zählers mit Rücksetzfunktionaliät

Als nächstes Testmodell soll ein typisches Modellfragment eines Zählers herangezogen werden, wie man es sehr häufig in automotiven Modellen findet. Eine mögliche Realisierungsvariante ist in Abb. 5.10 dargestellt. Der Grundgedanke zur Realisierung des Zählers besteht darin, die eigentliche Zählvariable in einem Unit Delay Block zu speichern und bei jeder Berechnung des Systems über eine Rückkopplungsschleife um einen Wert *Delta* zu erhöhen. Dieser neue Zählwert wird anschließend mit einem Maximalwert $MaxValue$ verglichen und entweder auf den Wert $ResetValue$ zurückgesetzt oder unverändert an den Eingang des Unit Delay Blockes weitergereicht. Der Initialwert $InitValue$ des Zählers ist hingegen durch den Initialwert der Zustandsvariablen z des Unit Delay Blockes gegeben.

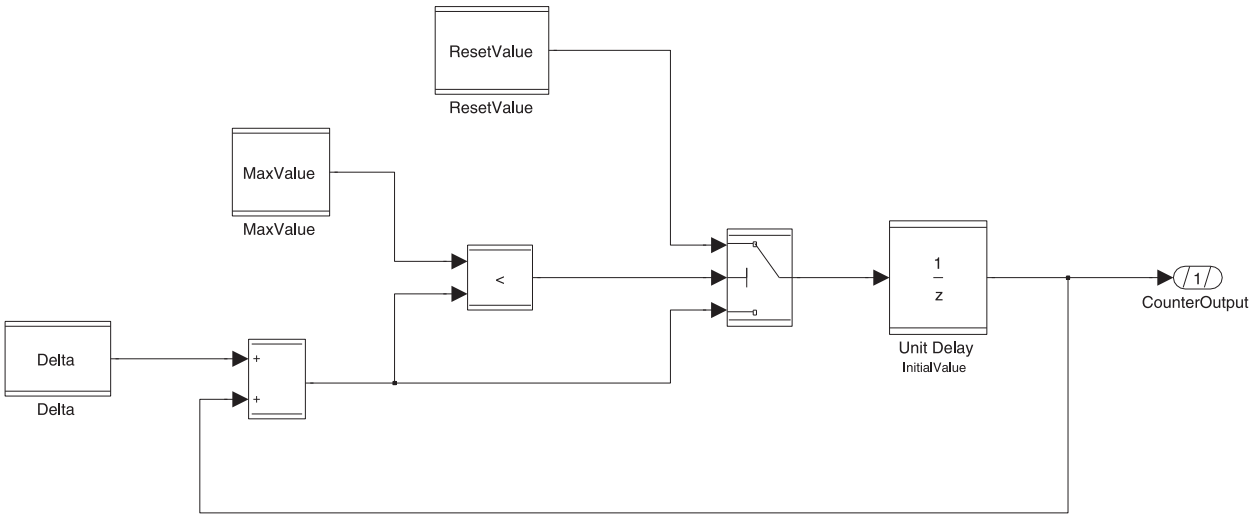


Abbildung 5.10: TargetLink System zur Implementierung eines Zählers mit automatischer Rücksetzfunktionalität, wie es sich vielfach in realen regelungstechnischen Modellen in der Automobilindustrie findet. Das System weist keinen Eingang auf und stellt am Ausgang *CounterOutput* den Wert der Zählvariablen zur Verfügung. Der Switch-Block (der Vorgänger des Unit Delay Blockes) schaltet entweder seinen obersten oder untersten Eingang auf seinen Ausgang durch, je nachdem ob der mittlere Eingang ungleich Null oder gleich Null ist.

Insgesamt ergeben sich damit die folgenden Differenzengleichungen für die Zustandsvariable z des Unit Delay Blockes und die Ausgangsvariable *CounterOutput* des Zähler-

Systems, siehe Abbildung 5.10:

$$\begin{array}{lcl} z(k+1) & = & \begin{cases} ResetValue & \text{falls } z(k)+Delta > MaxValue \\ z(k)+Delta & \text{sonst} \end{cases} \\ z(0) & = & InitValue \\ CounterOutput\,(k) & = & z(k) \end{array}$$

Exemplarisch sind in Abbildung 5.11 und Abbildung 5.12 für zwei unterschiedliche Parameter- und Initialwertkombinationen die entsprechenden Simulationssignale dargestellt, die sich bei Durchführung einer reinen Simulink-Simulation über einige Zeitschritte hinweg ergeben.

Bei der Anwendung der TalInt Analysen entsprechend Abschnitt 4.3.3 auf das Modell des Zählers ergibt sich folgendes Bild:

- Das System besitzt einen eindimensionale Zustandsraum, der durch die Zustandsvariable des Unit Delay Blockes, bzw. durch deren Zustandsupdatevariable für den nächsten Iterationsschritt gegeben ist. Die Zustandsvariable und die zugehörige Zustandsupdatevariable bilden entsprechend der TalInt Vorgabe Bereichsvariablen des Systems.
- Die Zustandsvariable befindet sich in einer Rückkopplungsschleife und wird somit im Allgemeinen einer Widening-Operation zur Erzwingung eines Fixpunktes unterzogen.
- Zur Reduzierung von Dependency-Effekten führt TalInt eine weitere Bereichsvariable am Ausgang des Summationsblockes ein. Ohne eine solche zusätzliche Bereichsvariable ergäbe sich für Intervalle der Breite $w\,([Delta]) \neq 0$ für den Parameter $Delta$ im Allgemeinen ein systematisches Überschätzen, welches nicht beliebig reduziert werden könnte.

Für die TalInt-Algorithmus Parameter $(NInt_{Expan}, NIter_{Wid}) = (5, 15)$ sind in Abbildung 5.13 die mit TalInt ermittelten Wertebereichsgrenzen über den einzelnen Iterationsschritten k für die Ausgangsgröße $CounterOutput$ dargestellt, wenn der Zähler wie in Abbildung 5.11 angegeben, konfiguriert ist. Ausgehend vom Initialwert 0 steigt das Maximum von $CounterOutput$ über den einzelnen Iterationen jeweils um $Delta = 1$ an und erreicht bei der Iteration $k = 10$ den Maximalwert 10. Bei der Berechnung der Zustandsvariable für den nächsten Zeitschritt $k = 11$ wird $Delta = 1$ auf den letzten Zählwert aufaddiert und das Resultat mit $MaxValue = 10$ verglichen. Beim gewählten Algorithmus-Parameter $NInt_{Expan} = 5$ wird das Intervall [10, 11] in 5 gleichmäßige Subintervalle unterteilt und nur noch für das Intervall [10, 10.2] gibt es die Möglichkeit, dass das Rücksetzen des Zählers unterbleibt. Für die weiteren Subintervalle [10.2, 10.4] etc. ist die Vergleichsbedingung wegen $MaxValue = 10$ hingegen grundsätzlich erfüllt und die Zustandsvariable des Unit-Delay Blockes wird auf den Wert $ResetValue = 0$ zurückgesetzt. Daher ergibt sich für $\left[\breve{CounterOutput}\right]$ im Iterationsschritt $k = 11$ der Wertebereich [0, 10.2] und nachfolgend keine Änderung mehr. Da $CounterOutput$ hinsichtlich des Wertebereiches bzw. der Diskretisierung mit der Zustandsvariablen des Unit Delay Blockes identisch ist, bedeutet dies, dass der Zustandsraum ab der Iteration $k = 11$ konstant ist. Folglich beendet TalInt die Berechnung und der ermittelte Wertebereich [0, 10.2] wird für alle Zeitpunkte k eingehalten.

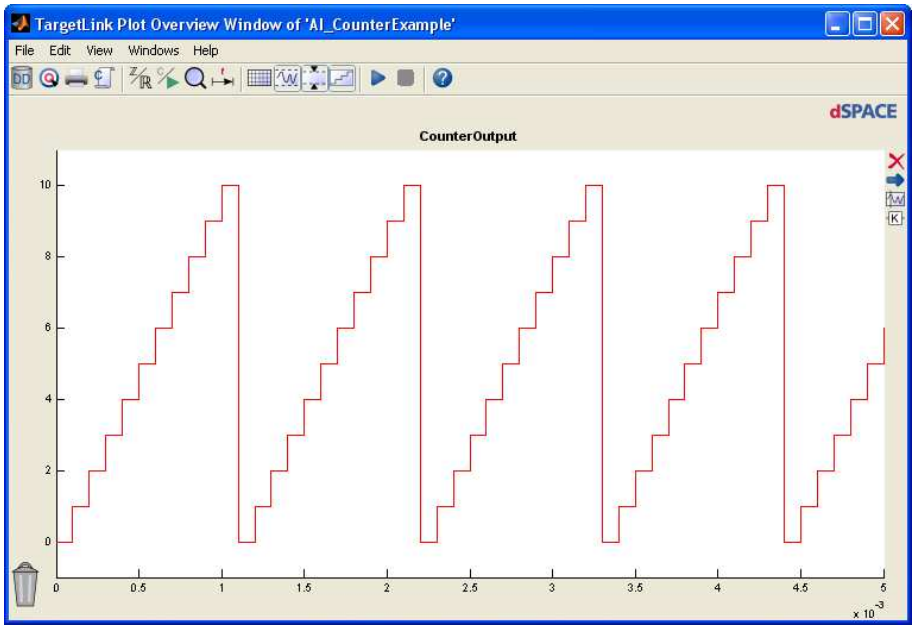


Abbildung 5.11: Signalverlauf des Zählerausgangs *CounterOutput* im Rahmen einer Simulink-Simulation für das Modell aus Abbildung 5.10 bei vorgegebenen Parameter- und Initialwerten $(Delta, MaxValue, ResetValue, InitValue) = (1, 10, 0, 0)$. Die Zählvariable wird vom Initialwert 0 jeweils um $Delta = 1$ bis zum Maximalwert $MaxValue = 10$ erhöht und anschließend auf den Rücksetzwert $ResetValue = 0$ zurückgesetzt.

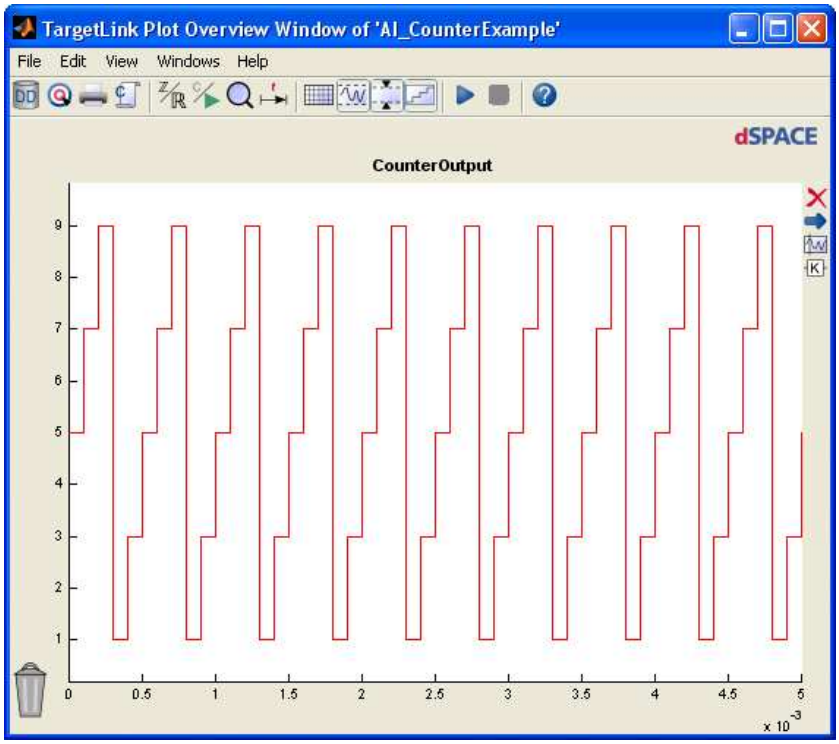


Abbildung 5.12: Signalverlauf des Zählerausgangs *CounterOutput* analog zu Abbildung 5.11, jetzt jedoch für die Parameter- und Initialwertkonfiguration $(Delta, MaxValue, ResetValue, InitValue) = (2, 9, 1, 5)$. Ersichtlich können die relevanten Eigenschaften eines Zählers durch geeignete Vorgabe der Parameter konfiguriert werden.

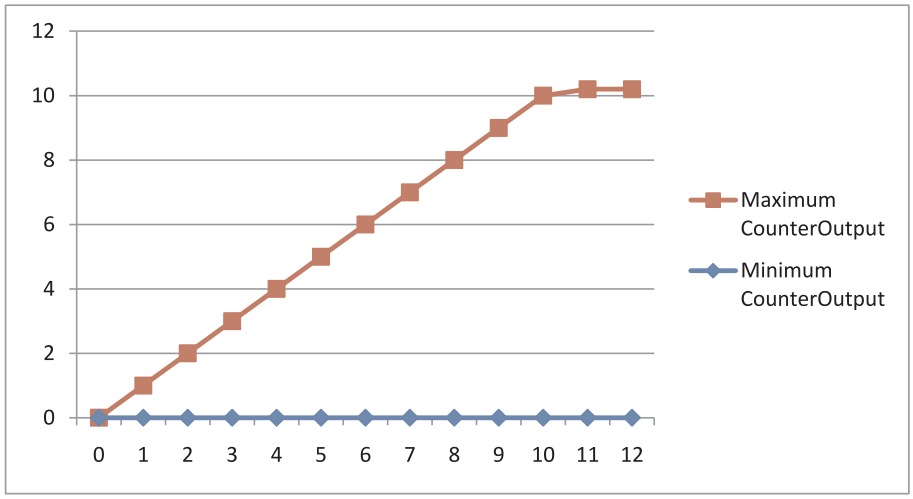


Abbildung 5.13: Die Wertebereichsgrenzen des Ausgangssignals *CounterOutput*, die sich bei der Anwendung von TalInt auf den mit $(Delta, MaxValue, ResetValue, InitValue) = (1, 10, 0, 0)$ konfigurierten Zähler ergeben (dies ist dieselbe Zähler-Konfiguration wie bei der Simulink-Simulation in Abbildung 5.11). Für TalInt wurden dabei die Algorithmenparameter $(NInt_{Expan}, NIter_{Wid}) = (5, 15)$ genutzt. Im Iterationsschritt $k = 11$ wird ein Fixpunkt erreicht und für *CounterOutput* ergibt sich der abgesicherte Wertebereich zu $CounterOutput \in [0, 10.2]$, also ein leichtes Überschätzen. Man beachte, dass in der Grafik nur die Minimal- und Maximalwerte angezeigt werden. Der Zwischenwertebereich ist auf Basis der TalInt Parameter, konkret $NInt_{Expan}$ diskretisiert.

$NInt_{Expan}$	1	2	3	4	5
$\breve{CounterOutput}$	[0, 11]	[0, 10.5]	[0, 10.33334]	[0, 10.25]	[0, 10.2]

Tabelle 5.3: Die von TalInt ermittelten, für alle Iterationsschritte k gültigen Wertebereichsgrenzen für *CounterOutput* in Abhängigkeit des TalInt-Parameters $NInt_{Expan}$. Das Überschätzen des tatsächlichen Wertebereichs [0, 10] lässt sich durch Vergrößern von $NInt_{Expan}$, d.h. durch Verfeinerung der Intervalle beliebig reduzieren.

Offensichtlich verursacht TalInt mit dem berechneten Wertebereich $[0, 10.2] \supset [0, 10]$ für *CounterOutput* ein gewisses Maß an Überschätzen, welches direkt vom Parameter $NInt_{Expan}$ abhängt. Tabelle 5.3 bzw. Abbildung 5.14 geben die von TalInt ermittelten, für alle k gültigen Wertebereichgrenzen für einige andere Werte des TalInt Parameters $NInt_{Expan}$ an, die anhand der obigen Überlegungen gedanklich verifiziert werden können. Man sieht, dass sich das Überschätzen durch Vergrößern von $NInt_{Expan}$ (bei gleichzeitiger Erhöhung des Rechenaufwandes) beliebig reduzieren lässt, obwohl das Zähler-Modell aus Abbildung 5.10 mit dem Relational Operator Block und Switch Block Elemente beinhaltet, die keine Lipschitz-stetigen Output-Funktionen aufweisen. Man beachte, dass für $NInt_{Expan} = 1$ mit den ermittelten Grenzen [0, 11] nur deshalb das Überschätzen so gering ist, weil der Zählerparameter $Delta = 1$ gesetzt wurde. Für beliebig großes *Delta* würde es im Allgemeinen zu beliebig großem Überschätzen kommen, sofern $NInt_{Expan}$ nicht ebenfalls hinreichend groß gewählt wird.

Als weiteres Anwendungsbeispiel sind in Abbildung 5.15 die von TalInt ermittelten Wer-

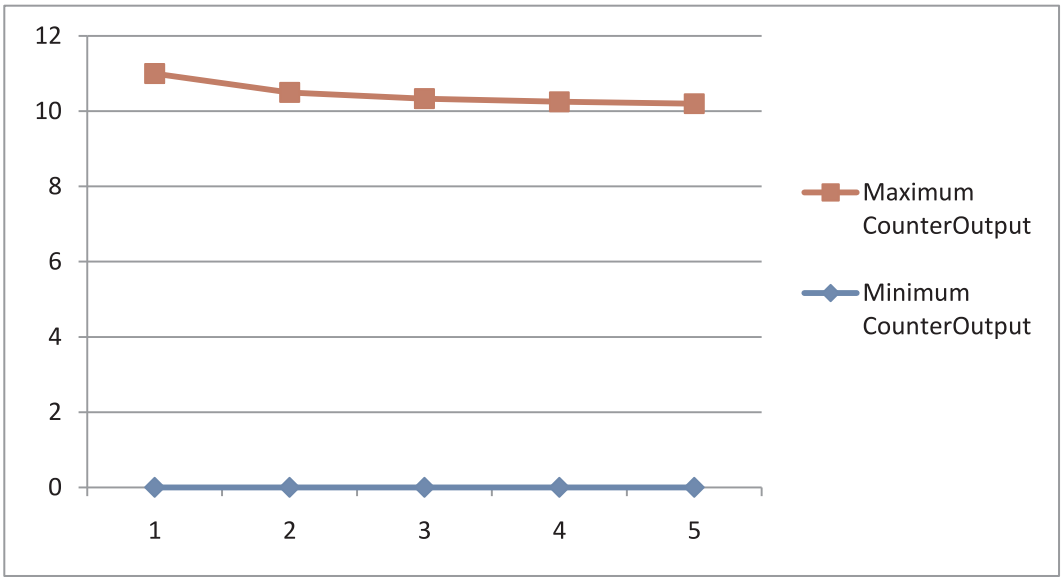


Abbildung 5.14: Darstellung der Wertebereichsgrenzen für *CounterOutput* in Abhängigkeit von $NInt_{Expan}$ entsprechend Tabelle 5.3.

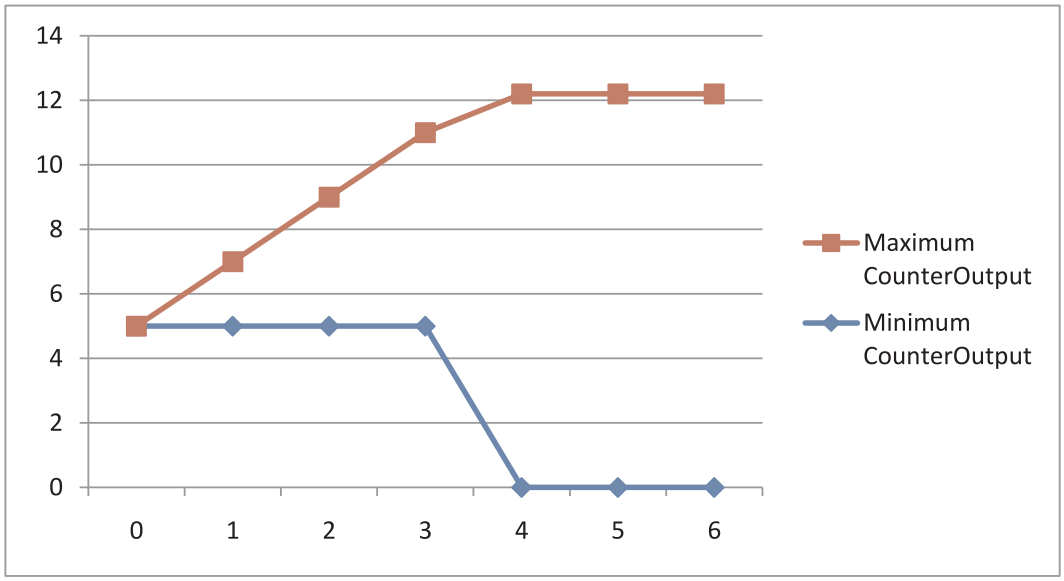


Abbildung 5.15: Die abgeschätzten Minimal- und Maximalwerte von *CounterOutput*, die sich bei der Anwendung von TalInt auf den mit $(Delta, MaxValue, ResetValue, InitValue) = ([0.5, 2]\,[10, 12], 0, 5)$ konfigurierten Zähler in Abhängigkeit des Iterationsschrittes k ergeben. Hier wird der Fixpunkt im Zustandsraum bereits bei der Iteration $k = 5$ erreicht. Die relevanten TalInt Parameter wurden wieder zu $(N_{ScExp}, N_{Wid}) = (5, 15)$ gesetzt.

tebereichsgrenzen für eine weitere Konfiguration des Zählers dargestellt, bei der nicht nur ein von Null verschiedener Initialwert sondern auch Intervalle statt Singletons für einzelne Zähler-Parameter vorgegeben sind. In diesem Fall ist die Konvergenz bereits für $k = 5$ erreicht.

5.1.4 Anwendung auf allgemeine lineare zeitinvariante Filter 1. Ordnung

Die TalInt Algorithmik soll nun auf die allgemeine Struktur eines linearen, zeitinvarianten Filters erster Ordnung angewandt werden, wie es in vielen Anwendungen der Signalverarbeitung auch im automotiven Bereich eingesetzt wird. Die allgemeinste Form eines solchen Filters, bestehend aus Addierer- und Verstärker-Blöcken sowie einem Verzögerungselement ist in Abbildung 5.16 dargestellt. Das System gehorcht den folgenden Differenzengleichungen, wobei z die Zustandsvariable des Unit-Delay Blockes bezeichnet:

$$\begin{aligned} z\,(k+1) &= Input\,(k) + b_1 \cdot z\,(k) \\ Output\,(k) &= a_0 \cdot Input\,(k) + (b_1 \cdot a_0 + a_1) \cdot z\,(k) \end{aligned}$$

Erkennbar ist die Rekursionsgleichung für die Zustandsvariable des Unit-Delay Blockes mit der des Beispiels aus Abschnitt 4.2 identisch, sofern der Verstärkungsparameter b_1 passend gewählt wird. Die TalInt Algorithmik kann also anhand dieses einfachen Beispiels nochmal explizit mit dem im Spezialfall $b_1 = 1/2$ und $Input\,(k) \in [-1, 1]$ gewonnenen Resultat verifiziert werden. Für den Spezialfall $a_0 = 1$ und $a_1 = 0$ ergibt sich sogar insgesamt die identische Struktur des Filters aus Abschnitt 4.2.

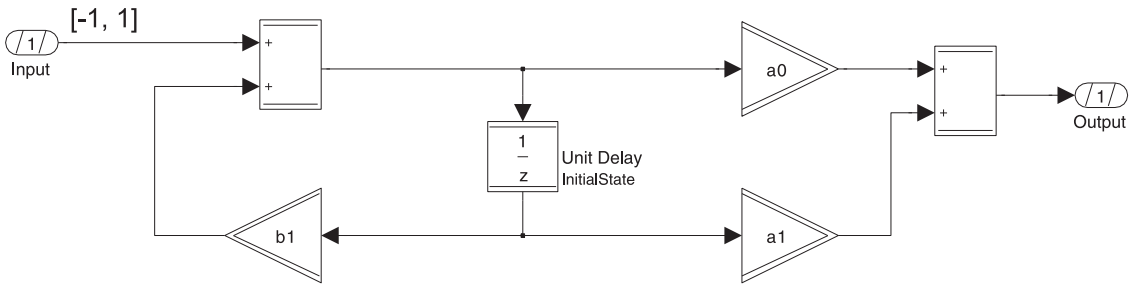


Abbildung 5.16: TargetLink System zur Implementierung eines allgemeinen, linearen, zeitinvarianten Filters 1.Ordnung unter Nutzung von Addierern, Verstärkern und einem Unit Delay Block (Verzögerungselement)

Bei der Anwendung der TalInt Analysen entsprechend Abschnitt 4.3.3 auf das Modell aus Abbildung 5.16 ergibt sich nun das folgende Bild:

- Der Zustandsraum des Systems besteht lediglich aus der Zustandsvariablen z des Unit Delay Blockes. Diese, sowie die Zustandsupdatevariable werden als Bereichsvariablen in die Systembeschreibung aufgenommen.

- Die Zustandsvariable liegt in einer Rückkopplungsschleife und wird daher prinzipiell einer Widening-Operation unterzogen.

k	0	1	2	3	4	5 (Widening)	6
$[z](k)$	[0, 0]	$[-1, 1]$	$[-1,5,\ 1,5]$	$[-1,75,\ 1,75]$	$[-1,875,\ 1,875]$	$[-2,105,\ 2,105]$	$[-2,105,\ 2,105]$

Tabelle 5.4: Minimal und Maximalwerte des Zustandes des Unit Delay Blockes bei der Anwendung von TalInt. Analog zum Beispiel in Abschnitt 4.2 wurde der erforderliche Widening Schritt zur Erzwingung der Konvergenz zum Iterationsschritt $k = 5$ durchgeführt. Die Konvergenz ist dann bei der Iteration $k = 6$ nachgewiesen.

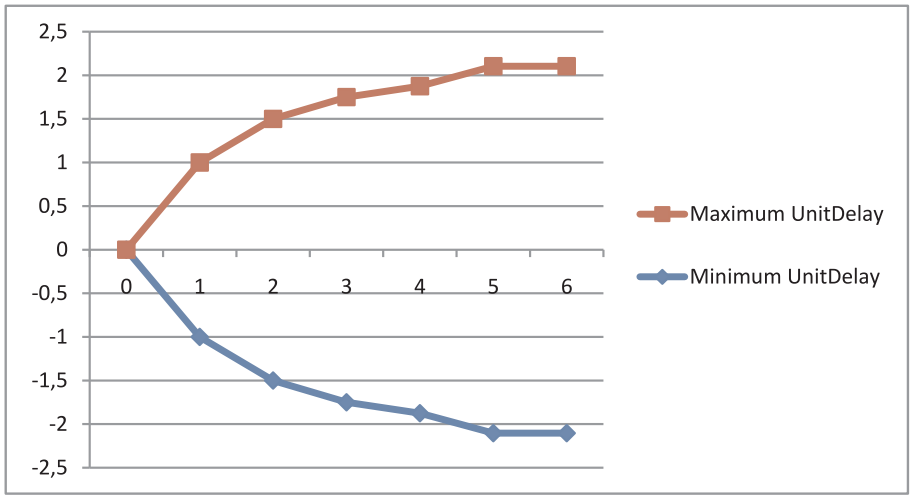


Abbildung 5.17: Minimal und Maximalwerte des Zustandes des Unit Delay Blockes in Abhängigkeit von der Iteration k entsprechend Tabelle 5.4.

- Es gibt keine weiteren Bereichsvariablen, da es zu keinerlei Überschätzen durch Dependency- oder Wrapping-Effekt kommen kann.

In Tabelle 5.4 bzw. Abbildung 5.17 sind die von TalInt ermittelten Wertebereichsgrenzen für die Zustandsvariable z des Unit Delay Blocks in Abhängigkeit von der Iteration k für den Parameter $b_1 = 1/2$ angegeben. Die TalInt-Parameter wurden dabei mit den Werten $(NIter_{Wid},\ F_{Over},\ NInt_{Wid}) = (5-1,\ 10.0,\ 3)$ so gewählt, dass analog zu Abschnitt 4.2 für $k = 5$ eine Widening-Operation durchgeführt wird, um die Konvergenz der Fixpunkt-Iteration zu erzwingen. Die Konvergenz ist dann im nächsten Zeitschritt $k = 6$ nachgewiesen, da der Fixpunkt erreicht ist. Damit ist dann für alle $k \in \mathbb{N}_0$ bewiesen, dass $z\,(k)$ den Wertebereich $[-2,105,\ 2,105]$ nicht überschreitet, wobei es aufgrund der Widening-Operation zu einem Überschätzen der tatsächlichen Wertebereichsgrenzen $[-2,\ 2]$ kommt, siehe Abschnitt 4.2. Die Genauigkeit kann durch Erhöhung der TalInt Parameter $NIter_{Wid}$ oder $NInt_{Wid}$ beliebig gesteigert werden, um die exakten Wertebereichsgrenzen $[-2,\ 2]$ besser zu approximieren.

Zur Reduzierung des Überschätzens wird dasselbe Modell nun nochmal mit anderen Parametern durchgerechnet, konkret mit $(NIter_{Wid},\ F_{Overshoot},\ NInt_{Wid}) = (10,\ 10.0,\ 3)$. Die Widening-Operation wird also nur zu einem späteren Iterationsschritt angewandt, wenn sich $\Delta = |z(k+1) - z(k)|$ weiter verringert hat. Dabei ergibt sich der in Abbildung 5.18 skizzierte Verlauf für die Wertebereichsgrenzen von z in Abhängigkeit von der Iteration k. Die Widening-Operation wird im Iterationsschritt $k = 10 + 1 = 11$ durchgeführt und die Fixpunkt-Konvergenz ist für $k = 13$ nachgewiesen. Damit ergibt sich ein für alle

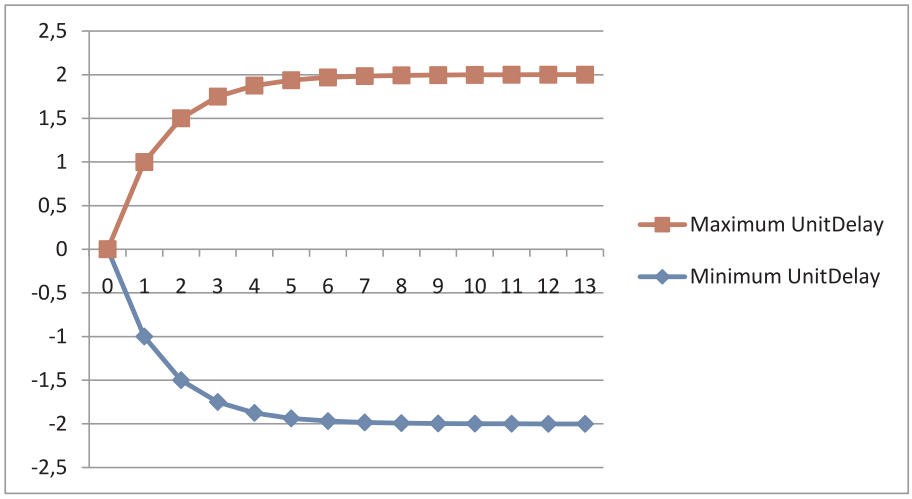


Abbildung 5.18: Minimal und Maximalwerte des Zustandes des Unit Delay Blockes in Abhängigkeit von der Iteration k für die TalInt Parameter $(NIter_{Wid}, F_{Overshoot}, NInt_{Wid}) = (10, 10.0, 3)$ zur Reduzierung des Überschätzens infolge der Widening-Operation.

$k \in \mathbb{N}_0$ gültiger Wertebereich $z \in [-2.000407, 2.000407]$, also eine deutliche Reduzierung des Überschätzens.

Im Allgemeinen kann die Konvergenz natürlich auch ohne Widening eintreten, etwa falls ein geeigneter Anfangswert für den Zustand des Unit-Delay Blocks vorgegeben wird. Für die Filterkonfiguration $b_1 \in [-0.5, 0.5]$ und $InitialState = 3$ ermittelt TalInt die Konvergenz beispielsweise schon zum Zeitschritt $k = 2$ und für den Wertebereich des Zustands des Unit-Delay Blocks findet TalInt $z \in [-2, 5, 3]$, was man auch anhand von Abbildung 5.16 auch verifizieren kann.

5.1.5 Anwendung auf allgemeine lineare, zeitinvariante Filter 2. Ordnung

Als weiteres Beispiel soll nun die Anwendung der TalInt-Algorithmik auf den Fall linearer zeitinvarianter Filter zweiter Ordnung diskutiert werden, wie sie im Bereich der Signalverarbeitung weite Verbreitung gefunden haben. Das in Abbildung 5.19 dargestellte TargetLink System stellt den allgemeinsten Fall eines solchen Filters dar, welches durch Modifikation der Verstärkungsfaktoren bzw. Filterparameter a_0 bis a_2 bzw. b_1 und b_2 beliebig an die gewünschte Filtercharakteristik (Hochpassfilter, Tiefpassfilter, Bandpassfilter, Bandsperrenfilter etc.) angepasst werden kann. Für $a_2 = b_2 = 0$ ergibt sich als Spezialfall das allgemeine lineare, zeitinvariante Filter 1. Ordnung, das bereits in Abschnitt 5.1.4 mit Hilfe von TalInt analysiert wurde. Filter beliebiger, d.h. höherer Ordnung als zwei werden üblicherweise durch Verkettung, d.h. "Hintereinanderschaltung" von Systemen entsprechend Abbildung 5.19 realisiert. Das Filter enthält zwei Verzögerungsglieder (Unit Delay Blöcke), die jeweils über Rückkopplungsschleifen und die Verstärkungsfaktoren b_1 bzw. b_2 auf den Eingang zurückgeführt werden. Das Filter gehorcht der Rekursionsgleichung

$$y(k+2) - b_1 \cdot y(k+1) - b_2 \cdot y(k) = a_0 \cdot x(k+2) + a_1 \cdot x(k+1) + a_2 \cdot x(k),$$

wobei sich die Anfangswerte aus den Initialwerten der Unit Delay Blöcke ergeben. In Analogie zum bereits diskutierten Fall eines rückgekoppelten Filters erster Ordnung ist auch dieses Filter nur für einen eingeschränkten Wertebereich von b_1 und b_2 stabil, d.h. nur in diesem Bereich liefert das Filter Werte, die nicht über alle Grenzen wachsen. Die Parameter a_0 bis a_2 haben hingegen keinen Einfluss auf die Stabilität des Filters, da sie sich nicht in Rückkopplungsschleifen befinden.

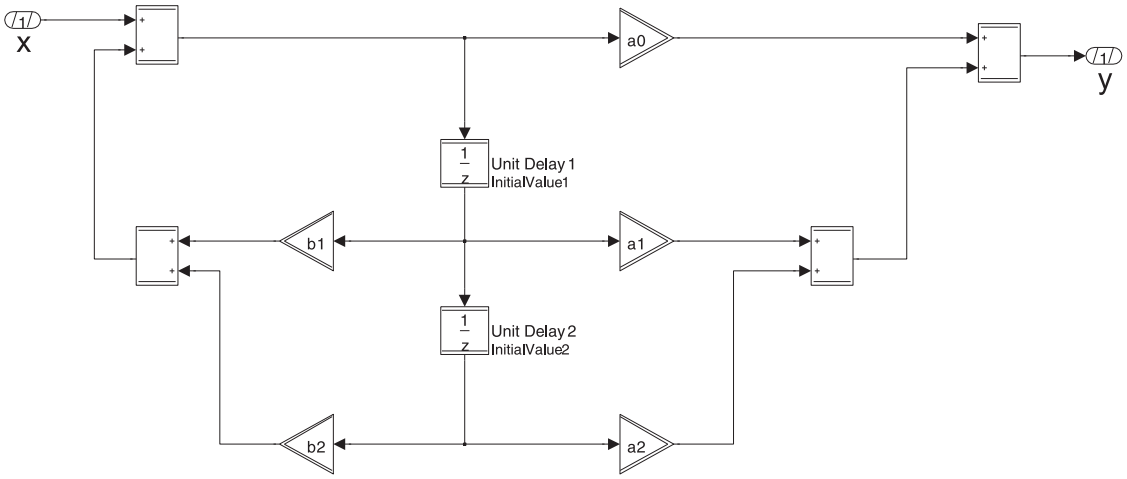


Abbildung 5.19: TargetLink Modell zur Implementierung eines beliebigen linearen, zeitinvarianten Filters zweiter Ordnung zeitdiskreten Zählers wie er sich vielfach in realen Regelungstechnischen Modellen in der Automobilindustrie findet. Das Modellfragment zählt jeweils von

Im Folgenden wird nun TalInt auf das Filter angewandt, wobei als Beispiel die Parameterbereiche $[b_1] = [b_2] = [-0.2, 0.2]$ verwendet werden (mit $[a_1] = [a_2] = [a_3] = 0.5$, wobei letztere Spezifikationen weitestgehend belanglos sind. Bei dieser Wahl von $[b_1]$ und $[b_2]$ ist das Filter stabil und die Analyse von TalInt entsprechend Abschnitt 4.3.3 liefert folgendes Bild:

- Das System beinhaltet zwei Zustandsvariablen in einem zweidimensionalen Zustandsraum, der nicht weiter unterteilt werden kann ohne Überschätzen hervorzurufen, welches prinzipiell nicht reduzierbar ist.
- Beide Zustandsvariablen liegen in einer Rückkopplungsschleife und werden daher prinzipiell einer Widening-Operation unterzogen.
- Außer den Zustandsvariablen bzw. der zugehörigen Zustandsupdatevariablen existieren keine weiteren Bereichsvariablen, da es zu keinem weiteren Überschätzen durch Dependency- oder Wrapping-Effekt kommen kann.

In Abbildung 5.20 und Abbildung 5.21 sind die von TalInt ermittelten Wertebereichsgrenzen für die beiden Zustandsvariablen der Blöcke *UnitDelay1* und *UnitDelay2* für eine Standard-Parameter Konfiguration von TalInt aufgetragen. Offensichtlich wird der gesuchte Fixpunkt erreicht und TalInt terminiert mit den erfolgreich bestimmten Wertebereichsgrenzen. Analoge Situationen ergeben sich z.B. bei Variation der Initialwerte der

Zustandsvariablen oder modifizierten Parametern b_1 und b_2 sofern das Filter nicht direkt am Rande der Stabilität (oder darüber hinaus) betrieben wird.

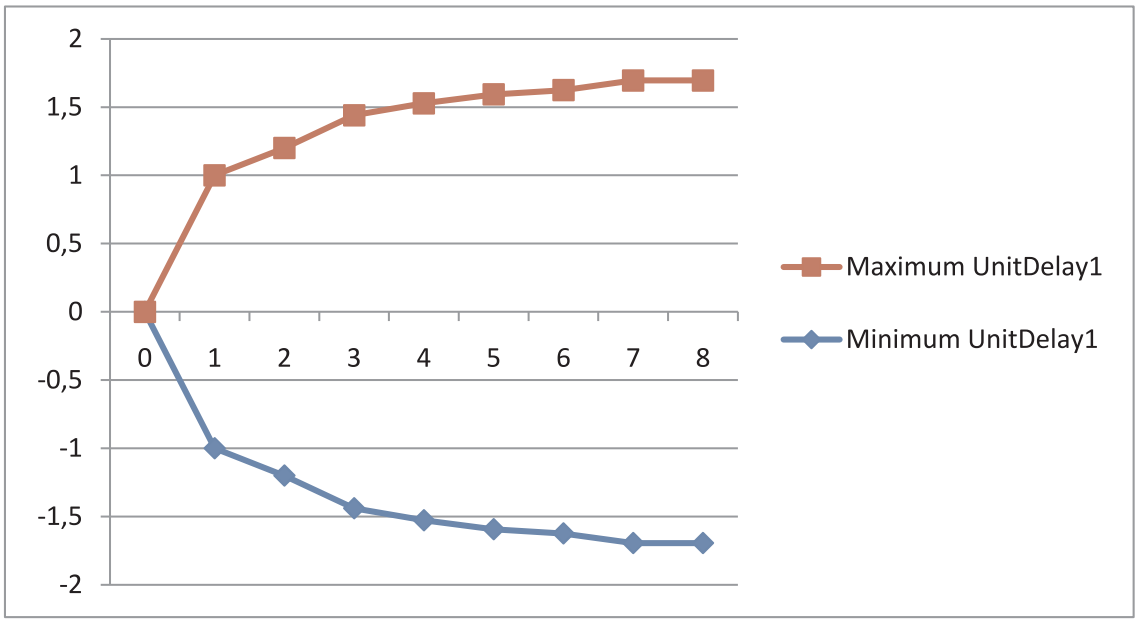


Abbildung 5.20: Minimal- und Maximalwerte der Zustandsvariablen des *UnitDelay1* Blockes in Abhängigkeit vom Iterationsschritt k.

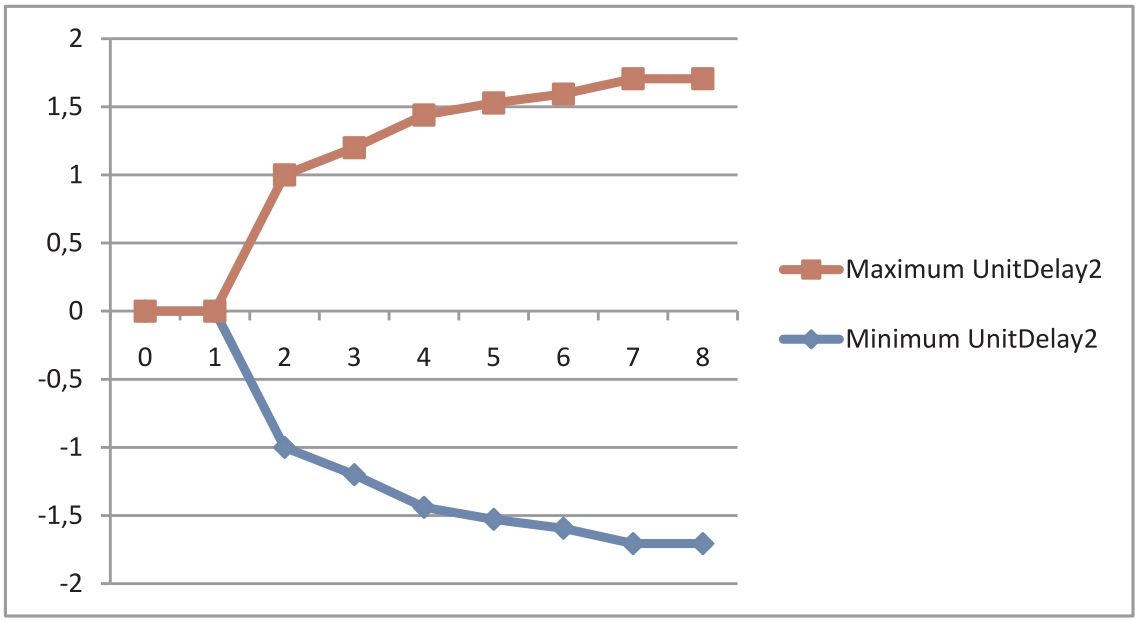


Abbildung 5.21: Minimal- und Maximalwerte der Zustandsvariablen des *UnitDelay2* Blockes in Abhängigkeit vom Iterationsschritt k.

5.1.6 Anwendung auf einen einfachen PI-Controller mit Vorverarbeitung

Als letztes, konkret betrachtetes Beispiel-Modell soll noch kurz die Anwendung von TalInt auf den bereits in Kapitel 2 vorgestellten, einfachen PI-Controller mit Sensorkompensation beschrieben werden. Das Modell des PI-Controllers, das nicht nur ein Modellfragment sondern bereits eine vollständige, wenn auch sehr kleine Anwendung darstellt, ist in Abbildung 5.22 noch einmal abgebildet. Vom Standpunkt der TalInt Analyse ist hier lediglich die Tatsache von Interesse, dass der rückgekoppelte Unit Delay Block in Abbildung 5.22 einen zeitdiskreten Integrator bildet, der jedoch mit Hilfe eines Sättigungsblocks hinsichtlich seiner Wertebereichsgrenzen saturiert wird. Ohne Sättigungsblock wären die Wertebereiche unbegrenzt und TalInt würde nach der vorgegebenen Maximalzahl an Iterationen $NIter_{Max}$ ohne gefundenen Fixpunkt und damit ohne gültiges Resultat terminieren.

Die Anwendung von TalInt auf das PI-Controller Modell gestaltet sich völlig unproblematisch und die Resultate sind weitestgehend unabhängig von der konkreten Wahl der TalInt Parameter. Abbildung 5.23 zeigt die von TalInt ermittelten Wertebereichsgrenzen für die Zustandsvariable des Unit Delay Blockes sowie des Ausgangs des Sättigungsblockes aus Abbildung 5.22. Ein Fixpunkt im Zustandsraum wird bereits im Iterationsschritt $k = 5$ erreicht und TalInt terminiert mit den gefundenen Wertebereichsgrenzen.

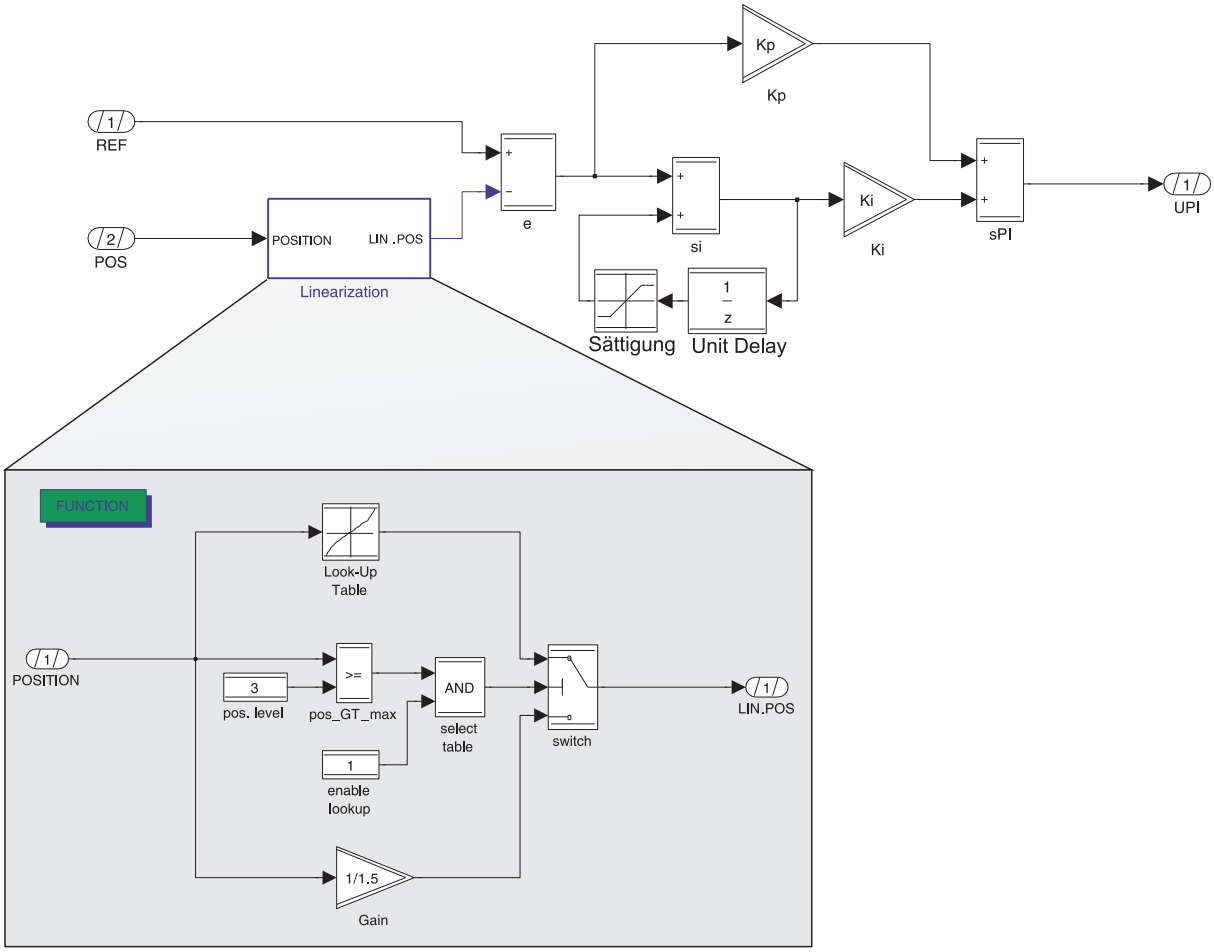


Abbildung 5.22: TargetLink Modell eines PI-Controllers mit Vorverarbeitung zur Kompensation der Kennlinie eines Sensors

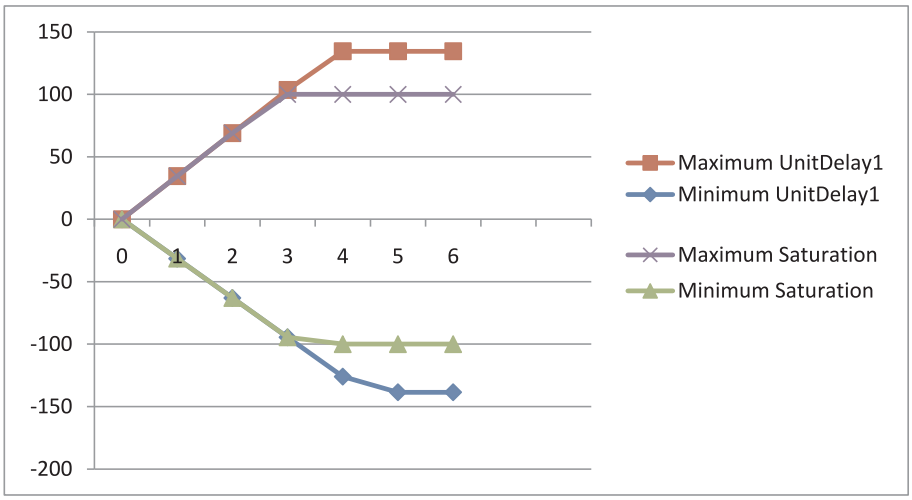


Abbildung 5.23: Minimal- und Maximalwerte der Zustandsvariablen des Unit Delay Blockes und des Ausgangs des Sättigungsblockes in Abhängigkeit vom Iterationsschritt k. Der gesuchte Fixpunkt ist im Schritt $k = 5$ erreicht.

5.2 Zusammenfassung der Erfahrungen und Verbesserungsvorschläge zur Anwendung von TalInt auf reale und idealisierte Modelle

In den vorhergehenden Abschnitten wurde die Funktionsweise und Leistungsfähigkeit von TalInt bei der Anwendung auf einige kleinere Modelle demonstriert, wobei die Modelle teils idealisierter Natur waren, teils extrahierte Modellfragmente aus größeren Modellen darstellten. Die letztere Kategorie hat also Praxisrelevanz und die dabei erzielten Ergebnisse sind prinzipiell sehr ermutigend. Die in den vorherigen Abschnitten präsentierten Benchmarks mit TalInt stellten auch nur eine kleinere Auswahl der tatsächlich durchgeführten Tests dar, da neben der Verwendung weiterer kleinerer Testmodelle oftmals mit Variationen der Modell-Parameter als auch der TalInt-Parameter experimentiert wurde.

Zudem wurde TalInt noch exemplarisch auf drei größere automotive Modelle aus der Praxis angewandt, die bei TargetLink Anwendern im praktischen Einsatz sind und die in Fahrzeugen Anwendung gefunden haben. Auch wenn die mit großen realen Modellen gemachten Erfahrungen noch nicht sehr umfangreich sind, so sollen diese hier kurz zusammengefasst werden:

- Der Versuch, ein Gesamtmodell mit Hilfe von TalInt zu analysieren, war jeweils nicht von Erfolg gekrönt. Das erste Hindernis dabei stellte bereits das von TalInt derzeit unterstützte Subset des TargetLink Sprachumfangs dar, das für eine Komplettanalyse des gesamten Modells grundsätzlich nicht ausreichend war. Es gab im jeden Fall Sprachkonstrukte, die von TalInt (noch) nicht unterstützt wurden.

- Als weiterer Aspekt stellte sich (erwartungsgemäß) heraus, dass die (derzeitige) Performance von TalInt mit Sicherheit nicht ausreichend sein wird, um ein Modell mit einer mittleren zweistelligen Zahl von Eingangs- und Ausgangsvariablen sowie einigen Zustandsvariablen in seiner Gesamtheit zu analysieren. Nur in Ausnahmefällen, etwa bei Vorhandensein von einer minimalen Anzahl von Zustandsvariablen,

keinen Rückkopplungen und einer sehr einfachen Modellstruktur ist dies prinzipiell denkbar.

- Die erfolgversprechende Anwendung von TalInt auf reale praktische Modelle besteht darin, einzelne Subsysteme der Gesamtfunktionalität getrennt zu analysieren und so die kombinatorische Explosion des Raumes der Systemgrößen zu vermeiden bzw. zu reduzieren. Dieser Ansatz erscheint gangbar und wurde exemplarisch an einigen Subsystemen der Modelle praktiziert, sofern das genutzte Sprachsubset hinreichend war. Der Ansatz, einzelne Subsysteme der Gesamtfunktionalität getrennt zu analysieren, wird derzeit von TargetLink Anwendern ohnehin verfolgt und das dadurch verursachte Überschätzen der Wertebereichsgrenzen erscheint oftmals tolerabel. Der Anwender muss dazu die Gesamtfunktionalität auch nicht manuell zerlegen, da ein großes Modell typischerweise ohnehin in einzelne Subsysteme reduzierter Komplexität partitioniert ist, auf welche die TalInt Analyse dann angewandt werden kann.

Im Folgenden sollen noch kurz einige Vorschläge zur Erweiterung und Verfeinerungen des gegenwärtigen Prototypen von TalInt aufgelistet werden, die anhand der gemachten Erfahrungen als weitestgehend zwingend oder geboten erscheinen, um TalInt einem größeren Anwenderkreis zugängig zu machen:

- Das derzeit unterstützte Sprachsubset von TalInt muss erweitert werden. Dies betrifft insbesondere die Unterstützung für konditional ausgeführte Subsysteme sowie sogenannte Simulink Function Calls
- Es sind generelle Konsolidierungen des Prototypen erforderlich, um dessen Robustheit und Benutzerfreundlichkeit zu steigern. Mindestens muss es im Falle von nicht unterstützten TargetLink Sprachkonstrukten geeignete Hinweise an den Benutzer geben. Ferner müssen die von TalInt ermittelten Wertebereichsgrenzen in das TargetLink-Modell automatisch übernommen werden, um den Anwender manuelle Arbeit zu ersparen.
- Hinsichtlich der Implementierung des Prototypen von TalInt wäre die wichtigste Maßnahme zur Performance-Verbesserung eine ausgefeiltere Analyse, welche Berechnungen redundant und daher überflüssig sind. In der derzeitigen Implementierung des Prototypen ist diese Analyse nur rudimentär vorhanden.
- Hinsichtlich der Optimierung der eigentlichen Algorithmik ist der Einsatz von Heuristiken zur adaptiven Festlegung der TalInt-Parametern ein wichtiger Aspekt. In der gegenwärtigen Lösung müssen alle TalInt Parameter vom Benutzer vorgegeben werden und diese sind weder adaptiv noch Signalspezifisch.
- Zur Vermeidung der kombinatorischen Explosion bei Zustandsräumen empfehlen sich Strategien, um deren Dimensionalität auf Kosten eines zu tolerierenden Überschätzens zu reduzieren und damit die Performance von TalInt deutlich zu steigern.

Kapitel 6

Zusammenfassung und Ausblick

Im Rahmen dieser Diplomarbeit wurde ein Verfahren namens TalInt (TargetLink Intervall Interpreter) entwickelt, mit dessen Hilfe Simulink/TargetLink Modelle als eine bestimmte Klasse zeitdiskreter, dynamischer Systeme hinsichtlich der angenommenen Wertebereiche analysiert werden können. Die Algorithmik liefert dabei robuste Wertebereichsgrenzen für alle Systemgrößen, die mit hundertprozentiger Sicherheit eingehalten, das heißt, grundsätzlich nie über- bzw. unterschritten werden. Das Anwendungsszenario von TalInt ist die Umsetzung von Fließkomma-Modellen in Festkomma-Code, für die robuste Wertebereichsinformationen eine absolute Notwendigkeit darstellen. Die entwickelte Algorithmik macht sich Methoden der Intervall-Analysis zu Nutze, um die Bildmenge von multidimensionalen Abbildungen konservativ abzuschätzen, indem die Bildmengen durch ein System von Intervallen bzw. Boxen eingehüllt werden. Da TargetLink Modelle im Allgemeinen dynamische, zustandsbehaftete Systeme darstellen, ist zur Sicherstellung der Robustheit der Wertebereichsgrenzen für alle Zeitpunkte die Lösung einer Fixpunktgleichung für den Zustandsraum des Systems erforderlich, wozu die Intervallanalytischen Methoden um spezielle Maßnahmen erweitert wurden. Zur effizienten Durchführung der Berechnungen von TalInt wird das betrachtete TargetLink Modell als Graph repräsentiert, wobei die Systemgrößen des Modells die Ecken des Graphen darstellen und Kanten Signalverbindungslinien bzw. mathematische Operationen der einzelnen TargetLink Blöcke repräsentieren. Analysen hinsichtlich der Relevanz einzelner Systemgrößen werden zur Dimensionsreduktion eingesetzt, um einer kombinatorischen Explosion des Raumes der Systemgrößen entgegenzuwirken und so den Rechenaufwand zur Bestimmung der Wertebereichsgrenzen zu reduzieren.

Im Rahmen der Diplomarbeit wurde eine prototypische Implementierung von TalInt umgesetzt, anhand derer die entwickelten Methoden an TargetLink-Modellen erprobt wurden, um Stärken und Schwächen der Algorithmik zu identifizieren und Verbesserungsvorschläge machen zu können. TalInt unterstützt dabei ein wohldefiniertes Subset aller TargetLink Sprachkonstrukte, das einerseits hinreichend ist, um TalInt auf selbstentwickelte, praxisrelevante Testmodelle anzuwenden, andererseits aber auch zumindest teilweise zur Analyse von Modellfragmenten existierender automotiver Modelle genutzt werden kann. Bei den im Rahmen der Diplomarbeit durchgeführten Tests mit dem TalInt Prototypen hat sich das Verfahren als prinzipiell sehr leistungsfähig und robust erwiesen, wobei die größte Schwäche von TalInt ganz eindeutig in der schlechten Skalierbarkeit bei der Anwendung auf große Modelle liegt, die sehr viel Rechenzeit erfordern. Dies ist zu einem großen Teil der TalInt-Algorithmik selbst geschuldet, teilweise jedoch auch durch die prototypische

Implementierung bedingt, die viel Raum für Performance-Verbesserungen lässt. Darüber hinaus ist TalInt im praktischen Einsatz auch dann ein Gewinn, sofern nur kleinere Teilmodelle in endlicher Zeit sinnvoll analysiert werden können, da sich auch umfangreiche Modelle immer in kleinere Einheiten unterteilen lassen, die dann mit TalInt einzeln sinnvoll analysiert werden können.

TalInt bietet insgesamt noch sehr viel Raum für weitere Verfeinerungen und Erweiterungen, die teilweise die Implementierung betreffen, teilweise jedoch auch die eigentliche Algorithmik selbst. Um den praktischen Einsatz von TalInt bei TargetLink Anwendern attraktiv zu machen, müsste das unterstützte Subset der TargetLink Sprachkonstrukte noch weiter ausgebaut und die Zahl der Limitierungen verringert werden. Eine Konsolidierung der Implementierung sollte neben einer Performance-Verbesserung auch eine Erhöhung der Robustheit und des Bedienungskomforts zum Ziel haben. Ferner gibt es im Bereich der Verfeinerung der Algorithmik diverse Möglichkeiten, um die Skalierbarkeit zu verbessern und Heuristiken einzusetzen, um die Algorithmen-Parameter von TalInt adaptiv anzupassen.

Anhang A

Spezifikation des Betrachteten Subsets von Simulink/TargetLink

In diesem Abschnitt soll das von TalInt unterstützte Subset von TargetLink aufgelistet werden, welches in Modellen zur Anwendung des Prototypen verwendet werden kann. Alle Angaben beziehen sich auf das Subsystem, das von TalInt analysiert werden soll, d.h. außerhalb des betrachteten Subsystems gelten diese Einschränkungen nicht.

A.1 Generelle Limitierungen

- Es werden nur skalare Signale für TargetLink Blöcke unterstützt, d.h. die Verwendung von vektorwertigen Signalen ist nicht zulässig. Alle Parameter müssen ebenfalls skalarwertig sein, mit Ausnahme der Parameter des Look-up Table Blockes, für den vektorwertige Parameter zugelassen sind.

- Die Verwendung von Simulink Bussen zur Zusammenfassung von einzelnen Signalen ist nur zwischen den einzelnen TargetLink Blöcken möglich. Busfähige Blöcke wie z.B. der Unit Delay Block können lediglich mit skalaren Eingangssignalen beschaltet werden, nicht jedoch mit Bussen. Analog können Mux und Demux Blöcke zum Zusammenfassen von skalaren Signalen zu Vektoren genutzt werden, jedoch müssen die Eingangssignale von TargetLink Blöcken selbst skalarwertig sein.

- Die Verwendung von sogenannten Simulink Function-Call Signalen ist grundsätzlich nicht zulässig. Weder dürfen solche Signale in das betrachtete Subsysteme hineinnoch herausgeführt werden. Innerhalb des betrachteten Subsystems dürfen ebenfalls keine Function-Call Signale verwendet werden.

- Alle Blöcke innerhalb des betrachteten Subsystems müssen eine identische Simulink Sample Rate aufweisen, d.h. sogenannte Multirate-Systeme innerhalb des betrachteten Subsystems sind nicht zulässig.

- Alle existierenden Simulink Datentypen sind zulässig, jedoch werden diese bei der Auswertung nicht berücksichtig, da der Prototyp mit Fließkomma-Signalen arbeitet und andere Datentypen ignoriert.

A.2 Unterstützte Blöcke und deren Spezifikationen

Die folgenden Blöcke werden von TalInt derzeit unterstützt:

- Virtuelle Subsysteme
- Atomare Subsysteme
- Sum-Block mit Addition und Subtraktion als möglichen Operationen. Dabei ist eine beliebige Zahl von Operanden und Kombinationen von Additionen und Subtraktionen in einem Block zulässig.
- Multiply-Block mit Multiplikationen und Divisionen als möglichen Operationen. Dabei ist eine beliebige Zahl von Operanden und Kombinationen von Multiplikationen und Divisionen in einem Block zulässig.
- Relational Operator Block mit den Vergleichsoperationen <, <=, >, >=, == (Test auf Gleichheit) und ~= (Test auf Ungleichheit)
- Logical Operator Block mit AND, OR, NAND, NOR, XOR, NOT als möglichen Operationen. Dabei wird eine beliebige Zahl von Operanden für die logischen Operationen in einem Block unterstützt. Naturgemäß weist der Logical Operator Block für die Negation (NOT) nur einen einzigen Operanden auf.
- Saturation Block
- Constant Block
- Gain Block
- Inport Block
- Outport Block
- MinMax Block mit den Operationen Minimum und Maximum
- Abs Block zur Bildung des Absolutbetrages
- Math Block mit den Operationen Exp (Exponentialfunktion), Log (Logarithmus zur Basis 10), 10^u (Zehnerpotenz), Square (Quadratbildung), Sqrt (Quadratwurzel), Pow (allgemeine Potenzfunktion)
- Trigonometric Block mit den Operationen Sin, Cos, Tan, Asin, Acos, Atan, Sinh, Cosh
- Unit Delay Block
- Switch Block mit den Operationen U2 Greater Equal Threshold, U2 Greater Threshold, U2 Equal 0

Grundsätzlich können die obigen Blöcke entsprechend der Simulink/TargetLink Regeln beliebig verschaltet werden.

Anhang B

Anmerkungen zur Bedienung von TalInt

B.1 Versionsabhängigkeit des TalInt-Prototypen

Die Nutzung des TalInt Prototypen erfordert sowohl eine MATLAB/Simulink- als auch eine TargetLink Installation auf einem handelsüblichen Windows PC. Entwickelt und evaluiert wurde der TalInt-Prototyp dabei mit den folgenden Versionen:

- MATLAB/Simulink Release R2007b von The MathWorks
- TargetLink 3.1 von dSPACE.

Wohingegen die Versionsabhängigkeit von MATLAB/Simulink eher gering sein dürfte, kann TalInt nicht mit älteren TargetLink Versionen als 3.1 genutzt werden. Für zukünftige TargetLink Versionen ist bei der Anwendung von TalInt jedoch nicht mit besonderen Inkompatibilitäten zu rechnen.

Die Sourcen des TalInt Main Programmes liegen in Form eines C++ Projekts des Microsoft Developer Studios in der Version 2008 vor.

B.2 Anwendung des TalInt-Prototypen

Wie in Abbildung 4.19 schon dargestellt wurde, besteht die Anwendung des TalInt Prototypen aus zwei separaten Schritten, nämlich

- der Extraktion der TargetLink System-Beschreibung als XML-File, die aus einem geöffneten TargetLink Modell in Simulink erfolgt
- der Ausführung der TalInt Main-Anwendung als einer Konsolenapplikation, welche die Abschätzung der Wertebereiche basierend auf der TargetLink System-Beschreibung durchführt.

Die konkrete Nutzung des Prototypen sieht nun folgendermaßen aus:

1. Zunächst muss MATLAB/Simulink gestartet werden.

2. In der laufenden MATLAB Session muss das MATLAB M-File AISetup.m ausgeführt werden, welches die MATLAB M-Files einrichtet und insbesondere die im Folgenden benutzten Kontext-Menü Einträge in Simulink verfügbar macht.

3. Ein TargetLink Modell muss in der laufenden MATLAB/Simulink Session geöffnet werden. Das Modell muss in Simulink initialisierbar sein.

4. Innerhalb des TargetLink Modells muss das sogenannte TargetLink Subsystem oder ein darunter befindliches Subsystem selektiert werden. Anschließend muss nach einem rechten Maus-Klick der Menü-Eintrag *AI Preparation* aus dem Kontext-Menü des Subsystems ausgewählt werden, siehe Abbildung B. Hierdurch wird der Simulink Debugger gestartet.

5. Im MATLAB Command Window muss anschließend das Kommando *slist* eingebeben werden.

6. Anschließend muss im MATLAB Command Window das Kommand *stop* eingegeben werden, wodurch der Simulink Debugger beendet wird.

7. Nun muss aus dem Kontextmenü des selektierten Subsystems der Menü-Eintrag *AI Start* ausgewählt werden. Hierdurch werden die benötigten Informationen des zu analysierenden Subsystems als XML-File mit dem Namen *TalIntSystem.xml* exportiert. Im selben Verzeichnis muss ein File namens *TalIntParameter.xml* liegen, in welchem die bei der Analyse zu verwendenden TalInt-Parameter vom Benutzer vorgegeben werden können. TalInt kann anschließend durch den Aufruf *TalInt* gestartet werden.

8. Nach Abschluss der Berechnungen von TalInt-Main wird in das aktuelle Verzeichnis ein Ausgabe-File *TalIntResults.txt* geschrieben, welches die berechneten Wertebereichsgrenzen im Textformat beinhaltet.

Hinweis: Der TalInt Prototyp in seiner bisherigen Form wurde ausschließlich dazu entwickelt, die Tauglichkeit der vorgeschlagenen Algorithmik in Bezug auf TargetLink Modelle zu testen und zu überprüfen. Er war also ausschlielich Mittel zum Zweck. Er wurde nicht dazu entwickelt, bereits im jetzigen Stadium von Anwendern wirklich genutzt zu werden. Es wurde ferner keinerlei Schwerpunkt auf Bedienerkomfort und Benutzerführung gelegt. Rückfragen können an ulrich_eisemann@yahoo.de gerichtet werden.

B.2.1 Zusammenfassung der TalInt-Parameter

In diesem Abschnitt sollen noch einmal kurz die Parameter von TalInt zusammengefasst werden:

- $NIter_{max}$ als die maximale Zahl der von TalInt durchzuführenden Iterationen. Nach Überschreiten dieser Zahl terminiert TalInt ohne Lösung der Aufgabenstellung 14, weil die Fixpunkt-Gleichung Gl. 4.4 nicht gelöst werden konnte.

- $NInt_{Init}$ als die initiale Anzahl der Intervallunterteilungen für jede Bereichsvariable, die für $k = 0$ durchgeführt wird, sofern es sich nicht um ein Singleton handelt

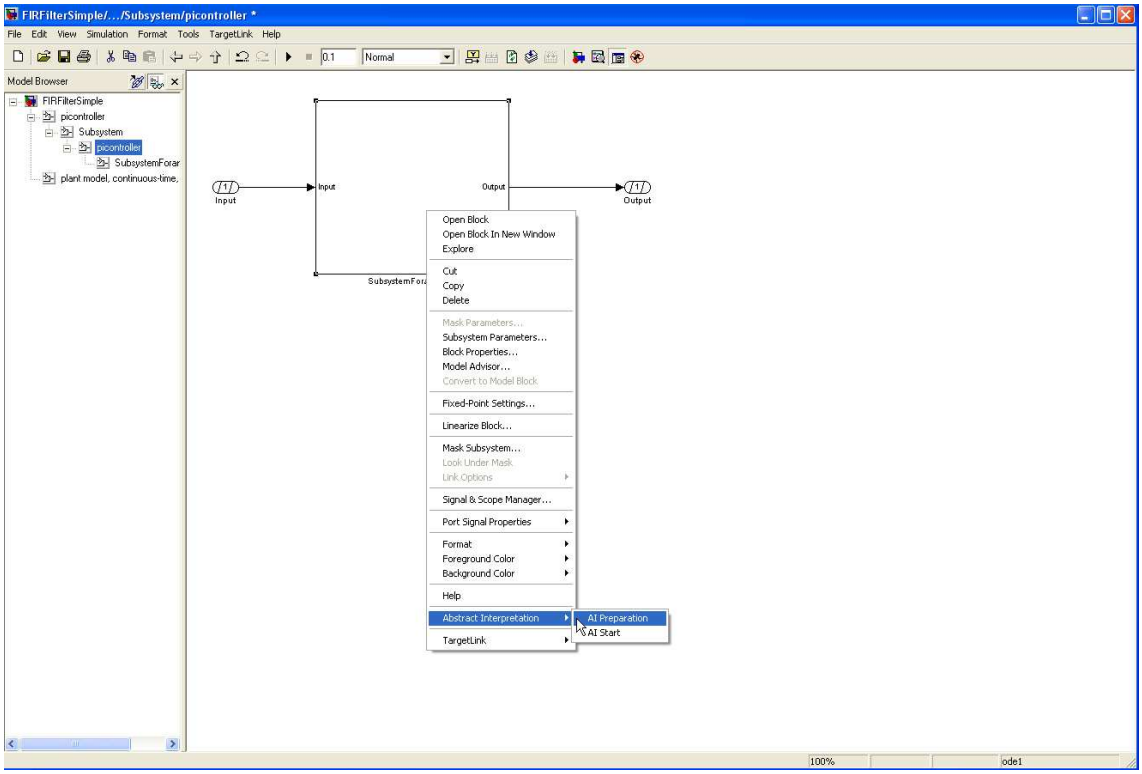


Abbildung B.1: Extraktion der benötigten Informationen über das von TalInt zu analysierende System.

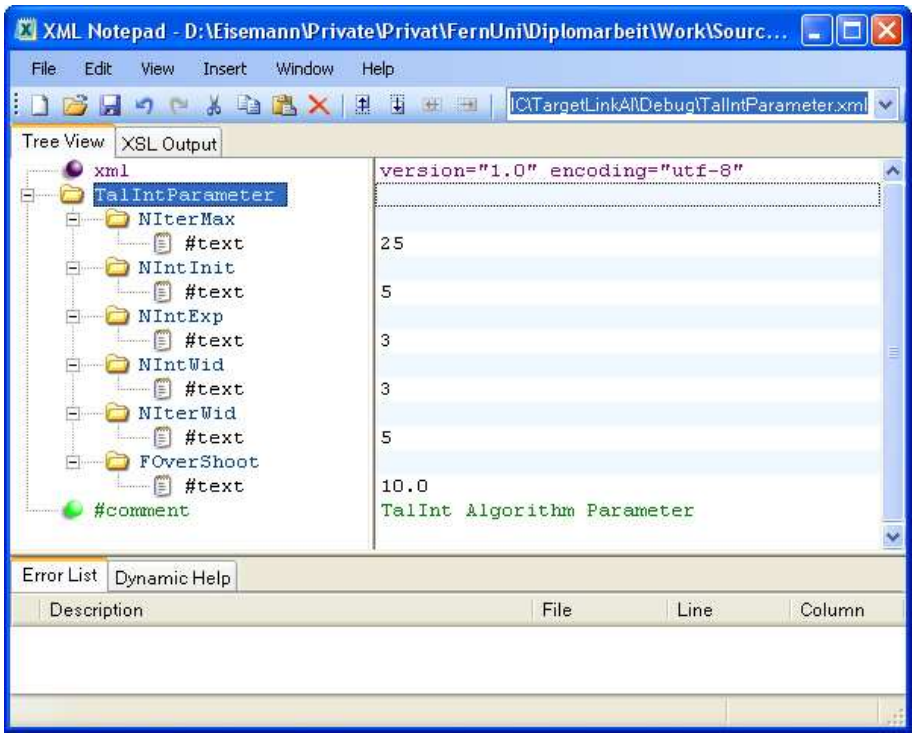


Abbildung B.2: Ansicht des TalInt-Parameter Files, welches die Genauigkeit und Rechenzeit bei der Anwendung von TalInt steuert. Im Hinblick auf die Bedeutung der einzelnen Parameter, siehe Abschnitt B.2.1.

- $NInt_{Expan}$ als die Anzahl der Intervall-Unterteilungen bei einer Ausweitung der Skala einer Bereichsvariablen ohne Widening.
- $NInt_{Wid}$ als die Anzahl der Intervall-Unterteilung bei einer Ausweitung der Skala einer Bereichsvariablen mit Widening.
- $NIter_{Wid}$ als die Anzahl der Iterationen von TalInt, aber der für Zustandsvariablen in Rückkopplungsschleifen eine Widening-Operation zur Vergrößerung des Zustandsraumes durchgeführt wird.
- F_{over} als Faktor zur Ausweitung der Skala einer Bereichsvariablen mit Widening.

Genauere Ausführungen zu den Parametern $NInt_{Init}$, $NInt_{Expan}$, $NInt_{Wid}$, $NIter_{Wid}$, F_{over}, die ausschließlich mit der Konstruktion bzw. Erweiterung der Skalen für einzelne Bereichsvariablen zu tun haben, finden sich im Anhang

B.2.2 TalInt Parameter zur Initialisierung und Erweiterung der Skalen für Bereichsvariablen

In diesem Abschnitt sollen die unterschiedlichen Arten dargestellt werden, wie Skalen für Bereichsvariablen in TalInt initialisiert und erweitert werden. Diese Vorgänge werden darüber hinaus durch TalInt-eigene Parameter gesteuert, welche vom Anwender vorgegeben werden.

B.2.3 Skaleninitialisierung

Sofern noch keine Skala für eine Bereichsvariable $bvar_i$ existiert und folglich kein existenter Maximal- oder Minimalwert, so wird die Skala der Bereichsvariablen nach dem folgenden Kriterium aufgebaut.

- Sofern $NewMin = NewMax$ gilt, wenn die Skala also zu einem einzigen Punkt degeneriert ist, wird die Skala der Bereichsvariablen $bvar_i$ mit einem einzigen Skalenbereich $bvar_i^0 = [NewMin,\ NewMin]$ initialisiert, also einem Intervall der Breite 0.
- Sofern $NewMin < NewMax$ erfüllt ist, wird die Skala der Bereichsvariablen $bvar_i$ mit $NInt_{Init}$ Intervallen gleicher Breite initialisiert. Die Skala deckt dann das Intervall $[NewMin,\ NewMax]$ ab, wobei dieser Bereich durch die $NInt_{Init}$ Intervalle identischer Breite unterteilt wird, welche die Skalenbereiche $bvar_i^0$ bis $bvar_i^{NInt_{Init}-1}$ bilden. Der ganzzahlige Wert $NInt_{Init}$ (Number of INTervals for INITialization) ist ein TalInt-Parameter, welcher vom Anwender unter der Nebenbedingung $NInt_{Init} \geq 1$ vorgegeben wird. Der Vorgang ist in Abbildung 4.13 veranschaulicht.

B.2.4 Skalenweiterung Ohne Widening

Sofern der neu errechnete Wertebereich (charakterisiert durch $NewMin$ und $NewMax$) der Bereichsvariablen $bvar_i$ den derzeitigen Wertebereich der Skala überschreitet, muss diese erweitert werden, wobei dies gegebenenfalls nach oben, nach unten oder in beiden Richtungen geschehen muss. Die Skalenerweiterung wird folgendermaßen vorgenommen:

- Sofern $NewMax > OldMax$ gilt, muss die Skala für $bvar_i$ nach oben erweitert werden. Dies wird durchgeführt, indem das Intervall $[OldMax, NewMax]$ in $NInt_{Expan}$ äquidistante Intervalle unterteilt wird, welche zum Skalenbereich von $bvar_i$ hinzugefügt werden. $NInt_{Expan}$ (Number of INTerval EXPANsion) ist ein ganzzahliger Parameter von TalInt, der vom Benutzer unter der Nebenbedingung $NInt_{Expan} \geq 1$ vorgegeben wird. Der Vorgang ist in der rechten Hälfte von Abbildung 4.14 veranschaulicht.

- Sofern $NewMin < OldMin$ gilt, muss die Skala für $bvar_i$ nach unten erweitert werden. Dies wird durchgeführt, indem das Intervall $[NewMin, OldMin]$ in $NInt_{Expan}$ äquidistante Intervalle unterteilt wird, welche zum Skalenbereich von $bvar_i$ hinzugefügt werden. Der Vorgang ist in der linken Hälfte von Abbildung 4.14 veranschaulicht.

B.2.5 Skalenerweiterung mit Widening

Die letzte Variante der Skalenerweiterung einer Bereichsvariablen $bvar_i$, die im Folgenden als Achsenerweiterung mit Widening bezeichnet werden soll, dient in Kombination mit der $Map2Scale$-Abbildung dazu, den Zustandsraum explizit auszudehnen (Widening-Operation). Wie in Abschnitt 4.2 dargestellt wurde, ist es im Falle von Rückkopplungen innerhalb des Graphen in der Regel erforderlich, den Zustandsraum "künstlich" auszudehnen, um die Konvergenz gegen einen Fixpunkt zu erzwingen. Die Skalenerweiterung mit Widening wird nur durchgeführt, sofern $NewMin$ und/oder $NewMax$ außerhalb der derzeitigen Skala für die Bereichsvariable $bvar_i$ liegen.

- Sofern $NewMax > OldMax$ gilt, muss die Skala für $bvar_i$ nach oben erweitert werden. Dies wird durchgeführt, indem die Skala auf den neuen Maximalwert

$$NewMaxWid = OldMax + (NewMax - OldMax) \cdot F_{Over}$$

  ausgedehnt wird. F_{Over} (Factor of OVERshoot) ist dabei ein reellwertiger TalInt-Parameter, welcher vom Anwender unter der Nebenbedingung $F_{Over} > 1$ vorgegeben werden muss. Der Wertebereich $[OldMax, NewMaxWid]$ wird dann in $NInt_{Wid}$ äquidistante Intervalle unterteilt, welche zum Skalenbereich von $bvar_i$ hinzugefügt werden. $NInt_{Wid}$ (Number of INTerval WIDening) ist ein ganzzahliger Parameter von TalInt, der vom Benutzer unter der Nebenbedingung $NInt_{Wid} \geq 1$ vorgegeben wird. Der Vorgang ist in der rechten Hälfte von Abbildung 4.15 veranschaulicht.

- Sofern $NewMin < OldMin$ gilt, muss die Skala für $bvar_i$ nach unten erweitert werden. Dies wird durchgeführt, indem die Skala auf den neuen Minimalwert

$$NewMinWid = OldMin + (NewMin - OldMin) \cdot F_{Over}$$

  ausgedehnt wird. Der Wertebereich $[NewMinWid, OldMin]$ wird dann in $NInt_{Wid}$ äquidistante Intervalle unterteilt, welche zum Skalenbereich von $bvar_i$ hinzugefügt werden. Der Vorgang ist in der linken Hälfte von Abbildung 4.15 veranschaulicht.

Literaturverzeichnis

[Han01] „Vom Modell zum Seriencode“, H. Hanselmann, ATZ 09/2001

[Bei03] ”Auf direktem Weg zum Seriencode“, M. Beine, G. Gruhn, Elektronik Automotive, Februar 2003

[SL09] ”Simulink User's Guide“, The MathWorks, September 2009

[TL09] ”TargetLink Production Code Generation Guide”, dSPACE, 2009

[Jau01] ”Applied Interval Analysis”, L. Jaulin, M. Kieffer, O. Didrit, E. Walter, Springer 2001

[Mo79] ”Methods and Applications of Interval Analysis”, R. Moore, SIAM Studies in Applied Mathematics, Philadelphia 1979

[Mo09] ”Introduction to Interval Analysis”, R. Moore, R. Kearfott, M. Cloud, SIAM, 2009

[Hy92] ”Constraint reasoning based on interval arithmetic; the tolerance propagation approach”, E. Hyvönen, Artificial Intelligence 58 (1-3):71-112, 1992

[St94] ”Affine Arithmetic”, M. Vinícius, A. Andrade, J. Comba, J. Stolfi, INTERVAL'94, St. Petersburg (Russland), 5-10 März, 1994

[Cou77] ”Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixed-Points”, P. Cousot, R. Cousot, Fourth ACM Symposium on Principles of Programming Languages, California, 1977

[Cou92] ”Comparing the Galois Connection and Widening/Narrowing Approaches to Abstract Interpretation”, P. Cousot, R. Cousot, Programming Language Implementation and Logic Programming, Volume 631, Springer, 1992

[MSXML60] ”Microsoft XML Version 6.0”, Microsoft, Version 6.0

[Leda51] ”LEDA - The LEDA User Manual”, Algorithmic Solutions, Version 5.1

[CXSC20] ”C-XSC 2.0 - A C++ Class Library for Extended Scientific Computing”, W. Hofschuster, W. Krämer, S. Wedner, A. Wiethoff, 2001

[Gl08] ”GloptiPoly 3: moments, optimization and semidefinite programming”, D. Henrion, J. Lasserre, J. Löfberg, Version 3.3, September 23, 2008

[TL08] ”Modeling Guidelines for Simulink/Stateflow and TargetLink”, Version 2.1, dSPACE GmbH, 2008