UNIVERSITÄT STUTTGART

Beschreibung Code-Generator Version 2.0 zur Erzeugung von DIVA-Grundelementen

R. Köhler

Report Nr. 00-2 September 2000

INSTITUT FÜR SYSTEMDYNAMIK
UND REGELUNGSTECHNIK

Prof. Dr.-Ing. Dr. h.c. mult. E.D. Gilles

Beschreibung Code-Generator Version 2.0 zur Erzeugung von DIVA-Grundelementen*

R. Köhler

Institut für Systemdynamik und Regelungstechnik, Universität Stuttgart
Postfach 80 11 40, D–70511 Stuttgart, Germany,
Fax: +49–711–685–6371, E–Mail: koehler@isr.uni–stuttgart.de

Stuttgart, den 20. September 2000

Kurzfassung

Die vorliegende Beschreibung des Code-Generators richtet sich an Modellentwickler für die Simulationsumgebung DIVA (**D**ynamische S**i**mulation **V**erfahrenstechnischer **A**nlagen) [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14]. Es wird davon ausgegangen, daß die Modellgleichungen für ein zu programmierendes Grundelement in der von DIVA geforderten linear-impliziten Differential-Algebra-Gleichungsform vorliegen. Der Code-Generator erzeugt die von DIVA benötigten FORTRAN-Unterprogramme zur Initialisierung, Funktionsauswertung, Datenausgabe und Ereignisdefinition [15] sowie ein formal passendes Datenfile, das die Werte der Modellparameter und Anfangsbedingungen für ein Simulationsexperiment mit dem Apparate-Modell enthält. Im einzelnen wird erläutert, wie eine deklarative Modellspezifikation in einer Datei (ASCII-File) angegeben werden kann und wie der Code-Generator für die Programmierung von neuen Apparate-Modellen und deren Implementierung in der DIVA-Modellbibliothek eingesetzt wird. Der Code-Generator ist in der Programmiersprache COMMON LISP auf Workstations der Firma SUN mit dem Betriebssystem UNIX implementiert.

In [16, 17, 18] wird der Code-Generator und als eigenständige Programmschicht zwischen der Modellierung und der Simulation von verfahrenstechnischen Prozeßmodellen mit DIVA beschrieben. Die Literaturstelle [18] gilt auch als Referenz für den Code-Generator.

---

*Frühere Berichte:
Code-Generator Version Beta.4, März 1995, S. Räumschüssel
Code-Generator Version Beta.6, Mai/Juni 1995, S. Räumschüssel
Code-Generator Version 1.0 Interner Report Nr. 96-6, September 1996 , S. Räumschüssel
Code-Generator Version 1.1, Mai 1997, S. Räumschüssel, R. Köhler
Code-Generator Version 1.2, Januar 1998., S. Räumschüssel, R. Köhler
Code-Generator Version 1.3, Interner Report Nr. 99-1, März 1999., R. Köhler
Code-Generator Version 1.3.1, Dezember 1999., R. Köhler

Inhaltsverzeichnis

Kurzfassung **i**

1 Einführung **1**

1.1 Simulationsumgebung DIVA . . . 1

1.2 Funktionalität des Code-Generators . . . 2

2 Interner Aufbau des Code-Generators **5**

2.1 Datenstruktur . . . 6

2.2 Programmablauf . . . 6

2.3 FORTRAN-Ausgabe . . . 7

2.4 Implementierungsphasen Version Beta.4 – Version 2.0 . . . 7

3 Modellspezifikationsdatei für den Code-Generator **10**

3.1 Angaben zur Modellbeschreibung . . . 12

3.2 Benennung der Integrationsvariablen (Zeit) . . . 13

3.3 Angaben zu Variablen und Parametern . . . 13

3.3.1 Parameter (parameters) . . . 14

3.3.2 Variablen (variables) . . . 16

3.3.3 Übersicht der Schlüsselworte zu Parameter und Variablen . . . 18

3.3.4 Parameterwerte und Anfangswerte von Zustandsvariablen . . . 19

3.4 Angaben zu Events . . . 23

3.4.1 Gemischt kontinuierlich-ereignisdiskrete Modelle in DIVA . . . 24

3.4.2 Petri-Netze zur Modellierung des diskreten Teilsystems . . . 28

3.4.3 Petri-Netze im Code-Generator . . . 33

3.4.4 Definition von Petri-Netzen . . . 36

3.4.5 Definition von Ereigniszeitpunkten . . . 39

3.5 Angaben zur Ansteuerung der Datenausgabe (process-values) . . . 42

3.6 Angaben zu Modellgleichungen . . . 46

3.6.1 Bausteine zum Aufbau der Modellgleichungen . . . 46

3.6.2 Eingabe von Gleichungen . . . 48

3.6.3 Günstige Gleichungsangabe und Zuweisung an Hilfsvariablen . . . 52

3.6.4 Berechnung von Hilfsvariablen mittels Unterprogramm-Aufrufen . . . 58

4 Initialisierung und Steuerbefehle des Code-Generators **65**

4.1 Initialisierung des Code-Generators . . . 65

4.2 Steuerbefehle des Code-Generators . . . 65

A Zusammenfassende Übersicht über LISP-spezifische Ausdrücke **69**

B Erläuterungen zur Grammatikschreibweise 70

C Definition arithmetischer und logischer Ausdrücke 71

D Aufruf des Code-Generators vom Betriebssystem 78

E Code-Generator Eingabe-File: Beispiel Behälter 81

F Code-Generator Eingabe-File: Beispiel Rohrreaktor 91

G CG-Eingabe-File: Beispiel Sättigungskennlinie 96

H Code-Generator Eingabe-File: Beispiel Ablaufsteuerung eines Batch-Prozesses 100

Literatur 111

1 Einführung

Bei der Simulation verfahrenstechnischer Prozesse und Anlagen steht der Anwender immer wieder vor dem Problem, daß in der Modellbibliothek des Simulationssystems die benötigten Apparate-Modelle nicht vorhanden sind. In diesem Fall müssen die fehlenden Modelle neu erstellt werden. Das erfordert im allgemeinen neben der Herleitung der Modellgleichungen die Vorverarbeitung der Gleichungen sowie deren Codierung in die vom Simulationssystem geforderten Form. Der nachfolgend beschriebene Code-Generator unterstützt die Codierung von Modellen für die Simulationsumgebung DIVA (**D**ynamische S**i**mulation **V**erfahrenstechnischer **A**nlagen) [4], die im nächsten Abschnitt beschrieben wird. Anschließend werden die Funktionalitäten des Code-Generator erläutert dem ein Überblick der Gliederung der vorliegenden Benutzeranleitung folgt.

1.1 Simulationsumgebung DIVA

Der erste Entwurf einer Software-Architektur des Modellierungswerkzeugs und der Simulationsumgebung DIVA [19] basiert auf den in [20, 21, 22, 23, 24] beschiebenen Konzepten. Davon ist die derzeitige Architektur der Simulationsumgebung DIVA abgeleitet worden, siehe Abb. 1. Der Code-Generator hat darin die Schnittstellenfunktion zwischen den Modellierungswerkzeugen bzw. dem Benutzer und der numerischen Simulation, Analyse und Optimierung mit DIVA. Die Simulationsumgebung DIVA verwendet bei der mathematischen Beschreibung von Anlagen und Anlagenkomponenten linear implizite Differential-Algebra-Gleichungssysteme (DAEs) der Form

$$B(x,u,p,\hat{p},m,t)\,\dot{x} = f(x,u,p,\hat{p},m,t) \qquad t > t_0 \tag{1}$$

$$x(t_0) = x_0 \tag{2}$$

$$y = \tilde{x} \tag{3}$$

wobei B die Linke-Seite-Matrix, f der Rechte-Seite-Vektor und x, u, y, m, p und $\hat{p}$ die Zustands-, Eingangs-, Ausgangs-, binäre Schaltparameter-, stetige und unstetige Parameter-Vektoren sind; der Vektor $\tilde{x}$ umfaßt eine Untermenge der Komponenten von x; x_0 ist der Vektor der Anfangswerte der Anfangsbedingung (2) ; t steht für Zeit. Da die Matrix B singulär sein darf, beinhaltet das Differential-Algebra-Gleichungssystemen (1)-(3) eine große Klasse von Prozeßmodellen. In der Simulationsumgebung DIVA werden die Prozeßmodelle in dieser Standardformulierung als sogenannte DIVA-Grundelemente in einer Modellbibliothek abgelegt, siehe Abb. 1. Ein Grundelement besteht aus mehreren FORTRAN-Unterprogrammen, die vom Code-Generator aus einer Code-Generator-Eingabedatei automatisch codiert werden können. Die Gleichungen müssen in der Eingabedatei ebenfalls in der differential-algebraischen Standardform vorliegen. Als weiteres Bindeglied zwischen der Modellierung und der Simulation ist die symbolische Vorverarbeitung [18, 25] abgebildet. Die symbolische Vorverarbeitung ist notwendig zur Transformation von Apparate-Modellen, die nicht in der von DIVA und vom Code-Generator geforderten Standardform (1) vorliegen. Dazu dienen Programme zur Diskretisierung von örtlich- und eigenschaftsverteilten Modellen, zur Indexanalyse und Indexreduktion sowie zur DAE-Umformung. Die Programme der symbolischen Vorverarbeitung werden mit dem Computer-Algebra-System MATHEMATICA [26] implementiert und befinden sich derzeit in der Entwicklung. Sie sind aber mit eingeschränktem Funktionsumfang bereits benutzbar.

Bei der Herleitung eines Modells kann der Benutzer von dem Modellierungswerkzeug PROMOT [27, 28] unterstützt werden. PROMOT ist ein wissensbasiertes Rechnerwerkzeug zur gleichungsbasierten, objektorientierten Modellierung von verfahrenstechnischen Apparaten auf der Phasenebene. Die objektorientierte Wissensbasis von PROMOT enthält verfahrenstechnisches Modellierungswissen in Form von strukturellen und verhaltensbeschreibenden Modellbausteinen. Durch den Mechanismus der Aggregation können Apparatemodelle aus strukturellen und verhaltensbeschreibenden Modellbausteinen zusammengesetzt werden. Der Vererbungsmechanismus ermöglicht die Definition neuer Modellvarianten als Spezialfälle bereits existierender Modellbausteine. Eine Beschreibung der Modellierungssprache von PROMOT und eine Modellierungsanleitung findet man in [29].

1.2 Funktionalität des Code-Generators

Für die Erstellung von Apparate-Modellen (DIVA-Grundelementen) und deren Implementierung in der DIVA-Modellbibliothek müssen — wie in Abb. 2 dargestellt — verschiedene FORTRAN-Unterprogramme zur Initialisierung, Funktionsauswertung, Ereignisdefinition, Ausgabe und analytischen Jacobi-Matrizenberechnung erstellt werden [15]. Im Initialisierungs-Programmteil wird u. a. der benötigte Speicherplatzbedarf berechnet und die Pattern der Linke-Seite-Matrix B sowie der Jacobi-Matrizen

$$\frac{\partial(B\dot{x})}{\partial x}, \frac{\partial(B\dot{x})}{\partial u}, \frac{\partial f}{\partial x} \text{ und } \frac{\partial f}{\partial u} \tag{4}$$

in der speicherplatzsparenden „sparse-matrix"-Technik bestimmt. Das Funktionsauswertungs-Programm enthält die funktionalen Ausdrücke der Elemente der Linke-Seite-Matrix B und des Rechte-Seite-Vektors f. Der Programmteil zur Ereignisdefinition enthält Schaltfunktionen zur Festlegung

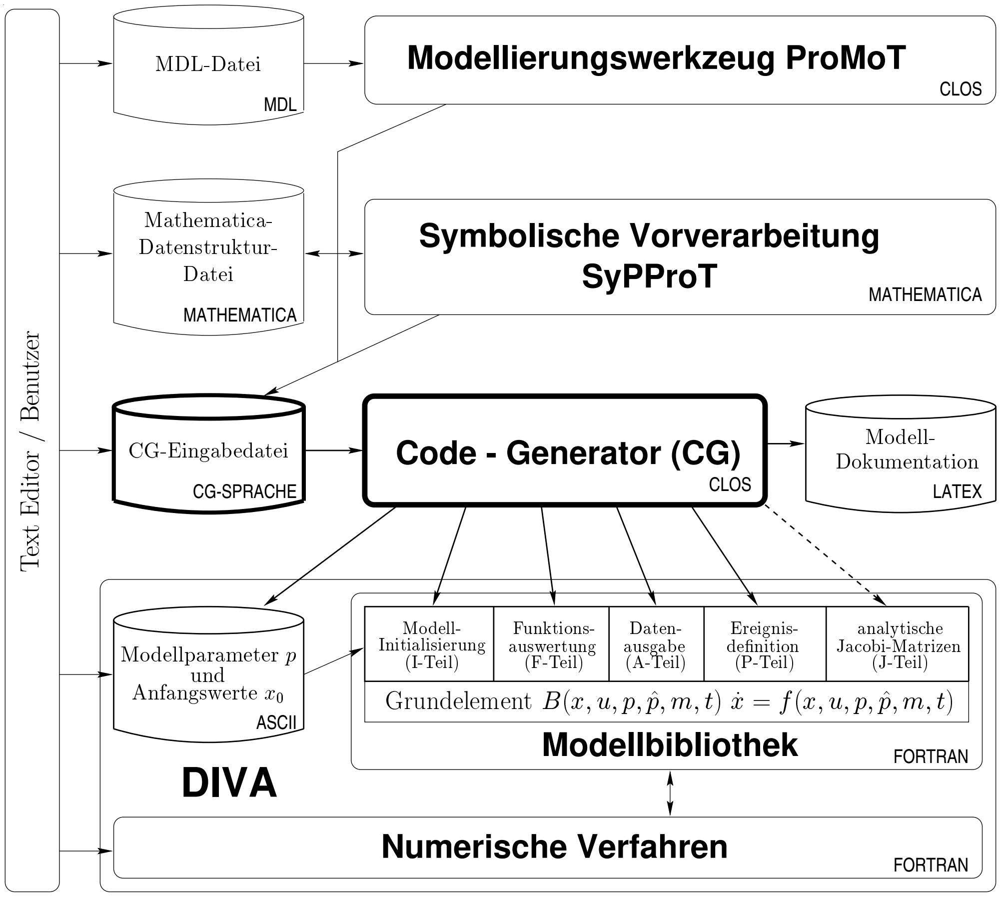


Abbildung 1: Architektur der Simulationsumgebung DIVA mit den numerischen Verfahren und der DIVA-Modellbibliothek, dem Code-Generator, dem Vorverarbeitungswerkzeug SYPPROT und dem Modellierungswerkzeug PROMOT.

von Modellumschaltungen. Der Ausgabe-Programmteil gibt ausgewählte Modellzustände und Modellgrößen für eine spätere Weiterverarbeitung, z. B. graphische Darstellung, aus. Alternativ zur Differenzen-Approximation der bei der numerischen Integration benötigten Differentialquotienten berechnet der optionale Programmteil zur analytischen Jacobi-Matrizenberechnung diese exakt. Zur Durchführung eines Simulationsexperimentes muß für jede Verwendung eines DIVA-Grundelements eine passende Datei (Blockdatenfile) erstellt werden. Darin stehen die Werte der Modellparameter und Anfangsbedingungen (Abb. 2). Seither mußten die FORTRAN-Unterprogramme „von Hand“ erstellt werden, was zeitraubend und fehleranfällig ist, da unter anderem Berechnungen des benötigten Speicherplatzes, Adreßrechnungen zur Positionsbestimmung von „Variablen“ in den DIVA-Arrays und Patternbestimmungen von vier Jacobi-Matrizen auszuführen sind. Außerdem wird durch die Repräsentation des Modellierungswissens in Form von FORTRAN-Programmen die Dokumentation, Wartung und Wiederverwendung von Prozeßmodellen wesentlich erschwert.

Der in Abb. 3 dargestellte Code-Generator soll den Benutzer bei der Implementierung von neuen Apparatemodellen zur Erweiterung der DIVA-Modellbibliothek unterstützen. Die Funktionalität des Code-Generators sieht vor, daß aus einer deklarativen Modellspezifikation, die als ASCII-File angegeben werden kann, automatisch die benötigten FORTRAN-Unterprogramme, ein Blockdatenfile und eine LATEX-Textdokumentation erzeugt werden. Die Modellspezifikation umfaßt neben den Differential-Algebra-Gleichungen (1) und Anfangswerten (2) ggfs. auch noch häufig vorkommende Teilausdrücke, um damit eine effiziente Funktionsauswertung und übersichtliche Modellformulierung zu realisieren. Zu jedem Parameter, Zustand, Eingang, Ausgang und jeder Modellgleichung gehört eine von dem Benutzer angegebene Textdokumentation (z. B. Temperatur der Flüssigphase auf Boden k). Die aktuelle Version des Code-Generators ermöglicht die Erzeugung der von DIVA benötigten FORTRAN-Programme für die Initialisierung, Funktions-Auswertung, Ansteuerung der Datenausgabe und Ereignisdefinition. Außerdem wird ein vollständig spezifiziertes Blockdatenfile generiert, der die Parameter und Startwerte für ein Simulationsexperiment enthält. Die Ansteuerung der im Simulator DIVA kodierten Stoffdatenkorrelationen wird bisher nicht unterstützt. Allerdings sollen zukünftig Stoffdatenkorrelationen von dem in Entwicklung befindlichen Modellierungswerkzeug [20, 21, 19, 30, 31] unterstützt und dem Code-Generator als explizite Gleichungen übergeben werden. Alternativ zum Text-Edieren der Modellspezifikationsdatei soll zukünftig eine graphische Bedienoberfläche zur Erstellung und Änderung der Modellspezifikation sowie zur Bedienung des Code-Generators eingesetzt werden können. Der Code-Generator ist zu Beginn der Entwicklung als Modul eines Präprozessors für die Simulationsumgebung DIVA konzipiert worden [25]. Es hat sich als zweckmäßig erwiesen die Codierung und die Funktionen des Präprozessors in getrennten Programmschichten zu realisieren. Dabei übernimmt die Symbolische Vorverarbeitung [17, 18, 32, 33] die Funktionen des Präprozessors zur Transformation eines Apparate-Modells in die von DIVA benötigte Standardform. Der Code-Generator [16, 17, 18] erzeugt aus dieser Standardform die FORTRAN-Unterprogramme für DIVA.

Der Code-Generator ist unter Verwendung der Konzepte der objektorientierten Programmierung in der Programmiersprache COMMON LISP [34] implementiert. Dabei wurde das Objektsystem CLOS (Common Lisp Object System) [35] verwendet, welches eine standardisierte objekt-orientierte Erweiterung der Programmiersprache LISP darstellt und in ANSI COMMON LISP[1] enthalten ist.

Kapitel 2 beschreibt den internen Aufbau und den Datenfluß im Code-Generator. Die Benutzung des Code-Generators durch Angabe einer Modellspezifikationsdatei und der genaue Aufbau dieser Modellspezifikation werden in Kapitel 3 erläutert. In Anhang A sind einige spezifische Ausdrücke in COMMON LISP zusammengestellt, eine ausführlichere Beschreibung der verwendeten Grammatikschreibweise (erweiterte Backus-Naur-Form) ist in Anhang B zu finden. Eine formale Definition der arithmetischen und logischen Ausdrücke, die im Eingabe-File des Code-Generators angegeben werden können, ist in Anhang C beschrieben. Anhang D erklärt den Aufruf des Code-Generators vom Betriebssystem. In Anhang E und F ist je ein Beispiel einer Modellspezifikationsdatei angeben.

[1]Seit 8. Dezember 1994 ist COMMON LISP offizieller ANSI Standard: ANSI X3.226:1994 American National Standard for Programming Language Common LISP (X3J13).

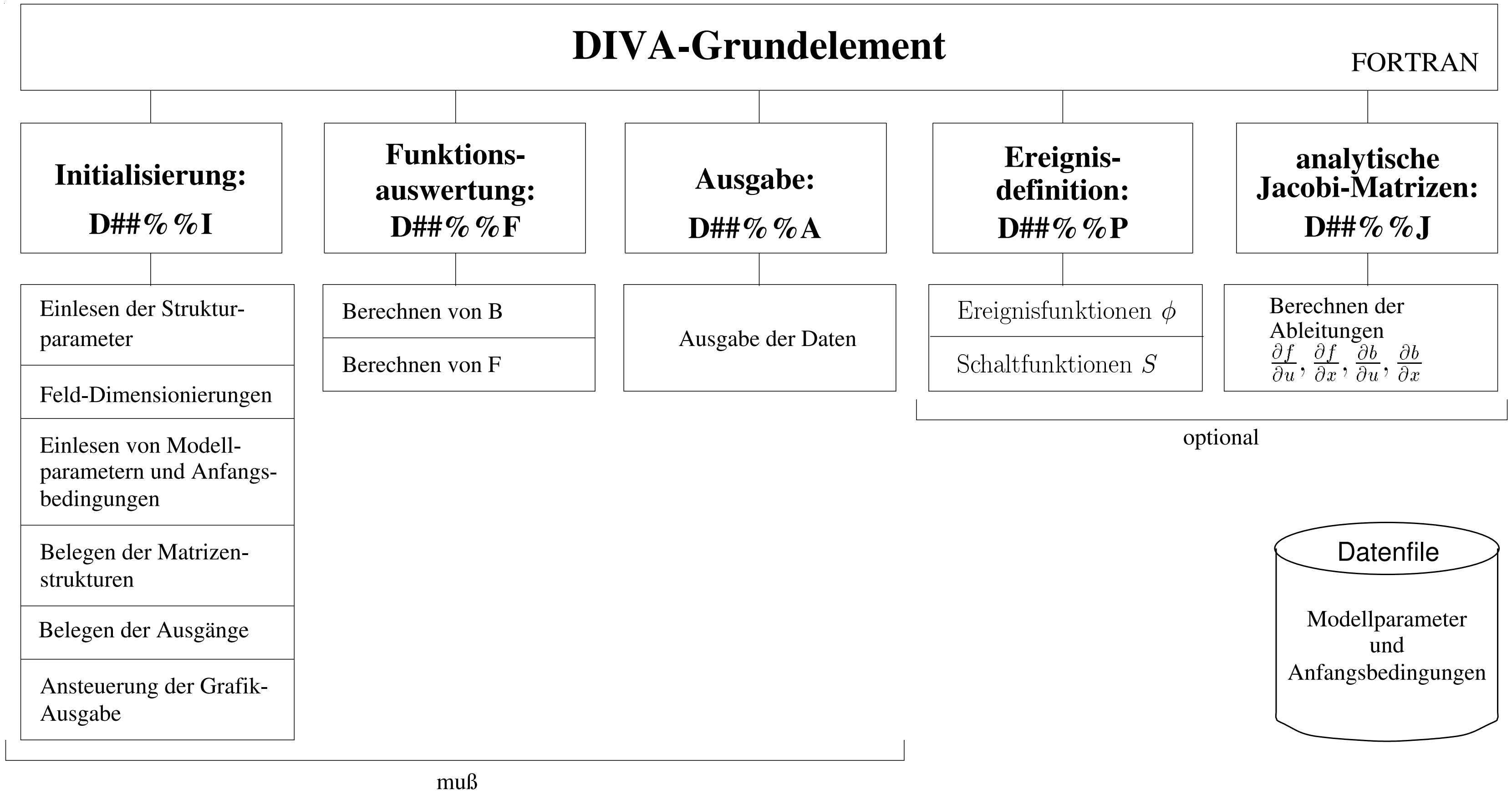


Abbildung 2: Implementierung eines DIVA-Grundelements in der DIVA-Modellbibliothek als FORTRAN-Unterprogramme zur Initialisierung, Funktionsauswertung, Ereignisdefinition, Ausgabe und analytischen Jacobi-Matrizenberechnung (Stoffdaten-Initialisierung und -Berechnungsanforderung sind nicht aufgeführt). Zur Verwendung eines DIVA-Grundelements in einem Simulationsexperiment gehört außerdem ein Datenfile mit den Werten der Modellparameter und Anfangsbedingungen [15].

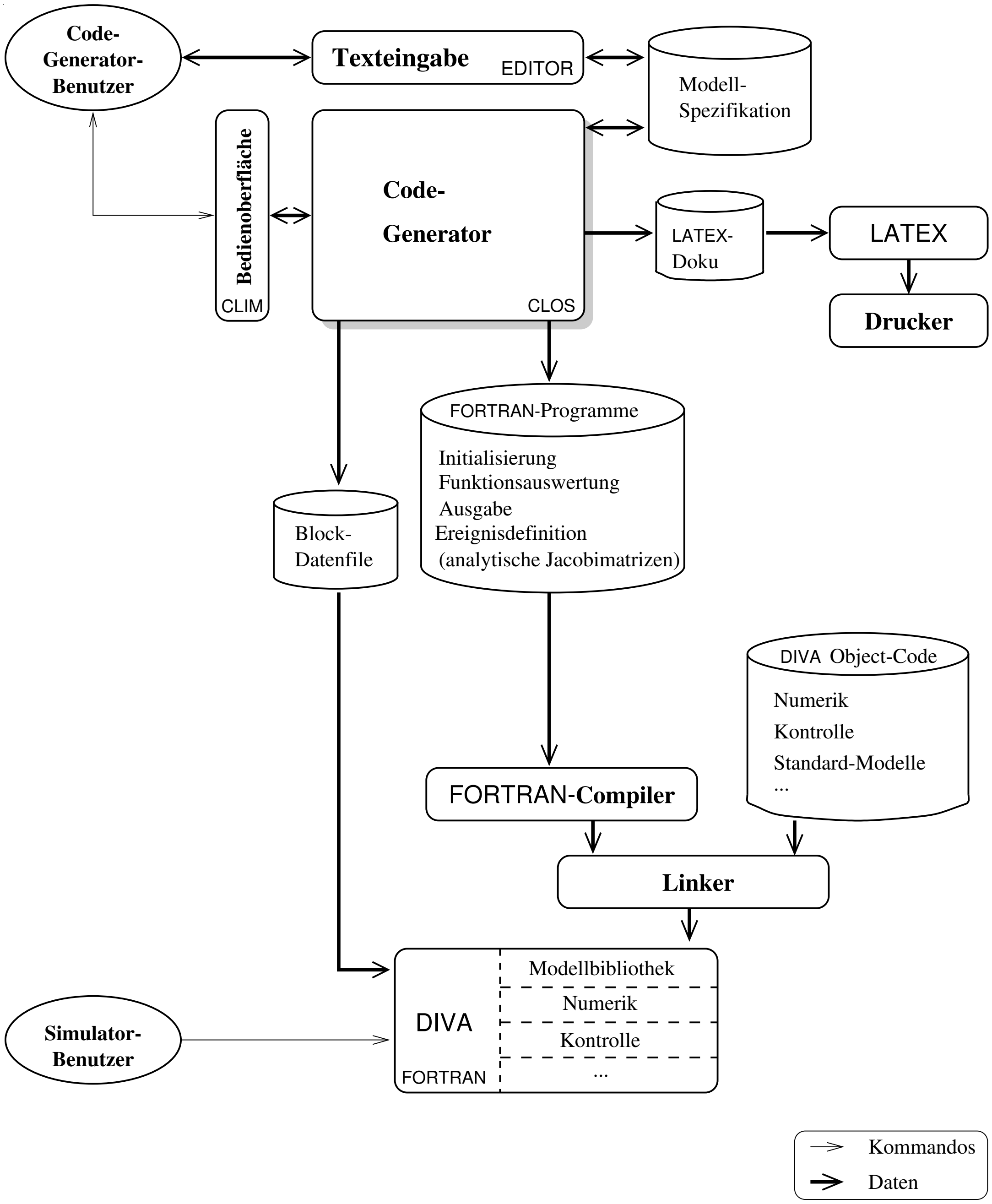


Abbildung 3: Datenfluß zwischen Code-Generator und Simulationsumgebung DIVA.

2 Interner Aufbau des Code-Generators

Dieses Kapitel beschreibt den internen Aufbau und den Datenfluß innerhalb des Code-Generators, soweit dies für den Modellentwickler von Bedeutung ist. Der Code-Generator ist unter Verwendung objekt-orientierter Konzepte programmiert. Dies zeigt sich z. B. in der Datenstruktur (Klassenhierarchie) und im Programmablauf (Versenden von Botschaften).

2.1 Datenstruktur

Als Datenstrukturen werden vor allem Klassen für Eingänge, Ausgänge, Zustände, Parameter, Gleichungen und B-Matrix-Einträge benutzt. Gemeinsame Eigenschaften (Attribute) dieser Klassen sind dabei in Oberklassen zusammengefaßt: Eingänge, Ausgänge und Zustände haben alle einen Namen und eine Position in einem FORTRAN-Array.

Zur Indexbeschreibung gibt es eine sogenannte Mixin-Klasse mit den Eigenschaften der Unter- und Obergrenze eines Index. Diese Mixin-Klasse wird zur Bildung neuer Unterklassen mit Hilfe der Mehrfachvererbung verwendet. Eine neue Klasse für Indizierte-Eingänge erbt von der Mixin-Klasse zur Indexbeschreibung und der Klasse für Eingänge. Analog entstehen neue Klassen für Indizierte-Ausgänge, Indizierte-Zustände, Indizierte-Parameter, Indizierte-Gleichungen und Indizierte-B-Matrix-Einträge.

Alle instantiierten Variablen und Parameter werden in geordneten Listen gespeichert und in eine Hash-Tabelle mit ihren Namen als Schlüssel eingetragen. Die Listen ermöglichen eine einfache Aufzählung aller Variablen und Parameter. Die Hash-Tabelle hilft bei der effizienten Zuordnung eines Namens (z. B. einer Variablen in einer Gleichung) zum zugehörigen Instanzobjekt.

2.2 Programmablauf

Zu Beginn werden alle Variablen- und Parameter-Deklarationen eingelesen. Dabei werden Initialisierungsmethoden benutzt, die einige Attribute, wie z. B. die Positionbestimmung in dem zugehörigen FORTRAN-Array, berechnen. Die Gleichungsdefinitionen werden gemeinsam eingelesen. Da die einzelnen bzw. in Schleifen zusammengefaßten Gleichungen aber sequentiell und unabhängig voneinander abgearbeitet werden, muß im Hauptspeicher immer nur eine einzelne Gleichung bzw. die Gleichungen einer Schleife gehalten werden. Dies hat den Vorteil, daß im Prinzip beliebig viele Gleichungen verarbeitet werden könnten.

Die sequentielle Behandlung der Modellgleichungen führt zu einer etwas anderen Reihenfolge der FORTRAN-Statements, als dies bei einer Programmierung „von Hand“ üblich ist.

- Bei der Programmierung „von Hand“ wird üblicherweise die Initialisierung aller Nicht-Null-Elemente der gesamten Linke-Seite-Matrix B programmiert. Ebenso geschieht die Initialisierung der Pattern der Jacobi-Matrizen (4) für jede Matrix blockweise mit den Anteilen **aller** Modellgleichungen.

- Der Code-Generator erzeugt pro Modellgleichung den Beitrag dieser Gleichung zur Initialisierung der Pattern der Linke-Seite-Matrix B und der Jacobi-Matrizen, sowie die Teilberechnung des Rechte-Seite-Vektors f und der Nicht-Null-Elemente der Linke-Seite-Matrix B. Ebenso wird gleichzeitig der für diese Gleichung benötigte Speicherplatzanteil der Matrizen-Informationen akkumulierende FORTRAN-Code erzeugt.

  Das gleichungsweise Vorgehen wird bei der Programmierung „von Hand“ normalerweise nur bei den Anteilen zur Berechnung des Rechte-Seite-Vektors f und der Nicht-Null-Elemente der Linke-Seite-Matrix B angewendet.

Die FORTRAN-Unterprogramme werden in einer quasi-parallelen Weise erzeugt. So kann identischer Code gleichzeitig auf mehr als ein FORTRAN-File ausgegeben werden und muß nicht ein weiteres Mal erzeugt werden. Beispielsweise wird identischer Code zur Verwaltung der Positionen von Variablen und Parametern in den DIVA-Arrays in den FORTRAN-Unterprogrammen zur Funktionsauswertung und Ausgabe benötigt. Das FORTRAN-Unterprogramm zur Initialisierung ist in einen Hauptteil und einen Include-Teil aufgeteilt. Der Include-Teil enthält Berechnungen zur Bestimmung des benötigten Speicherplatzes und jede Modellgleichung liefert hierzu einen Beitrag. Es ist natürlich auch möglich, die FORTRAN-Unterprogramme oder das Datenfile einzeln zu erzeugen. In diesem Fall werden die Programmteile, die die anderen Files erzeugen würden, nicht durchlaufen.

Alle Angaben arithmetischer Ausdrücke werden von der benutzerfreundlichen Infixnotation in die programmtechnisch besser verarbeitbare Präfixnotation vom Scanner und Parser umgewandelt. Der Scanner erkennt kleinere syntaktische Einheiten (lexikalische Elemente), wie z. B. numerische Konstanten, Variablennamen oder vordefinierte Funktionsnamen. Der Parser analysiert gemäß einer Grammatik die Struktur arithmetischer und logischer Ausdrücke, wie diese in Anhang C beschrieben sind. Dabei fordert der Parser bei Bedarf vom Scanner das nächste anstehende lexikalische Elemente an. Der Parser ist programmtechnisch nach dem Prinzip des rekursiven Abstiegs implementiert und kann optional während der syntaktischen Analyse gleichzeitig eine Typprüfung (Integer-, Real- oder Boolean-Typ) der Teilausdrücke vornehmen.

Die Ausgabe der FORTRAN-Statements erfolgt datengesteuert. Je nach Art der Operation eines Teil-Statements wird eine entsprechende Ausgaberoutine ausgeführt. Dabei werden Statements, die länger als 72 Zeichen sind, automatisch als Folgezeile (Continuation-Line) auf die nächste Zeile umgebrochen.

Gemäß DIVA-Konvention müssen Variablen oder Parameter in den FORTRAN-Unterprogrammen über Positionsangaben in vordefinierten DIVA-Arrays angesprochen werden [15]. Um diese arithmetischen Ausdrücke besser lesbar zu machen, wird jeweils eine Zeile vor dem zugehörigen FORTRAN-Code der arithmetische Ausdruck mit den ursprünglichen im Modell angegebenen Variablen- und Parameternamen als Kommentar angegeben.

2.3 FORTRAN-Ausgabe

Der erzeugte FORTRAN-Code hält sich streng an den FORTRAN 77 Standard mit den folgenden Ausnahmen:

1. Variablennamen werden nicht auf sechs Zeichen begrenzt, da dies sonst zu einem unleserlichen Code führen würde.
2. Momentan gibt es keinen Mechanismus, der verhindert, daß bei überlangen Statements mehr als 17 Folgezeilen (Continuation-Lines) auftreten.
3. Include-Befehle werden einmal im FORTRAN-Unterprogramm zur Initialisierung und mehrfach im FORTRAN-Unterprogramm zur Unstetigkeitsverwaltung verwendet.
4. Die implizite Typendeklaration von Variablen wird durch den Befehl `IMPLICIT NONE` ausgeschaltet. Dies verhindert die unabsichtliche Verwendung einer nicht explizit definierten Variablen.

2.4 Implementierungsphasen Version Beta.4 – Version 2.0

Nachfolgend werden die bisherigen Implementierungsphasen kurz beschrieben:

- Momentan umfaßt das Code-Generator-Programm etwa 29.000 COMMON LISP-Programmzeilen, etwa 5000 Zeilen Kommentare und ca. 1000 Zeilen FORTRAN-Text enthalten sind.
- Die character-parameters werden nicht unterstützt, da diese momentan keine spezifische Anwendung besitzen.

- Der key :write-as-infix der Funktion `cg-cg` ist ohne Wirkung. Es wird immer die Infix-Notation benutzt.

Änderungen der Code-Generator Version Beta.4 (Mitte März 1995) zur Version Beta.5 (Ende März 1995):

1. Die Namen von Parametern innerhalb von Prozeßgrößen werden jetzt richtig initialisiert.
2. Ein einfacher Syntax-Checker prüft die Modellspezifikation für den Code-Generator auf syntaktisch formale Korrektheit. Dieser Syntax-Checker wird jetzt vor dem Einlesen einer Modellspezifikationsdatei automatisch aufgerufen.

Änderungen zur Version Beta.6 (Mai 1995):

1. Ein früher bestehender Einfluß zwischen der Patternbelegung und der Gleichungs- und Zuweisungsformulierung wurde behoben. Jetzt wird eine Abhängigkeitsanalyse der in Hilfsgrößen eingehenden Variablen vorgenommen.
2. Die Patternbelegung wurde vollständig überarbeitet, und es gibt jetzt neue Arten von Hilfsgrößenzuweisungen in Schleifen, um z. B. bestimmte Größen über alle Diskretierungspunkte einer Eigenschaftskoordinate aufzusummieren, oder Größen, die an benachbarten Orstdiskretisierungspunkten liegen und schon berechnet wurden, wiederzuverwenden. Dies erlaubt eine wesentlich effizientere Auswertung der Modellgleichungen (s. Kapitel 3.6.3).

Änderungen zur Version 1.0 (August 1996):

1. Die Berechnung von Hilfsvariablen mittels Unterprogramm-Aufrufen ist jetzt möglich. Es wird die automatische Erzeugung von Programmköpfen, die zu den Unterprogramm-Aufrufen passen, sowie eine Bibliothek zur Beschreibung und einfachen Konsistenzprüfung der Programmparameter von vorhandenen externen Unterprogrammen unterstützt (s. Kapitel 3.6.4 und Steuerfunktion `cg-lib` in Kapitel 4.2).
2. Die Patternbelegung wird dahingehend optimiert, indem versucht wird, eine Variable, die an mehreren Stellen in einer Gleichung auftritt, auch nur einmal bei der Patternbelegung zu berücksichtigen. Eine Mehrfachbelegung kann im allgemeinen nicht völlig verhindert werden, wird aber in mehr Fällen als seither vermieden (s. Steuerfunktion `cg-fortran`, key `:pattern-opt-level` in Kapitel 4.2).

Änderungen zur Version 1.1 (März 1997):

1. Weitere Mechanismen der Patternoptimierung sorgen für eine zuätzliche Verringerung von Mehrfachbelegungen der Patterneintäge. Außerdem kann die Patternbelegung der Linke-Seite-Matrix B und des Funktionsvektors f nun auch von Hand vorgegeben werden. Dazu wurde der key `:provided-pattern` eingeführt. Die Belegung der Pattern von Hand ist nur in seltenen Ausnahmefällen bei starker Mehrfachbelegung sinnvoll und geschieht auf eigenes Risiko des Benutzers.
2. Durch die Angabe von Stutzendefinitionen mittels `(define-interfaces...)` kann die Eingabedatei des Code-Generators von Modellbeschreibungssprache EULER zur Verschaltung von DIVA-Grundelementen verwendet werden. Näheres hierzu ist einer momentan getrennnten Beschreibung zu entnehmen.
3. Für :default-values können jetzt auch array-Listen zur Definition von <u>mehrdimensionalen</u> Feldern verwendet werden. Die bisher notwendigen keys `:lower-boundary` und `:upper-boundary` einer array-Liste werden nicht mehr benötigt (s. Kapitel 3.3.4).

Änderungen zur Version 1.2 (Oktober 1997):

1. Unterstützung gemischt kontinuierlich-erignisdiskreter Modelle. Die Eingabe von expliziten und impliziten Events führt zur automatischen Generierung einer Schablone für das DIVA-Unterprogramm zur Unstetigkeitsverwaltung (P-Teil) . Außerdem werden das Blockdatenfile und das FORTRAN-Unterprogramm zur Initialisierung mit den ereignisspezifischen Informationen codiert (s. Kapitel 3.4).

2. Alle als Real angegebenen Zahlenwerte werden automatisch umformatiert und als Double ausgegeben.

Änderungen zur Version 1.3 (Februar 1999):

1. Vollständige Definition von Ereigniszeitpunkten expliziter und impliziter Events. Erweiterung der automatischen Codierung des DIVA-Unterprogramms zur Unstetigkeitsverwaltung (P-Teil). Ersetzung der Definitionsfunktion für Events (define-events) durch eine Definitionsfunktion für Eventzeitpunkte (define-event-dates) und eine noch nicht implementierte Definitionsfunktion des Schaltwerks (define-event-schedule) (s. Kapitel 3.4).

2. Patternoptimierung konditionaler Ausdrücke: Bei der Patternbelegung werden Abhängigkeit von Strukturparametern oder numerischen Werten berücksichtigt, wenn dem key `:pattern-opt-level` ein Wert ≥ 4 zugewiesen wird (s. Kapitel 4.2).

3. Vereinheitlichung der Definition von Index-Namen durch Verwendung des keys `:index-names` auch für Ausgangsvariablen (s. Kapitel 3.3.2).

4. Die FORTRAN-Unterprogramme werden mit der Endung „.f“ statt „.for“ erzeugt.

5. Implementierung der Fakultätsfunktion `!`.

Änderungen zur Version 1.3.1 (Dezember 1999):

1. Definition von Zeichen-Parametern (character-parameters) implementiert (s. Kapitel 3.3.1).

2. LaTeX-Dokumentation für Events erweitert.

3. Vereinheitliche Ausgabe der Reihenfolge von Pattern der Jacobi-Matrizen für verschiedene Lisp-Implementationen.

Änderungen zur Version 2.0 (September 2000):

1. Vollständige Untersützung gemischt kontinuierlich-ereignisdiskreter Modelle (s. Kapitel 3.4). Eingabe des Schaltwerks mit Hilfe der Definition von Petri-Netzen. Eine ausführliche Beschreibung dieser Erweiterung ist in [36] nachzulesen.

2. Geänderte interne Behandlung von `lambda`-Ausdrücken. Damit ist deren Verwendung auch für aktuelle C-Lisp Versionen möglich[2].

3. Geänderter FORTRAN-Code im I-Teil betreffend dem Einlesen des Blockdatenfiles lässt die Ausgabe von diesbezüglichen Fehlermeldungen zu[3].

[2]Dank an M. Ginkel, der den entsprechenden Patch beigesteuert hat.
[3]Dank an M. Mangold für die entsprechende Anregung

3 Modellspezifikationsdatei für den Code-Generator

Die benötigten Informationen über ein Modell werden dem Code-Generator in Form einer einzulesenden Modellspezifikationsdatei (CG-File) gegeben. In diesem Kapitel wird beschrieben, wie die Modellgleichungen und Variablendeklarationen in dem CG-File in einer COMMON LISP-orientierten Syntax angegeben werden müssen und wie die Ansteuerung des Code-Generators zur Erzeugung verschiedener Files erfolgt.

Die Modellspezifikation in dem CG-File erfolgt in folgender Reihenfolge, wobei manche Angaben optional sind, also weggelassen werden können:

1. Angaben zur Modellbeschreibung
2. Benennung der Integrationsvariablen (Zeit)
3. Angaben zu Struktur-, Integer-, Real- und Character-Parametern
4. Angaben zu Eingangs-, Zustands- und Ausgangs-Variablen
5. Angaben zu Events[4]
6. Angaben zur Ansteuerung der Datenausgabe
7. Angaben zu Modellgleichungen

Das folgende Beispiel für einen CG-File als Eingabe für den Code-Generator vermittelt einen ersten konkreten Eindruck der anzugebenden Informationen. Die volle Funktionalität bei der Informationsangabe ist in diesem Beispiel nicht ersichtlich, wird aber in den folgenden Abschnitten ausführlich beschrieben.

```
;;; Input-File for the Code-Generator

;;; 0.1 Documentation for head of FORTRAN programs
;;; ---------------------------------------------------------------------------
(define-header
  :program "pt2"
  :documentation "PT2-Glied"
  :date "25.10.93"
  :filename "pt2.cg")

;;; 0.2 Name of time variable
;;; ---------------------------------------------------------------------------
;; not needed



;;; 1. Parameters:
;;; ---------------------------------------------------------------------------

;;; 1.1 Structure Parameters:
;;; - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
;; not needed

;;; 1.2 Integer Parameters (FORTRAN vector RPAR):
```

[4] Dieser Abschnitt muss seit Version 2.0 vor den Angeben zur Ansteuerung der Datenausgabe stehen. Andernfalls wäre eine Ausgabe von Stellen und Eventfunktionen nicht möglich.

```
;;; - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
;; not needed

;;; 1.3 Real Parameters (FORTRAN vector IPAR):
;;; - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
(define-real-parameters
  (:name "a0" :meaning "Schwingungsparameter 0" :default-value 0)
  (:name "a1" :meaning "Schwingungsparameter 1" :default-value 1/2)
) ; end of real-parameters

;;; 1.4 Character Parameters (FORTRAN vector CPAR):
;;; - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
;; not needed



;;; 2. Model Inputs (FORTRAN vector U):
;;; -----------------------------------------------------------------------
(define-input-variables
   (:name "u1" :meaning "Eingang U")
) ;; end of input-variables



;;; 3. Model States (FORTRAN vector X):
;;; -----------------------------------------------------------------------
(define-state-variables
  (:name "x1" :meaning "is X" :default-state-value 0)
  (:name "x2" :meaning "is dX/dt" :default-state-value 1)
) ;; end of state-variables



;;; 4. Model Outputs (FORTRAN vector IOUT):
;;; -----------------------------------------------------------------------
(define-output-variables
  (:name "y" :meaning "is X" :is-state-variable-with-name "x1")
) ;; end of output-variables



;;; 5. Help Variables:
;;; -----------------------------------------------------------------------
;; not needed



;;; 7. Events:
;;; -----------------------------------------------------------------------
;; not needed



;;; 7. Process Values (grouping of data output):
;;; -----------------------------------------------------------------------
(define-process-values
```

```
  (:name "PROC1" ; maximum of 5 characters !!
   :meaning "first and second variable"
   :components ("x1" "x2")
  )
) ;; end of process-values



;;; 8. Model Equations
;;; ---------------------------------------------------------------------------
(define-equations
  (:equation-name "(1)"
   :meaning "
      first equation:
      (1) :diff(x1,t) = x2"
   :b-values ((:state-variable-reference "x1" :b-value 1))
   :f-value "x2"
  )
  (:equation-name "(2)"
   :meaning "
     second equation:
     (2) :diff(x2,t) = - a1 * x2 - a0 * x1 + u1"
   :b-values ((:state-variable-reference "x2" :b-value 1))
   :f-value "- a1 * x2 - a0 * x1 + u1"
  )
) ;; end of equations
```

Syntax-Notation

Folgende Syntax-Notation wird für die Beschreibungen in den folgenden Unterkapiteln verwendet:

- Drei Punkte (`...` `...`) deuten auf den vorangegangen (bzw. folgenden) Zusammenhang.
- Beliebig viele Symbole (bzw. mindestens ein Symbol), die durch Leerzeichen getrennt sind, werden durch geschweifte Klammern mit hochgesetzten Stern (bzw. Pluszeichen) beschrieben (z. B. $\{$`symbol1`$\}^*$ (bzw. ($\{$`symbol2`$\}^+$))).
- Optionale Ausdrücke werden in eckigen Klammern geschrieben ([`expression`])
- Alternativen werden durch einen senkrechten Strich(|) gekennzeichnet.

3.1 Angaben zur Modellbeschreibung

Am Anfang des CG-Files steht die Definition eines Headers. In diesem werden die allgemeine Filedaten wie Filename, Eingabedatum und Programmname sowie eventuelle Erläuterungen dokumentiert[5] Durch die Funktion (`define-header...`) läßt sich ein solcher Header definieren.

Beispiel für eine Header-Beschreibung im CG-File eines PT2-Gliedes:

```
(define-header
  :program "pt2"
  :documentation "PT2-Glied"
  :date "20.8.92"
  :filename "pt2.cg") ;end of header
```

[5]Außerdem können hier vom Benutzer programmierte include-Dateien für gemischt-kontinuierlich ereignisdiskrete Modell angegeben werden, die automatisch eingebunden werden sollen. Diese Option wird aber nur benötigt, wenn das Schaltwerk für ereignisdiskrete Modellumschaltungen von Hand codiert wird, wie es für die Code-Generator Version 1.3 erforderlich war. Bei der Eingabe des Schaltwerks mit Hilfe der Definition von Petri-Netzen (`define-event-schedule`) darf diese Option nicht verwendet werden.

Der prinzipielle Aufbau von solchen Definitionsfunktionen (deren Namen grundsätzlich mit `define-...` beginnt) orientiert sich an der Programmiersprache COMMON LISP. So werden entsprechend der COMMON LISP-Syntax *auszuwertende* Funktionen `(Fkt-Name arg1 arg2 ...)` mit den durch ein Leerzeichen getrennten Argumenten in Klammern geschrieben. Im Falle der `define-...` Funktionen werden die Argumente durch sogenannte keys (Schlüsselworte) gekennzeichnet. Solche keys beginnen immer mit einem Doppelpunkt und treten paarweise mit einem dazugehörigen Wert[6] (Funktionsargument) auf (key value pair). Die angegebenen Funktionsargumente werden somit durch die zugehörigen keys näher bezeichnet. Die Verwendung von keys birgt einige wesentliche Vorteile. Verschiedene key-Werte-Paare dürfen in beliebiger Reihenfolge innerhalb einer `define-..` Funktion angegeben werden. Bestimmte Angaben sind optional, das heißt, falls eine Information nicht benötigt wird, muß sie und der dazugehörige key nicht im `define-...` Aufruf angegeben werden (vgl. Tabelle 1). Es bleibt noch anzumerken, daß ein Semikolon (`;`) einen Kommentar definiert, der bis zum Zeilenende geht. Mehrere Kommentarzeilen werden zwischen `#||` und `||#` eingeschlossen.

3.2 Benennung der Integrationsvariablen (Zeit)

Kommt in den Modellgleichungen die Zeit explizit als Variable vor, so muß festgelegt werden, wie der Variablenname dafür heißen soll. Durch den Aufruf der Funktion `(define-time-variable :name ...)` wird der Variablenname definiert.

Beispiel für Benennung der Zeit mit dem Namen `SimTime`:

```
(define-time-variable :name "SimTime")
```

Kommt die Zeitvariable in den Modellgleichungen nicht explizit vor, so braucht die Benennung durch die Funktion `define-time-variable` nicht angegeben werden, und es wird ein Default-Name gewählt.

3.3 Angaben zu Variablen und Parametern

Die Variablen (variables) und Parameter (parameters) werden im folgenden mit variables-and-parameters[7] bezeichnet. Die Unterscheidung zwischen variables und parameters wurde so gewählt, daß unter dem Begriff parameters Größen beschrieben werden, die beim Erzeugen des FORTRAN-Programms in den Quelltext eingehen und dessen Aufbau mitgestalten (structur-parameters) oder unter DIVA konstante Werte beschreiben (real-parameter, integer-parameter). Im Gegensatz dazu werden mit dem Begriff variables unter DIVA veränderliche Daten verstanden (Modelleingänge (input-variables), Modellzustände (state-variables), Modellausgänge (output-variables) und Hilfsvariablen (help-variables), die als Implementierungsparameter verstanden werden können). Die Hilfsvariablen sind eine im Kontext der Code-Generation neu eingeführte Klassifizierung. Sie dienen zur übersichtlicheren Angabe von Modellgleichungen und für eine effiziente Berechnung von mehrfach auftretenden Teilausdrücken. (Weitere Beschreibung siehe Kapitel 3.3.2.)

Die Reihenfolge, in der die variables-and-parameters angegeben werden, bestimmt die Reihenfolge, in der sie in den entsprechenden FORTRAN-Feldern abgespeichert werden. Diese Reihenfolge ist z. B. bei den Eingangs- und Ausgangsvariablen für die Verschaltung von Apparaten zu Anlagen von Bedeutung.

Im allgemeinen sind sowohl (einfache) variables-and-parameters als auch (möglicherweise mehrfach) indizierte variables-and-parameters zugelassen. Für jeden Index einer indizierten variables-and-parameters ist die untere und obere Grenze des Laufindexes anzugeben. Die Grenzen dürfen mathematische Ausdrücke in Abhängigkeit von Strukturparametern sein.

Im CG-File werden die variables-and-parameters wie folgt definiert.

[6] Dieser Wert besteht grundsätzlich aus einem LISP-Ausdruck, z. B. aus einem String, einem Atom (einer Zahl, Variablen ...) oder einer Liste. Eine Liste wird durch runde Klammern gekennzeichnet und besteht aus unterschiedlichen, durch Leerzeichen getrennte Elementen: `(element1 element2 ...)`. Je nach der internen Weiterverarbeitung kann diese Liste ganz, teilweise (z. B. Macro) oder nicht (Programmdaten) ausgewertet werden. So ist z. B. die `(define-header ...)` Funktion nichts anderes als eine auszuwertende Liste. Für weitere Informationen vgl. Literatur zu COMMON LISP.

[7] Die Bezeichnung und Unterteilung von variables-and-parameters entsprechen der internen Klassifizierung von DIVA [4, 15].

3.3.1 Parameter (parameters)

Über die `(define-...-parameters)` Funktionen lassen sich die einzelnen parameters erzeugen und in die interne Tabelle der bekannten Namen von variables-and-parameters eintragen.

Strukturparameter

Strukturparameter dienen zur Dimensionierung von Parametern, Variablen und Modellgleichungen. Sie können während eines Simulationsexperiments nicht verändert werden.

(**define-structure-parameters**
{(**:name** ⟨name⟩
:meaning ⟨string⟩
:default-value ⟨expression⟩)}*
)

Zum Beispiel kann ein einfacher Strukturparameter folgendermaßen definiert sein :

```
(define-structure-parameters
  (:name "NC"
   :meaning "Anzahl der Komponenten"
   :default-value 3)
)
```

Dabei enthalten die entsprechenden keys folgende Informationen:[8]

:name ⟨name⟩

- Benennt den Parameter mit einem String. Wie bei allen Namen von variables und parameters wird nicht zwischen Groß- und Kleinschreibung unterschieden, da dies in den zu erzeugenden FORTRAN-Programmen ebenfalls unterbleibt. Beachten Sie bitte, daß die Zeichen `&, #` und `\` problematisch beim Übersetzen der LaTeX-Dokumentation sein können.

:meaning ⟨string⟩

- Dieser key wird von einem möglicherweise mehrzeiligen String gefolgt, der eine nähere Beschreibung enthält und zur Dokumentation benutzt wird. Beachten Sie bitte, daß die Zeichen `&, #` und `\` problematisch beim Übersetzen der LaTeX-Dokumentation sein können.

:default-value ⟨expression⟩

- Dieser Wert wird auf dem Datenfile dem Parameter zugeordnet. Spätere Änderungen dieses Wertes müssen auf dem Blockdatenfile selbst vorgenommen werden oder mit dem geänderten Wert muß die Datenfileerzeugung neu aufgerufen werden. Der Wert des default-values läßt sich auch in Abhängigkeit von Indizes oder anderen Parametern definieren (vgl. Kapitel 3.3.4).

Integer-, Real- und Character-Parameter

Die Syntax der Definitionsfunktionen und key-Wörter stimmen mit denen der Strukturparameter überein. Neben den im obigen Beispiel für Strukturparameter aufgeführten keys besitzen die Integer/Real/Character-Parameter noch einen weiteren key `:indices` für die Indizierung:

[8] Früher gab es zusätzlich den key **:process-value-info-name**, wobei **:process-value-info-name** ⟨name⟩ die physikalische Bedeutung der Größe innerhalb des Prozesses, d. h. dessen Gruppierung innerhalb des Modells, beschreibt (vgl. dazu das Prozeßgrößenkonzept von DIVA [15]). **ACHTUNG:** Der String ⟨name⟩ der diese genauere Beschreibung der Variablen oder Parameter erlaubt darf höchstens 5 Buchstaben umfassen. Es ist allerdings konzeptionell sinnvoller den Namen einer Prozeßgröße nicht mit dem **:process-value-info-name** zu bestimmen, sondern bei der Prozeßgröße selbst zu definieren (vgl. Kapitel 3.5).

(**define-integer-parameters** | **define-real-parameters** | **define-character-parameters**
{(**:name** ⟨name⟩
:meaning ⟨string⟩
:indices ({⟨index-description⟩}$^+$)
:default-value ⟨expression⟩)}*
)

⟨index-description⟩::=
(**:lower-boundary** ⟨lower-boundary⟩
:upper-boundary ⟨upper-boundary⟩)

Ein Beispiel zur Indizierung:

```
... (... :indices ((:lower-boundary 2 :upper-boundary "nc")) ))
```

Nach dem `key` `:indices` folgt eine Liste mit Angaben von Indexlaufbereichen. Ein solcher Indexlaufbereich ist wiederum eine Liste, die jetzt zwei `keys` beinhalten muß: Der eine `key` `:lower-boundary` bezeichnet die untere Laufgrenze des Indexes und der andere `key` `:upper-boundary` die obere Laufgrenze. Dabei können durchaus die Laufgrenzen variabel (in Abhängigkeit von Strukturparametern) gestaltet werden, wobei natürlich darauf zu achten ist, daß den Grenzen ganzzahlige Werte zugewiesen werden. So wird im obigen Beispiel der entsprechende Parameter von der unteren Grenze 2 bis zur oberen Grenze `nc` (Strukturparameter `NC`) indiziert. Durch das Erweitern der Index-Liste läßt sich eine beliebige Indizierung realisieren.

```
(define-integer-parameters
(... :indices ((:lower-boundary 2 :upper-boundary "nc")
               (:lower-boundary -10 :upper-boundary 20)) ))
```

Entsprechend der Reihenfolge der Index-Liste wird dem *ersten* Index der Bereich von 2 bis `nc` und dem *zweiten* Index der Bereich -10 bis 20 zugewiesen. Den jeweiligen Indizes wird explizit keine Laufvariable zugewiesen, doch durch die Reihenfolge der Index-Liste werden die einzelnen Indizes den Elementen eindeutig zugeordnet. So hätte z. B. das erste Element den Index `_2_-10` und nicht `_-10_2`[9].

Die Definitionsfunktion `(define-...-parameters (...))` hat bislang nur einen einzelnen Parameter mit seinen Attributen definiert. Beabsichtigt man, mehrere Parameter zu definieren, so muß man dies über einen einmaligen Aufruf der Definitionsfunktion tun:

```
(define-real-parameters
  (:name "a0"
   :meaning "Schwingungsparameter 0"
   :default-value 1)
  (:name "a1"
   :meaning "Schwingungsparameter 1"
   :default-value 1)
)
```

ACHTUNG: Ein und dieselbe `(define-...)` Funktion darf nicht zweimal hintereinander aufgerufen werden. Sollen die in einer `(define-...)` Funktion gemachten Angaben rückgängig gemacht werden, so kann die ursprüngliche Definition mit einer entsprechenden `(redefine-...)` Funktion überschrieben werden. Die parameters (oder variables) selbst werden entsprechend ihrer Reihenfolge in der `(define-... ...)` Funktion in den Code-Generator übernommen und später in derselben Reihenfolge in das entsprechende FORTRAN-Feld geschrieben.

[9] Im folgenden wird die interne Index-Schreibweise, die auch im Blockdatenfile erscheint, übernommen.

3.3.2 Variablen (variables)

Die Definition von variables geschieht analog zu der schon erläuterten Definition von parameters. Es ändern sich lediglich einige keys und der Name der Definitionsfunktion (`define-...-variables`).

Eingangsvariablen (input-variables)

Eingangsvariablen werden wie Parameter definiert, lediglich der key **:default-value** existiert nicht.

Zustandsvariablen (state-variables)

Zustandsvariablen (state-variables) haben zusätzliche Attribute, die im Zusammenhang zur Numerik stehen und auf das Blockdatenfile geschrieben werden. Folgende keys erlauben die Beschreibung dieser Attribute:

:default-state-value ⟨expression⟩

- Anfangswert, mit dem beim Simulatorstart die Zustandsvariable initialisiert wird. ⟨expression⟩ wird in Kapitel 3.3.4 näher beschrieben.

:default-min-value ⟨numerical-value⟩

- Erlaubter minimaler Wert.

:default-max-value ⟨numerical-value⟩

- Erlaubter maximaler Wert.

:default-numerical-error-value ⟨numerical-value⟩

- Maximaler numerischer absoluter Fehler.

  ACHTUNG: Bei der Verwendung indizierter Zustandsvariablen die für externe Unterprogrammaufrufe als Argumente übergeben werden ist der Hinweis auf Seite 61 zu beachten.

Ausgangsvariablen (output-variables)

Nach Gleichung (3) muß eine Ausgangsvariable einer Zustandsvariablen entsprechen. Die Definition der Ausgangsvariablen erfolgt mit dem zusätzlichen key **:is-state-variable-with-name**. Ist eine Ausgangsvariable indiziert, so werden zur Beschreibung der Indizes neben den oberen und unteren Laufgrenzen des keys :indices zusätzlich die dazugehörigen Laufvariablen mit dem key :index-names angegeben. In Abhängigkeit dieser Laufvariablen werden die einzelnen Komponenten des Ausgangsvektors angegeben:

```
(define-output-variables
  {(:name ⟨name⟩
    :meaning ⟨string⟩
    :indices ({⟨index-description⟩}+)
    :index-names ({⟨string⟩}+)
    :is-state-variable-with-name ⟨name⟩)}*
)
```

```
⟨index-description⟩::=
(:lower-boundary ⟨lower-boundary⟩
 :upper-boundary ⟨upper-boundary⟩)
```

Dabei bedeuten die keys:

:index-names ⟨index-name-list⟩

- Liste von Indexnamen, die durch Leerzeichen getrennt sind. Die Liste muß nur dann angegeben werden, wenn die Ausgangsvariable indiziert ist Für jeden Index der Ausgangsvariable muß genau ein Indexname angeben werden. Die Indexnamen können bei der Angabe von **:is-state-variable-with-name** benutzt werden. Beachten Sie bitte, daß die Zeichen `&`, `#` und \ problematisch beim Übersetzen der LaTeX-Dokumentation sein können.

:is-state-variable-with-name ⟨name⟩

- Name der auszugebenden Zustandsvariablen.

  Beispiel:

  Die Konzentrationen eines örtlich eindimensional verteilten Systems sollen den Ausgangsvektor des Blocks bilden. Dabei ist die Konzentration `x` zweifach indiziert. Der erste Index beschreibt die Komponente und der zweite die örtliche Stützstelle. Der structure-parameter `NC` (bzw. `ND`) beschreibt die Anzahl der auftretenden Komponenten (bzw. der Stützstellen).

  Da man den output-variables nur state-variables zuweisen kann, müssen zuvor die Konzentrationen `x` als solche definiert worden sein.

```
(define-output-variables
  (:name "x_out"
   :meaning "beschreibt die Konzentrationen aller Komponenten an jeder
             Stuetzstelle"
   :indices ((:lower-boundary 1 :upper-boundary "ND")
             (:lower-boundary 1 :upper-boundary "NC"))
   :index-names ("i" "j")
   :is-state-variable-with-name "x[j,i]"     ; Beide Indizes werden durchlaufen
                                             ; (x[1,1], x[2,1], x[3,1],...).
  )
  (:name "xpart_out"
   :meaning "gibt die Konzentration der Komponente NC an jeder Stuetzstelle an"
   :indices ((:lower-boundary 1 :upper-boundary "ND"))
   :index-names ("i")
   :is-state-variable-with-name "x[NC,i]"    ; Der erste Index ist fest.
  )
  (:name "xonevalue_out"
   :meaning "gibt die Konzentration der Komponente NC an der
             2. Stuetzstelle an"
   :is-state-variable-with-name "x[NC,2]"    ; Beide Indizes sind fest.
  )
  (:name "x1konst_out"
   :meaning "gibt die Konzentration der Komponente NC-1 an der
             3. Stuetzstelle 11 mal an"
   :indices ((:lower-boundary 0 :upper-boundary "5+5"))
   :index-name ("i")
   :is-state-variable-with-name "x[NC-1,3]" ; Beide Indizes sind fest.
  )
)
```

Hilfsvariablen (help-variables)

Hilfssvariablen (help-variables) dienen dazu, Werte von Teilausdrücken aufzunehmen, und werden einerseits für eine effiziente Funktionsauswertung mehrfach vorkommender Teilausdrücke, andererseits für eine übersichtliche Formulierung von komplizierten Funktionen benötigt. Folgende keys sind zusätzlich erlaubt:

:index-names ⟨index-name-list⟩

- Liste von Indexnamen, die durch Leerzeichen getrennt sind. Die Liste muß nur dann angegeben werden, wenn die Hilfsvariable indiziert ist und der key **:value** ebenfalls angegeben ist. In diesem Falle muß für jeden Index der Hilfsvariable genau ein Indexname angeben werden. Die Indexnamen können bei der Angabe von **:value** benutzt werden.

:value ⟨expression⟩

- Der angegebene Ausdruck wird zu Beginn eines Simulationsexperiments der Hilfsvariable zugewiesen. Der Ausdruck sollte zeitinvariant sein. Während des Simulationsexperiments muß dann dieser Ausdruck nicht wiederholt ausgewertet werden.

  ACHTUNG: Diese Art von Hilfsgrößenwerten sind wie die Modellparameter konstant, können aber nicht während eines Simulationsexperiments wie die Integer- oder Realparameter verändert werden. Außerdem muß davon ausgegangen werden, daß der Wert einer Hilfsgröße, deren **:value** von einem Parameter p abhängt, nicht aktualisiert wird, wenn sich der Wert von p während eines Simulationsexperiments ändern sollte. Zuweisungen von zeitabhängigen Berechnungen an Hilfsvariablen werden bei den Simulationsgleichungen angegeben und in Abschnitt 3.6.3 beschrieben.

  Soll der Hilfsvariablen ein zeitvarianter Ausdruck zugewiesen werden hat dies im Gleichungsabschnitt zu erfolgen, siehe Kapitel 3.6.3.

  Beispiel:
  Der Hilfsvariable `h_ge` werden erst bei den Gleichungsdefinitionen Werte zugewiesen. `Pi` wird im FORTRAN-Programm mit der Maschinengenauigkeit berechnet. `xsi0` wird in Abhängigkeit der im Blockdatenfile angegebenen Werte der Parameter `delta_pn` und `Fn` berechnet. Die indizierte Hilfsvariable `z` wird folgendermaßen berechnet:

$$z[i] = (i-1)dz \qquad \text{für } i = 1, ..., NP$$

  Eine nur teilweise Belegung, z. B. nur $z[2]$, ist nicht möglich, da sonst die Patternbelegung sehr viel schwieriger werden würde.

```
(define-help-variables
  (:name "h_ge"
   :meaning "molare Gemischenthalpie"
  )
  (:name "Pi"
   :meaning "Kreiskonstante"
   :value "4.0 * :arctan(1.0)"
   )
  (:name "xsi0"
   :meaning "Druckverlustbeiwert"
   :value "0.1*delta_pn/Fn"
  )
  (:name "z"
   :meaning "Orte der Ortsstuetzstellen"
   :indices ((:lower-boundary 1 :upper-boundary "NP"))
   :index-names ("i")
   :value "(i-1)*dz"
  )
)
```

  ACHTUNG: Der Wert der Hilfsvariablen, wie z. B. `xsi0`, wird nur zu Beginn einer Simulationssitzung berechnet. Deshalb ist eine interaktive Änderung des Parameterwertes von `delta_pn` während eines Simulationsexperiments ohne Auswirkung auf den Wert der Hilfsvariablen `xsi0`.

3.3.3 Übersicht der Schlüsselworte zu Parameter und Variablen

Zum Überblick sind in Tabelle 1 zusammenfassend die zu den entsprechenden variables und parameters gehörigen keys zusammengestellt.

PARAMETER	SCHLÜSSELWORTE (KEYs)			
structure	*:name*	*:meaning*		:default-value
integer	*:name*	*:meaning*	:indices	:default-value
real	*:name*	*:meaning*	:indices	:default-value
character	*:name*	*:meaning*	:indices	:default-value
VARIABLEN				
input	*:name*	*:meaning*	:indices	
state	*:name*	*:meaning*	:indices	:default-state-value
				:default-min-value
				:default-max-value
				:default-numerical-error-value
output	*:name*	*:meaning*	:indices	:index-names
				:is-state-variable-with-name
help	*:name*	*:meaning*	:indices	:index-names
				:value

Tabelle 1: Schlüsselworte (keys) zu Variablen und Parameter. Nicht alle keys müssen verwendet oder angegeben werden. Die zwingend *erforderlichen* keys sind *kursiv* geschrieben.

Bei einer Definition müssen nicht alle keys verwendet oder angegeben werden. In der Tabelle sind die zwingend *erforderlichen* keys *kursiv* geschrieben. Auf jeden Fall müssen die keys `:name`, `:meaning` und (im Falle von output-variables) `:is-state-variable` mit einem Wert angegeben werden. Die default-values werden beim Fehlen eines Wertes meist zu 1 gesetzt.

3.3.4 Parameterwerte und Anfangswerte von Zustandsvariablen

Der Wert `:default-value` ist je nach Definition des Parameters eine Integer-Zahl (integer-parameters), eine Real-Zahl (real-parameters, state-variables ...) oder eine Zeichenkette bzw. String (character-parameters). Der Anfangswert `:default-state-value` einer Zustandsvariablen ist eine Real-Zahl.

ACHTUNG: Für character-parameters darf die Zeichenkette keine Leer- oder Sonderzeichen enthalten.

Für indizierte Integerparameter, Realparameter oder Zustandsvariablen ist es wünschenswert, die Werte in Abhängigkeit der Indizes explizit oder gar als Berechnungsvorschrift vorzugeben. Dies ermöglicht zum Beispiel die Angabe des Anfangsprofils eines örtlich verteilten Zustands in kurzer, aber von Strukturparametern abhängiger Form.

Folgende Möglichkeiten für die Angabe der Werte bei `:default-value` oder `:default-state-value` stehen zur Auswahl:

- Eine Möglichkeit ist ein sogenannter lambda-Ausdruck. Dieser Funktional-Ausdruck berechnet in Abhängigkeit von Indizes oder anderen Parametern den gewünschten Wert. Ein solcher lambda-Ausdruck besteht aus einer Liste mit drei Elementen:
  :default-value (**lambda** ($\{\langle$symbols$\rangle\}^+$) $\langle$expression$\rangle$)

  lambda

  - Kennzeichnet die Liste als einen Funktional-Ausdruck.

  $\{\langle$symbols$\rangle\}^+$

  - Die darauf folgende Liste (sog. Argumentliste) beschreibt (*hier*) die Indizes, die beim lambda-Aufruf in den arithmetischen Ausdruck eingehen. Die Symbole dieser Liste sind frei wählbar, wobei jedoch darauf zu achten ist, daß die Zahl der Symbole in der Liste mit der Anzahl von Indizes des Parameters übereinstimmt. So wird dem ersten Symbol der Liste der erste Index zugewiesen... usw.

  $\langle$expression$\rangle$

- Dies ist der arithmetische Ausdruck (sog. Funktionsrumpf), der den `:default-value` in Abhängigkeit von Parametern und Indizes berechnet. Wie in allen arithmetischen Ausdrücken, die in COMMON LISP geschrieben werden, muß man auch hier darauf achten, daß man die Präfix-Notation[10] verwendet.

  Im Funktionsrumpf können (vorher definierte) Parameter verwendet werden, die wiederum durch ihren `:default-value` ersetzt werden. Allerdings sind nur nicht-indizierte Parameter mit einfachen arithmetischen Ausdrücken (kein lambda-/array-list- oder spline-Ausdruck) als `:default-value` zulässig.

  Beispiel:

  Der `:default-value` eines zweifach indizierten integer-parameters (a0) soll aus der Summe von `PAR` und der Differenz der Indizes berechnet werden. Dabei soll `PAR` den Wert eines vorher definierten integer-parameters `PAR` beschreiben. Der Parameter `a0` selbst hat zwei Indices, deshalb steht im lambda-Ausdruck eine zweielementige Liste mit den hier verwendeten, frei wählbaren Indexnamen `i` und `j`. Die letzte Liste des lambda-Ausdrucks enthält die mathematische Berechnung des `:default-value` in Präfix-Notation.

```
(define-real-parameters
  (:name "a0"
   :meaning "Schwingungsparameter 0"
   :indices ((:lower-boundary 2 :upper-boundary 5)
             (:lower-boundary -1 :upper-boundary 3))
   :default-value (lambda ("i" "j") (+ "PAR" (- "i" "j")))
  )
)
```

  Wenn der Parameter `PAR` den `:default-value` 100 besitzt, dann hätte der `:default-value` des Parameters `a0_2_-1` den Wert $100 + (2 - (-1)) = 103$, der `:default-value` des Parameters `a0_5_3` hätte den Wert $100 + (5 - 3) = 102$.

  Wenn eine lokale Indexvariable in der Argumentliste in doppelten Anführungszeichen (") eingeschlossen steht, so sollte sie auch im Funktionsrumpf in die Anführungszeichen eingeschlossen stehen. Steht sie ohne Anführungszeichen in der Argumentliste, so darf sie im Funktionsrumpf nur ohne Anführungszeichen stehen:

```
   (lambda ("i" j) (+ "i" j "PAR"))
```

  Zur Belegung einer Matrix mit verschiedenen konstanten Werten kann folgender lambda-Ausdruck nützlich sein:

```
(define-real-parameters
  (:name "aij"
   :meaning "wilson interaction parameters [K]"
   :indices ((:lower-boundary 1 :upper-boundary 2)
             (:lower-boundary 1 :upper-boundary 4))
   :default-value (lambda (i j) (nth (- j 1) (nth (- i 1)
      '( (0.0     -130.65  1749.93  2.03)
         (101.65  0.0      288.20   198.17)
       ) )))
  )
)
```

  Hierbei erhält z. B. `aij[1,2]` den Wert -130.65.

  Bei skalaren (nicht-indizierten) Variablen oder Parametern wird beim `:default-value` einfach die Präfix-Notation der mathematischen Berechnungsvorschrift angegeben. Wäre der Parameter `a0` skalar und der `default-value` sollte das Minimum des Parameters `PAR` und 4.1 zugewiesen bekommen, würde der lambda-Ausdruck folgendermaßen lauten:

[10] Vgl. Anhang A.

```
(define-real-parameters
  (:name "a0"
   :meaning "Schwingungsparameter 0"
   :default-value (min "PAR" 4.1)
  )
)
```

- Statt obigem lambda-Ausdruck ist auch eine benutzerfreundlichere — nicht mit derselben Vielfalt wie bei COMMON LISP versehene — Funktionsangabe möglich. Dazu muß ein String mit dem Schlüsselwort function gefolgt von den in eckigen Klammern eingeschlossenen und durch Komma getrennten Funktionsargumenten angegeben werden. Nach der schließenden eckigen Klammer folgt der Funktionsausdruck.

  Beispiel:
  Obige vier Beispiele können auch folgendermaßen geschrieben werden:

```
   ...
   :default-value "function[i,j] PAR + i - j"
   ...
   "function[i,j] i + j + PAR"
   ...
   :default-value
     "function[i,j]
       :cond(i==1, :cond(j==1, 0.0,    j==2, -130.65, j==3, 1749.93, 2.03),
                   :cond(j==1, 101.65, j==2, 0.0,     j==3, 288.20,  198.17))"
   ...
   :default-value ":min(PAR,4.1)"
   ...
```

  Weiteres Beispiel:
  Die Zustandsvariable c beschreibt die Konzentration von NC örtlich eindimensional verteilten Komponenten an NP Ortsstützstellen. Der :default-state-value von c wird nur in Abhängigkeit der Komponente i, aber konstant für alle Ortsstützstellen j belegt. Die dritte Komponente erhält z. B. den Wert 0.33 die vierte und jede weitere Komponente — falls vorhanden — erhält den Wert 0.0.

```
(define-state-variables
  (:name "c"
   :meaning "c[i,j]: Konzentration der Komponente i an der Ortsstuetzstelle j"
   :indices ((:lower-boundary 1 :upper-boundary "NC")
             (:lower-boundary 1 :upper-boundary "NP"))
   :default-state-value "function[i,j]
     :cond (i==1, 0.1,
            i==2, 0.2,
            i==3, 0.33,
            0.0)"
   :default-min-value 0.0
   :default-max-value 1.0
   :default-numerical-error-value 1.0e-6
  )
)
```

- Man kann nun aber auch den :default-value von indizierten parameters-and-variables durch Felder definieren. In der aufsteigenden Reihenfolge einer Wertliste werden den entsprechenden indizierten Variablen oder Parametern die Werte zugewiesen. Ein mehrdimensionales Feld wird durch eine Werteliste bestehend aus mehreren entsprechend verschachtelten weiteren Wertelisten definiert.

  Für die Zuweisung von Feldern muß bei dem :default-value eine sogenannte array-Liste angegeben werden:

(**array-list**
  :values ({⟨expression⟩}$^+$)
)

array-list

- Kennzeichnet die Liste als eine array-Liste.

:values ({⟨expression⟩}$^+$)

- Auf diesen key folgt die Werteliste bzw. die Wertelisten, die aus arithmetischen Ausdrücken (meist nur Zahlen) besteht.

  Beispiel:
  Angenommen, der drei-dimensionale Parameter PAR der Dimension 3x2x3 ist folgendermaßen definiert:

```
(:name "PAR"
 :meaning "3-dim array-list"
 :indices ((:lower-boundary 1 :upper-boundary 3)
           (:lower-boundary 3 :upper-boundary 4)
           (:lower-boundary 3 :upper-boundary 5))
 :default-value
    (array-list :values (((111 112 113) (121 122 123))
                         ((211 212 213) (221 222 223))
                         ((311 312 313) (321 322 323))) )
)
```

  Entsprechend der Indexdefinition enthält die dem key `:values` nachfolgende Werteliste des `:default-value` drei Wertelisten, die jeweils aus zwei Wertelisten mit je drei Werten bestehen. So hat beispielsweise `PAR_1_3_3` den `:default-value` 111, `PAR_2_4_4` den `:default-value` 222 und `PAR_3_4_5` den `:default-value` 323. Die Werte dieses Beispiels sind so gewählt, daß die einzelnen Ziffern den Positionen innerhalb der 3x2x3-Matrix entsprechen.
  Wichtig ist, daß die Längen der einzelnen Wertelisten mit der entsprechenden Indexdimension des indizierten Parameters übereinstimmen.

- Die Funktion `spline` ermöglicht es den default-value einer einfach indizierten Variablen (bzw. eines Parameters) zu interpolieren.

  (**spline**
    :start ⟨value-description⟩
    :values ({⟨value-description⟩}$^+$)
    :end ⟨value-description⟩
)

  ⟨value-description⟩::=
  (⟨strucpar-expression⟩ ⟨expression⟩)

Die keys **:start**, **:values** und **:end** definieren die Wertepaare, die zur Interpolation nötig sind. Ein solches Wertepaar wird durch eine zweielementige Liste (vgl. ⟨value-description⟩) beschrieben. Hierbei ist ⟨strucpar-expression⟩ ein arithmetischer Ausdruck, der nur von Strukturparametern abhängen darf. Das erste Element der Liste ⟨value-description⟩ beschreibt die "Index-Koordinate", das zweite Element den dazugehörigen Wert. Die "Index-Koordinate" bezieht sich auf einen kontinuierlichen Index, muß also nicht ganzzahlig sein.

  Beispiel:

Die Anfangswerte eines örtlich verteilten Systems mit einer Ortskoordinate sollen durch ein Profil beschrieben werden. Falls zur Beschreibung des Profils Meßwerte zur Verfügung stehen, berechnet sich die "Index-Koordinate" aus der Lage der Meßstelle und der Diskretisierung. Wird z. B. ein Meter in 10 (bzw. 40) äquidistante Sützstellen unterteilt, so beträgt die "Index-Koordinate" für den ersten Meter 10 (bzw. 40) und für 25cm folglich 2.5 (bzw. 10). Das heißt der fiktive Parameter $p_{2.5}$ (bzw. p_{10}) würde den Meßwert an der Stelle 25cm bezeichnen.

Die örtlichen Anfangs- und Endwertpaare werden gesondert durch den key **:start** und **:end** angegeben. Die anderen Wertepaare müssen in aufsteigender Reihenfolge zu einer Liste zusammengefaßt und dem key **:values** zugewiesen werden.

Folgende Eingabe:

```
(define-real-parameters
  (:name "p" :meaning "koeff"
   :default-value (spline :start (0 1)
                          :values ((2 -3) (3.5 -6) (4 0))
                          :end (6 1) )
   :indices ((:lower-boundary 0 :upper-boundary 5))
  )
)
```

bewirkt im Blockdatenfile folgende Ausgabe:

```
...

C**********************************************************************
C
C   Real-Parameter
C   Anzahl: 6
REAL_PARAMETER
  6                 #berechneter Wert
C
  1.0               #p_0
  0.34202766        #p_1
  -3.0              #p_2
  -7.882874         #p_3
  0.0               #p_4
  4.5762796         #p_5
C
...
```

Dem Parameter p_0 wird 1, p_2 wird -3, und p_4 wird 0 zugewiesen, wie es die Definition verlangt. Die Wert von p_1, p_3 und p_5 werden interpoliert.

3.4 Angaben zu Events

Events oder Ereignisse treten zu ausgezeichneten Zeitpunkten auf und beschreiben Änderungen des Modells hinsichtlich der Parameterwerte, der Eingangsgrößen oder der Modellgleichungen. Ausgelöst werden Events durch innere Unstetigkeiten oder Unstetigkeiten durch Interaktion mit der Umgebung. Inner Unstetigkeiten sind beispielsweise physikalische Effekte, wie die Änderung des Strömungszustands. Typische Unstetigkeiten durch Interaktion mit der Umgebung treten bei Prozeßführungen auf, beispielsweise bei Ventilansteuerungen. Die veränderten Parameter- und/oder Eingangswerte und Umschaltungen der Modellgleichungen können zu Unstetigkeiten der Modellzustände führen, die ihrerseits Probleme für die numerischen Lösung darstellen. Aus diesem Grund ist eine spezielle numerische Behandlung von Events erforderlich.

3.4.1 Gemischt kontinuierlich-ereignisdiskrete Modelle in DIVA

Mit den in DIVA zur Verfügung stehenden numerischen Integrationsverfahren können nur kontinuierliche DAE-Systeme simuliert werden. Für Prozess-Modelle mit unstetigem Verhalten zu bestimmten Zeitpunkten (Ereignissen, Events) existiert zu diesem Zweck ein integratorunabhängiges Konzept zur Behandlung von ereignisdiskreten Vorgängen [37].

Ereignisdiskrete Vorgänge in einem Anlagenteil können im Blockmodell enthalten sein oder durch Unstetigkeiten im Eingangsvektor hervorgerufen werden. Da die Ereignisbehandlung bei der Simulation auf Anlagenebene erfolgt, d.h. zentral für alle verschalteten Blockmodelle, spielt diese Unterscheidung im folgenden keine Rolle. Es sind jedoch nur ereignisdiskrete Vorgänge zugelassen, die keine strukturellen Unstetigkeiten in den Modellen verursachen.

DAE-Modell mit unstetigen Anteilen

Für die Programmierung gemischt kontinuierlich-ereignisdiskreter Apparate-Modelle in DIVA muss das Gleichungssystem in der linear-impliziten DAE-Form gemäß Gleichung (1) vorliegen. Eine DIVA-spezifische Erweiterung sind sogenannte Hilfsgrößen $\boldsymbol{h}$ zur Abkürzung wiederholt auftretender Terme in den Elementen der $\boldsymbol{B}$-Matrix und des $\boldsymbol{f}$-Vektors. Dabei wird zwischen zeitvarianten und zeitinvarianten Hilfsgrößen unterschieden, siehe Kapitel 3.3.2.

Zeitvariante Hilfsgrößen lassen sich wie Elemente der $\boldsymbol{B}$-Matrix und des $\boldsymbol{f}$-Vektors darstellen. Auf die Darstellung der $\boldsymbol{B}$-Matrix und $\boldsymbol{f}$-Vektor Elemente wird noch näher eingegangen.

Zeitinvariante Hilfsgrößen sind dagegen wie Modellparameter zu betrachten. Analog zu unstetigen Modellparametern $\hat{\boldsymbol{p}}$ werden im folgenden mit $\hat{\boldsymbol{h}}$ unstetige zeitinvariante Hilfsgrößen bezeichnet, deren Werte in Abhängigkeit der Schaltparameter m_i ausgewählt werden.

Ein gemischt kontinuierlich-ereignisdiskretes Modell in DIVA wir mit folgenden Gleichungen beschrieben:

$$\boldsymbol{B}(\boldsymbol{x},\boldsymbol{u},\boldsymbol{h},\hat{\boldsymbol{h}},\boldsymbol{p},\hat{\boldsymbol{p}},\boldsymbol{m},t)\;\dot{\boldsymbol{x}} = \boldsymbol{f}(\boldsymbol{x},\boldsymbol{u},\boldsymbol{h},\hat{\boldsymbol{h}},\boldsymbol{p},\hat{\boldsymbol{p}},\boldsymbol{m},t) \quad t > t_0\ ,\ \boldsymbol{x}(t_0) = \boldsymbol{x}_0 \tag{5}$$

mit

$$m_i \in \{0,1\}\ ,\ i = 1(1)M \qquad \text{binäre Schaltparameter} \tag{6}$$

$$\hat{h}_i = \hat{h}_i(\boldsymbol{m})\ ,\ i = 1(1)H \qquad \text{unstetige zeitinvariante Hilfsvariablen} \tag{7}$$

$$\hat{p}_i = \hat{p}_i(\boldsymbol{m})\ ,\ i = 1(1)P \qquad \text{unstetige Modellparameter} \tag{8}$$

wobei

$$\hat{h}_i = \hat{h}_{i,k} \quad \text{falls} \quad m_k = 1\ \wedge\ m_l = 0\ \ \forall l \in K_h \text{ mit } l \neq k \quad \text{für} \ \ k \in K_h \subseteq \{1,2,\ldots,M\} \tag{9}$$

$$\hat{p}_i = \hat{p}_{i,k} \quad \text{falls} \quad m_k = 1\ \wedge\ m_l = 0\ \ \forall l \in K_p \text{ mit } l \neq k \quad \text{für} \ \ k \in K_p \subseteq \{1,2,\ldots,M\} \tag{10}$$

Das bedeuted, dass für eine Teilmenge der Schaltparameter m_k mit $k \in K \subseteq \{1,2,\ldots,M\}$ ein unstetiger Parameter $\hat{p}_i$ verschiedene Werte $\hat{p}_{i,k}$ annimmt, je nachdem, welcher der Schaltparameter m_k gerade den Wert 1 hat. Analoges gilt auch für unstetige zeitinvariante Hilfsvariablen.

Die Elemente von $\boldsymbol{B}$ bzw. $\boldsymbol{f}$ lassen sich aufteilen in stetige Anteile $\bar{b}_{ij}$ bzw. $\bar{f}_i$ und unstetige Anteile $\hat{b}_{ij}$ bzw. $\hat{f}_i$.

$$b_{ij} = \bar{b}_{ij}(\boldsymbol{x},\boldsymbol{u},\boldsymbol{h},\boldsymbol{p},t) + \hat{b}_{ij}(\boldsymbol{x},\boldsymbol{u},\boldsymbol{h},\hat{\boldsymbol{h}},\boldsymbol{p},\hat{\boldsymbol{p}},\boldsymbol{m},t)\ ,\quad i,j = 1(1)n \tag{11}$$

$$f_i = \bar{f}_i(\boldsymbol{x},\boldsymbol{u},\boldsymbol{h},\boldsymbol{p},t) + \hat{f}_i(\boldsymbol{x},\boldsymbol{u},\boldsymbol{h},\hat{\boldsymbol{h}},\boldsymbol{p},\hat{\boldsymbol{p}},\boldsymbol{m},t)\ ,\quad i = 1(1)n \tag{12}$$

wobei

$$\hat{b}_{ij}(\boldsymbol{x},\boldsymbol{u},\boldsymbol{h},\hat{\boldsymbol{h}},\boldsymbol{p},\hat{\boldsymbol{p}},\boldsymbol{m},t) = \begin{cases} \hat{b}_{ij,1}(\boldsymbol{x},\boldsymbol{u},\boldsymbol{h},\hat{\boldsymbol{h}},\boldsymbol{p},\hat{\boldsymbol{p}},t) & \text{falls } \underline{\text{Bedingung}_{b,1}} \\ \hat{b}_{ij,2}(\boldsymbol{x},\boldsymbol{u},\boldsymbol{h},\hat{\boldsymbol{h}},\boldsymbol{p},\hat{\boldsymbol{p}},t) & \text{falls } \underline{\text{Bedingung}_{b,2}} \\ \ldots & \\ \hat{b}_{ij,L}(\boldsymbol{x},\boldsymbol{u},\boldsymbol{h},\hat{\boldsymbol{h}},\boldsymbol{p},\hat{\boldsymbol{p}},t) & \text{falls } \underline{\text{Bedingung}_{b,L}} \end{cases} \tag{13}$$

bzw.

$$\hat{f}_i(\boldsymbol{x},\boldsymbol{u},\boldsymbol{h},\hat{\boldsymbol{h}},\boldsymbol{p},\hat{\boldsymbol{p}},\boldsymbol{m},t) = \begin{cases} \hat{f}_{i,1}(\boldsymbol{x},\boldsymbol{u},\boldsymbol{h},\hat{\boldsymbol{h}},\boldsymbol{p},\hat{\boldsymbol{p}},t) & \text{falls } \underline{\text{Bedingung}}_{f,1} \\ \hat{f}_{i,2}(\boldsymbol{x},\boldsymbol{u},\boldsymbol{h},\hat{\boldsymbol{h}},\boldsymbol{p},\hat{\boldsymbol{p}},t) & \text{falls } \underline{\text{Bedingung}}_{f,2} \\ \ldots & \\ \hat{f}_{i,L}(\boldsymbol{x},\boldsymbol{u},\boldsymbol{h},\hat{\boldsymbol{h}},\boldsymbol{p},\hat{\boldsymbol{p}},t) & \text{falls } \underline{\text{Bedingung}}_{f,L} \end{cases} \tag{14}$$

mit beliebigen Wertekombinationen (Bitmuster) der Schaltparameter m_i als Bedingungen, beispielsweise

$$\begin{aligned} \underline{\text{Bedingung}}_{b,1} &= (m_4 = 1) \wedge (m_5 = 1) \wedge (m_6 = 0) \\ \underline{\text{Bedingung}}_{b,2} &= (m_4 = 0) \wedge (m_5 = 1) \wedge (m_6 = 1) \\ \ldots & \\ \underline{\text{Bedingung}}_{b,K} &= (m_4 = 0) \wedge (m_5 = 0) \wedge (m_6 = 1) \end{aligned}$$

Die Bedingungen $\underline{\text{Bedingung}}_{b,k}$ bzw. $\underline{\text{Bedingung}}_{f,k}$ für $k = 1(1)K$ müssen sich jedoch gegenseitig ausschließen, d.h. es muss immer jeweils genau eine dieser Bedingungen zutreffen, so dass eine eindeutige Auswahl erfolgen kann.

Die Unstetigkeiten in den unstetigen Modellparametern $\hat{p}_i$ und in den unstetigen Anteilen $\hat{b}_{ij}$ und $\hat{f}_i$ in den Modellgleichungen sind also zurückzuführen auf diskrete Änderungen an den Werten der Schaltparameter m_i zu Ereigniszeitpunkten t_{ev}.

Diese Schaltzeitpunkte $t_{ev} > t_0$ werden mit Hilfe von E Schalteingangs- oder φ-Funktionen

$$\varphi_e = \varphi_e(\boldsymbol{x},\dot{\boldsymbol{x}},\boldsymbol{u},\boldsymbol{p},t)\ ,\ e = 1(1)E \tag{15}$$

durch den Nulldurchgang einer φ-Funktion definiert:

$$\begin{aligned} t_{ev}: \quad & \varphi_e[\boldsymbol{x}(t_{ev}),\dot{\boldsymbol{x}}(t_{ev}),\boldsymbol{u}(t_{ev}),\boldsymbol{h},\hat{\boldsymbol{h}},\boldsymbol{p},\hat{\boldsymbol{p}},t_{ev}] = 0 \\ \wedge \quad & sgn[\varphi_e(t_{ev}^-)] \neq sgn[\varphi_e(t_{ev}^+)]\ ,\ e \in \{1,2,\ldots E\} \end{aligned} \tag{16}$$

Die Änderungen an den Schaltparametern m_i werden durch Schaltfunktionen $\boldsymbol{S}$ bestimmt, die in Abhängigkeit von den Werten $\boldsymbol{m}(t_{ev}^-)$ der Schaltparameter vor einem Ereignis und den eingetretenen Vorzeichenwechseln der φ-Funktionen (15) neue Werte $\boldsymbol{m}(t_{ev}^+)$ für die Schaltparameter bestimmen:

$$\boldsymbol{m}(t_{ev}^+) = \boldsymbol{S}[\boldsymbol{m}(t_{ev}^-),\varphi_1^+,\varphi_1^-,\varphi_2^+,\varphi_2^-,\ldots,\varphi_E^+,\varphi_E^-] \tag{17}$$

$$\boldsymbol{m}(t_0) = \boldsymbol{m}_0 \tag{18}$$

Hier bezeichnet φ_e^+ wiederum einen Vorzeichenwechsel der Eventfunktion φ_e von $(-)$ nach $(+)$. Die Anfangswerte $\boldsymbol{m}(t_0) = \boldsymbol{m}_0$ für die Schaltparameter sind vorzugeben.

Definition des Schaltzeitpunkts

Der Schaltzeitpunkt t_{ev} eines ereignisdiskreten Vorgangs wird entsprechend Gleichung (16) durch den Vorzeichenwechsel einer Schalteingangs- oder φ-Funktion (15) definiert. Je nach Struktur dieser φ-Funktion (15) unterscheidet man zwischen impliziten und expliziten Ereignissen.

- **Implizite Ereignisse**

  Die φ-Funktion eines <u>impliziten Ereignisses</u> (<u>state event</u>) hängt von den Zuständen oder den Eingängen ab und ist damit eine mittelbare Funktion der Zeit. Der Schaltzeitpunkt t_{ev} ist also nicht explizit bekannt, sondern muss während der Simulation durch wiederholtes Auswerten der φ-Funktion und Erkennen von Nulldurchgängen bestimmt werden.

  Man unterscheidet weiterhin zwischen unidirektionalen und bidirektionalen impliziten Ereignissen. Während <u>unidirektionale</u> implizite Ereignisse nur bei einem Vorzeichenwechsel der φ-Funktion in eine Richtung stattfinden, kann bei <u>bidirektionalen</u> impliziten Ereignissen ein beliebiger Vorzeichenwechsel einen Schaltvorgang auslösen. In DIVA finden unidirektionale Ereignisse nur bei einem Vorzeichenwechsel von $(+)$ nach $(-)$ statt.

  Üblicherweise werden bidirektionalen Ereignissen für reversible Unstetigkeiten und unidirektionalen Ereignisse für irreversible Unstetigkeiten verwendet.

- **Explizite Ereignisse**

  Streng genommen sind explizite Ereignisse (time events) nur ein Sonderfall impliziter Ereignisse, deren φ-Funktionen (15) folgende Struktur besitzen:

$$\varphi = t_{ev} - t \tag{19}$$

  Der Schaltzeitpunkt t_{ev} ist damit explizit bekannt, so dass die Angabe des Schaltzeitpunkts t_{ev} alleine gleichwertig mit der Definition einer φ-Funktion wie in Gleichung (19) ist.

  Bei expliziten Ereignissen unterscheidet man nach [38] und [39] weiterhin zwischen unmittelbaren oder exogenen Ereignissen, deren Schaltzeitpunkt unabhängig von anderen Schaltvorgängen angegeben werden kann, und mittelbaren oder endogenen Ereignissen. Bei letzteren ergibt sich der Schaltzeitpunkt erst während der Simulation in Abhängigkeit von anderen Schaltvorgängen.

  Eine Unterscheidung zwischen impliziten und expliziten Ereignissen ist wichtig für eine effektive Darstellung und Berechnung von Schaltzeitpunkten, da für explizite Ereignisse die aufwendige Erkennung und Bestimmung von Schaltzeitpunkten nicht erforderlich ist. Aus diesem Grund sieht die Simulationsumgebung DIVA für implizite und explizite Ereignisse eine getrennte Definition und Behandlung vor.

Behandlung von Ereignissen in DIVA

Da die numerische Integration gewisse Stetigkeitsanforderungen an das zugrundeliegende DAE-System stellt, muss die Integration bei Eintreten eines Ereignisses unterbrochen werden, um unstetige Vorgänge auszuführen. Es wird also genau bis zum Ereigniszeitpunkt t_{ev} integriert, dann ein Schaltvorgang ausgeführt, anschließend konsistente Anfangsbedingungen für das veränderte Modell berechnet (Abgleich) und erst dann weiterintegriert.

Für ein solches Vorgehen ist es notwendig, dass die Ereigniszeitpunkte t_{ev} rechtzeitig erkannt werden, damit die Integrationschritte an diese Zeitpunkte angepasst werden können. Während die Zeitpunkte expliziter Ereignisse bekannt sind, müssen die Zeitpunkte impliziter Events durch wiederholtes Auswerten der Schalteingangsfunktionen (φ-Funktionen) erkannt und genau bestimmt werden.

Das allgemeine Vorgehen der Ereignisverwaltung in DIVA lässt sich in drei Schritte aufteilen:

1. Erkennung von Ereignissen
2. Bestimmung des Schaltzeitpunkts t_{ev}
3. Ausführung des Schaltvorgangs

Dabei fallen für explizite Ereignisse die ersten beiden Schritte weg. Stattdessen muss nur der jeweils nächste explizite Ereigniszeitpunkt t_{ex} bestimmt werden, so dass parallel zur Erkennung impliziter Ereignisse höchstens bis zum nächsten expliziten Ereigniszeitpunkt t_{ex} integriert wird.

Im folgenden werden die drei Schritte zur Behandlung impliziter Ereignisse einzeln erläutert.

zu 1. **Erkennung von Ereignissen**

Die φ-Funktionen (15), durch deren Vorzeichenwechsel implizite Ereignisse definiert sind, werden nach jedem Integrationsschritt ausgewertet. Durch Vergleich mit dem zuletzt berechneten Wert werden Vorzeichenwechsel erkannt.

zu 2. **Bestimmung des Schaltzeitpunkts t_{ev}**

Durch einen iterativen Suchalgorithmus wird das Intervall, in dem der erkannte Vorzeichenwechsel liegt, verkleinert. Es handelt sich also um eine Näherungslösung. Der Zeitpunkt t_{ev} des Ereignisses gilt als hinreichend genau bestimmt, wenn der Betrag der φ-Funktion einen vom Benutzer definierten Toleranzwert ϵ_φ nicht überschreitet und die φ-Funktion ihr Vorzeichen gegenüber dem letzten Integrationsschritt geändert hat:

$$|\varphi(t_{ev})| \leq \epsilon_\varphi \text{ und } \mathrm{sgn}[\varphi(t_{k-1})] \neq \mathrm{sgn}[\varphi(t_{ev})] \tag{20}$$

zu 3. **Ausführung des Schaltvorgangs**

Ist die Integration am Schaltzeitpunkt angelangt, so werden für alle Ereignisse, die zu diesem Zeitpunkt stattfinden, die zugehörigen Schaltaktionen ausgeführt, d.h. es werden die Schaltfunktionen $\boldsymbol{S}$ aus (17) ausgewertet. Für die Ereignisse gelten folgende Bedingungen:

- Ein implizites Ereignis findet zum Schaltzeitpunkt t_{ev} statt, wenn die φ-Funktion das Schaltintervall entsprechend Bedingung (20) erfüllt.
- Ein explizites Ereignis findet zum Schaltzeitpunkt t_{ev} statt, wenn der gegebene Ereigniszeitpunkt t_{ex} innerhalb von einem bestimmten Maschinenungenauigkeitsintervall um den augenblicklichen Simulationszeitpunkt liegt.

Zunächst muss also festgestellt werden, welche Ereignisse zum augenblicklichen Zeitpunkt aktiviert sind, d.h. stattfinden und somit einen Schaltvorgang verursachen dürfen. Anschließend werden die folgenden Einzelschritte ausgeführt:

1. Die Aktionen der aktivierten Ereignisse werden ausgeführt, also entsprechend den Schaltfunktionen $\boldsymbol{S}$ die Werte der Schaltparameter $\boldsymbol{m}$ geändert. Anschließend werden diese Ereignisse wieder deaktiviert.
2. Falls $\mathbf{x}(t)$ und $\dot{\mathbf{x}}(t)$ das veränderte DAE-System nicht mehr erfüllen, wird ein Abgleich zur Bestimmung konsistenter Anfangsbedingungen für die weitere Integration vorgenommen.
3. Die Schaltaktionen oder der Abgleich können weitere Vorzeichenwechsel in den φ-Funktionen verursachen und damit weitere implizite Ereignisse aktivieren. In diesem Fall spricht man von statischem Schalten, da der Vorzeichenwechsel nicht während der Integration entstanden ist. Für diese Ereignisse werden die Schritte 1. bis 3. erneut ausgeführt.

Die drei Schritte werden so lange wiederholt, bis sich keine weiteren Vorzeichenwechsel mehr in den φ-Funktion ergeben. Um einen eventuellen deadlock zu vermeiden, wird das Verfahren nach einer bestimmten Zahl von Wiederholungen abgebrochen. Somit liegt nach einem Schaltvorgang ein neues Anfangswert-Problem für das DAE-System vor.

Programmtechnische Realisierung von Grundelementen mit Ereignissen

Die von den Schaltparametern $\boldsymbol{m}$ abhängigen Elemente b_{ij} der linken und f_i der rechten Seite des DAE-Systems (5) werden entsprechend (11) bis (14) im F-Teil definiert. Ebenso werden optional die von den Schaltparametern $\boldsymbol{m}$ abhängigen analytischen Ausdrücke für die Jacobi-Matrix-Einträge in den J-Teil codiert. In einem zusätzlichen FORTRAN-Unterprogramm, dem sogenannten P-Teil, werden die Ereigniszeiten entsprechend (16) oder durch Angabe expliziter Ereigniszeitpunkte t_{ex} definiert, sowie das Schaltwerk bestehend aus den Schaltfunktionen (17) und der Initialisierung (18) zum Zeitpunkt t_0. Außerdem werden im P-Teil die Werte der unstetigen zeitinvarianten Hilfsgrößen $\hat{\boldsymbol{h}}$ und der unstetigen Modellparameter $\hat{\boldsymbol{p}}$ in Abhängigkeit der Schaltparameter m_i entsprechend (9) bzw. (10) definiert. Im einzelnen hat der P-Teil also folgende Aufgaben zu erfüllen:

- Berechnung der φ-Funktionswerte (15)
- Bestimmung der nächsten expliziten Ereigniszeit t_{ex}
- Zuweisung der Werte der unstetigen zeitinvarianten Hilfsgrößen $\hat{\boldsymbol{h}}$ (9) und der unstetigen Modellparameter $\hat{\boldsymbol{p}}$ (10)
- Auswertung der Schaltfunktionen (17) zu Ereigniszeitpunkten t_{ev}
- Initialisierung des Schaltwerks nach (18) bei Simulationsstart und Wiederaufnahme der Integration

Je nachdem, wann der P-Teil von der DIVA-Ereignisverwaltung aufgerufen wird, hat er unterschiedliche der oben genannten Aufgaben zu erfüllen. Aus diesem Grund gibt es sogenannte Schaltflags, die anzeigen, ob zum Beispiel ein Ereignis eingetreten ist oder eine Initialisierung stattfinden soll, und somit das Verhalten des P-Teils bestimmen. Für jedes implizite Ereignis ist auf dem Statusfeld SWITCH ein Eintrag vorgesehen, für das jeweils nächste explizite Ereignis existiert die Statusvariable TTFLAG.

Die DIVA-Ereignisverwaltung ruft nach jedem Integrationschritt den P-Teil auf, der je nach Werten der Schaltflags verschiedene Aufgaben zu erfüllen hat. Man unterscheidet dabei drei Aufrufarten:

1. „Auswerten“: alle SWITCH-Einträge und TTFLAG haben den Wert 2
2. „Initialisieren“: alle SWITCH-Einträge haben den Wert 0, TTFLAG den Wert 3
3. „Schalten“: mindestens ein SWITCH-Eintrag oder TTFLAG hat den Wert 3

Ein Aufruf der Art „Auswerten“ erfolgt unmittelbar nach jedem Integrationsschritt. Hierbei sind sämtliche φ-Funktionen (15) auszuwerten und die nächste explizite Ereigniszeit t_{ex} zu bestimmen. Der nächste explizite Ereigniszeitpunkt t_{ex} wird auf der globalen Variable EVETIM abgelegt. Zudem werden die Werte der unstetigen zeitinvarianten Hilfsgrößen $\hat{\boldsymbol{h}}$ und der unstetigen Modellparameter $\hat{\boldsymbol{p}}$ in Abhängigkeit der Schaltparameter m_i zugewiesen.

Die Aufrufart „Initialisieren“ tritt beim Beginn einer Simulation auf oder bei Wiederaufnahme der Integration, falls zuvor auf der DIVA-Kommandoebene Parameter oder Zustandsgrößen verändert worden sind und dadurch ein Abgleich des Simulationsmodells erforderlich ist.

In diesem Modus erfolgt neben der Auswertung der φ-Funktionen, der nächsten expliziten Ereigniszeit und der Werte der unstetigen zeitinvarianten Hilfsgrößen und Modellparameter auch die Initialisierung des Schaltwerks, indem die Anfangswerte der Schaltparameter $\boldsymbol{m}(t_0)$ entsprechend (18) zugewiesen werden.

Die Ereignisverwaltung erkennt und bestimmt während der Integration anhand der Werte der φ-Funktionen (15) und EVETIM auftretende Ereignisse und steuert die Integration entprechend. Falls ein Ereignis stattfindet, leitet sie einen Schaltvorgang ein, wie er in Abschnitt 3.4.1 beschrieben wurde. Zur Ausführung aller notwendigen Schaltaktionen ruft sie den P-Teil im Modus „Schalten“ auf. Bei dieser Aufrufart werden wiederum die Werte aller φ-Funktionen (15), der nächste explizite Ereigniszeitpunkt t_{ex} sowie die Werte der unstetigen zeitinvarianten Hilfsgrößen und Modellparameter bestimmt. Zusätzlich werden aber auch in Abhängigkeit der Schaltflags die Schaltaktionen aller aktivierten Ereignisse ausgeführt, indem die Schaltfunktionen (17) ausgewertet werden. Dabei findet ein implizites Ereignis statt, falls dessen SWITCH-Eintrag den Wert 3 hat, und ein explizites Ereignis, falls TTFLAG den Wert 3 hat und die explizite Ereigniszeit t_{ex} nahe genug an der aktuellen Simulationszeit liegt.

Die möglichen Werte der Schaltflags und deren Bedeutung für die zugehörigen Ereignisse sind in nachfolgender Tabelle zusammengefasst.

Flag	Wert	Bedeutung
SWITCH	0	unmittelbar nach Neustart oder Wiederaufnahme der Integration zur Initialisierung des Schaltwerks
	2	implizites Ereignis darf nicht schalten
	3	zugehöriges implizites Ereignis darf schalten
TTFLAG	2	kein Schalten expliziter Ereignisse
	3	alle expliziten Ereignisse, die nahe genug an der Simulationszeit liegen, dürfen schalten

Tabelle 2: Mögliche Werte auf dem Statusfeld SWITCH mit Einträgen für jedes implizite Ereignis und der Statusvariable TTFLAG für die nächste explizite Ereigniszeit sowie deren Bedeutungen

Das Zusammenspiel der DIVA-Ereignisverwaltung mit den FORTRAN-Unterprogrammen zur Ereignisdefinition (P-Teil) und Funktionsauswertung (F-Teil), sowie mit den Numerikprogrammen zum Abgleich und Integrieren ist in Abbildung 4 schematisch dargestellt.

Für eine ausführliche Beschreibung der DIVA-Eventverwaltung wird auf [15, 37] verwiesen.

3.4.2 Petri-Netze zur Modellierung des diskreten Teilsystems

Insbesondere bei Prozess-Steuerungen hat man es in der Regel mit sehr komplexen Strukturen im Schaltwerk zu tun: als Basiskonstrukte treten Sequenzen, Nebenläufigkeiten und Alternativen auf. Diese sind in der nachfolgenden Abbildung in der Form von Petri-Netzen dargestellt. Die Kreise (Stellen) symbolisieren dabei die Einzelschritte eines Prozessablaufs. Durch die schwarzen Punkte (Marken) werden die jeweils aktuellen Schritte angezeigt. Die Balken (Transitionen) stellen Ereignisse dar, bei deren Eintreten entsprechend der

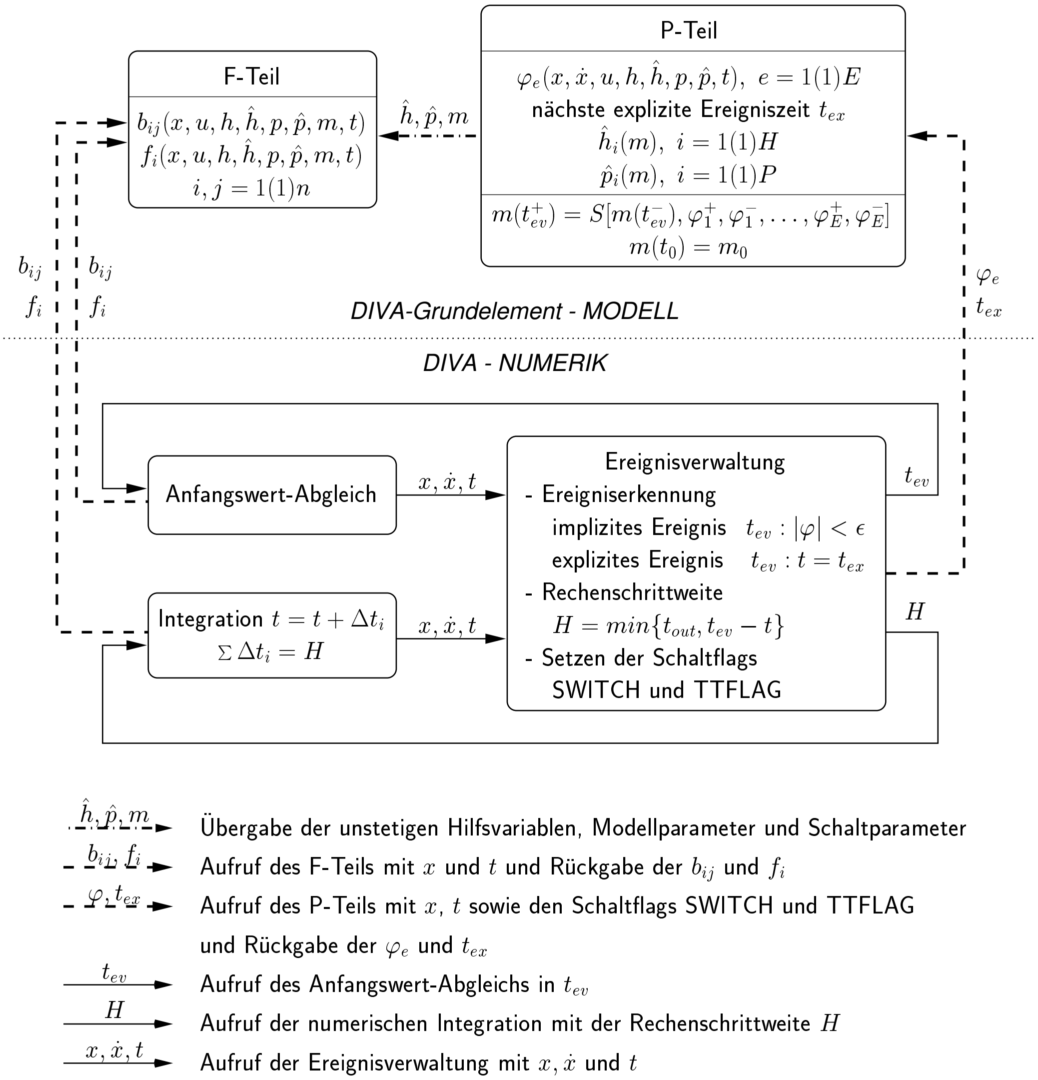


Abbildung 4: Zusammenspiel der DIVA-Ereignisverwaltung mit den FORTRAN-Unterprogrammen zur Ereignisdefinition (P-Teil) und Funktionsauswertung (F-Teil), sowie mit den Numerikprogrammen zum Abgleich und Integrieren

Pfeile (Flussrelation) die Marke in der Stelle vor der Transition in die Stelle nach der Transition verschoben wird. Der Schaltzeitpunkt wird dabei durch eine φ-Funktion definiert. Somit wird bei Eintreten des Ereignisses ein Schritt des Prozessablaufs beendet und der nächste eingeleitet. Dabei können auch mehrere Schritte gleichzeitig beendet bzw. eingeleitet werden, wenn eine Transition durch Pfeile mit mehreren Stellen gleichzeitig verbunden ist.

Mit Sequenzen modelliert man lineare Ketten von Abläufen, die nacheinander durchschritten werden: einem Teilschritt folgt der nächste und so weiter.

Bei Nebenläufigkeiten (Parallelitäten) finden mehrere Teilprozesse unabhängig voneinander statt, können aber zu einem späteren Zeitpunkt synchronisiert werden. Bei einer Synchronisation wird erst fortgefahren, wenn alle nebenläufigen Teilprozesse einen bestimmten Endzustand erreicht haben. Dies wird in Abbildung 5 durch die Transition φ_d dargestellt, die erst dann weiterschalten kann, wenn beide Stellen davor eine

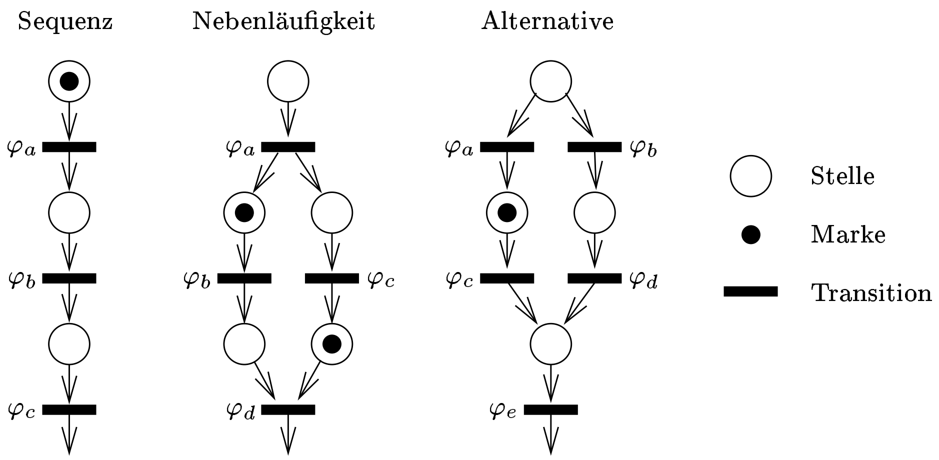


Abbildung 5: Petri-Netz-Darstellung der Basiskonstrukte ereignisdiskreter Abläufe in Prozess-Steuerungen

Marke enthalten, also die beiden parallelen Teilabläufe beendet sind.

Bei Alternativen schließlich stehen mehrere weitere Verläufe miteinander im Konflikt. Je nachdem, welches Ereignis zuerst eintritt, folgt der Prozess einem der möglichen weiteren Verläufe. In Abbildung 5 kann nur eine der beiden Transitionen φ_a und φ_b schalten, wodurch der weitere Prozessablauf bestimmt ist.

Aus diesen Basiskonstrukten lassen sich sehr komplexe Strukturen aufbauen. Dabei können auch Strukturen mit iterativen oder zyklischen Vorgängen vorkommen.

Mit Petri-Netzen, genauer Bedingungs/Ereignis-Systemen, lassen sich insbesondere nebenläufige Prozesse effizient und anschaulich modellieren. Daher werden Petri-Netze bei der Eingabe von Schaltwerken gemischt kontinuierlich-ereignisdiskreter Modelle im Code-Generator verwendet. In diesem Abschnitt wird zunächst die Art von Petri-Netz definiert, die mit Hilfe der Eingabesprache des Code-Generator dargestellt werden kann. Außerdem werden wichtige Begriffe im Zusammenhang mit Petri-Netzen eingeführt. Anschließend wird der Bezug zur Darstellung gemischt kontinuierlich-ereignisdiskreter Modelle hergestellt. Näheres zu den folgenden Definitionen ist [40] zu entnehmen.

Definition von Petri-Netzen

Zunächst wird der Begriff eines Netzes oder Netzgraphen geklärt:

Definition 1 *Ein Netz (oder Netzgraph) ist ein Tripel $N = (S, T, F)$ mit*

$$S \cap T = \emptyset \tag{21}$$

$$F \subseteq (S \times T) \cup (T \times S) \tag{22}$$

Die Elemente von S heißen Stellen, die von T heißen Transitionen. Beide zusammen werden als Knoten bezeichnet. Die Elemente der Flussrelation F heißen Kanten.

Stellen werden graphisch als Kreise, Transitionen als Balken dargestellt. Die Elemente der Flussrelation (Pfeile) verbinden nach (22) jeweils eine Stelle mit einer Transition oder umgekehrt. Für eine Transition $t \in T$ heißt die Menge aller Stellen $s \in S$ mit $(s, t) \in F$ Eingangsstellen oder Vorbereich, die Menge aller Stellen s mit $(t, s) \in F$ Ausgangsstellen oder Nachbereich der Transition. In Abbildung 6 werden diese Begriffe verdeutlicht.

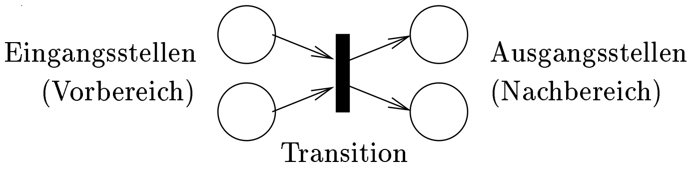


Abbildung 6: Zur Erklärung der Begriffe Ein- und Ausgangsstellen

Eine spezielle Klasse von Netzen sind sogenannte Stellen/Transitions-Netze (S/T-Netze), die zusammen mit einer Anfangsmarkierung ein Stellen/Transitions-System oder S/T-System nach folgender Definition ergeben:

Definition 2 *Ein 6-Tupel* $Y = (S, T, F, K, W, M_0)$ *heißt Stellen/Transitions-System oder S/T-System, falls*

- (S, T, F) *ein Netz ist,*
- $K : S \to \mathbb{N} \cup \{\infty\}$ *(Kapazitäten der Stellen),*
- $W : F \to \mathbb{N}$ *(Kantengewichte der Kanten) und*
- $M_0 : S \to \mathbb{N}_0$ *(Anfangsmarkierung), wobei* $\forall s \in S : M_0(s) \leq K(s)$.

S/T-Systeme werden häufig auch als Petri-Netze bezeichnet. Für die Darstellung von Schaltwerken gemischt kontinuierlich-ereignisdiskreter Modelle kann man sich jedoch auf einen Spezialfall, sogenannte Bedingungs/Ereignis-Systeme oder B/E-Systeme, beschränken. Im folgenden sind daher mit Petri-Netzen immer B/E-Systeme nach folgender Definition gemeint:

Definition 3 *Ein Bedingungs/Ereignis-System oder B/E-System ist ein S/T-System* $Y = (S, T, F, K, W, M_0)$ *mit schlingungsfreiem Netz* $N = (S, T, F)$ *und sowohl* $K = 1$ *als auch* $W = 1$.

Ein Netz N heißt schlingungsfrei, wenn keine zwei Knoten $s \in S$ und $t \in T$ existieren, die in der Flussrelation sowohl durch $(s, t) \in F$ als auch durch $(t, s) \in F$ miteinander verbunden sind. Zur Verdeutlichung des Begriffs ist in der nachfolgenden Abbildung links als Gegenbeispiel eine Schlinge dargestellt.

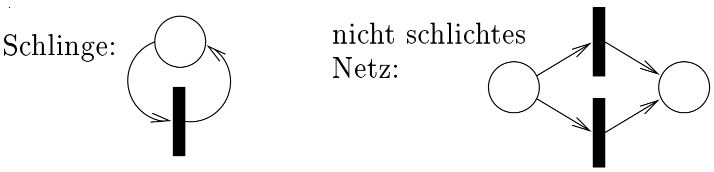


Abbildung 7: Zur Erklärung der Begriffe Schlinge und Schlichtheit

In [40] wird für B/E-Systeme zusätzlich noch die Schlichtheit des Neztes N verlangt. Schlichtheit bedeutet, dass keine zwei Knoten sowohl denselben Vor- als auch denselben Nachbereich haben. Das nicht schlichte Netz in Abbildung 7 (rechts) soll den Begriff der Schlichtheit verdeutlichen: hier haben zwei Transitionen denselben Vor- und denselben Nachbereich. Die Forderung der Schlichtheit führt bei der Modellierung eines Schaltwerks zu unerwünschten Einschränkungen und wurde deshalb in Definition 3 weggelassen. Auf diesen Aspekt wird in Abschnitt 3.4.2 noch näher eingegangen.

Ein B/E-System ohne Anfangsmarkierung heißt B/E-Netz. Eine Markierung eines B/E-Netzes ist eine Abbildung $M : S \to \{0, 1\}$, d.h. jede Stelle enthält keine oder eine Marke. Eine Stelle, die eine Marke enthält, wird auch als markiert bezeichnet. Die Anfangsmarkierung M_0 eines B/E-Systems ist eine Markierung des zugehörigen B/E-Netzes. Graphisch werden Marken als schwarze Punkte dargestellt.

Die Dynamik von B/E-Systemen wird durch folgende Definition bestimmt:

Definition 4 *Eine Transition $t \in T$ eines B/E-Systems heißt aktiviert unter M, wenn*

- *alle Eingangsstellen $s_{ein} \in S$ von t markiert und*
- *alle Ausgangsstellen $s_{aus} \in S$ von t nicht markiert sind.*

Eine aktivierte Transition t schaltet von M in die sogenannte (unmittelbare) Folgemarkierung M', wenn

- *die Marken aller Eingangsstellen entfernt werden und*
- *alle Ausgangsstellen Marken erhalten.*

Die Aktivierungsbedingung und die vorstehende Definition der Folgemarkierung wird als Schaltregel bezeichnet.

Das Schalten einer Transition wird im folgenden auch als Auslösen bezeichnet.

Da Stellen in B/E-Systemen nur höchstens eine Marke enthalten können, zeigen Marken die Gültigkeit von „Bedingungen“ an. Transitionen entfernen bzw. setzen Marken in ihrem Vor- bzw. Nachbereich und verändern so die Bedingungen des Netzes. Daher kann man Transitionen als „Ereignisse“ im Petri-Netz bezeichnen. Wie Abschnitt 3.4.2 zeigen wird, stimmt bei der Modellierung von gemischt kontinuierlich-ereignisdiskreten Systemen mit Hilfe eines Petri-Netzes eine Transition nicht unbedingt mit einem Ereignis im Modell überein. Aus diesem Grund wird der Begriff „Ereignis“ im folgenden nur noch im Sinne eines gemischt kontinuierlich-ereignisdiskreten Systems verwendet.

Petri-Netz eines Schaltwerks

Bei der Modellierung gemischt kontinuierlich-ereignisdiskreter Systeme kann das Schaltwerk als Petri-Netz dargestellt werden, indem man die Stellen als Schaltparameter $\boldsymbol{m}$ des DAE-Systems (5) auffasst. Eine Marke weist dem von der Stelle repräsentierten Schaltparameter den Wert 1 zu, ohne Marke hat er den Wert 0. Somit entspricht jede Markierung des Petri-Netzes einem Zustand des Schaltwerks und damit einem gültigen DAE-System. Die Anfangsmarkierung $\boldsymbol{m}(t_0)$ bestimmt dementsprechend das DAE-System zum Zeitpunkt t_0, also zu Beginn einer Simulation.

Ein Petri-Netz stellt ereignisdiskrete Vorgänge nach den obigen Definitionen nur in einen logischen Zusammenhang. Um ein Schaltwerk als Petri-Netz zu modellieren, muss darüber hinaus ein zeitlicher Aspekt mit eingebracht werden. Dies geschieht dadurch, dass der Schaltzeitpunkt aktivierter Transitionen an Ereignisse gekoppelt werden kann: solche Transitionen lösen nach ihrer Aktivierung erst aus, wenn das zugehörige Ereignis eintritt. Durch Transitionen einschließlich der Flussrelation des Netzes und eine Zuordnung zu Ereignissen werden somit die Schaltfunktionen $\boldsymbol{S}$ aus (17) repräsentiert: in Abhängigkeit von der alten Markierung $\boldsymbol{m}(t_{ev}^-)$ und einem stattfindenden Ereignis zum Zeitpunkt t_{ev} schalten Transitionen und stellen dadurch eine neue Markierung $\boldsymbol{m}(t_{ev}^+)$ (Folgemarkierung) her.

An dieser Stelle sei auf die genaue Unterscheidung zwischen Transitionen und Ereignissen hingewiesen: Während Ereignisse oder Events diskrete Zustandsänderungen im gemischt kontinuierlich-ereignisdiskreten System sind, die einen unstetigen Verlauf der Lösung verursachen, sind Transitionen Elemente eines Petri-Netzes, die die Markierung und damit den Schaltwerkszustand verändern. In vielen Fällen stimmen beide miteinander überein. Es kann aber auch Ereignisse geben, die in mehreren nebenläufigen Teilprozessen gleichzeitig aktivierte Transitionen auslösen können. Weiterhin kann eine Transition auch direkt – ohne das Eintreten eines Ereignisses – schalten, sobald sie aktiviert ist, um zum Beispiel mehrere nebenläufige Teilprozesse zu synchronisieren (vgl. Abb. 5): sobald alle Teilprozesse ihre letzte Stelle erreicht haben, wird in eine gemeinsame Folgestelle weitergeschaltet, ohne dass ein weiteres Ereignis eintritt.

Man kann also zwei Klassen von Transitionen unterscheiden: solche, die mit Ereignissen auslösen, und solche, die direkt auslösen, sobald sie aktiviert sind. Für eine programmtechnische Umsetzung und Codierung gemischt kontinuierlich-ereignisdiskreter Modelle mit Hilfe von Petri-Netzen ist eine Betrachtung der Schaltdynamik, wie sie bislang vorliegt, noch nicht ausreichend. Im nächsten Unterkapitel folgen deshalb weitere Definitionen von Schaltregeln, die die Aspekte einer programmtechnischen Umsetzung mit berücksichtigen.

Beim Modellieren eines Schaltwerks mit Hilfe eines Petri-Netzes kann es vorkommen, dass das Auslösen einer Transition nicht nur von der Markierung des Petri-Netzes und einem Ereignis abhängt, sondern auch von weiteren Bedingungen wie zum Beispiel den Vorzeichen verschiedener anderer φ-Funktionen. Solche Bedingungen können mit den hier zur Verfügung gestellten Mitteln durch weitere Stellen modelliert werden, die den Zustand der weiteren Bedingungen repräsentieren. Diese zusätzlichen Stellen enthalten dann keine Information für das auszuwählende Gleichungssystem, repräsentieren also keinen Schaltparameter, sondern dienen nur der Definition der Schaltwerksdynamik.

Des weiteren kann es auch sinnvoll sein, wenn ein bestimmter Schaltvorgang durch das Eintreten verschiedener Ereignisse verursacht werden kann. Transitionen werden hier jedoch nur an ein Ereignis gekoppelt, so dass ein solcher Fall nur mit mehreren Transitionen modelliert werden kann, die alle jeweils denselben Vor- und Nachbereich haben. Das resultierende Netz ist damit nicht schlicht. Aus diesem Grund wurde in der Definition 3 bei B/E-Systemen keine Schlichtheit verlangt.

3.4.3 Petri-Netze im Code-Generator

Zur Definition eines Schaltwerks können mehrere unabhängige Petri-Netze verwendet werden. Ein Petri-Netz besteht nach Kapitel 3.4.2 aus Stellen, Transitionen, einer Flussrelation und Anfangsmarken. Für die Umsetzung von Petri-Netzen in Apparatemodellen für die Simulationsumgebung DIVA wird im folgenden die Schaltdynamik in Petri-Netzen definiert, die auch das Verhalten bei Konflikten zwischen verschiedenen Transitionen eindeutig bestimmt und Probleme im Zusammenhang mit der Bindung an Ereignisse löst. Ausserdem wird eine Erweiterung der hier verwendeten Petri-Netze eingeführt: bei Eintreten eines Ereignisses kann unabhängig von der Markierung des Petri-Netzes eine bestimmte neue Markierung definiert werden, womit sich beispielsweise Notfallsteuerungen in Prozessen modellieren lassen. Weitere Abschnitte stellen den Zusammenhang zwischen dem Schaltwerk in Form von Petri-Netzen und endogenen expliziten Ereignissen her und beschreiben die Darstellung der Abhängigkeit unstetiger zeitinvarianter Hilfsgrößen und Modellparameter von Stellen des Petri-Netzes.

Schaltdynamik eines Petri-Netzes

In Kapitel 3.4.2 wurde die Schaltregel für Petri-Netze in Definition 4 eingeführt. Zwei Aspekte wurden an dieser Stelle allerdings nicht beachtet: Konfliktsituationen zwischen verschiedenen Transitionen und der zeitliche Zusammenhang während Schaltvorgängen. Für die Behandlung dieser Aspekte spielt nämlich die programmtechnische Realisierung eine entscheidende Rolle.

Ein Konflikt ist gegeben, wenn zwei Transitionen gleichzeitig aktiviert sind, aber wegen gemeinsamer Stellen im Vor- oder Nachbereich nicht beide schalten können. In Abbildung 8 ist links ein Beispiel für einen Konflikt dargestellt: die Transitionen T_1 und T_2 haben die gemeinsame Eingangsstelle S_1. Durch das Schalten der einen Transition ist jeweils die andere nicht mehr aktiviert.

Solange die beteiligten Transitionen an verschiedene Ereignisse gekoppelt sind und diese nicht zufällig gleichzeitig eintreten, ist die Dynamik durch die Reihenfolge der Ereigniszeitpunkte bestimmt. Dürfen die in Konflikt stehenden Transitionen jedoch zum selben Ereigniszeitpunkt auslösen, so ist das Verhalten zunächst unbestimmt. Im Code-Generator soll die Reihenfolge bei der Definition der Transitionen entscheiden: die zuerst definierte Transition schaltet, die andere nicht.

Anhand der beiden anderen Beispiele in Abbildung 8 wird ein weiterer offener Punkt in der Schaltdynamik gezeigt, der den zeitlichen Zusammenhang von Schaltvorgängen in Petri-Netzen betrifft. Es stellt sich die Frage, zu welchem Zeitpunkt der Simulation bzw. an welcher Stelle des auszuwertenden Schaltwerkcodes eine Transition als aktiviert gilt, wenn der dafür erforderliche Markierungszustand durch ein Ereignis hergestellt worden ist. Dabei muss zwischen direkt auslösenden Transitionen und solchen, die mit einem Ereignis auslösen, unterschieden werden (vgl. Kapitel 3.4.2).

Im folgenden wird die Transition T_4 betrachtet, die zunächst mit einem Ereignis auslöst. Schaltet die Transition T_3, so wird die Stelle S_5 markiert, und damit ein Markierungszustand hergestellt, in dem die Transition T_4 aktiviert ist. Für das weitere Verhalten ergeben sich zwei Möglichkeiten:

1. Die Transition T_4 gilt sofort als aktiviert, kann also noch zu demselben Ereigniszeitpunkt auslösen, falls ihr Ereignis gerade stattfindet.

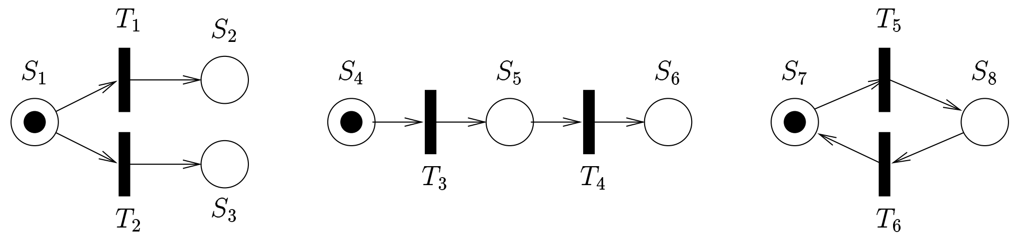


Abbildung 8: Problemfälle bei der Schaltdynamik in Petri-Netzen (links eine Konfliktsituation zwischen zwei Transitionen, in der Mitte eine Sequenz von Transitionen und rechts das Petri-Netz einer reversiblen Unstetigkeit)

2. Die Transition T_4 gilt erst nach der Auswertung des kompletten Schaltwerkcodes als aktiviert, also unmittelbar nach dem Schaltzeitpunkt von T_3. T_4 kann somit erst beim nächsten Eintreten des Ereignisses auslösen.

Hierbei können durchaus beide Varianten sinnvoll erscheinen. Betrachtet man hingegen das rechte Beispiel in Abbildung 8, so wird das erwünschte Verhalten klar. Hier ist eine reversible Unstetigkeit dargestellt, die Transitionen T_5 und T_6 sind also beide an dieselbe φ-Funktion gekoppelt. Bei einem Nulldurchgang dieser φ-Funktion schaltet T_5 und markiert dadurch die Stelle S_8. Würde die Transition T_6 nun sofort als aktiviert gelten, so müsste sie noch zum selben Ereigniszeitpunkt auslösen und somit die ursprüngliche Markierung wieder herstellen, was aber nicht erwünscht ist.

Die Aktivierung von Transitionen, die mit Ereignissen auslösen, soll daher erst gültig werden, nachdem der erforderliche Markierungszustand hergestellt und alle weiteren Schaltaktionen desselben Ereigniszeitpunkts ausgeführt worden sind, also erst nach der Auswertung aller Transitionen, die mit Ereignissen auslösen.

Zusammen mit dem gewünschten Verhalten in Konfliktsituationen ergibt sich die folgende Schaltregel für eine Transition, deren Auslösen an ein Ereignis gekoppelt ist:

Definition 5 *Eine Transition, die mit einem Ereignis auslöst, gilt als aktiviert, wenn vor dem Auswerten des gesamten Schaltwerkcodes* <u>*und*</u> *zu dem Zeitpunkt, an dem der entsprechende Programmcode für die Transition ausgewertet wird, alle Eingangsstellen markiert und alle Ausgangsstellen nicht markiert sind.*

Hat also in der Zwischenzeit eine andere Transition die Aktivierung aufgehoben, so darf nicht geschaltet werden. Wurde der Markierungszustand, der für eine Aktivierung erforderlich ist, erst in diesem Schaltvorgang hergestellt, so gilt die Transition noch nicht als aktiviert.

In der folgenden Betrachtung handelt es sich bei T_4 in Abbildung 8 um eine direkt auslösende Transition. Schaltet wiederum T_3, so wird die Stelle S_5 markiert. Da eine direkt auslösende Transitionen nach ihrer Aktivierung sofort schaltet, ist es hier sinnvoll, dass T_4 unmittelbar nach dem Markieren der Stelle S_5 als aktiviert gilt, um noch zum selben Ereigniszeitpunkt schalten zu können. Direkt auslösende Transitionen dürfen also auch schalten, wenn sie beim selben Auswerten des Schaltwerkcodes erst aktiviert worden sind. Es ergibt sich die folgende Schaltregel:

Definition 6 *Eine direkt auslösende Transition gilt als aktiviert, wenn zu dem Zeitpunkt, an dem der entsprechende Programmcode für die Transition ausgewertet wird, alle Eingangsstellen markiert und alle Ausgangsstellen frei sind.*

Für eine Realisierung dieser Schaltregel ist es notwendig, dass der Programmcode für direkt auslösende Transitionen <u>hinter</u> dem Code für die übrigen Transitionen programmiert wird, um sicherzustellen, dass sie rechtzeitig, also vor ihrer Auswertung, aktiviert worden sind. Außerdem ist beim Entwerfen eines Petri-Netzes darauf zu achten, dass sich keine Stelle unmittelbar zwischen zwei direkt auslösenden Transitionen befindet. In Abbildung 8 dürfen beispielsweise T_3 und T_4 nicht beide als direkt auslösende Transitionen definiert werden. In einem solchen Fall kann nämlich bei der Umsetzung im FORTRAN-Programmcode für DIVA keine korrekte Schaltwerksdynamik garantiert werden.

Unter Beachtung dieser Schaltregeln erfolgt die Codierung von Petri-Netzen durch den Code-Generator.

Markierungsdefinition durch ein Ereignis

In Prozess-Steuerungen kann es vorkommen, dass zu jedem Zeitpunkt unabhängig von der aktuellen Markierung des zugehörigen Petri-Netzes bei Eintreten eines bestimmten Ereignisses eine bestimmte Markierung hergestellt werden soll. Ein Beispiel hierfür ist eine Notfall-Steuerung oder ein „Not-Aus“, das bei Eintreten einer bestimmten Gefahrensituation in jedem Fall aktiviert werden soll.

Um einen solchen Vorgang in einem Petri-Netz zu modellieren, müsste eine Vielzahl von Transitionen definiert werden, die sämtliche möglichen Kombinationen an Stellen als Eingangsstellen hätten und den erwünschten Zustand als Ausgangsstellen. Anders interpretiert entspricht dies auch einer Neuinitialisierung des Petri-Netzes – unabhängig von der aktuellen Markierung – bei Eintreten eines bestimmten Ereignisses.

Um solche Vorgänge einfach angeben zu können, werden Petri-Netze im Code-Generator um ein Element erweitert: eine Markierungsdefinition durch ein Ereignis wird unabhängig von der aktuellen Markierung des Petri-Netzes vorgenommen, falls das Ereignis eintritt. Die Angabe dieser Markierung erfolgt analog zur Angabe der Anfangsmarkierung durch eine Menge von Stellen, die Marken enthalten sollen.

Stellen, die auf diese Weise bei der Auswertung des Schaltwerkscodes eine neue Marke erhalten haben, können zum selben Ereigniszeitpunkt lediglich direkt auslösende Transitionen aktivieren und damit deren Auslösen verursachen. Für eine Transition, die mit einem Ereignis auslöst, kann die Schaltregel nach Definition 5 nicht erfüllt werden.

Triggern endogener expliziter Ereignisse

Die Schaltzeitpunkte endogener (mittelbarer) expliziter Ereignisse ergeben sich erst während der Simulation, da sie von anderen Schaltvorgängen abhängen. Diese Abhängigkeit von anderen Schaltvorgängen wird im Code-Generator dadurch realisiert, dass endogene explizite Ereignisse durch Stellen „getriggert“ werden, was nach folgender Definition erfolgt:

Definition 7 *Ein endogenes explizites Ereignis wird durch eine Stelle getriggert, indem beim Markieren dieser Stelle die Ereigniszeit auf die aktuelle Simulationszeit zuzüglich einer positiven Verzögerungszeitspanne Δt festgesetzt wird:*

$$t_{ev} = t_{sim} + \Delta t \tag{23}$$

Das Triggern kann beliebig oft erfolgen, aber nicht, solange sich das Ereignis im getriggerten Zustand befindet, also während des Ablaufs der Verzögerungszeit.

Das bedeutet, dass das endogene explizite Ereignis genau Δt Zeiteinheiten nach der Markierung der triggernden Stelle eintritt. Zur Definition eines endogenen expliziten Ereigniszeitpunktes sind also eine triggernde Stelle aus dem Petri-Netz sowie eine positive Verzögerungszeit $\Delta t > 0$ anzugeben. Eine Verzögerungszeit $\Delta t = 0$ würde Probleme bei der DIVA-Ereignisverwaltung verursachen und ist daher unzulässig.

Charakteristische Werte für unstetige Hilfsgrößen und Parameter

Die Auswahl der Werte für die unstetigen zeitinvarianten Hilfsgrößen $\hat{h}_i$ und unstetigen Modellparameter $\hat{p}_i$ erfolgt nach (9) bzw. (10) in Abhängigkeit von einer Teilmenge K_h bzw. K_p der Stellen, in der jeweils nur eine Stelle markiert sein darf. Je nachdem, welche Stelle m_k gerade markiert ist, wird der unstetigen Hilfsgröße $\hat{h}_i$ der Wert $\hat{h}_{i,k}$ bzw. dem unstetigen Modellparameter $\hat{p}_i$ der Wert $\hat{p}_{i,k}$ zugewiesen. Die Werte $\hat{h}_{i,k}$ bzw. $\hat{p}_{i,k}$ charakterisieren sozusagen die Stelle m_k in Bezug auf $\hat{h}_i$ bzw. $\hat{p}_i$. Daher wird im folgenden von charakteristischen Werten einer Stelle m_k für unstetige zeitinvariante Hilfsgrößen bzw. Modellparameter gesprochen. Die charaketristischen Werte werden den einzelnen Stellen zugeordnet und im Code-Generator zusammen mit den Stellen definiert.

Codierung des Unterprogramms zur Ereignisdefinition

Das Unterprogramm zur Ereignisdefinition wird generiert, wenn der nachfolgend beschriebene Abschnitt zur Definition der Eventzeitpunkte in der Eingabedatei vorhanden ist. Im Unterprogramm zur Initialisierung (I-Teil) werden dann die Abschnitte zur Speicherplatzverwaltung und zur Adressberechnung ergänzt. Diese Ergänzungen betreffen zum einen die φ-Funktionen für die in DIVA das PHI-Feld vorgesehenen ist und zum anderen die Parameter der expliziten Events, die einen weiteren Teil des RPAR-Felds belegen. Das Unterprogramm zur Ereignisdefinition stellt den Zeitpunkt des nächsten expliziten Events fest, enthält die codierten φ-Funktionen sowie das Schaltwerk S bestehend aus mehreren Petri-Netzen. Im Daten-File werden die Werte des Vektors der Schaltschranken ϵ_φ sowie der expliziten Eventzeitpunkte codiert. **ACHTUNG: Optionale Berechnung von Hilfsvariablen.** Da es sinnvoll sein kann, φ-Funktionen in Abhängigkeit von Hilfsvariablen zu definieren, müssen die Werte der Hilfsvariablen bekannt sein. Deshalb ist der Code zu deren Berechnungen im P-Teil enthalten. Werden die Hilfsvariablen hier nicht benötigt, kann deren Berechnung übersprungen werden. Dazu dient die Abfrage der lokalen boole'schen Variablen `EVLHV` (Evalualte Help-Variables), die vor deren Berechnungsteil steht. Per default ist `EVLHV` auf `TRUE` gesetzt. Um die Berechnung der Hilfsvariablen zu überspringen, muß `EVLHV` auf `FALSE` gesetzt werden.

3.4.4 Definition von Petri-Netzen

Die Definition von Petri-Netzen im Code-Generator umfasst nach den Abschnitten 3.4.2 und 3.4.3 die Angabe von

1. Stellen,
2. einer Anfangsmarkierung,
3. Markierungsdefinitionen durch Ereignisse,
4. Transitionen, die mit Ereignissen auslösen und
5. direkt auslösenden Transitionen.

Die Flussrelation des Petri-Netzes wird zusammen mit den Transitionen angegeben. Für eine vollständige Petri-Netz-Definition sind auf jeden Fall die Punkte 1., 2. und 4. anzugeben, die restlichen Elemente sind optional, können also weggelassen werden.

Ein Schaltwerk kann aus mehreren Petri-Netzen bestehen, daher werden in der Definitionsfunktion `(define-event-schedule ...)` für das Schaltwerk ein oder mehrere Petri-Netz-Definitionen eingegeben:

(**define-event-schedule** {⟨petrinet⟩}$^+$)

Eine solche Petri-Netz-Definition besteht aus einer Liste mit verschiedenen „key-value"-Paaren, mit deren Hilfe dem Petri-Netz die o.g. Elemente sowie ein Name und ein Kommentar zugewiesen werden. Folgende Schlüsselwörter (keys) stehen zur Verfügung:

⟨petrinet⟩::=
(**:petrinet-name** ⟨name⟩
:meaning ⟨string⟩
:places ({⟨place⟩}$^+$)
:initial-marks ({⟨string⟩}$^+$)
:event-initiated-status ({⟨event⟩}*)
:event-fired-transitions ({⟨event⟩}$^+$)
:directly-fired-transitions ({⟨transition⟩}*))

:petrinet-name ⟨name⟩

- Ordnet dem Petri-Netz einen Namen (String) zu. Beachten Sie bitte, daß die Zeichen `&`, `#` und `\` problematisch beim Übersetzen der LaTeX-Dokumentation sein können.

:meaning ⟨string⟩

- Dieser key wird von einem möglicherweise mehrzeiligen String gefolgt, der eine nähere Beschreibung enthält und zur Dokumentation benutzt wird. Beachten Sie bitte, daß die Zeichen `&`, `#` und \ problematisch beim Übersetzen der LaTeX-Dokumentation sein können.

:places ({⟨place⟩}$^+$)

- Die Stellen des Petri-Netzes werden in einer Liste definiert. Eine Stelle kann während der Simulation in DIVA markiert oder frei sein. Anhand des Markierungszustandes der Stelle können in den Modellgleichungen entsprechend Gleichungen (11) bis (14) Teile von Elementen der $\boldsymbol{B}$-Matrix oder des $\boldsymbol{f}$-Vektors ausgewählt werden. Bei der Definition der Modellgleichungen können dafür im Code-Generator konditionale Ausdrücke `:cond` verwendet werden. Konditionale Ausdrücke bestehen aus Paaren einer Bedingung und eines Terms. Ein Term wird ausgewertet, wenn die zugehörige Bedingung zutrifft. Statt eines logischen Ausdrucks kann nun auch eine Stelle oder eine logische Verknüpfung mehrerer Stellen als Bedingung verwendet werden: die Bedingung trifft genau dann zu, wenn die Stelle markiert ist. Die Definition einer Stelle sieht wie folgt aus:

  ⟨place⟩::=
  (**:place-name** ⟨name⟩
  :meaning ⟨string⟩
  :set-characteristic-values ({⟨characteristic-value⟩}*))

  :place-name ⟨name⟩

  - Ordnet der Stelle eines Petri-Netzes einen Namen (String) zu. Beachten Sie bitte, daß die Zeichen `&`, `#` und \ problematisch beim Übersetzen der LaTeX-Dokumentation sein können.

  :meaning ⟨string⟩

  - Dieser key wird von einem möglicherweise mehrzeiligen String gefolgt, der eine nähere Beschreibung enthält und zur Dokumentation benutzt wird. Beachten Sie bitte, daß die Zeichen `&`, `#` und \ problematisch beim Übersetzen der LaTeX-Dokumentation sein können.

  :set-characteristic-values ({⟨characteristic-value⟩}*)

  - Zu den einzelnen Stellen gehören optional auch Zuweisungen charakteristischer Werte für unstetige zeitinvariante Hilfsgrößen oder Modellparameter. Charakteristische Werte können für Integer-Parameter, Real-Parameter und für Hilfsvariablen definiert werden. Die Werte werden bei der Simulation zugewiesen, solange die Stelle markiert ist. Dabei ist zu beachten, dass die Werte auch dann gesetzt bleiben, wenn die Stelle nicht mehr markiert ist. Sie können nur durch andere Stellen wieder verändert werden.

    ACHTUNG: Bei der Angabe von charakteristischen Wertezuweisungen ist darauf zu achten, dass sich beim Setzen der charakteristischen Werte keine Konflikte zwischen verschiedenen Stellen ergeben können. Letztlich entscheidet bei der Zuweisung der Werte die Reihenfolge bei der Definition der Stellen.

    1. **Integer-Parameter:**

       ⟨characteristic-integer-parameter-value⟩::=
       (**:integer-parameter-name** ⟨string⟩
       :value ⟨expression⟩)

       :integer-parameter-name ⟨string⟩

       - Bezeichner eines zuvor definierten Integer-Parameters.

       :value ⟨expression⟩

       - Wert der dem Integer-Parameter beim Belegen der Stelle zugewiesen wird.

    2. **Real-Parameter:**

       ⟨characteristic-real-parameter⟩::=

(**:real-parameter-name** ⟨string⟩
:value ⟨expression⟩)

:real-parameter-name ⟨string⟩
- Bezeichner eines zuvor definierten Real-Parameters.

:value ⟨expression⟩
- Wert der dem Real-Parameter beim Belegen der Stelle zugewiesen wird.

3. **Hilfsvariablen:**

⟨characteristic-help-variable⟩::=
(**:help-variable-name** ⟨string⟩
:value ⟨expression⟩)

:help-variable-name ⟨string⟩
- Bezeichner einer zuvor definierten Hilfs-Variablen.

:value ⟨expression⟩
- Wert der der Hilfs-Variablen beim Belegen der Stelle zugewiesen wird.

:initial-marks ($\{\langle$string$\rangle\}^+$)
- Definition der Anfangsmarkierung des Petri-Netzes. Dazu werden in einer Liste die Bezeichner der zu Anfang markierten Stellen als Strings eingegeben.[11]

:event-initiated-status ($\{\langle$event$\rangle\}^*$)
- Liste von Ereignissen zur Umschaltung auf vollständig neue Markierungsdefinitionen des Petri-Netzes, siehe Beschreibung Seite 35. Die Liste enthält Ereignis-Definitionen mit zugeordneten Markierungen. Eine Beschreibung der Ereignisse folgt ab Seite 39. Die Markierung, die durch ein bestimmtes Ereignis im Petri-Netz hergestellt wird, ist innerhalb der Definition des zugehörigen Ereignisses anzugeben.

:event-fired-transitions ($\{\langle$event$\rangle\}^+$)
- Definition von Ereignissen, die Transitionen auslösen. Die Liste enthält Ereignis-Definitionen mit zugeordnete Transitionen.

:directly-fired-transitions ($\{\langle$transition$\rangle\}^*$)
- Liste mit Definitionen von Transitionen, die direkt auslösen.

  Eine Transition wird durch die Flussrelation nach Abschnitt 3.4.2 mit einer Menge Eingangs- und einer Menge Ausgangsstellen verbunden. Die Definition einer Transition beinhaltet die Angabe dieser Ein- bzw. Ausgangsstellen:

  ⟨transition⟩::=
  (**:from** ($\{\langle$string$\rangle\}^+$)
  :to ($\{\langle$string$\rangle\}^+$))

  :from ($\{\langle$string$\rangle\}^+$)
  - Liste mit den Bezeichnern mindestens einer Eingangsstelle.

  :to ($\{\langle$string$\rangle\}^+$)
  - Liste mit den Bezeichnern mindestens einer Ausgangsstelle.

  ACHTUNG: Dabei ist zu beachten, dass keine Stelle gleichzeitg Ein- und Ausgangsstelle sein

[11] Alternativ dazu sollen in zukünftigen Versionen auch logische Ausdrücke verwendet werden können, deren Resultat wiederum immer ein Bezeichner einer zu Anfang markierten Stelle ist.

darf, da sonst eine Schlinge vorliegen würde, die nach Definition 3 in den hier verwendeten Petri-Netzen, nämlich B/E-Systemen unzulässig ist.

3.4.5 Definition von Ereigniszeitpunkten

Für die Definition des Zeitpunkts eines Ereignisses gibt es drei Möglichkeiten: die implizite Angabe mit Hilfe einer φ-Funktion, die exogene (unmittelbar) explizite Angabe eines Schaltzeitpunkts oder die endogene (mittelbare) explizite Angabe mit Hilfe einer triggernden Stelle und einer Verzögerungszeit:

⟨event⟩::=
⟨implicit-event⟩|⟨exogenous-explicit-event⟩|⟨endogenous-explicit-event⟩

Ereignisse können sich auf das Petri-Netz in zweierlei Weise auswirken und treten deshalb unter zwei verschiedenen Schlüsselwörtern einer Petri-Netz-Definition auf: entweder lösen sie eine oder mehrere aktivierte Transitionen aus (`:event-fired-transitions`) oder sie stellen entsprechend Kapitel 3.4.3 unabhängig von der aktuellen Markierung des Petri-Netzes eine bestimmte neue Markierung her (`:event-initiated-status`). Je nachdem, wo das Ereignis definiert wird, sind daher entweder eine Liste mit zugehörigen Transitionen (`:transitions`) oder eine Markierung für das Petri-Netz (`:marks`) anzugeben. Letztere besteht wie die Anfangsmarkierung aus einer Liste mit den Bezeichnern der Stellen, die Marken enthalten sollen.

Weiterhin gemeinsam ist allen Ereignisdefinitionen ein Bezeichner (`:implicit-event-name` bzw. `:explicit-event-name`) und ein String zur Dokumentation (`:meaning`). Mit Hilfe des Bezeichners können die Ereigniszeitpunkte während einer Simulation in DIVA verändert werden.

Die verschiedenen Ereignisarten werden wie folgt definiert:

Implizite Ereignisse

⟨implicit-event⟩::=
(**:implicit-event-name** ⟨name⟩
:meaning ⟨string⟩
:type ⟨direction-description⟩
:phi-value ⟨expression⟩
:default-numerical-error-value ⟨numerical-value⟩
:transitions ({⟨transition⟩}$^+$) | **:marks** ({⟨string⟩}$^+$))

Die einzelnen Schlüsselwörter haben folgende Bedeutung:

:implicit-event-name ⟨name⟩
- Ordnet dem impliziten Event einen Namen (String) zu. Beachten Sie bitte, daß die Zeichen `&`, `#` und \ problematisch beim Übersetzen der LaTeX-Dokumentation sein können.

:meaning ⟨string⟩
- Dieser key wird von einem möglicherweise mehrzeiligen String gefolgt, der eine nähere Beschreibung enthält und zur Dokumentation benutzt wird. Beachten Sie bitte, daß die Zeichen `&`, `#` und \ problematisch beim Übersetzen der LaTeX-Dokumentation sein können.

:type ⟨direction-description⟩
- Die Art des Events wird festgelegt. Möglich sind uni- und bidirektionale Events, die mit entsprechenden Strings angegeben werden:

⟨direction-description⟩::=
"unidirectional" | **"bidirectional"**

:phi-value ⟨expression⟩

- Ein arithmetischer Ausdruck zur Berechnung der φ-Funktion. Sie definiert durch einen Vorzeichenwechsel den Schaltzeitpunkt des impliziten Events. Dieser Ausdruck kann von den Zuständen, Eingängen, sämtlichen Parametern, Hilfsgrößen und der Simulationszeit abhängen.

:default-numerical-error-value ⟨numerical-value⟩

- Hiermit wird der Toleranzwert ϵ_φ und damit die Genauigkeit bei der numerischen Bestimmung des Ereigniszeitpunkts festgelegt (siehe Abschnitt 3.4.1). <u>Es dürfen nur Zahlenwerte angegeben werden, Ausdrücke mit Parametern sind nicht erlaubt.</u>

Exogene explizite Ereignisse

⟨exogenous-explicit-event⟩::=
(**:explicit-event-name** ⟨name⟩
:meaning ⟨string⟩
:default-value ⟨expression⟩
:transitions ({⟨transition⟩}$^+$) | **:marks** ({⟨string⟩}$^+$))

:explicit-event-name ⟨name⟩

- Ordnet dem exogenen expliziten Event einen Namen (String) zu. <u>Beachten Sie bitte, daß die Zeichen `&`, `#` und \ problematisch beim Übersetzen der LaTeX-Dokumentation sein können.</u>

:meaning ⟨string⟩

- Dieser `key` wird von einem möglicherweise mehrzeiligen String gefolgt, der eine nähere Beschreibung enthält und zur Dokumentation benutzt wird. <u>Beachten Sie bitte, daß die Zeichen `&`, `#` und \ problematisch beim Übersetzen der LaTeX-Dokumentation sein können.</u>

:default-value ⟨expression⟩

- Weist dem expliziten Event einen Wert zu. Es können die in Kapitel 3.3.4 beschriebenen Möglichkeiten zur Wertedefinition verwendet werden.

:transitions ({⟨expression⟩}$^+$)

- Liste mit Definitionen von Transitionen die aufgrund des hier definierten Events auslösen, siehe Seite 38.

  <u>**ACHTUNG:** Dieser `key` darf nur in Verbindung mit `:event-fired-transitions` verwendet werden!</u>

:marks ({⟨string⟩}$^+$)

- Definiert eine neue Markierung des Petri-Netzes, siehe Seite 35. Die Liste enthält die Bezeichner von zuvor definierten Stellen.

  <u>**ACHTUNG:** Dieser `key` darf nur in Verbindung mit `:event-initiated-status` verwendet werden!</u>

Endogene explizite Ereignisse

⟨endogenous-explicit-event⟩::=
(**:explicit-event-name** ⟨name⟩
:meaning ⟨string⟩
:trigger-place ⟨string⟩
:default-delay-time ⟨expression⟩
:transitions ({⟨transition⟩}$^+$) | **:marks** ({⟨string⟩}$^+$))

:explicit-event-name ⟨name⟩

- Ordnet dem endogenen expliziten Event einen Namen (String) zu. Beachten Sie bitte, daß die Zeichen `&`, `#` und \ problematisch beim Übersetzen der LaTeX-Dokumentation sein können.

:meaning ⟨string⟩

- Dieser key wird von einem möglicherweise mehrzeiligen String gefolgt, der eine nähere Beschreibung enthält und zur Dokumentation benutzt wird. Beachten Sie bitte, daß die Zeichen `&`, `#` und \ problematisch beim Übersetzen der LaTeX-Dokumentation sein können.

:trigger-place ⟨string⟩

- Zuweisung der triggernden Stelle. Wird diese Stelle markiert findet dieser explizite Event mit einer Verzögerungszeitspanne statt, siehe Abschnitt 3.4.3.

:default-delay-time ⟨expression⟩

- Weist dem endogenen expliziten Event eine Verzögerungszeitspanne zu. Es können die in Kapitel 3.3.4 beschriebenen Möglichkeiten zur Wertedefinition verwendet werden.

:transitions ({⟨expression⟩}$^+$)

- Liste mit Definitionen von Transitionen die aufgrund des hier definierten Events auslösen, siehe Seite 38.

  ACHTUNG: Dieser key darf nur in Verbindung mit `:event-fired-transitions` verwendet werden!

:marks ({⟨string⟩}$^+$)

- Definiert eine neue Markierung des Petri-Netzes, siehe Seite 35. Die Liste enthält die Bezeichner von zuvor definierten Stellen.

  ACHTUNG: Dieser key darf nur in Verbindung mit `:event-initiated-status` verwendet werden!

Ein Beispiel der Definitionsfunktion für Petri-Netze zeigt einen Ausschnitt eines Petri-Netzes mit Definition zweier Stellen und Transitionen die durch implizite und explizite Ereignisse bzw direkt auslösen. Das komplette Beispiel ist in Anhang H beschrieben.

```
(define-event-schedule
  (:petrinet-name "Prozess-Steuerung"
   :meaning "Steuerung des zyklischen Batch-Prozesses"
   :places
   ;; ----------------------------------
   ;;  Stellen der Prozess-Steuerung:
   ;; ----------------------------------
   ((:place-name "Start"
     :meaning "Beginn eines Zyklus")
    (:place-name "VA1_auf"
     :meaning "Vorlage VA ablassen"
     :set-characteristic-values
     ((:real-parameter-name "J_Azu"      :value 0.005)))
     ...
    )
   ;; ----------------------------------
   ;;  Anfangsmarkierung:
   ;; ----------------------------------
   :initial-marks ("Start" "HA_aus" "HB_aus")
   :event-fired-transitions
   ;; ----------------------------------
   ;;  Ereignisse mit Transitionen:
   ;; ----------------------------------
```

```
   ((:implicit-event-name "phi1"
     :meaning "Behaelter VA leer"
     :phi-value "nA_v - 0.01"
     :type "unidirectional"
     :default-numerical-error-value 0.0001
     :transitions
     ((:from ("VA2_auf")                :to ("VA2_zu")))
    )
    ...

    (:explicit-event-name "phi7"
     :meaning "Reaktionszeit delta_T nach Kat-Zugabe abgelaufen"
     :default-delay-time 1500.0
     :trigger-place "Kat_Zugabe"
     :transitions
     ((:from ("Kat_Zugabe")             :to ("Reaktionsende"))
      (:from ("Abkuehlen")              :to ("VR_auf")))
    )
   )
   :directly-fired-transitions
   ;; ---------------------------------
   ;;  direkt ausloesende Transitionen:
   ;; ---------------------------------
   ((:from ("Start" "HA_aus" "HB_aus")   :to ("VA2_auf" "VB2_auf"))
    (:from ("VA2_zu" "VB2_zu")           :to ("VA1_auf" "VB1_auf" "Kat_Zugabe"))
    (:from ("VA1_zu" "VB1_zu")           :to ("HA_an" "HB_an"))
   )
  )
) ; end of event-schedule
```

3.5 Angaben zur Ansteuerung der Datenausgabe (process-values)

Die Ausgabe, die die für den Benutzter interessanten Informationen (wie z. B. Profile thermodynamischer Größen,...) bereitstellt, muß im CG-File über die Definition von Ausgabeformaten (sogenannten process-values) beschrieben werden. Es können Eingangs-, Zustands-, Ausgangs- oder Hilfsvariablen (evtl. mit Index und Laufbereich) sowie Integer-, Real- oder Strukturparameter zu Sinneinheiten, den Prozeßgrößen, gruppiert werden (vgl. dazu das Prozeßgrößenkonzept von DIVA [15]). Diese Prozeßgrößen werden mit der `(define-process-values ...)` Funktion konstruiert. Aufbau und Syntax von `(define-process-values ...)` (abgesehen von den unterschiedlichen keys) unterscheiden sich nicht von anderen `(define-... ...)` Funktionen[12].

Die Eingabe der process-values wird in zweierlei Hinsicht unterstützt. Zum einen können durch beliebig verschachtelte Laufbereiche indizierte Variablen ausgegeben werden. Außerdem ist es möglich, örtliche Bereiche durch Aneinandersetzung örtlicher Teilbereiche in flexibler und effizienter Weise anzugeben. Die Laufbereichsgrenzen dürfen von structure-parameters abhängen. Folgende keys stehen zur Verfügung:[13]

:name ⟨name⟩

- Der key `:name` bezeichnet die auszugebende Prozeßgröße mit einem maximal 5 Buchstaben langen String.[14] Beachten Sie bitte, daß die Zeichen `&`, `#` und \ problematisch beim Übersetzen der

[12] Vgl. Kapitel 3.3.1.

[13] Der key `:dimension` stand im Code-Generator bis zur Version Beta.2 zur Verfügung. Erst ab Version Beta.3 können örtliche Teilbereiche aneinandergesetzt werden.

[14] Diese Angabe ist jetzt zwingend erforderlich. Früher war diese Angabe optional, d.h. wenn der Name nicht explizit angegeben war, las der Code-Generator aus dem Blockdatenfile (`.DAT`-File) jeweils die ersten 5 Buchstaben des Namens der ersten auszugebenden Größe ein. Dies kann aber zu Problemen führen, wenn derselbe Zustand in zwei verschiedenen Prozeßgrößen vorkommt und diese Prozeßgrößen dann unerlaubterweise denselben Namen erhalten.

LaTeX-Dokumentation sein können.

:meaning ⟨string⟩

- Dieser key wird von einem möglicherweise mehrzeiligen String gefolgt, der eine nähere Beschreibung enthält und zur Dokumentation benutzt wird. Beachten Sie bitte, daß die Zeichen &, # und \ problematisch beim Übersetzen der LaTeX-Dokumentation sein können.

Neben den beiden keys **:name** und **:meaning** gibt es einen weiteren key, der in Abhängigkeit der örtlichen Verteilung (Dimension) der Prozeßgröße zu wählen ist:

Dimension	key
0	**:components**
1	**:x-scope**
2	**:y-scope**
3	**:z-scope**

Diese vier keys werden nachfolgend beschrieben:

:components ({⟨component⟩}$^+$)

- Hinter dem key `:components` werden die auszugebenden Komponenten (Prozeßgrößenzustände) in einer Liste aufgeführt. Man muß zwischen nicht-indizierten (≙einfachen) und indizierten Komponenten unterscheiden.

  ⟨component⟩::=
  ⟨simple-component⟩ | ⟨indexed-component⟩

1. **Einfache Komponenten:**

   ⟨simple-component⟩::=
   "⟨name⟩"
   | "⟨name⟩[⟨dimension-expression⟩ {,⟨dimension-expression⟩}*]"

   Als String wird der Name ⟨name⟩ des auszugebenden Prozeßgrößenwertes evtl. gefolgt von in eckigen Klammern ("[]") eingeschlossene Ausdrücke von Indizes ⟨dimension-expression⟩, angegeben.

   Beispiel:

```
(define-process-values
  (:name "TFluid"
   :meaning "Alle wichtigen Groessen im Fluid."
   :components (
     "Temperatur"
     "Druck"
     "Konzentration[NC-1]"
   )
  ) ;; Ende Prozessgroesse TFluid
)
```

2. **Indizierte Komponenten:**

   Für indizierte Komponenten, die also nicht nur ortsabhängig sind, stehen nun zusätzliche keys zur Verfügung:

   ⟨indexed-component⟩::=
   (:index ⟨short-index-description⟩
   :comp-scope ({⟨component⟩}$^+$ **)**
)

:index ⟨short-index-description⟩

- Den key `:index` verwendet man zur Beschreibung des ortsunabhängigen Indexes. Ihm folgt folgende drei- oder vierelementige Liste:

  ⟨short-index-description⟩::=
  (⟨index-name⟩ ⟨lower-index-boundary⟩ ⟨upper-index-boundary⟩
  [⟨step-width⟩] **)**

  Dabei benennt ⟨index-name⟩ den Index, der innerhalb den Komponenten, die durch den key `:comp-scope` angegebenen sind, auftritt. Die Laufbereichsgrenzen des Indexes werden durch ⟨lower-index-boundary⟩ und ⟨upper-index-boundary⟩ beschrieben. Die Angabe von ⟨step-width⟩ ist optional und wird bei mehrdimensionalen Prozeßgrößen (siehe **:x-scope**, **:y-scope** und **:z-scope**) verwendet, um nur jede ⟨step-width⟩-te Prozeßgröße auszugeben.

:comp-scope ({⟨component⟩}$^+$)

- In Abhängigkeit der oben beschriebenen Indizes und der örtlichen Ausdehnung können die einzelnen Komponenten angegeben werden. Dazu verwendet man den key `:comp-scope`. Sein Argument ist eine Liste von einfachen oder wieder indizierten Komponenten.

  Folgendes Beispiel definiert die Komponenten `Temperatur`, alle `Konz[1]` bis `Konz[NC]` und `Druck`:

```
(define-process-values
  (:name "TFluid"
   :meaning "Alle wichtigen Groessen im Fluid."
   :components (
     "Temperatur"
     (:index ("i" 1 "NC")
      :comp-scope (
        "Konz[i]"
     ))
     "Druck"
   )
  ) ;; Ende Prozessgroesse TFluid
)
```

:x-scope ({⟨x-part⟩}$^+$)

- Der key `:x-scope` beschreibt eine örtlich 1-dimensionale Prozeßgröße und wird von einer Liste mit Teilbereichsangaben ⟨x-part⟩ gefolgt:

  ⟨x-part⟩::=
  (:index ⟨short-index-description⟩
  :components ({⟨component⟩}$^+$)
)

  Folgendes Beispiel definiert einen örtlich 1-dimensionalen Temperaturbereich, der aus einem Heizungs- und Reaktionsbereich zusammengesetzt ist. In jedem Teilbereich müssen jeweils gleich viele Komponenten, hier die Temperatur, ausgegeben werden.

```
(define-process-values
  (:name "KFBR2"
   :meaning "Temperatur im Heizungsrohr und anschliessendem Reaktionsrohr."
   :x-scope (
     (:index ("x" 1 "N1")
      :components ( "THeiz[x]" )
```

```
        )
        (:index ("x" 1 "N2")
         :components ( "TReak[x]" )
        )
      ) ;; Ende x-scope
    ) ;; Ende Prozessgroesse KFBR2
  )
```

Wenn der Strukturparameter `NSTEP` im folgenden Beispiel den Wert 2 besitzt, so wird `THeiz` nur an jeder dritten Ortsstützstelle ausgegeben, also `THeiz[1]`, `THeiz[4]`, `THeiz[7]`, ...

```
(define-process-values
  (:name "KFBR2"
   :meaning "Temperatur im Heizungsrohr und anschliessendem Reaktionsrohr."
   :x-scope (
     (:index ("x" 1 "N1" "NSTEP + 1")
      :components ( "THeiz[x]" )
     )
   ) ;; Ende x-scope
  ) ;; Ende Prozessgroesse KFBR2
)
```

:y-scope ({⟨y-scope⟩}$^+$)

- Der key `:y-scope` beschreibt eine örtlich 2-dimensionale Prozeßgröße und wird von einer Liste mit Teilbereichsangaben ⟨y-part⟩ gefolgt:

  ⟨y-part⟩::=
  (:index ⟨short-index-description⟩
  :x-scope ({⟨x-part⟩}$^+$)
)

:z-scope ({⟨z-scope⟩}$^+$)

- Der key `:z-scope` beschreibt eine örtlich 3-dimensionale Prozeßgröße und wird von einer Liste mit Teilbereichsangaben ⟨z-part⟩ gefolgt:

  ⟨z-part⟩::=
  (:index ⟨short-index-description⟩
  :y-scope ({⟨y-part⟩}$^+$)
)

Beispiel für 2 örtliche Dimensionen und wie die Reihenfolge der Angabe von ⟨dimension-expression⟩ als Indizes die Reihenfolge der auszugebenden Prozeßgrößenzustände beeinflußt.

```
(define-process-values
  (:name "TFluid"
   :meaning "Alle wichtigen Groessen im Fluid."
   :y-scope (
     (:index ("y" 1 2)
      :x-scope (
        (:index ("x" 1 3)
         :components (
           "Temp[x,y]"
           "Temp[y,x]"
        ))
      ) ;; Ende x-scope
     )
```

```
    ) ;; Ende y-scope
  )
)
```

Bewirkt folgende Ausgabe:

```
"Temp[x,y]"
      "Temp[y,x]"
-----------------
Temp[1,1]
          Temp[1,1]
Temp[2,1]
          Temp[1,2]
Temp[3,1]
          Temp[1,3]
Temp[1,2]
          Temp[2,1]
Temp[2,2]
          Temp[2,2]
Temp[3,2]
          Temp[2,3]
```

3.6 Angaben zu Modellgleichungen

Bevor die Syntaxbeschreibung zur Angabe von Modellgleichungen und Zuweisungen an Hilfsvariablen erfolgt, soll eine kurze Übersicht über einige Bausteine zum Aufbau der Modellgleichungen gegeben werden.

3.6.1 Bausteine zum Aufbau der Modellgleichungen

Die Gleichungen müssen nach einer bestimmten Form aufgebaut sein, wobei folgende Bausteine verwendet werden können.

- Summen und Produkte sind in kompakter Form erlaubt.

$$\sum_{Laufindex=Untergrenze}^{Obergrenze} Term \tag{24}$$

$$\prod_{Laufindex=Untergrenze}^{Obergrenze} Term \tag{25}$$

  Obige Summe bzw. Produkt wird mit `:sum` bzw. `:prod` angegeben. Für die Unter- und Obergrenze dürfen nur Strukturparameter oder Integerwerte verwendet werden.

$$\texttt{:sum<}Laufindex, Untergrenze, Obergrenze\texttt{>}Term \tag{26}$$

$$\texttt{:prod<}Laufindex, Untergrenze, Obergrenze\texttt{>}Term \tag{27}$$

- Konditionale Ausdrücke (conditionals) werden z. B. für unstetige Anteile $\hat{b}_{ij}$ bzw. $\hat{f}_i$ der B-Matrix bzw des f-Vektors oder zusammengesetzte Parameterkennlinien benötigt und dürfen innerhalb von Termen verwendet werden.

$$\left\{\begin{array}{ll} Term_1 & \text{, falls } Bedingung_1 \\ Term_2 & \text{, falls } Bedingung_2 \\ \vdots & \vdots \\ Term_n & \text{, falls } Bedingung_n \\ Term_{sonst} & \text{, sonst} \end{array}\right\} \tag{28}$$

Dieser Ausdruck nimmt folgenden Wert an: Es wird der Reihe nach $Bedingung_1$ bis $Bedingung_n$ geprüft. Sobald eine $Bedingung_i$ erfüllt ist, erhält der Ausdruck den Wert von $Term_i$ und es wird gestoppt. Ist keine *Bedingung* erfüllt erhält der Ausdruck den Wert von $Term_{sonst}$.
Obiges conditional wird angegeben durch:

$$\begin{array}{ll} \text{:cond(} & Bedingung_1 \, , Term_1 \; , \\ & Bedingung_2 \, , Term_2 \; , \\ & \quad \vdots \qquad\quad \vdots \\ & Bedingung_n \, , Term_n \; , \\ & Term_{sonst} \qquad\quad) \end{array} \tag{29}$$

ACHTUNG: Die zu einem conditional gehörenden Patterns werden je nach Optimierungsstufe der Patternbelegung unterschiedlich bestimmt, siehe Kapitel 4.2 Optionen für `cg-fortran`.

- Optimierungsstufe 1–3: Berücksichtigt wird die Vereinigungsmenge der zu allen $Term_i$ gehörenden Patterns, ungeachtet aller $Bedingung_i$. Der Ausdruck `:cond(i==0, 2.3, x[i])` ergibt das Pattern `x[i]`, was für folgenden Fall inkorrekt ist: `i` nimmt den Wert 0 an und `x[j]` ist nur für $j = 1, ..., 5$ definiert.
- Optimierungsstufe ≥ 4: Die Pattern werden unter Berücksichtigung aller $Bedingung_i$ bestimmt, sofern $Bedingung_i$ nur von Strukturparametern oder numerischen Werten abhängt. Das o.g. Beispiel ergäbe somit für den kritischen Wert von $i = 0$ keinen Patterneintrag und führt zu keinen falschen Pattern.

Zur Formulierung der *Bedingung*, siehe Beispiel (Seite 72) und Syntax von Vergleichsoperatoren (Seite 77).

- Indizierte Gleichungen sind — auch in geschachtelter Form — erlaubt:

$$Laufindex = Untergrenze \ldots Obergrenze:$$

$$\quad Term = Term \tag{30}$$

$$\quad Laufindex = Untergrenze \ldots Obergrenze:$$

$$\qquad Term = Term \tag{31}$$

$$\quad Term = Term \tag{32}$$

- Für die Berechnung von Zwischenergebnissen, die in Hilfsvariablen gespeichert werden, haben Zuweisungen an Hilfsvariablen "links" vom Gleichheitszeichen die Hilfsvariable stehen und "rechts" einen Term.

$$Hilfsvariable := Term \tag{33}$$

- Die mathematischen Terme werden vom Code-Generator intern in der Präfixnotation gespeichert, da diese Repräsentationsform für die Verarbeitung am geeignetsten ist. Der Code-Generator akzeptiert arithmetische Ausdrücke sowohl in Präfixnotation als auch in Infixnotation innerhalb eines Strings (s. Anhang C). Z. B.:

```
":sin y + :sum<i, 1, nc - 2> x[i]"                ;; Infixnotation
(+ (sin "y") (sum "i" 1 (- "nc" 2) (aref "x" "i")))  ;; Praefixnotation
```

Folgendes Beispiel zeigt, welche Art von Gleichungen angegeben werden kann.

$$i = 1 \ldots N_C - 1: \tag{34}$$

$$n \sum_{j=3}^{N_C+2} \frac{dx_{zu_{i,j}}}{dt} + \left(\sum_{k=j+2}^{N_C-2i} x_{i,j,k^2} \right) \frac{dn}{dt} = P_{ab} x_{zu_{i,3}} - g \left\{ \begin{array}{ll} 0 & \text{, falls } T \leq 273 \\ k_1 p^2 & \text{, falls } (T > 0) \wedge (T < T_m) \\ k_2 - e^{-k_3 p} & \text{, sonst} \end{array} \right\} T \tag{35}$$

3.6.2 Eingabe von Gleichungen

Die Eingabe der Gleichungen erfolgt über eine `(define-equations ...)` Funktion[15].

(**define-equations** {⟨equation-or-assignment⟩}$^+$)

Das linear implizite Differential-Algebra-Gleichungssystem (1) auf Seite 1 lautet ausführlich:

$$
\begin{array}{l}
b_{11}\dot{x}_1 + b_{12}\dot{x}_2 + \cdots + b_{1j}\dot{x}_j + \cdots + b_{1n}\dot{x}_n = f_1 \\
b_{21}\dot{x}_1 + b_{22}\dot{x}_2 + \cdots + b_{2j}\dot{x}_j + \cdots + b_{2n}\dot{x}_n = f_2 \\
\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots \\
b_{i1}\dot{x}_1 + b_{i2}\dot{x}_2 + \cdots + b_{ij}\dot{x}_j + \cdots + b_{in}\dot{x}_n = f_i \\
\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots \\
b_{n1}\dot{x}_1 + b_{n2}\dot{x}_2 + \cdots + b_{nj}\dot{x}_j + \cdots + b_{nn}\dot{x}_n = f_n
\end{array}
$$

Bei der Eingabe wird unterschieden zwischen:

einfachen (nicht-indizierten) Gleichungen: Eine Zeile von (3.6.2).

indizierten Gleichungen: Eine kompakte Angabe mehrerer, aufeinanderfolgender Zeilen von (3.6.2) durch Formulierung in Abhängigkeit eines Zeilenindexes.

Zuweisungen an Hilfsgrößen: Bildung von Zwischenwerten aus Teilausdrücken.

⟨equation-or-assignment⟩::=
⟨simple-equation⟩ | ⟨indexed-equation⟩ | ⟨assignment⟩

1. **Einfache Gleichungen:**
   Für die Angabe einer einfachen Gleichung stehen folgende keys zur Verfügung:

   ⟨simple-equation⟩::=
   (**:equation-name** ⟨name⟩
   :meaning ⟨string⟩
   :f-value ⟨expression⟩
   :provided-pattern ⟨expression⟩
   :b-values ({⟨b-value-or-assignment⟩}$^+$))
)

   :equation-name ⟨name⟩

   - Ordnet der jeweiligen Gleichung einen Namen (String) zu. Dieser Name stellt eine Kurzbezeichnung der Gleichung dar und sollte nur ein oder zwei Stichworte oder eine Nummer enthalten, wie z. B. `"KMB"`, `"Komponentenmaterialbilanz"`, `"Druck Zulauf"` oder `"(7a)"`. Für eine genauere Beschreibung ist der key **:meaning** vorgesehen. Beachten Sie bitte, daß die Zeichen `&`, `#` und \ problematisch beim Übersetzen der LaTeX-Dokumentation sein können.

   :meaning ⟨string⟩

   - Dieser key wird von einem möglicherweise mehrzeiligen String gefolgt, der eine nähere Beschreibung enthält und zur Dokumentation benutzt wird. Beachten Sie bitte, daß die Zeichen `&`, `#` und \ problematisch beim Übersetzen der LaTeX-Dokumentation sein können.

[15] Die Funktion unterscheidet sich nicht von anderen `(define-... ...)` Funktionen, abgesehen von unterschiedlichen keys. Vgl. auch Anhang A.

:f-value ⟨expression⟩

- Definiert die rechte Seite des Differentialgleichungssystems. Wollte man z. B. die i-te Gleichung des obigen Differentialgleichungssystems beschreiben, müßte jetzt hier der Ausdruck für f_i aufgeführt werden.

  ACHTUNG: Real-Werte müssen mit Punkt und mindestens einer Nachkommastelle angegeben werden. Es genügt nicht wie in FORTRAN z. B. `"2."` anzugeben, sondern es muß immer `"2.0"` geschrieben werden. Es wird außerdem davor gewarnt `"2/3"` in **:f-value** oder **:b-value** zu schreiben, da dies in FORTRAN als Integer-Division behandelt wird und gleich Null wird.

:provided -pattern ⟨expression⟩

- Der hier eingegebene Ausdruck wird anstatt des beim key **f-value** eingebenen Ausdrucks zur Bestimmung der Patternbelegung verwendet. Wird jedoch **nil** eingegeben, erfolgt die Patternbestimmung wie bisher. Dieser key ist optional und sollte nur in Ausnahmefällen bei starker Mehrfachbelegung der Pattern verwendet werden. Hinweis: Zur Überprüfung der Patternbelegung kann der DIVA-Befehl **testpat** verwendet werden.

:b-values ({⟨b-value-or-assignment⟩}$^+$)

- Dieser key ordnet der Differentialgleichung die linke Seite zu. Das Argument des keys ist eine Liste mit allen von Null verschiedenen zur Gleichung gehörigen Einträgen der Linke-Seite-Matrix B. Für die i-te Gleichung müßten alle Terme $b_{ik} \neq 0,\ k = 1, 2, ..., n$ in die Liste aufgenommen werden. Analog zu den Gleichungsangaben wird auch bei den B-Matrix-Einträgen unterschieden zwischen:

  Einfachen B-Matrix-Einträgen: Ein Term b_{ik} als Linearfaktor zu $\dot{x}_j$.

  Indizierten B-Matrix-Einträgen: Eine kompakte Angabe mehrerer, B-Matrix-Einträge b_{ik} durch Formulierung in Abhängigkeit eines Spaltenindexes $k = k_1, k_1 + 1, ..., k_2$.

  Zuweisungen an Hilfsgrößen: Bildung von Zwischenwerten aus Teilausdrücken.

  ⟨b-value-or-assignment⟩::=
  ⟨simple-b-value⟩ | ⟨indexed-b-value⟩ | ⟨assignment⟩

(a) **Einfache B-Matrix-Einträge:**
Ein einfacher B-Matrix-Eintrag wird durch zwei keys angegeben:

⟨simple-b-value⟩::=
(**:state-variable-reference** ⟨symbol⟩
:b-value ⟨expression⟩
:provided-pattern⟨expression⟩)

:state-variable-reference ⟨symbol⟩

- Das Argument gibt die dem Term b_{ij} zugehörige, nach der Zeit abzuleitende, Zustandsvariable x_i an.

:b-value ⟨expression⟩

- Das Argument beschreibt den Term b_{ij}.

:provided-pattern ⟨expression⟩

- Der hier eingegebene Ausdruck wird anstatt des beim key **b-value** eingebenen Ausdrucks zur Bestimmung der Patternbelegung verwendet. Wird jedoch **nil** eingegeben, erfolgt die Patternbestimmung wie bisher. Dieser key ist optional und sollte nur in Ausnahmefällen bei starker Mehrfachbelegung der Pattern verwendet werden. Hinweis: Zur Überprüfung der Patternbelegung kann der DIVA-Befehl **testpat** verwendet werden.

(b) **Indizierte B-Matrix-Einträge:**

Oftmals ist es zweckmäßig die linke Seite indiziert anzugeben. Hängen zum Beispiel die b_{ik} ($k = 1, 2, ..., n$) in der i-ten Gleichung von k ab, ist eine indizierte Darstellung übersichtlicher und sogar erforderlich, wenn die Anzahl der anzugebenden b_{ik} von einem Strukturparameter abhängt.[16]

⟨indexed-b-value⟩::=
(**:index** ⟨short-index-description⟩
:meaning ⟨string⟩
:b-elem-scope ({⟨b-value-or-assignment⟩}$^+$))

:index ⟨short-index-description⟩

- Der Laufbereich wird durch die nachfolgende Liste (bestehend aus dem Laufvariablennamen, der oberen und unteren Laufgrenze) beschrieben. Die durch den `key` beschriebene Laufvariable ist innerhalb des `:b-elem-scope key` gültig.

  ⟨short-index-description⟩::= (⟨index-name⟩ ⟨lower-boundary⟩ ⟨upper-boundary⟩)

  ⟨index-name⟩::=⟨string⟩

  ⟨lower-boundary⟩::=⟨expression⟩

  ⟨upper-boundary⟩::=⟨expression⟩

:meaning ⟨string⟩

- Dieser `key` wird von einem möglicherweise mehrzeiligen String gefolgt, der eine nähere Beschreibung enthält und zur Dokumentation benutzt wird. Beachten Sie bitte, daß die Zeichen `&`, `#` und \ problematisch beim Übersetzen der LaTeX-Dokumentation sein können.

:b-elem-scope ({⟨b-value-or-assignment⟩}$^+$)

- Dieser `key` beschreibt in Abhängigkeit des oben definierten Indexes die Elemente der Linke-Seite-Matrix B. Diese Elemente können einen Eintrag beschreiben (⟨simple-b-value⟩) oder wiederum indizierte Elemente von B (⟨indexed-b-value⟩).

Mit diesen `keys` werden dem Code-Generator alle Informationen über die Differentialgleichung angegeben. Dies wird nachfolgend für die Eingabe eines PT_2-Systems gezeigt:

$$\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} \dot{x}_1 \\ \dot{x}_2 \end{pmatrix} = \begin{pmatrix} x_2 \\ -a_1 x_2 - a_0 x_1 + u_1 \end{pmatrix} \tag{36}$$

Die Eingabe lautet wie folgt :

```
(define-equations
  (:equation-name "Gleichung1"
   :meaning "Die Gleichung lautet:
             1.0 * :diff(x1,t) == x2"
   :f-value "x2"
   :b-values ((:state-variable-reference "x1"
                :b-value 1.0))
  )
  (:equation-name "Gleichung2"
   :meaning "Zu der Gleichung gehoert nur ein B-Matrix-Eintrag."
   :f-value "- a1 * x2 - a0 * x1 + u1"
   :b-values ((:state-variable-reference "x2"
                :b-value 1.0))
```

[16]In Kapitel 3.6.3 werden zwei hier nicht aufgeführte, optionale `keys` erläutert.

```
  )
) ;; end of equations
```

Ändert sich beispielsweise die zweite Gleichung von (36) zu

$$5x_1\dot{x}_1 + \dot{x}_2 = -a_1x_2 - a_0x_1 + u_1 \quad , \tag{37}$$

dann bedeutet dies für die Eingabe der "Gleichung2":

```
...
  (:equation-name "Gleichung2"
   :meaning "Zu dieser Gleichung gehoeren zwei B-Matrix-Eintraege."
   :f-value "- a1 * x2 - a0 * x1 + u1"
   :b-values ((:state-variable-reference "x1"      ;; beschreibt b_2_1
               :b-value "5.0 * x1")
              (:state-variable-reference "x2"      ;; beschreibt b_2_2
               :b-value 1.0))
  )
...
```

ACHTUNG: Die Reihenfolge für die B-Matrix-Einträge $5x_1$ und 1 muß beachtet werden. Ein zu $\dot{x}_1$ gehörender Eintrag muß vor einem zu $\dot{x}_2$ gehörenden Eintrag angegben werden, wenn x_1 vor x_2 bei der Definition der Zustände angegeben wurde. Der Grund liegt darin, daß die in DIVA verwendete Numerik diese Reihenfolge benötigt und eine Umsortierung der angegebenen B-Matrix-Elemente, die bei jeder Funktionsauswertung, d. h. i. a. mehrmals bei jedem Integrationsschritt durchzuführen wäre, aus Effizienzgründen momentan nicht vorgenommen wird. Diese Sortierung kann auch vom Code-Generator nicht vorgenommen werden. Dies liegt an der Nicht-Entscheidbarkeit, ob z. B. x_{m+n} vor x_{2m-n} liegt, wenn die Werte der Strukurparameter m und n nicht bekannt sind (array disambiguity).

Ein Beispiel für indizierte B-Matrix-Einträge ist in dem Beispiel für indizierte Gleichungen enthalten.

Beabsichtigt man nur eine algebraische Gleichung, d. h. $0 = f_i$, einzugeben, so sind b_{i1},b_{i2},...,b_{in} alle 0, und man kann auf die Eingabe von `:b-values` verzichten.

2. **Indizierte Gleichungen:**
Indizierte Gleichungen werden vom Code-Generator durch einen zusätzlichen key unterstützt. Wie bei den indizierten Elementen der Linke-Seite-Matrix B wird die Indizierung durch den key `:index` und zusätzliche keys beschrieben.[17]

⟨indexed-equation⟩::=
(**:index** ⟨short-index-description⟩
:meaning ⟨string⟩
:help-var-accumulate ⟨accumulate⟩
:help-var-remember ⟨remember⟩
:eq-scope ({⟨equation-or-assignment⟩}$^+$))

:index ⟨short-index-description⟩

- die nachfolgende Liste (bestehend aus dem Laufvariablennamen, der oberen und unteren Laufgrenze) beschreibt den Laufbereich[18]. Die durch den Laufbereich definierte Laufvariable ist innerhalb des `:eq-scope` key gültig.

:meaning ⟨string⟩

- Dieser key wird von einem möglicherweise mehrzeiligen String gefolgt, der eine nähere Beschreibung enthält und zur Dokumentation benutzt wird. Beachten Sie bitte, daß die Zeichen `&`, `#` und \ problematisch beim Übersetzen der LaTeX-Dokumentation sein können.

[17] In Kapitel 3.6.3 werden zwei hier nicht aufgeführte, optionale keys erläutert.
[18] Vgl. ⟨indexed-b-value⟩ auf Seite 49.

:help-var-accumulate ⟨accumulate⟩

- Wird in Abschnitt 3.6.3 bei „zu merkende Hilfsvariablen“ beschrieben.

:help-var-remember ⟨remember⟩

- Wird in Abschnitt 3.6.3 bei „Akkumulierungs-Hilfsvariablen“ beschrieben.

:eq-scope ({⟨equation-or-assignment⟩}$^+$)

- Die Differentialgleichungen können nun hier in Abhängigkeit des oben definierten Indexes eingegeben werden. Mehrfache Indizierungen realisiert man durch eine weitere ⟨indexed-equation⟩-Angabe.

  Beispiel:
  Es soll eine einfache und eine mit $k = 1, 2, ..., n-1$ indizierte Differentialgleichung eingegeben werden:

$$\dot{x}_1 = x_2 \tag{38}$$

$$k = 1, 2, ..., n-1: \quad \sum_{i=k+1}^{n} b_i \dot{x}_i = k \cdot x_k \tag{39}$$

```
(define-equations
  (:equation-name "einfache Gl."
   :meaning "Gleichung 1.0*:diff(x[1],t) == x[2]"
   :f-value "x[1]"
   :b-values ((:state-variable-reference "x[1]"
               :b-value 1.0))
  )                                      ;Ende der einfachen Gleichung
  (:index ("k" 1 "n - 1")               ;Definition von Index k=1,...,n-1
   :meaning "Die folgende Gleichung gilt fuer alle k=1,...,n-1"
   :eq-scope (                          ;Beginn Gueltigkeitsbereich Index k
     (:equation-name "indizierte Gl."
      :meaning "Gl. mit indizierten B-Matrix-Elementen"
      :f-value "k * x[k]"
      :b-values ((:index ("i" "k + 1" "n")
                  :meaning "Index i zur Beschreibung der B-Matrix-Elemente"
                  :b-elem-scope (        ;Beginn Gueltigkeitsbereich Index i
                    (:state-variable-reference "x[i]"
                     :b-value "b[i]"))   ;Ende Gueltigkeitsbereich Index i
                 ))                      ;Ende Index i und b-values
     ))                                  ;Ende der indizierten Gleichung
                                         ;     und Gueltigkeitsbereich Index k
  )                                      ;Ende Index k
) ;; end of equations
```

3.6.3 Günstige Gleichungsangabe und Zuweisung an Hilfsvariablen

Der Code-Generator führt keine Optimierung hinsichtlich der Laufzeit des erzeugten Codes oder aus numerischen Gesichtspunkten durch, indem Modellgleichungen oder ihre Reihenfolge abgeändert werden. Die Modellgleichungen können aber in günstiger Weise angegeben werden, so daß sie in einen optimalen FORTRAN-Code umgesetzt werden. Diese günstige Form kann entweder direkt vom Modellierer kommen oder durch Zwischenschaltung eines noch zu entwickelnden Optimierungswerkzeuges erreicht werden.

Konstante Ausdrücke sollten möglichst weit ausgeklammert werden:

$$\text{statt: } \sum_{i=1}^{n}(2 \cdot x_i) \qquad \text{besser: } 2 \cdot \sum_{i=1}^{n} x_i \tag{40}$$

Invariante Gleichungen oder Terme sollten aus indizierten Gleichungen herausgezogen werden:

$$\text{statt:} \qquad i = 1, ..., n: \quad 2\dot{x}_i = \sin(x_0 + x_1^2) - x_{i+1} \tag{41}$$

$$\text{besser:} \qquad H := \sin(x_0 + x_1^2) \tag{42}$$

$$i = 1, ..., n: \quad 2\dot{x}_i = H - x_{i+1} \tag{43}$$

Hierbei ist H eine Hilfsvariable, die eine Mehrfachberechnung eines Terms vermeidet. Ebenso kann die Einführung einer Hilfsvariablen die Lesbarkeit eines großen Terms vereinfachen. Ein und derselben Hilfsvariablen können zur Speicherplatzeinsparung auch mehrmals hintereinander verschiedene Werte zugewiesen werden.

Zuweisungen an Hilfsvariablen dürfen überall dort stehen, wo auch eine Gleichung (⟨simple-equation⟩, ⟨indexed-equation⟩) oder ein B-Matrix-Eintrag (⟨simple-b-value⟩, ⟨indexed-b-value⟩) vorkommen dürfen. Eine Zuweisung wird folgendermaßen angeben:

⟨assignment⟩::=
(**:assignment-name** ⟨string⟩
:meaning ⟨string⟩
:help-variable ⟨help-variable⟩
:h-value ⟨expression⟩
:provided-pattern⟨expression⟩)

Die keys haben folgende Bedeutung:

:assignment-name ⟨string⟩

- Kurzbezeichnung für diese Zuweisung. Analog zum **:equation-name** ist eine wesentlich kürzere Bezeichnung zu wählen als bei der Beschreibung nach dem key **:meaning**. Im einfachsten Fall kann die Kurzbezeichnung der Name der zu berechnenden Hilfsvariable sein. <u>Beachten Sie bitte, daß die Zeichen `&`, `#` und \ problematisch beim Übersetzen der LaTeX-Dokumentation sein können.</u>

:meaning ⟨string⟩

- Dieser key wird von einem möglicherweise mehrzeiligen String gefolgt, der eine nähere Beschreibung enthält und zur Dokumentation benutzt wird. <u>Beachten Sie bitte, daß die Zeichen `&`, `#` und \ problematisch beim Übersetzen der LaTeX-Dokumentation sein können.</u>

:help-variable ⟨help-variable⟩

- Die Hilfsvariable, der ein Wert des Ausdrucks **:h-value** zugewiesen werden soll. Ist die Hilfsvariable indiziert, so muß hinter dem Namen der Hilfsvariablen eine Liste der Indexnamen in eckigen Klammern und durch Kommata getrennt stehen. Im Ausdruck **:h-value** können diese Indexnamen dann verwendet werden.

:h-value ⟨expression⟩

- Dieser Ausdruck wird berechnet und der Hilfsvariablen zugewiesen. Als Indexnamen dürfen die Bezeichnungen verwendet werden, die bei **:help-variable** dem Namen der Hilfsvariablen folgend in eckigen Klammern angegeben worden sind.

:provided-pattern ⟨expression⟩

- Der hier eingegebene Ausdruck wird anstatt des beim key **h-value** eingebenen Ausdrucks zur Bestimmung der Patternbelegung verwendet. Wird jedoch **nil** eingegeben, erfolgt die Patternbestimmung wie bisher. Dieser key ist optional und sollte nur in Ausnahmefällen bei starker Mehrfachbelegung der Pattern verwendet werden. Hinweis: Zur Überprüfung der Patternbelegung kann der DIVA-Befehl **testpat** verwendet werden.

  Beispiel:

  Bei der Gleichungsdefinition erfolgt zuerst eine Zuweisung an alle Elemente der indizierten Hilfsvariablen `h`. Obwohl keine explizite Indexbereichs-Angabe bei der Zuweisung steht, wird `h[i]` für `i=1,...,NC` vollständig belegt. Die Hilfsvariable `h` wird dann in der nachfolgenden Gleichung (Gl3) im `:f-value` und in den `:b-values` benutzt. Die Hilfsvariable `h_ge` wird zwischen den Angaben zu den `:b-values` zugewiesen, direkt vor der ersten Benutzung im darauffolgenden `:b-value`.

```
(define-help-variables
  (:name "h"
   :meaning "molare Reinstoffenthalpie der Komponente"
   :indices ((:lower-boundary 1 :upper-boundary "NC"))
  )
  (:name "h_ge"
   :meaning "molare Gemischenthalpie"
  )
) ;; end of help-variables

(define-equations
  (:assignment-name "Enthalpie-Berechnung h[i]"
   :meaning "Jedem Element h[i] des Vektors h wird der Wert von h-value
             zugewiesen. D. h. es wird automatisch eine Indexbereichs-Angabe
             :index i = 1,...,NC angenommen !"
   :help-variable "h[i]"
   :h-value "(A[i] + (B[i] + C[i]*T)*T)*T*4.186"
  )
  (:equation-name "(Gl3)"
   :meaning "Die Hilfsvariable h[1] wird im Element des
             Rechte-Seite-Vektors f benutzt."
   :f-value "h[1] + 34.0"
   :b-values ((:index ("i" 1 "NC - 1")
               :meaning "Die Hilfsvariable h[i] wird in Elementen der
                         Linke-Seite-Matrix B benutzt."
               :b-elem-scope ((:state-variable-reference "x[i]"
                               :b-value "2.7*(h[i] - h[NC])"))
              )
              (:assignment-name "Enthalpie-Berechnung h_ge"
               :meaning "Berechnung der Gesamt-Enthalpie h_ge direkt vor ihrer
                         Benutzung im Element der Linke-Seite-Matrix B"
               :help-variable "h_ge"
               :h-value ":sum<i,1,NC>(h[i]*x[i])"
              )
              (:state-variable-reference "T"
               :b-value "h_ge + 2.0 + :prod<i,22,23>(x[i])"))
  )
) ;; end of equations
```

Die Gleichungsdefinition lautet in anderer Schreibweise:

$$\begin{aligned} i = 1, ..., NC: \quad h_i &:= (A_i + (B_i + C_i \cdot T) \cdot T) \cdot T \cdot 4.186 \\ \left(\sum_{i=1}^{NC-1} 2.7 \cdot (h_i - h_{NC}) \cdot \dot{x}_i \right) + \left(h_{ge} + 2.0 \cdot + \left(\prod_{i=22}^{23} x_i \right) \right) \cdot \dot{T} &= h_1 + 34.0 \\ mit \quad h_{ge} &:= \sum_{i=1}^{NC-1} (h_i \cdot x_i) \end{aligned}$$

Bei der Verwendung von indizierten Gleichungen $f_p, ..., f_q$ — die z. B. bei der Ortsdiskretisierung von partiellen Differentialgleichungen (örtlich verteilte Prozeßmodelle) oder bei der Diskretisierung einer Eigenschaftskoordinate (populationsdynamische Modelle) entstehen können — sind zwei spezielle Arten von Hilfsvariablen erlaubt:

zu merkende Hilfsvariablen deren Wert für eine Gleichung f_i berechnet wird und für n vorige Gleichungen $f_{i-1}, f_{i-2}, ..., f_{i-n}$ gespeichert wird.

Akkumulierungs-Hilfsvariablen zur Summation einer Größe, die für jede Gleichung f_i $(i = p, ..., q)$ berechnet wird.

1. **zu merkende Hilfsvariablen:**
   Die folgenden Beispiele von Gleichungsformulierungen, sollen die Verwendung dieser Art von Hilfsvariablen erläutern.

$$\begin{aligned} &i = 2, ..., n-1: \\ &\qquad hl := a \cdot x_i - b \cdot x_{i-1} && (44) \\ &\qquad hr := a \cdot x_{i+1} - b \cdot x_i && (45) \\ &\qquad \dot{x}_i = c \cdot hl + d \cdot hr && (46) \end{aligned}$$

   Hierbei ist für die Konzentration x_i an der i-ten Ortsstützstelle eine Differentialgleichung für die Ortsstützstellen 2 bis $n-1$ gegeben. a, b, c, d sind Parameter und hl, hr Hilfsvariablen. Da sich die Berechnung von hl und hr nur durch die Ortsstützstelle i unterscheidet, kann hl für die Ortsstützstelle i aus hr der Ortsstützstelle $i-1$ durch Umspeichern erhalten werden:

$$\begin{aligned} &hl := a \cdot x_2 - b \cdot x_1 && (47) \\ &i = 2, ..., n-1: \\ &\qquad hr := a \cdot x_{i+1} - b \cdot x_i && (48) \\ &\qquad \dot{x}_i = c \cdot hl + d \cdot hr && (49) \\ &\qquad hl := hr && (50) \end{aligned}$$

   Diese Formulierung ermöglicht eine effizientere Berechnung als die vorige. Allerdings gestaltet sich die Patternbestimmung schwieriger, da die zeitliche Zuweisungsreihenfolge an die Hilfsvariablen durch die Schleifenangabe $i = 2, ..., n-1$ beeinflußt wird. Im allgemeinen kann durch mehrfache Umspeicherungen von Hilfsgrößen die Patternbestimmung so kompliziert werden, daß hier folgende Konvention eingehalten werden muß: Wird einer Hilfsvariablen innerhalb einer Schleifenangabe ein Wert zugewiesen, so muß diese Zuweisung vor der ersten Verwendung dieser Hilfsvariablen stehen. Im obigen Beispiel wird hl am Schleifenende (50) berechnet, aber schon in Gleichung (49) verwendet. Dieser Sachverhalt wird vom Code-Generator durch eine Abhängigkeitsanalyse (welche Hilfsgrößen einen Initialwert besitzen und in die Berechnung einer anderen Hilfsgröße eingehen) erkannt, und der Benutzer wird darauf hingewiesen, daß daraus eine falsche Patternbelegung folgen kann. Um die Patternbestimmung korrekt durchzuführen, muß dem Code-Generator mitgeteilt werden, daß der Wert von hr für den nächsten Schleifendurchlauf in hl gemerkt werden soll. Die Gleichung (50) wird vom Benutzer weggelassen und vom Code-Generator selbständig hinzugefügt. Folgende Gleichungsangabe ist für den Code-Generator:

```
(define-equations
  (:index ("i" 2 "n - 1")
   :meaning "Remember Schleifendefinition"
   :help-var-remember
   (("hr" :sequentially-in                         ; Merke hr in hl
          (("hl" :with-init "a*x[2] - b*x[1]"))    ; Berechnung hl vor Schleife
   ))
   :eq-scope
   ((:assignment-name "Groesse hr"
     :meaning "Remember-Hilfsvariable"
     :help-variable "hr"
     :h-remember "a*x[i+1] - b*x[i]"               ; h-remember statt h-value
    )
    (:equation-name "Dgl. fuer x[i]"
     :meaning "Verwendung von hl und hr innerhalb Schleife"
     :b-values ((:state-variable-reference "x[i]" :b-value "1.0"))
     :f-value "c*hl + d*hr")
   ))
) ;; end of equations
```

Zu merkende Hilfsvariablen werden durch den `key` `:help-var-remember` (siehe auch Abschnitt 3.6.2 bei „indizierte Gleichungen") angegeben:

:help-var-remember ⟨remember⟩

- Nach diesem `key` folgt eine Liste von Tripeln (dreielementige Listen). Diese Tripel bestehen aus dem Namen der zu merkenden Hilfsvariable, dem `key` `:sequentially-in` und der Liste von Angaben, in welchen Hilfsvariablen Werte zu speichern sind:

  ⟨remember⟩::=
  ({ (⟨string⟩ **:sequentially-in** (⟨remember-in⟩*)) }*
)

  Die Angaben, in welchen Hilfsvariablen Werte zu speichern sind, sind wiederum Tripel. Sie bestehen aus dem Namen der Hilfsvariablen, die den gespeicherten Wert aufnehmen soll, dem `key` `:with-init` und dem Berechnungsausdruck für die Initialisierung der Hilfsvariablen vor dem Schleifendurchlauf:

  ⟨remember-in⟩::=
  (⟨string⟩ **:with-init** ⟨expression⟩)

  Die Angaben

```
:help-var-remember
(("h1" :sequentially-in
       (("h1_1" :with-init "expr1")
        ("h1_2" :with-init "expr2"))

 )
 ("h2" :sequentially-in
       (("h2_1" :with-init "expr3"))
 )
)
```

  besagen, daß vor dem ersten Schleifendurchlauf `h1_1` den Wert `expr1`, `h1_2` den Wert `expr2` und `h2_1` den Wert `expr3` zugewiesen bekommen. Am Ende des ersten Schleifendurchlaufes wird der Wert von `h1_1` in `h1_2`, der im ersten Schleifendurchlauf berechnete Wert von `h1` in `h1_1` und der ebenfalls im ersten Schleifendurchlauf berechnete Wert von `h2` in `h2_1` umgespeichert. Für den zweiten Schleifendurchlauf steht dann der aus dem ersten Schleifendurchlauf gemerkte Wert von `h1` in der Hilfsvariablen `h1_1` zur Verfügung. Ähnliche Umspeicherungen erfolgen am Ende jedes Schleifendurchlaufes. Nach dem Durchlauf der Schleife stehen die im letzten Schleifendurchlauf berechneten Werte von `h1` und `h2` zur Verfügung. Die Hilfsvariablen `h1_1`, `h1_2` und `h2_1` haben dann keine definierte Werte mehr, da erstens für eine korrekte Patternbelegung ein erheblicher Aufwand getrieben werden müßte und zweitens eine Erforderlichkeit nicht gegeben scheint.

Vollständigerweise bleibt anzumerken, daß die Gleichungen (47) bis (50) durch algebraische Umformulierung in alternativer Weise formuliert werden können. Durch Einsetzen der Hilfsvariablen-Ausdrücke für *hl* und *hr*, Bestimmung der Vorfaktoren zu x_{i-1}, x_i, x_{i+1} und das Einführen neuer Hilfsvariablen u, v, w erhält man:

$$u := -b \cdot c \tag{51}$$

$$v := a \cdot c - b \cdot d \tag{52}$$

$$w := a \cdot d \tag{53}$$

$$i = 2, ..., n-1 :$$

$$\dot{x}_i = u \cdot x_{i-1} + v \cdot x_i + w \cdot x_{i+1} \tag{54}$$

Diese letzte Formulierung ergibt (in diesem Fall bei konstanten Werten von a, b, c, d) sogar eine noch effizientere Berechnung.

2. **Akkumulierungs-Hilfsvariablen:**
Wird eine kompliziert zu berechnende Größe h_i $(i = p, ..., q)$ für jede Gleichung f_i benötigt und geht zusätzlich die Summe dieser Größen in eine weitere Gleichung ein, so kann die Größe h_i in der Schleife über die Gleichungen f_i als Akkumulierungs-Hilfsvariable definiert werden und wird bei der Berechnung der Gleichungen aufsummiert. Es wird eine zweite Berechnung der Größe h_i für eine explizite Summenberechnung gespart. Akkumulierungs-Hilfsvariablen werden durch den `key` `:help-var-accumulate` (siehe auch Abschnitt 3.6.2 bei „indizierte Gleichungen") angegeben:

:help-var-accumulate ⟨accumulate⟩

- Nach diesem `key` folgt eine Liste von Tripeln. Diese Tripel bestehen aus dem Namen der Akkumulierungs-Hilfsvariable, dem `key` `:with-init` und dem Berechnungsausdruck zur Initialisierung der Summe:

  ⟨accumulate⟩::=
  ({ (⟨string⟩ **:with-init** ⟨expression⟩) }*
)

  Beispiel:
  Die folgenden Gleichungen sollen berechnet werden:

$$i = 2, ..., n-1 :$$

$$h_{ge,i} := x_i \cdot \sum_{j=1}^{m} (a_j \cdot x_j) \tag{55}$$

$$h_{a,i} := \sum_{j=2}^{i} \frac{h_{ge,j}}{w} \tag{56}$$

$$\dot{x}_i = k_1 \cdot h_{ge,i} + k_2 \cdot h_{a,i} \tag{57}$$

$$\dot{T} = h_{a,n} + w \tag{58}$$

  Sie werden dem Code-Generator wie folgt angegeben:

```
(define-equations
  (:index ("i" 2 "n - 1")
   :meaning "Accumulate Schleifendefinition"
   :help-var-accumulate
   (("h_a" :with-init "0"                          ; Akkumuliere h1
```

```
      ))                                          ; mit Initialisierung 0
      :eq-scope
      ((:assignment-name "Groesse h_ge"
        :meaning "komplizierte Berechnung"
        :help-variable "h_ge"
        :h-value "x[i] * :sum<j,1,m>(a[j]*x[j])"
       )
       (:assignment-name "h_a"
        :meaning "Akkumulations-Hilfsvariable"
        :help-variable "h_a"
        :h-add "h_ge / w"                         ; h-add statt h-value
       )
       (:equation-name "Dgl. fuer x[i]"
        :meaning "Verwendung von h_a und h_ge innerhalb Schleife"
        :b-values ((:state-variable-reference "x[i]" :b-value "1.0"))
        :f-value "k1*h_ge + k2*h_a")
      ))
     (:equation-name "Dgl. fuer T"
      :meaning "Accumulate-Hilfsvar h_a nach Schleife"
      :b-values ((:state-variable-reference "T" :b-value "1.0"))
      :f-value "h_a + w"
     )
   ) ;; end of equations
```

Eine Zuweisung an eine zu merkende Hilfsvariable (bzw. an eine Akkumulierungs-Hilfsvariable) innerhalb der Schleife erfolgt durch das Assignment-Konstrukt, wobei allerdings anstatt des keys `:h-value` der key `:h-remember` (bzw. `:h-accumulate`) angegeben werden muß.[19]

Alle zu merkenden Hilfsvariablen und Akkumulierungs-Hilfsvariablen, sowie alle Hilfsvariablen die dabei zur Speicherung und Akkumulierung benutzt werden müssen als Hilfsvariablen definiert sein.

3.6.4 Berechnung von Hilfsvariablen mittels Unterprogramm-Aufrufen

Alternativ zu den expliziten Zuweisungen an Hilfsvariablen, bietet der Code-Generator die Möglichkeit, Hilfsvariablen mittels von FORTRAN-Unterprogrammen zu berechnen. Dies hat verschiedene Vorteile: Zum einen kann der Code der Modellspezifikationsdatei minimiert werden, sofern verschiedene Hilfsvariablen durch einen identischen Algorithmus berechnet werden sollen. Vor allem aber ermöglichen Unterprogramme die Hilfsvariablen völlig unabhängig von der Code-Generator-Syntax, mittels eines vom Benutzer meist selbstgeschriebenen FORTRAN-Algorithmus zu berechnen. Unterprogramm-Aufrufe sind in den Fällen unverzichtbar, in denen die Syntax des Code-Generators eine geeignete mathematische Beschreibung des Problems bislang nicht erlaubt. Dies ist beispielsweise bei einer speziellen Interpolation oder Heuristiken der Fall.

Unterprogramme, die durch den Code-Generator aufgerufen werden, müssen in FORTRAN-Programmdateien definiert sein. Die Beschreibung der Programmköpfe dieser FORTRAN-Programmdateien werden in Bibliotheken in LISP-Code abgelegt. Damit wird eine Verwaltung der Unterprogramme möglich. Zur Verwendung von Unterprogrammen in Verbindung mit dem Code-Generator sind unterschiedliche Vorgehensweisen möglich.

1. Das Unterprogramm kann vom Benutzer als FORTRAN-Programmdatei komplett von Hand geschrieben werden. Dabei muß der Benutzer darauf achten, daß die Argumente des Unterprogramms hinsichtlich der Anzahl und des Typs mit dem Eintrag des Programmkopfes in der Bibliothek und mit dem Unterprogramm-Aufruf des Code-Generators übereinstimmen.
2. Die Erstellung eines benötigten Unterprogramms unterstützt der Code-Generator durch die Option der Generierung eines entsprechenden Programmkopfes einer FORTRAN-Programmdatei. Die so

[19] Dies dient zur Sicherheit für den Benutzer, damit das zu einer normalen Zuweisung unterschiedliche Verhalten durch eine unterschiedliche Syntax explizit gemacht wird.

erstellte FORTRAN-Programmdatei muß anschließend vom Benutzer mit dem Quellcode der eigentlichen Berechnung ergänzt werden. Dieser FORTRAN-Quellcode kann auch innerhalb der Modellspezifikationsdatei angegeben werden, was eine vollständige Generierung der FORTRAN-Programmdatei erlaubt. Bei dieser Art der Unterprogrammerstellung übernimmt der Code-Generator die Aktualisierung der Bibliothekseinträge. Dadurch wird die Verwaltung der Unterprogramme sicherer, weshalb diese Vorgehensweise empfohlen wird.

Die so erstellten FORTRAN-Programmdateien werden zusammen, mit den vom Code-Generator erzeugten Programmdateien in ein gemeinsames Verzeichnis abgelegt. Bei der Generierung des Simulators greift DIVA auf alle FORTRAN-Programmdateien des Verzeichnisses zu (also auch auf die Unterprogramme) und bindet diese in den Simulator ein. Die FORTRAN-Routinen, die der Code-Generator erzeugt hat, können somit auf die jeweiligen Unterprogramme zugreifen.

Unterprogramm-Aufruf im Code-Generator:

Analog zu den Zuweisungen an Hilfsvariablen können Unterprogramm-Aufrufe im Code-Generator bei den Gleichungen (⟨simple-equation⟩, ⟨indexed-equation⟩) oder bei den B-Matrix-Einträgen (⟨simple-b-value⟩, ⟨indexed-b-value⟩) definiert werden.

Hierzu muß folgende Syntax verwendet werden:

⟨procedure-call⟩::=
(**:procedure-name** ⟨string⟩
:meaning ⟨string⟩
:compute ⟨help-variable⟩
:args ⟨list-of-variables-or-parameters-or-expressions⟩
[**:subroutine-names** ⟨list-of-strings⟩]
[**:fortran-code** ⟨string⟩]
:provided-pattern⟨expression⟩)

Zum Aufruf eines Unterprogramms sind folgende keys vorgesehen:

:procedure-name ⟨string⟩

- Name des aufzurufenden Unterprogramms.

  ACHTUNG: Um eindeutig die Unterprogramme in einer Modellspezifikationsdatei zu kennzeichnen, wurde die Konvention vereinbart, daß die ersten vier Buchstaben des Namens mit `CGP_` (**C**ode **G**enerator **P**rocedure) übereinstimmen müssen. Ist dies nicht der Fall gibt der Code-Generator eine Warnung aus und ignoriert den entsprechenden Aufruf.

:meaning ⟨string⟩

- Dieser key wird von einem möglicherweise mehrzeiligen String gefolgt, der eine nähere Beschreibung enthält und zur Dokumentation benutzt wird. Beachten Sie bitte, daß die Zeichen `&`, `#` und `\` problematisch beim Übersetzen der LaTeX-Dokumentation sein können.

:compute ⟨help-variable⟩

- Hilfsvariable, die durch den Unterprogramm-Aufruf berechnet wird.

  ACHTUNG: Eine indizierte Hilfsvariable muß – aus Gründen der internen Pattern-Belegung – über den ganzen Laufbereich eines jeden Indexes deklariert sein. Folglich muß jeder Index mit dem Ausdruck **:all** belegt werden (vgl. die folgenden Erläuterungen).

:args ⟨list-of-variables-or-parameters-or-expressions⟩

- Die diesem key folgende Liste kann beliebige algebraische Ausdrücke, Variablen oder Parameter enthalten, deren Werte dem Unterprogramm übergeben und dort zur Berechnung der Hilfsvariable benötigt werden. Jedes Element der Liste muß entsprechend der LISP Notation durch ein Leerzeichen vom folgenden Element getrennt werden.

Die Elemente müssen in der Reihenfolge angegeben werden, wie sie dem Unterprogramm übergeben werden sollen. Es ist also darauf zu achten, daß der Typ (**double precision**, **integer** oder **character**) und – im Falle von indizierten Größen – die Anzahl der Indizes mit den Deklarationen im Unterprogramm übereinstimmen.

:subroutine-names ⟨list-of-strings⟩

- Liste mit den Kurzbezeichungungen der Argumente eines Unterprogramms. Dieser key ist optional und wird ausschließlich bei der Generierung des Unterprogramm-Kopfes verwendet (vgl. S. 62f).

:fortran-code ⟨string⟩

- Mehrzeiliger optionaler String, der den FORTRAN-Quell-Code enthält (vgl. S. 62f). Beachten Sie dabei, daß die ersten 5 Spalten einer FORTRAN-Zeile leer sein müssen[20].

  ACHTUNG: Beachten Sie die üblichen FORTRAN-Konventionen, bspw. die Reservierung der ersten fünf Spalten für Anweisungsmarken.

:provided-pattern ⟨expression⟩

- Der hier eingegebene Ausdruck wird anstatt der beim key **args** eingebenen Argumente zur Bestimmung der Patternbelegung verwendet. Wird jedoch **nil** eingegeben, erfolgt die Patternbestimmung wie bisher. Dieser key ist optional und sollte nur in Ausnahmefällen bei starker Mehrfachbelegung der Pattern verwendet werden. Hinweis: Zur Überprüfung der Patternbelegung kann der DIVA-Befehl **testpat** verwendet werden.

Mittels der oben beschriebenen Angaben erzeugt der Code-Generator in den FORTRAN-Programmen einen Unterprogramm-Aufruf und belegt die Code-Generator-Pattern in Abhängigkeit der mit dem key **:args** angegebenen Argumente.

Übergabe von Feldern oder Teilen von Feldern an ein Unterprogramm:

Eine Besonderheit, die nur bei Unterprogramm-Aufrufen verwendet werden kann, ist der Ausdruck **:all**. Er kennzeichnet bei einer indizierten Größe eine interne Laufvariable, die den ganzen Laufbereich eines einzelnen Indexes beschreibt. Somit ist es möglich dem Unterprogramm ganze Felder oder Teile davon zu übergeben.

Beispiel:

`M[:all, 2]` bezeichnet alle Elemente der zweiten Spalte der Matrix `M`.

ACHTUNG: Völlig unsinnig ist die Verwendung des Ausdrucks **:all** in einem arithmetischen Ausdruck, da der Code-Generator keine vektoriellen Operationen kennt.

ACHTUNG: Im derzeitigen Implementierungsstand können von Hilfsvariablen keine Felder oder Teile von Feldern übergeben werden.

Die Verwendung des Ausdrucks **:all** wird durch die spezifische Verwaltung von Feldern im Code-Generator (bzw. in FORTRAN) eingeschränkt: Im Code-Generator ist ein Feld intern als Vektor abgespeichert. So ist beispielsweise eine zweidimensionale Matrix Spalte für Spalte in einem internen Vektor nacheinander abgespeichert. Auf ein einzelnes Element wird zugegriffen, indem aus den Indizes des Feldes der interne Index des internen Vektors berechnet wird.

Falls man nur einen Teil eines Feldes übergeben will, muß man beachten, daß in FORTRAN einem Unterprogramm nur das erste Element des Feldes übergeben wird. Um im Unterprogramm zu vermeiden, daß bei einem Feldzugriff auf ein falsches Element des Teil-Feldes zugegriffen wird, muß darauf geachtet werden, daß alle Elemente des Teil-Feldes in einer ununterbrochenen Reihenfolge im internen Vektor abgespeichert sind. Im Beispiel einer zwei dimensonalen Matrix bedeutet dies, daß eine beliebige Spalte (**:all** Ausdruck im ersten Index) dem Unterprogramm übergeben werden kann, jedoch keine beliebige Zeile (**:all** Ausdruck im zweiten Index). Grundsätzlich muß also bei der Verwendung des Ausdrucks **:all** folgendes beachtet werden:

ACHTUNG: Der Ausdruck **:all** darf nur dann in einem Index verwendet werden, wenn der vorangegangene Index auch durch einen **:all** Ausdruck definiert wurde!

Beispiel:
Von einem vierdimensionalen Feld foo_{cg} könnten z. B. die Felder foo_{cg} [:all, :all, :all, 1], foo_{cg} [:all, :all, $2+NC$, 1] oder foo_{cg} [:all, $NC*2$, $2+NC$, 1] dem Unterprogramm übergeben werden. Jedoch *unzulässig* ist beispielsweise die Übergabe von foo_{cg} [2, :all, $2+NC$, 1]!

Beispiel für einen Unterprogramm-Aufruf unter Verwendung des Ausdrucks **:all** zur Feldübergabe:

```
(:procedure-name
    ;--------------------------------------------
     "CGP_DUMMY"
    ;--------------------------------------------
    :compute  "foo_cg[:all,:all]"
    :meaning "Beispiel fuer einen Unterprogramm-Aufruf"
    :args  ("x[:all,:all,:all,1]"
            ":sum<i,1,NC>(:sum<j,1,2>(y[i,j,1]))"
            )
    )
```

ACHTUNG: Werden doppelt indizierte Zustandsvariablen als Argumente von Unterprogrammen verwendet, ist die Reihenfolge der Dimensionsdefinition wichtig hinsichtlich eines einfachen Zugriffs.

Mit einem Unterprogramm-Aufruf soll eine Stoffdatenberechnung durchgeführt werden. Dazu werden Konzentrationen aller Stoffe benötigt. Im Fall eines Kolonnenmodells sind die Konzentrationen über die Komponentenanzahl und die Bodenzahl der Kolonne definiert. Um auf alle Komponenten eines Bodens einfach zugreifen zu können ist es sinnvoll die Konzentration $x_{i,j}$, $i = 1(1)NC$, $j = 1(1)NB$ in der ersten Dimension über der Anzahl der Komponenten NC und in der zweiten Dimension über der Bodenanzahl NB zu definieren. Mit `x[:all,j]` erhält man dann alle Komponenten des Bodens j.

In FORTRAN wird einem Unterprogramm jeweils nur das erste Element des Feldes und die Parameter für die Längen der Laufbereiche der Indizes übergeben. Der Code-Generator erzeugt diese Parameter automatisch und schreibt sie in den Unterprogramm-Aufruf. Falls der Benutzer nicht die automatische Erzeugung eines Unterprogramm-Kopfes verwendet, muß er in seinem FORTRAN-Unterprogramm beachten, daß neben dem ersten Element des Feldes auch die Parameter für die Längen der Laufbereiche übergeben werden.

Beispiel:
Der Code-Generator erzeugt aus dem Unterprogramm-Aufruf aus dem vorigen Beispiel folgenden FORTRAN-Code bei der Generierung eines DIVA-Grundelements:

```
C     Prozeduraufruf: CGP_DUMMY , berechnet  foo_cg[:all,:all]
C
C     Beispiel fuer einen Unterprogramm-Aufruf
C
C     Berechnung der Doppelsumme und Speicherung in SUM01
...

C     CALL CGP_DUMMY (foo_cg ( 1, 1 ) , foo_cg2LEN , foo_cg1LEN , x ( 1, 1, 1,
C                    1 ) , x3LEN , x2LEN , x1LEN , SUM01)
      CALL CGP_DUMMY (RPAR ( ((1 * foo_cg2LEN + 1) + foo_cgVIR) ) ,
     #               foo_cg2LEN , foo_cg1LEN , X ( ((((1 * x2LEN + 1
     #               ) * x3LEN + 1) * x4LEN + 1) + xVIR) ) , x3LEN ,
     #               x2LEN , x1LEN , SUM01)
```

Die Berechnung der Indizes der Variablen RPAR und X zur Argumentberechnung innerhalb des Aufrufs von CPG_DUMMY sind DIVA spezifische Adressrechnungen, die hier nicht weiter interessieren.

Zu diesem Unterprogramm-Aufruf könnte ein vom Benutzer geschriebenes Unterprogramm mit folgender FORTRAN-Deklaration existieren:

```
      SUBROUTINE CGP_DUMMY (foo_up , foo_up1LEN , foo_up2LEN,
#                          x_up   , x_up1LEN   , x_up2LEN   , x_up3LEN,
#                          SUMME)
C
C  Hier folgen die Variablen-Deklarationen und der eigentliche
C  Programm-Code
```

Neben den Feldern foo_{up} und x_{up} (foo_up, x_up) müssen zusätzlich die Parameter foo_up1LEN, foo_up2LEN, x_up1LEN, x_up2LEN, und x_up3LEN definiert werden. Der Wert des mathematischen Ausdrucks wird der Variablen SUMME übergeben.

Wie im Code-Generator, sind auch in FORTRAN die Felder als interne Vektoren dargestellt. FORTRAN legt ein zweidimensionales Feld Spalte für Spalte, der Code-Generator hingegen Zeile für Zeile hintereinander ab. Das Feld des Unterprogramms (FORTRAN-Feld) entspricht somit dem transponierten Feld im Code-Generator. Bei der Verwendung des :all Ausdrucks muß deshalb folgendes beachtet werden:

ACHTUNG: Wird ein Feld oder ein Teil eines Feldes (jedoch kein einzelnes Feld-Element!) einem Unterprogramm übergeben, so vertauscht sich die Reihenfolge der Indizes im Unterprogramm!

Beispiel:

Wird ein Unterprogramm mit dem Teil-Feld $x_{\mathrm{cg}}\,[:\!\mathrm{all}, :\!\mathrm{all}, :\!\mathrm{all}, 2+NC]$ aufgerufen und im Unterprogramm der nur noch dreidimensionalen Variablen x_{up} übergeben, so entspricht dem Element $x_{\mathrm{cg}}\,[A, B, C, 2+NC]$ das Element $x_{\mathrm{up}}\,[C, B, A]$ im Unterprogramm. Für die Indizes gilt also folgendes:

$x_{\mathrm{CG}}\,[i_{\mathrm{CG}}, j_{\mathrm{CG}}, k_{\mathrm{CG}}, l_{\mathrm{CG}}]$	$x_{\mathrm{UP}}\,[i_{\mathrm{UP}}, j_{\mathrm{UP}}, k_{\mathrm{UP}}]$
mit $i_{\mathrm{CG}} = 1 \ldots NI_{\mathrm{CG}}$	mit $i_{\mathrm{UP}} = 1 \ldots (NI_{\mathrm{UP}} = NK_{\mathrm{CG}})$
$j_{\mathrm{CG}} = 1 \ldots NJ_{\mathrm{CG}}$	$j_{\mathrm{UP}} = 1 \ldots (NJ_{\mathrm{UP}} = NJ_{\mathrm{CG}})$
$k_{\mathrm{CG}} = 1 \ldots NK_{\mathrm{CG}}$	$k_{\mathrm{UP}} = 1 \ldots (NK_{\mathrm{UP}} = NI_{\mathrm{CG}})$
$l_{\mathrm{CG}} = 1 \ldots NL_{\mathrm{CG}}$	

Generierung von Unterprogramm-Köpfen:

Der Code-Generator bietet die Möglichkeit, mittels der Steuerfunktion cg-lib, die Unterprogramm-Aufrufe die sich in der Modellspezifikationsdatei befinden, auf ihre Konsistenz zu überprüfen. Außerdem wird, falls noch kein FORTRAN-Unterprogramm existiert, dieses erzeugt. Dabei wird lediglich der Unterprogramm-Kopf und nur im Fall der Angabe des keys :fortran-code zusätzlich auch der Unterprogramm-Rumpf erzeugt.

Zur Konsistenzprüfung sind eine globale Bibliothek und lokale Bibliotheken mit den Beschreibungen der Unterprogramm-Köpfe vorgesehen. Die globale Bibliothek umfaßt Unterprogramme, auf die alle Benutzer zugreifen können. Eine lokale Bibliothek verwaltet die vom Benutzer definierten Unterprogramme und befindet sich genau in dem Verzeichnis, in dem auch die dazugehörigen Unterprogramme abgespeichert sind. Der Dateiname der Bibliothek ist durch die Erweiterung .lib gekennzeichnet.

Ist ein Unterprogramm-Aufruf nicht in der Bibliothek eingetragen, wird davon ausgegangen, daß noch kein FORTRAN-Unterprogramm hierfür existiert. In diesem Falle wird der Unterprogramm-Kopf des Unterprogramms erzeugt und im folgenden *sofort* in die lokale Bibliothek geschrieben. Sollte dennoch unerwarteter Weise ein FORTRAN-Unterprogramm existieren, wird kein Eintrag in die Bibliothek geschrieben sondern eine entsprechende Fehlermeldung angezeigt. Bereits vorhandene Bibliothekseinträge und FORTRAN-Programmdateien werden prinzipiell nicht überschrieben.

Es sind zwei unterschiedliche Vorgehensweisen bei der Verwaltung der Unterprogramme möglich:

1. Wenn der Benutzer ein Unterprogramm komplett selbst erstellt, muß die lokale Bibliothek entsprechend ergänzt werden. Nähere Erläuterungen zu einem Eintrag in eine solche lokale Bibliothek befinden sich am Anfang einer jeden Bibliotheksdatei.
2. Der Code-Generator unterstützt die Generierung eines Unterprogramms und nimmt den entsprechenden Bibliothekseintrag vor. Diese Vorgehensweise wird empfohlen.

   Für die Generierung des Unterprogramm-Kopfes des Unterprogramms wird eine FORTRAN-Programmdatei angelegt, die nach dem Unterprogramm-Aufruf benannt wird. Dieser Unterprogramm-Kopf wird jedoch nur dann in die Datei geschrieben, sofern noch keine gleichnamige Datei existiert.

   Jeweils nur der *allererste* Unterprogramm-Aufruf ist relevant für die Generierung eines Unterprogramm-Kopfes und für den Eintrag in die lokale Bibliothek.

Ein vom Code-Generator generierter Unterprogramm-Kopf des Unterprogramms besteht aus der Deklaration des Unterprogramms mit seinen Argumenten (`subroutine`) und den Deklarationen der Argumente als Variablen. Die Argumente des Unterprogramms setzen sich aus der zu berechnenden Hilfsvariable (`:compute`) und den zur Berechung benötigten Argumente (`:args`) zusammen. Die Reihenfolge der Argumente des Unterprogramms muß den Definitionen in der Modellspezifikationsdatei entsprechen. Handelt es sich bei einem Argument um ein Feld, so muß beachtet werden, daß zusätzlich zum ersten Feldelement die Längen der Laufbereiche übergeben werden müssen. Folglich müssen hierfür zusätzlich Parameter als Argumente eingefügt werden (vgl. vorangegangener Abschnitt). Für die Variablen und Parameter werden die gleichen Namen verwendet, wie sie im Unterprogramm-Aufruf deklariert wurden. Im Falle von Feldern wird das erste Feldelement einer gleichnamigen Variablen übergeben. Die Parameter für die Längen der Laufbereiche bestehen aus dem Namen des Feldes, der Nummer des Indexes und den drei Buchstaben `LEN`. Speziell für Variablen, denen Werte von algebraischen Ausdrücken zugewiesen werden, werden Symbole generiert, die aus den vier Buchstaben `EXPR` und einer zweistelligen Nummer bestehen. Diese Nummer erhöht sich für jeden weiteren übergebenen algebraischen Ausdruck.

Beispiel:
Für den Unterprogramm-Aufruf von Seite 61 schreibt der Code-Generator folgende Deklaration in eine FORTRAN-Programmdatei, namens "`CGP_DUMMY.for`":

```
      SUBROUTINE CGP_DUMMY (foo_cg , foo_cg1LEN , foo_cg2LEN,
#                           x      , x1LEN      , x2LEN     , x3LEN,
#                           EXPR01)
C
C  Hier folgen die Variablen-Deklarationen und der eigentliche
C  Programm-Code
```

Bei einem Unterprogramm-Aufruf in der Modellspezifikationsdatei kann der Benutzer, mittels dem optionalen `key :subroutine-names` eigene Bezeichnungen für die Argumente im Unterprogramm wählen. Zu beachten ist jedoch, daß sowohl für jeden algebraischen Ausdruck als auch für die Parameter der Längen der Laufbereiche von Indizes Namen anzugegeben sind.

Beipiel:
Ergänzt man das vorangegangene Beispiel (vgl. Seite 61) durch

`:subroutine-names   ( "x" "nx1" "nx2" "y" "ny1" "ny2" "ny3" "z" )`

in der Modellspezifikationsdatei, so ergibt sich folgende Deklaration im Unterprogramm:

```
      SUBROUTINE CGP_DUMMY (x , nx1 , nx2 ,
#                           y , ny1 , ny2 , ny3,
#                           z)
C
C  Hier folgen die Variablen-Deklarationen und der eigentliche
C  Programm-Code
```

Der optionale key `:fortran-code` ermöglicht dem Benutzer eine vollständige Spezifikation des FORTRAN-Unterprogrammes. Dem key folgt ein mehrzeiliger String, der den eigentlichen FORTRAN-Code des Unterprogrammes enthält und direkt hinter die Deklarationen geschrieben wird. Es erfolgt jedoch keine Überprüfung der Syntax oder der Konsistenz des FORTRAN-Codes. Der String wird ledglich an das Ende der FORTRAN-Datei geschrieben. Bei der Verwendung dieses keys sollte darauf geachtet werden, daß diese Angabe nur bei der ersten Definition eines Unterprogramm-Aufrufs in der Modellspezifikationsdatei sinnvoll ist, denn nur dann wird auch wirklich der FORTRAN-Code in die Datei geschrieben. Bei allen weiteren Unterprogramm-Aufrufen in der Modellspezifikationsdatei werden nur die Unterprogramm-Aufrufe in den FORTRAN-Dateien erzeugt.

Sollen die Argumente hinsichtlich Anzahl oder Typ eines Unterprogramm-Aufrufs durch die Steuerfunktion `cg-lib` geändert werden, muß der Bibliothekseintrag <u>und</u> die bestehende FORTRAN-Programmdatei gelöscht werden, da bereits vorhandene Bibliothekseinträge und FORTRAN-Programmdateien nicht überschrieben werden. Dabei ist zu bedenken, daß das vorhandene Unterprogramm in mehreren Modellspezifikationsdateien stehen kann. Sollen die Änderungen auch diese Modellspezifikationsdateien betreffen, so müssen die dortigen Unterprogramm-Aufrufe natürlich auch geändert werden. Sollen sich diese Modellspezifikationsdateien jedoch auf das ursprüngliche Unterprogramm beziehen, empfiehlt es sich bei dem modifizierten Unterprogramm-Aufruf einen neuen Namen des Unterprogramms in der Modellspezifikationsdatei anzugeben, um damit ein neues Unterprogramm zu generieren.

4 Initialisierung und Steuerbefehle des Code-Generators

In diesem Abschnitt werden die Kommandos zur Initialisierung und zum Aufruf des Code-Generators beschrieben, wie sie in einen COMMON LISP-Steuerfile geschrieben werden können. Der Aufruf des Code-Generators vom Betriebssytem mit Übergabe eines Steuerfiles oder mit interaktiver Abfrage wird in Anhang D erläutert.

4.1 Initialisierung des Code-Generators

Der Code-Generator wird durch die Funktion `cg-init` initialisiert. Dabei werden alle bisherigen Definitionen gelöscht, und es kann beim selben Aufruf des Code-Generators eine weitere Modellspezifikationsdatei geladen und übersetzt werden. Die Funktion `cg-init` bietet darüberhinaus die Möglichkeit, über den key `:verbose-level` die Häufigkeit der informativen Ausgaben zu steuern.

:verbose-level ⟨number⟩

- Die dem key folgende ganze Zahl (0...100) ist ein Maß für die Anzahl der Bildschirmausgaben, die über den Fortschritt des Code-Generators informieren. Je kleiner die Zahl ist, um so weniger Meldungen werden von dem Code-Generator ausgegeben, d.h unwichtige Meldungen werden unterdrückt.

4.2 Steuerbefehle des Code-Generators

Zur Generierung verschiedener Files stehen folgende Funktionen zur Verfügung. Alle diese Funktionen haben je drei weiter unten angegebene keys.

- Die Funktion `cg-fortran-dat` erzeugt die FORTRAN-Unterprogramme und das Blockdatenfile (DAT-File). Der Name des DAT-Files ergibt sich aus dem in `define-header` angegebenen Programm-Namen. Die Funktion hat folgenden key:

  :default-output-subname ⟨string⟩

  - Der String gibt den internen DIVA-Namen des zu generierenden DIVA-Blockes an.

- Die Funktion `cg-dat` erzeugt nur das DAT-File. Prozeßgrößen und Gleichungen werden nicht eingelesen. Die Funktion hat folgenden key:

  :default-dat-name ⟨string⟩

  - Der String gibt den Namen des zu erzeugenden DAT-Files an.

- Die Funktion `cg-fortran` erzeugt die FORTRAN-Unterprogramme. Die Funktion hat neben denselben keys wie die Funktion `cg-fortran-dat` und den zusätzlichen key `:pattern-opt-level`. Dieser key steuert die Generierung der FORTRAN-Unterprogramme hinsichtlich des Grads der Optimierung der Patternbelegung. Die Optimierung erfolgt bezüglich der Vermeidung von Mehrfachbelegungen, die aber im allgemeinen nicht völlig verhindert werden können.

  :pattern-opt-level ⟨number⟩

  - Die dem key folgende ganze Zahl (0...4) gibt den Grad der Optimierung der Patternbelegung an. Je kleiner die Zahl, um so weniger wird optimiert. Der voreingestellte Wert ist die zweithöchste Optimierungsstufe 3. Die Bedeutung der einzelnen Optimierungsstufen sind nachfolgend beschrieben, wobei jede Stufe die Optimierungsschritte der niedrigeren Stufen beinhaltet:

    0 Pattern von nicht indizierten Größen, erzeugt durch die Verwendung von Summen, Produkten und `help-var-accumulate` Variablen, die nicht vom Laufindex derselben abhängen werden aus den Schleifen der Summen und Produkte herausgezogen.

    1 Mehrfach vorkommende Pattern innerhalb einer Gleichungsebene, beispielsweise innerhalb einer Summe, werden zusammengefaßt.

2 Pattern von mehrfach verschachtelten Schleifen, erzeugt durch die Verwendung von `help-var-accumulate` Variablen, werden auf Index-Abhängigkeit untersucht und gegebenenfalls werden die Schleifen aufgeteilt. Dadurch werden überflüssige Mehrfachbelegungen reduziert. Pattern erzeugt durch die Verwendung von Summen und Produkten, die nicht vom Laufindex derselben abhängen werden aus den Schleifen der Summen und Produkte herausgezogen. Damit wird eine Mehrfachbelegung durch verschachtelte Summen und Produkte reduziert.

3 Summen und Produkte mit gleicher Ober- und Untergrenze des Laufbereichs werden zusammengefaßt und nur deren Schnittmenge der Pattern wird belegt.

4 Für konditionale Ausdrücke werden bei der Patternbelegung die Abhängigkeiten berücksichtigt, die sich während der Laufzeit einer Simulation nicht ändern können. Das sind Abhängigkeiten von Strukturparametern und von rein numerischen Werten. Aufgrund der möglichen komplexen Verschachtelung konditionaler Bedingungen können in verschiedenen Abschnitten der Pattern konditionaler Ausdrücke, identische Patterneinträge vorkommen, die bei kleineren Optimierungsstufen nicht mehrfach auftreten. Dadurch kann es in ungünstigen Fällen bei Optimierungstufe 4 zu einer größeren Mehrfachbelegung als bei Optimierungsstufe 3 kommen.
Die Verwendung der Optimierungsstufe 4 empfiehlt sich außerdem bei folgender Kostellation (hier kann es zu Bereichsüberschreitungen für Felder, durch die bei niedrigere Optimierungsstufen verwendete Patternbestimmung kommen): Innerhalb von Gleichungsschleifen werden in Hilfsgrößenzuweisungen oder Gleichungen konditionale Ausdrücke in Verbindung mit indizierten Größen verwendet, dergestalt daß der Definitionsbereich der indizierten Größe überschritten wird. Dazu muß man wissen, daß bei der Patternbestimmung für konditionale Ausdrücke, bis einschließlich Optimierungsstufe 3, die Obermenge aller Teilausdrücke verwendet wird und eben keine Fallunterscheidung durchgeführt wird, siehe auch Kapitel 3.6.1.

- Die Funktion `cg-latex` erzeugt eine Dokumentation in LaTeX-Format. Die Funktion hat folgenden key:

  :default-latex-name ⟨string⟩

  - Der String gibt den Namen des zu schreibenden LATEX-Files an.

- Die Funktion `cg-cg` erzeugt ein CG-File. Diese Funktion wird von einer Benutzungsoberfläche benötigt. Diese Funktion kann aber auch vom Benutzer aus zwei Gründen direkt aufgerufen werden: Erstens kann anhand der Ausgabe des CG-Files überprüft werden, ob die eingegebene Information wie erwartet vom Code-Generator eingelesen wurde. Zweitens kann das CG-File in einer genormten Formatierung (pretty print) ausgegeben werden. Die Funktion hat folgende keys:

  :default-cg-name ⟨string⟩

  - Der String gibt den Namen des zu erzeugenden CG-Files an.

  :write-as-infix ⟨t or nil⟩

  - Falls t angegeben wurde wird die vom Benutzer gewohnte Infix-Notation bei der Ausgabe gewählt. Dies ist auch die Voreinstellung, falls dieser optionale key nicht angeben wird. Wird nil angeben, so wird die schneller zu lesende und zu schreibende interne COMMON LISP-Präfix-Notation bei der Ausgabe verwendet.

- Die Funktion `cg-lib` sollte aufgerufen werden, wenn der Benutzer einen Unterprogramm-Aufruf in der Modellspezifikationsdatei verwendet. Jeder einzelne Unterprogramm-Aufruf wird mit den Einträgen in der globalen und lokalen Unterprogramm-Bibliothek verglichen. Falls noch kein Eintrag in einer Bibliothek existiert, ergänzt die Funktion die lokale Bibliothek und generiert einen Unterprogramm-Kopf. Existiert bereits ein Eintrag in einer der Bibliotheken, dann wird die Konsistenz des Unterprogramm-Aufrufs geprüpft. Die Funktion hat folgende keys:

:local-library-name ⟨string⟩

- Der String gibt den Namen der aktuellen lokalen Bibliothek an.

Alle soeben beschriebenen `cg-...`-Funktionen besitzen dieselben drei keys:

:input-file ⟨string⟩

- Der String beschreibt das CG-File, das eingelesen werden soll. Ein vollständiger Pfad darf im Namen des CG-Files enthalten sein. Dieses CG-File beinhaltet alle nötigen Definitionen zur Modellierung des gewünschten Blocks. Wird dieser key nicht angegeben, so wird kein CG-File geladen, sondern angenommen, daß dieses schon vorher geladen wurde.

:default-output-pathname ⟨string⟩

- Der bzw. die vom Code-Generator erzeugten Files werden unter dem hier angegebenen Pfad abgelegt. Der Pfad muß grundsätzlich mit einem Dummy-Namen enden.

:do-overwrite ⟨nil oder t⟩

- Falls t angegeben wird, so überschreiben die vom Code-Generator erzeugten Files eventuell schon existierende Files, die dieselben Namen besitzen.

  Beispiel:

```
(cg-init :verbose-level 1)                    ; Nur die wichtigsten
                                              ; Meldungen werden ausgegeben.
(cg-fortran-dat
  :input-file "pt2.cg"                        ; CG-File
  :default-output-subname "ab01"              ; DIVA-Modellname
  :default-output-pathname "out/foo.for")     ; out/         <- Ausgabepfad
                                              ;     foo.for  <- Dummy-Name

(cg-init)                                     ; Voreingestellte Anzahl von
                                              ; Meldungen werden ausgegeben.
(cg-fortran
  :input-file "efb.cg"                        ; CG-File
  :default-output-subname "ab01"              ; DIVA-Modellname
  :default-output-pathname "out/foo.for")

(cg-dat
  ;; :input-file "efb.cg"                     ; CG-File nicht laden
  :default-dat-name "EFB.DAT"                 ; Name des DAT-Files
  :default-output-pathname "out/FOO.DAT")     ; out/         <- Ausgabepfad
                                              ;     foo.DAT  <- Dummy-Name

(cg-lib
  ;; :input-file "efb.cg"                     ; CG-File nicht laden
  :local-library-name "procedure-calls-local" ; lokaler Bibliotheksname
  :default-output-pathname "out/foo.lib")     ; out/         <- Ausgabepfad

(cg-latex
  ;; :input-file "efb.cg"                     ; CG-File nicht laden
  :default-latex-name "refb.tex"              ; LATEX-Filename
  :default-output-pathname "out/foo.tex")

(cg-cg
  ;; :input-file "efb.cg"                     ; CG-File nicht laden
  :write-as-infix t                           ; Infix-Ausgabe ist Voreinstellung
```

```
:default-cg-name "efb.cg"                ; Name des CG-Files
:default-output-pathname "out/foo.cg")
```

A Zusammenfassende Übersicht über LISP-spezifische Ausdrücke

Diese kurze Zusammenfassung von COMMON LISP-spezifischen Ausdrücken erhebt nicht den Anspruch, in COMMON LISP einzuführen, vielmehr soll das Lesen einer Einführung in COMMON LISP erleichtert werden.

Argumente Dies können Atome, Listen, Symbole, "Operatoren" oder Funktionen sein.

Atome Dies sind z. B. Symbole, Zahlen oder Strings.

Funktion Ein im Programm implementierter Algorithmus, der in Listenform angegeben und aufgerufen wird. Diese Liste besteht *hier* aus dem Funktionsname und entsprechenden key-Wert-Paaren: `(Funktionsname :key1 value1 :key2 value2 ...)`

key Schlüsselwort innerhalb von Funktionsaufrufen. Schlüsselworte beginnen immer mit einem Doppelpunkt und treten paarweise mit einem dazugehörigen Wert (Funktionsargument) auf (key value pair). Die angegebenen Funktionsargumente werden somit durch die zugehörigen keys näher bezeichnet. Ein key und sein Argument sind oftmals optional (vgl. Variables). Verschiedene key-Wert-Paare dürfen in beliebiger Reihenfolge innerhalb eines Funktionaufrufes angegeben werden, d. h. `(Fktname :key1 val1 :key2 val2 ...)` und `(Fktname :key2 val2 :key1 val1 ...)` sind gleichwertig.

Leerzeichen Das Leerzeichen trennt verschiedene Ausdrücke (seien es Symbole, Operatoren, Argumente ...) innerhalb von Listen.

Liste Eine in runden Klammern stehende Auflistung von Argumenten: `(element1 element2 ...)`

Präfix-Notation Darstellung mathematischer Ausdrücke in COMMON LISP. In dieser Darstellung steht der Operator am Anfang einer Liste von Argumenten und verknüpft diese entsprechend seiner Bedeutung:

`(+ 1 2 3)` $\hat{=}$ 1+2+3 = 6

`(- 1 2 3)` $\hat{=}$ 1-2-3 = -4

`(* 1 2 3)` $\hat{=}$ 1*2*3 = 6

`(/ 1 2 3)` $\hat{=} 1/2/3 = \frac{1}{6}$

Zusammengesetzte arithmetische Ausdrücke werden durch geschachtelte Listen dargestellt:

`(* (+ 1 2) 3)` $\hat{=} (1+2)3 = 6$

Operationen eines Elements (transzendente Operatoren) stellt man wie folgt dar:

`(sin 0)` $\hat{=} \sin(0) = 0$

`(cos 0)` $\hat{=} \cos(0) = 1$ usw.

B Erläuterungen zur Grammatikschreibweise

In der Code-Generator-Beschreibung wird die folgende Grammatikschreibweise (erweiterte Backus-Naur-Form) zur Beschreibung der Syntax verwendet:

- Symbole in spitzen Klammern (⟨nonterminal⟩) bezeichnen Nicht-Terminale, die gemäß einer Grammatikregel/Ableitungsregel ersetzt werden müssen.
- Symbole in Fettdruck (**terminal**) bezeichnen Terminale, die genau wie angegeben übernommen werden müssen.
- Eine Grammatikregel/Ableitungsregel (⟨nonterminal⟩ :== right side) beschreibt, auf welche Weise ein Nicht-Terminal ersetzt werden darf. Rechts von ':==' steht eine Folge von Terminalen und Nicht-Terminalen, die an Stelle des Nicht-Terminals geschrieben werden darf. Zum Beispiel wird durch die Grammatik

  ⟨A⟩ :==
  ⟨A⟩ :== **a**
  ⟨A⟩ :== ⟨A⟩**a**⟨B⟩

  beschrieben, daß ausgehend von dem Nicht-Terminal ⟨A⟩ die leere Zeichenkette und die Zeichenkette 'a' abgeleitet werden kann. Wird die dritte Regel zweimal angewendet erhält man zuerst '⟨A⟩a⟨B⟩' und dann '⟨A⟩a⟨B⟩a⟨B⟩'. Die beiden ⟨A⟩s können je durch eine der ersten beiden Regel ersetzt werden, aber für ⟨B⟩ gibt es keine Ableitungsregel und man kann deshalb durch Anwenden der dritten Regel keine vollständig abgeleitete Zeichenkette erhalten. In diesem Falle ist die dritte Regel überflüssig.
- Verschiedene Ableitungsregeln ein und desselben Nicht-Terminals können durch eine Ableitungsregel abgekürzt werden, indem die rechten Seiten der Regeln durch '|' getrennt zusammengefaßt werden. Zum Beispiel kann obige Grammatik folgendermaßen gleichwertig geschrieben werden:

  ⟨A⟩ :== | **a** | ⟨A⟩**a**⟨B⟩

- Eine weitere Abkürzungsmöglichkeit ist das Einschließen einer Folge von Terminalen und Nicht-Terminalen in eckige Klammern ('$\Big[$'...'$\Big]$') und bedeutet das optionale Auftreten der eingeschlossenen Symbole. (Achtung: '[' bezeichnet ein Terminal und '$\Big[$' das eben beschriebene Metasymbol.)
- Das Einschließen einer Folge von Terminalen und Nicht-Terminalen in geschweifte Klammern mit nachgestelltem Plus oder Mal ('$\Big\{$'...'$\Big\}^{+}$', '$\Big\{$'...'$\Big\}^{*}$') kürzt das beliebig oft wiederholte Auftreten der eingeschlossenen Symbole ab. Mit nachgestelltem Plus müssen die eingeschlossenen Symbole mindestens einmal auftreten, mit nachgestelltem Mal ist auch die leere Zeichenkette erlaubt. (Achtung: '**{**' bezeichnet ein Terminal und '$\Big\{$' das eben beschriebene Metasymbol.)
- Zusätzlich zu den Ableitungsregeln einer Grammatik muß ein ausgezeichnetes Nicht-Terminal (Startsymbol) festgelegt werden, mit dem die Ableitung einer Zeichenkette immer begonnen wird. Wenn das Startsymbol nicht explizit angegeben wird, so ist es implizit das erste Nicht-Terminal der ersten Grammatikregel.
- Die folgende kurze Grammatik beschreibt alle Zeichenketten, die mindestens ein 'a' enthalten und am Ende ein 'b' enthalten dürfen ('a', 'ab', 'aa', 'aab', 'aaa', 'aaab', ...).

  ⟨S⟩ :== ⟨A⟩ | ⟨A⟩**b**
  ⟨A⟩ :== **a** | ⟨A⟩**a**

  Die beiden folgenden Grammatikregeln beschreiben jeweils dieselbe Menge von Zeichenketten.

  $\langle \mathrm{S} \rangle :== \Big\{\mathbf{a}\Big\}^{*}\mathbf{a}\Big[\mathbf{b}\Big]$

  $\langle \mathrm{S} \rangle :== \Big\{\mathbf{a}\Big\}^{+}\Big[\mathbf{b}\Big]$

C Definition arithmetischer und logischer Ausdrücke

Die in den folgenden Tabellen angegebene Grammatik beschreibt die Syntax des am ISR entwickelten verfahrenstechnischen Datenmodells [41]. Arithmetische (`<arex>`) und logische (`<logical-expr>`) Ausdrücke dürfen, wie in den Tabellen 5 und 6 angegeben, im Eingabe-File des Code-Generators notiert werden. Die Tabellen 3 und 4 beinhalten alphabetisch geordnete Nicht-Terminale, von denen aber im Zusammenhang mit arithmetischen und logischen Ausdrücken nur `<boole>`, `<integer>`, `<real>` und `<word>` von Bedeutung sind.

Mit der Festlegung der Priorität und Assoziativität der arithmetischen und logischen Operatoren (Tabelle 7) ist die implizite Klammerung der Ausdrücke eindeutig beschrieben.

Zusätzlich zu den in FORTRAN vorhandenen Operationen werden auch Summen und Produkte (`:sum`, `:prod`[21]) in FORTRAN-Code umgesetzt. Die Differential- und Integral-Operationen (`:diff`, `:int`) müssen vor Weitergabe an den Code-Generator beseitigt werden.

Eine ausführlichere Beschreibung der hier verwendeten Grammatikschreibweise (erweiterte Backus-Naur-Form) ist in Anhang B zu finden.

Beispiele für arithmetische Ausdrücke:

1. Mehrfaches Differential

$$\frac{d^2\ \sin(x\,y)}{dx\,dy}$$

   notiert als `:diff<x,y> :sin(x * y)`

2. Unbestimmtes Integral

$$\int x\,dx$$

   notiert als `:int<x> x`

3. Bestimmtes Integral

$$\int_{-\infty}^{2\pi} \ln(e + x^2)\,dx$$

   notiert als `:int<x, -:inf, 2 * :pi> ln(:e + x^2)`

4. Implizite Klammerung bei Produkt und Summe

   `:prod` und `:sum` werden als unäre Präfixoperatoren (wie `:sin`) behandelt. Deshalb ergibt sich hier eine implizite Klammerung, die in folgenden Beispielen verdeutlicht wird.

   (a) `:prod<i, 1, m> x[i] * y` bedeutet

$$\prod_{i=1}^{m} x[i] \cdot y \equiv \left(\prod_{i=1}^{m} x[i]\right) \cdot y$$

   (b) `:sum<i> x[i] * y + z` bedeutet

$$\sum_i x[i] \cdot y + z \equiv \left(\sum_i x[i]\right) \cdot y + z \not\equiv \sum_i (x[i] \cdot y) + z$$

[21] Summen oder Produkte in Angaben von Indizes eines FORTRAN-Feldes, z.B. $x[1, 2, \sum_{i=1}^{n} p[i], 4]$, werden nicht unterstützt.

(c) `:prod<i> 3 x` bedeutet

$$\prod_i 3x \equiv \prod_i (3x)$$

(d) `:prod<i> 3 :sin x` bedeutet

$$\prod_i 3\, sin\, x \equiv \left(\prod_i 3\right) sin\, x$$

Um falsche Deutungen der impliziten Klammerung zu verhindern, wird empfohlen bei Verwendung von `:prod` oder `:sum` in Verbindung mit `+`, `-`, `*` oder `/` explizit Klammern zu setzen.

5. Produkt, Summe

$$\prod_{i=0}^{1} \sum_{j=i}^{n+m} \sin^2 x[i,j] \cdot \cos y + 2 + 3 \cdot 4 \cdot 5^{6^{7!!}}$$

$$\equiv \left(\left(\prod_{i=0}^{1}\left(\sum_{j=i}^{n+m}(\sin(x[i,j]))^2\right)\right) \cdot \cos y + 2\right) + \left((3 \cdot 4) \cdot \left(5^{\left(6^{((7!)!)}\right)}\right)\right)$$

notiert als
`:prod<i,0,1> :sum<j,i,n + m> (:sin x[i,j])^2 * :cos y + 2 + 3 * 4 * 5 ^ 6 ^ 7!!`

6. Bedinungsabhängiger Teilausdruck
Die Bedingungen dieser Teilausdrücke können auf zwei verschiedene formuliert werden:

(a) Anwendung von Vergleichsoperatoren auf arithmetische Ausdrücke; bspw. wird

$$k_0 \left\{ \begin{array}{ccl} 0 & , & \text{falls } T \leq 273 \\ k_1 p^2 & , & \text{falls } (T > 0) \wedge (T < T_m) \\ k_2 - e^{-k_3 p} & , & \text{sonst} \end{array} \right\} T$$

notiert als
`k0 * :cond( T <= 273, 0, (T > 0) :and (T < Tm), k1 * p^2, k2 - :exp(-k3 * p) ) * T`

(b) Angabe von Stellen eines Petri-Netzes, siehe auch Anhang G

$$\left\{ \begin{array}{lcl} x + b & : & m_1 = 1 \\ ax - bu & : & m_2 = 1 \\ x - b & : & m_3 = 1 \end{array} \right. \tag{59}$$

wird notiert als
`:cond(m_1 , x + b , m_2 , a*x - b*u , x - b)`

7. Definition von `:div` und `:mod` (r reell, z ganzzahlig nichtnull: $r \in \mathbb{R}$, $z \in \mathbb{Z} \setminus \{0\}$)

	Beschreibung	FORTRAN	COMMON LISP
`:div(r,z)`	ganzzahliger Anteil bei Division von r/z	`INT(r/z)`	`(truncate r z)`
`:mod(r,z)`	Rest der Division r/z d. h. $r - (int(r/z) * z)$	`MOD(r,z)`	`(rem r z)`
`:div(r)`	≡`:div(r,1)` Nachkommastellen wegstreichen d. h. Vorzeichen und Vorkommastellen	`INT(r)`	`(truncate r)`
`:mod(r)`	≡`:mod(r,1)` Vorzeichen und Nachkommastellen d. h. $r - int(r)$	`MOD(r,1)`	`(rem r 1)`

Beispiel	Auswertung
`:div(5,3)`	1
`:div(-5,3)`	-1
`:div(5,-3)`	-1
`:div(-5,-3)`	1
`:mod(5,3)`	2
`:mod(-5,3)`	-2
`:mod(5,-3)`	2
`:mod(-5,-3)`	-2
`:div(5.67,3)`	1
`:div(-5.67,3)`	-1
`:div(5.67,-3)`	-1
`:div(-5.67,-3)`	1
`:mod(5.67,3)`	2.67
`:mod(-5.67,3)`	-2.67
`:mod(5.67,-3)`	2.67
`:mod(-5.67,-3)`	-2.67
`:div(5.67)`	5
`:div(-5.67)`	-5
`:mod(5.67)`	0.67
`:mod(-5.67)`	-0.67

⟨actor⟩ ::= ⟨word⟩

⟨ausym⟩ ::= − | _

⟨axes-name⟩ ::= z | z_1 | z_2 | z_3 | r | ϕ | ψ

⟨boole⟩ ::= **:t** | **:f** | **:true** | **:false**

⟨capital-letter⟩ ::= **A** | **B** | **C** | . . . | **Z**

⟨capital-word⟩ ::= ⟨capital-letter⟩ | ⟨capital-word⟩ ⟨capital-letter⟩
| ⟨capital-word⟩ ⟨ausym⟩**)**

⟨clock-time⟩ ::= ⟨hour⟩**:**⟨minute⟩**:**⟨second⟩

⟨composite-concept⟩† ::= ⟨capital-word⟩

⟨'composite-concept⟩† ::= ⟨word⟩

⟨composite-relation⟩† ::= ⟨capital-word⟩

⟨'composite-relation⟩† ::= ⟨word⟩

⟨concept⟩† ::= ⟨capital-word⟩

⟨'concept⟩† ::= ⟨word⟩

⟨day⟩ ::= **1** | **2** | . . . | **31**

⟨date⟩ ::= ⟨day⟩**.**⟨month⟩**.**⟨year⟩

⟨differential⟩ ::= ⟨ordinary-differential⟩ | ⟨partial-differential⟩

⟨digit⟩ ::= **0** | **1** | **2** | . . . | **9**

⟨equation⟩ ::= ⟨arex⟩ **:eq** ⟨arex⟩ | ⟨arex⟩ = ⟨arex⟩

⟨hour⟩ ::= **1** | **2** | . . . | **24**

⟨integer⟩ ::= ⟨pos-int⟩ | ⟨neg-int⟩

⟨integro-differential⟩ ::= **...**

⟨letter⟩ ::= ⟨small-letter⟩ | ⟨capital-letter⟩

⟨metaclass⟩† ::= **elementary-concept-properties**
| **composite-concept-properties**
| **elementary-relation-properties**
| **composite-relation-properties**
| **sufficiently-defined** | **not-sufficiently-defined**
| **abstract-class** | **non-abstract-class**

⟨method⟩† ::= ⟨capital-word⟩

⟨'method⟩† ::= ⟨word⟩

Tabelle 3: Definition systemdefinierter Datentypen in BNF. Mit † gekennzeichnete Daten sind die Namen von Datentypen und nicht die Datentypen selbst.

⟨minute⟩ ::= **1** | **2** | . . . | **60**

⟨month⟩ ::= **Jan** | **Feb** | **Mar** | **Apr** | **May** | . . . | **Dec**

⟨neg-int⟩ ::= – ⟨pos-int⟩

⟨number⟩ ::= ⟨boole⟩ | ⟨real⟩ | ⟨integer⟩

⟨object⟩† ::= ⟨concept⟩ | ⟨relation⟩

⟨'object⟩† ::= ⟨'concept⟩ | ⟨'relation⟩

⟨ordinary-differential⟩ ::= **'ordinary-differential-ivp**
| **'ordinary-differential-bvp**

⟨partial-differential⟩ ::= **'partial-differential-ibvp**
| **'partial-differential-bvp**

⟨pos-int⟩ ::= ⟨digit⟩ | ⟨pos-int⟩⟨digit⟩

⟨punctuation⟩ ::= **.** | **,** | **:** | **;**

⟨predicate⟩ ::= Notation für Prädikate

⟨r-body⟩ ::= Notation für komplexe Relationen

⟨real⟩ ::= ⟨integer⟩**.**⟨pos-int⟩ | ⟨integer⟩**.**⟨pos-int⟩**E**⟨integer⟩

⟨relation⟩† ::= ⟨capital-word⟩

⟨'relation⟩† ::= ⟨word⟩

⟨role⟩ ::= ⟨word⟩

⟨r-type⟩ ::= **equation** | . . .

⟨second⟩ ::= **1** | **2** | . . . | **60**

⟨slot⟩ ::= ⟨word⟩ **:**

⟨small-letter⟩ ::= **a** | **b** | . . . | **z**

⟨tensor-equation-type⟩ ::= **'algebraic** | **'integral**
| ⟨differential⟩ | ⟨integro-differential⟩

⟨text⟩ ::= ⟨word⟩ | ⟨text⟩ ⟨word⟩ | ⟨text⟩ ⟨punctuation⟩ | ⟨punctuation⟩ ⟨text⟩

⟨time⟩ ::= ⟨date⟩ ⟨clock-time⟩

⟨trend⟩ ::= **+** | **–** | **0**

⟨value⟩ ::= primitiver Datentyp

⟨'value⟩ ::= Instanz eines primitiven Datentyps

⟨word⟩ ::= ⟨letter⟩ | ⟨word⟩⟨small-letter⟩ | ⟨word⟩⟨digit⟩ | ⟨word⟩⟨ausym⟩

⟨year⟩ ::= ⟨pos-int⟩

Tabelle 4: Definition systemdefinierter Datentypen in BNF (Fortsetzung). Mit † gekennzeichnete Daten sind die Namen von Datentypen und nicht die Datentypen selbst.

⟨arex⟩ ::=
  ⟨real⟩ | ⟨variable-ref⟩
  | **(** ⟨arex⟩ **)**
  | ⟨arex⟩ ⟨binary-infix-op⟩ ⟨arex⟩
  | ⟨arex⟩ ⟨unary-postfix-op⟩
  | ⟨unary-prefix-op⟩ ⟨arex⟩
  | ⟨extended-un-pre-op⟩ ⟨arex⟩
  | ⟨binary-prefix-op⟩**(** ⟨arex⟩ **,** ⟨arex-of-int⟩ **)**
  | ⟨multiple-arg-function⟩**(** ⟨arex-list⟩ **)**
  | **:cond(** ⟨if-then-else-list⟩ **)**

⟨arex-of-int⟩ ::=
  ⟨integer⟩ | ⟨variable-ref⟩
  | **(** ⟨arex-of-int⟩ **)**
  | ⟨arex-of-int⟩ ⟨binary-infix-op-of-int⟩ ⟨arex-of-int⟩
  | ⟨arex-of-int⟩ ⟨unary-postfix-op⟩
  | ⟨unary-prefix-op-of-int⟩ ⟨arex-of-int⟩
  | ⟨extended-un-pre-op-of-int⟩ ⟨arex-of-int⟩
  | **:div(** ⟨arex⟩ **,** ⟨arex-of-int⟩ **)**
  | **:mod(** ⟨arex-of-int⟩ **,** ⟨arex-of-int⟩ **)**

⟨logical-expr⟩ ::=
  ⟨boole⟩
  | **(** ⟨logical-expr⟩ **)**
  | ⟨arex⟩ ⟨relational-bin-inf-op⟩ ⟨arex⟩
  | ⟨logical-expr⟩ ⟨logical-bin-inf-op⟩ ⟨logical-expr⟩
  | ⟨logical-un-pre-op⟩ ⟨logical-expr⟩

⟨variable-ref⟩ ::= ⟨word⟩ | ⟨word⟩**[** ⟨index-list⟩ **]**

⟨index-list⟩ ::= ⟨arex⟩ | ⟨index-list⟩ (**;** | **,**) ⟨arex⟩

⟨arex-list⟩ ::= ⟨arex⟩ | ⟨arex-list⟩ **,** ⟨arex⟩

⟨if-then-else-list⟩ ::= ⟨logical-expr⟩ **,** ⟨arex⟩ **,** ⟨if-then-else-list⟩ | ⟨arex⟩

⟨binary-infix-op⟩ ::= ⟨binary-infix-op-of-int⟩ | **/**

⟨binary-infix-op-of-int⟩ ::= **^** | * | **+** | **−**

⟨unary-postfix-op⟩ ::= **!**

⟨unary-prefix-op⟩ ::=
  ⟨unary-prefix-op-of-int⟩ | **:sqrt** | **:log** | **:ln** | **:exp** | **:sin** | **:cos** | **:tan**
  | **:arcsin** | **:arccos** | **:arctan** | **:sinh** | **:cosh** | **:tanh**
  | **:arsinh** | **:arcosh** | **:artanh** | **:abs**

⟨unary-prefix-op-of-int⟩ ::= **−** | **:div** | **:mod**

Tabelle 5: Definition arithmetischer (arex = 'arithmetic expression') und logischer (logical-expr) Ausdrücke.

⟨extended-un-pre-op⟩ ::=
⟨extended-un-pre-op-of-int⟩
| **:diff**< ⟨var-list⟩ >
| **:int**< ⟨variable-ref⟩ >
| **:int**< ⟨variable-ref⟩ **,** ⟨arex⟩ **,** ⟨arex⟩ >

⟨extended-un-pre-op-of-int⟩ ::=
⟨sum-or-prod⟩< ⟨word⟩ >
| ⟨sum-or-prod⟩< ⟨word⟩ **,** ⟨arex-of-int⟩ **,** ⟨arex-of-int⟩ >

⟨var-list⟩ ::= ⟨variable-ref⟩ | ⟨var-list⟩ **,** ⟨variable-ref⟩

⟨sum-or-prod⟩ ::= **:sum** | **:prod**

⟨binary-prefix-op⟩ ::= **:div** | **:mod**

⟨multiple-arg-function⟩ ::= **:max** | **:min**

⟨relational-bin-inf-op⟩ ::=
:lt | **:eq** | **:neq** | **:le** | **:ge** | **:gt**
| < | == | != | <> | <= | >=

⟨logical-bin-inf-op⟩ ::= **:and** | **:or**

⟨logical-un-pre-op⟩ ::= **:not**

Tabelle 6: Definition arithmetischer und logischer Ausdrücke (Fortsetzung).

	Prioritätsreihenfolge	Assoziativität
1.	Klammern	innen nach außen
2.	! ⟨unary-postfix-op⟩	links nach rechts
3.	ˆ	rechts nach links
4.	[nicht verwendet]	
5.	⟨unary-prefix-op⟩ ⟨extended-un-pre-op⟩ ⟨binary-prefix-op⟩ ⟨logical-un-pre-op⟩	rechts nach links
6.	[nicht verwendet]	
7.	* /	links nach rechts
8.	+ −	links nach rechts
9.	:and	links nach rechts
10.	:or	links nach rechts
11.	⟨relational-bin-inf-op⟩	—
12.	= Zuweisung	rechts nach links

Tabelle 7: Priorität und Assoziativität der arithmetischen und logischen Operatoren.

D Aufruf des Code-Generators vom Betriebssystem

File /usr/local/lisp/code-generator/README:

```
09.05.94,S.R.
               How to load the Code-Generator on the SUN
               =========================================

========================================================================
Setting up your environment once:
========================================================================
Following line must be in your .cshrc file:

set LISP_USER          # if you want to use Allegro Common Lisp


========================================================================
Interactive usage of the Code-Generator:
========================================================================

Type at a terminal window:

runcg
-----

You are then asked for the Code-Generator input-file and where to put
the files, which should be generated.


========================================================================
Batch usage of the Code-Generator:
========================================================================

Put four calls into a Lisp command file <cg-command-file> to
- load the Code-Generator
- initialise the Code-Generator
- generate the files
- exit Lisp

Example:

;;; begin of <cg-command-file>
(load "/usr/local/lisp/code-generator/load-cg")
(cg-init :quiet nil :verbose-level 1)   ; max. verbose-level is 100
(cg-fortran-dat
  :input-file "../in/pt2.cg"
  :default-output-subname "ab01"
  :default-output-pathname "../out/foo.for")
:exit
;;; end of <cg-command-file>

Then type at a terminal window:

runcg <cg-command-file>
-----------------------
```

```
========================================================================
Interactive usage of the Code-Generator via the editor emacs:
========================================================================

1.) To start emacs, type at a terminal window:

emacs &
-------

2,) To load Allegro Common Lisp, type at the emacs window:

<ESC> x allegro
---------------

3.) To load the Code-Generator, type at the emacs-lisp-buffer:

(load "/usr/local/lisp/code-generator/load-cg")
-----------------------------------------------

4.) Now you can call the Code-Generator functions, e. g. (cg-init).

...

5.) To exit Allegro Common Lisp, type:

:exit
-----

6.) To exit emacs, type:

<Ctrl-x> <Ctrl-c>


========================================================================
Remarks:
========================================================================

Control of loading the Code-Generator:
======================================

In step 3.) defsystem is used to control the loading of the
Code-Generator. Alternativly the file "make.lisp" can be loaded. This
file defines some more functions that can be used for further
loading. At the moment this kind of loading is not updated or
supported.


Short commands and user individual abbrevations:
================================================

If you want to use short commands, then you
must copy the file ".clinit.cl" into your home directory.
This file is called when allegro common lisp is loaded.
Instead of step 3.) you can type:
           :load-cg
```

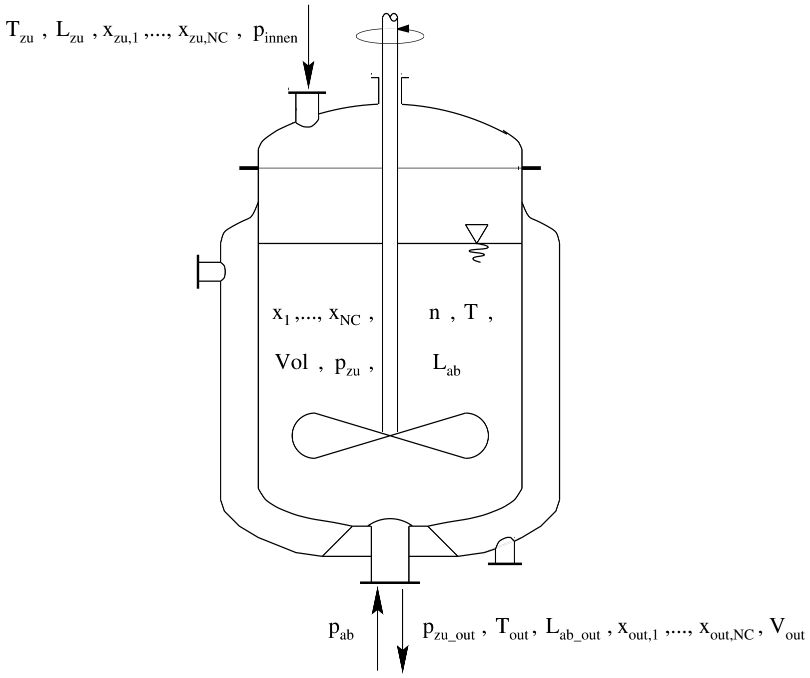
```
          --------
You can also edit your copy of the file and define your own short
commands.


Code-Generator development:
===========================

For Code-Generator development purposes, you can copy the file
"my-load-cg.lisp" [or "make.lisp" for alternative loading] into your
directory and alter it so that your development files instead of the
official Code-Generator files are going to be loaded.  Instead of step
3.) you would load your "my-load-cg" [or "make"] file.
```

E Code-Generator Eingabe-File: Beispiel Behälter

Die Modellgleichungen zu diesem Einphasen-Flüssigkeits-Behälter sind in der zugehörigen DIVA-Blockbeschreibung enthalten.




```
;;; Input-File for the Code-Generator

;;; 0.1 Documentation for head of FORTRAN programs
;;;---------------------------------------------------------------------------
(define-header
  :program "efb"
  :documentation "Einphasen-Fl\"ussigkeits-Beh\"alter
x[1], x[2] und n sind dynamische Zust\"ande.

Gleichungen (1), (11) und (14) lassen sich unter einem Index 1,...,NC-1
  zusammenfassen, wobei (11) und (14) explizit fuer NC dazu muss.

Diverse Stoffdatenberechnungen werden mittels externer
  FORTRAN-Unterprogramme berechnet.
"
  :date "26.02.99"
  :filename "efb.cg")

;;; 0.2 Name of time variable
;;; --------------------------------------------------------------------------
(define-time-variable :name "SimTime")



;;; 1. Parameters:
```

```
;;; ---------------------------------------------------------------------------

;;; 1.1 Structure Parameters:
;;; - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
(define-structure-parameters
  (:name "NC"
   :meaning "Anzahl der Komponenten"
   :default-value 3
  )
) ;; end of structure-parameters

;;; 1.2 Integer Parameters (FORTRAN vector IPAR):
;;; - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
;; not needed

;;; 1.3 Real Parameters (FORTRAN vector RPAR):
;;; - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
(define-real-parameters
  (:name "Fn"
   :meaning "Normalmolenstrom bei delta_pn"
   :default-value 0.1
  )
  (:name "delta_pn"
   :meaning "Normaldruckverlust bei Fn"
   :default-value 200
  )
  (:name "AA"
   :meaning "Flaeche"
   :default-value 0.5
  )
  (:name "alpha"
   :meaning "Waermeuebergangskoeffizent"
   :default-value 0.0 ;?
  )
  (:name "Tu"
   :meaning "Umgebungstemperatur"
   :default-value 293.0
  )
  (:name "M"
   :meaning "Molmasse der Komponente
Die Daten wurden aus der Zeile 1,18,35 des Stoffdatenfiles entnommen."
    :indices ((:lower-boundary 1 :upper-boundary "NC"))
    :default-value (array-list :values (46.07 18.02 60.1))
  )
  (:name "vb"
   :meaning "molares Reinstoffvolumen bei Bezugstemperatur Tb der Komponente
Die Daten wurden aus der Zeile 7,24,41 des Stoffdatenfiles entnommen."
   :indices ((:lower-boundary 1 :upper-boundary "NC"))
   :default-value
      (array-list :values (0.586804E-01 0.180742E-01 0.751532E-01))
  )
  (:name "Tb"
   :meaning "Bezugstemperatur der Komponente
Der konstante Wert wurde aus der Zeile 8,25,42 des Stoffdatenfiles
entnommen."
    :indices ((:lower-boundary 1 :upper-boundary "NC"))
```

```
   :default-value 298.15
  )
  (:name "Tkr"
   :meaning "kritische Temperatur der Komponente
Die Daten wurden aus der Zeile 3,20,37 des Stoffdatenfiles entnommen."
   :indices ((:lower-boundary 1 :upper-boundary "NC"))
   :default-value (array-list :values (513.9 647.3 536.7))
  )
  (:name "A"
   :meaning "Parameter zur Berechnung der Enthalpie der Komponente"
   :indices ((:lower-boundary 1 :upper-boundary "NC"))
   :default-value (array-list :values (24.82 18.04 31.92))
  )
  (:name "B"
   :meaning "Parameter zur Berechnung der Enthalpie der Komponente"
   :indices ((:lower-boundary 1 :upper-boundary "NC"))
   :default-value (array-list :values (0.4296E-01 0.7465d-03 0.449E-01))
  )
  (:name "C"
   :meaning "Parameter zur Berechnung der Enthalpie der Komponente"
   :indices ((:lower-boundary 1 :upper-boundary "NC"))
   :default-value (array-list :values (0.1474E-03 0.1001d-05 0.9663E-04))
  )
  (:name "g"
   :meaning "Erdbeschleunigung"
   :default-value 9.81
  )
) ;; end of real-parameters

;;; 1.4 Character Parameters (FORTRAN vector CPAR):
;;; - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
;; not needed



;;; 2. Model Inputs (FORTRAN vector U):
;;; ------------------------------------------------------------------------
(define-input-variables
  (:name "Tzu"
   :meaning "Zulauftemperatur")
  (:name "Lzu"
   :meaning "Zulaufmolenstrom")
  (:name "xzu"
   :meaning "Molenbruch der Komponente im Zulauf"
   :indices ((:lower-boundary 1 :upper-boundary "NC"))
  )
  (:name "pab"
   :meaning "Druck im Ablauf")
  (:name "pinnen"
   :meaning "Druck ueber der Fluessigkeit")
) ;; end of input-variables



;;; 3. Model States (FORTRAN vector X):
;;; ------------------------------------------------------------------------
```

```
(define-state-variables
  (:name "x"
   :meaning "Molenbruch der Komponente"
   :indices (( :lower-boundary 1 :upper-boundary "NC"))
   :default-state-value  ;Anfangswert
     (array-list :values (0.10728 0.40722 0.48550))
   :default-min-value 0
   :default-max-value 1
   :default-numerical-error-value 0.001
  )
  (:name "n"
   :meaning "Molinhalt"
   :default-state-value 0.1140261971E+02 ;Anfangswert
   :default-min-value 0
   :default-max-value 1000
   :default-numerical-error-value 0.0001
  )
  (:name "T"
   :meaning "Temperatur"
   :default-state-value 364.311 ;Anfangswert
   :default-min-value 300
   :default-max-value 550
   :default-numerical-error-value 0.1
  )
  (:name "Vol"
   :meaning "Volumen des Behaelters"
   :default-state-value 0.6 ;Anfangswert
   :default-min-value 0
   :default-max-value 1000
   :default-numerical-error-value 0.0001
  )
  (:name "pzu"
   :meaning "Druck im Zulauf"
   :default-state-value 6.20E+5 ;Anfangswert
   :default-min-value 1.0E+5
   :default-max-value 1.0E+6
   :default-numerical-error-value 1
  )
  (:name "Lab"
   :meaning "Ablaufmolenstrom"
   :default-state-value 0.1 ;Anfangswert
   :default-min-value 0
   :default-max-value 1
   :default-numerical-error-value 0.0001
  )
) ; end of state-variables



;;; 4. Model Outputs (FORTRAN vector IOUT):
;;; -------------------------------------------------------------------------
(define-output-variables
  (:name "pzu_out"
   :meaning "Druck im Zulauf"
   :is-state-variable-with-name "pzu"
  )
```

```
  (:name "T_out"
   :meaning "Temperatur"
   :is-state-variable-with-name "T"
  )
  (:name "Lab_out"
   :meaning "Ablaufmolenstrom"
   :is-state-variable-with-name "Lab"
  )
  (:name "x_out"
   :meaning "Konzentration der Komponente"
   :indices ((:lower-boundary 1 :upper-boundary "NC"))
   :index-names ("i")
   :is-state-variable-with-name "x[i]"
  )
  (:name "V_out"
   :meaning "Volumen"
   :is-state-variable-with-name "Vol"
  )
) ;; end of output-variables



;;; 5. Help Variables:
;;; ---------------------------------------------------------------------------
(define-help-variables
  (:name "xsi0"
   :meaning "Druckverlustbeiwert1"
   :value "0.1*delta_pn/Fn"
  )
  (:name "xsi"
   :meaning "Druckverlustbeiwert2"
   :value "0.9*delta_pn/Fn^2"
  )
  (:name "h_ge"
   :meaning "molare Gemischenthalpie"
   )
  (:name "h"
   :meaning "molare Reinstoffenthalpie der Komponente"
   :indices ((:lower-boundary 1 :upper-boundary "NC"))
  )
  (:name "M_ge"
   :meaning "Molmasse des Gemischs"
  )
  (:name "v_ge"
   :meaning "molares Gemischvolumen"
  )
  (:name "v"
   :meaning "molares Reinstoffvolumen der Komponente"
   :indices ((:lower-boundary 1 :upper-boundary "NC"))
  )
  (:name "hzu_ge"
   :meaning "molare Gemischenthalpie im Zulauf"
   )
  (:name "hzu"
   :meaning "molare Reinstoffenthalpie im Zulauf der Komponente"
   :indices ((:lower-boundary 1 :upper-boundary "NC"))
```

```
  )
  (:name "cp_ge"
   :meaning "molare Waermekapazitaet (Fluessigkeit)"
   )
  (:name "cp"
   :meaning "molare Waermekapazitaet (Fluessigkeit) der Komponente"
   :indices ((:lower-boundary 1 :upper-boundary "NC"))
  )
  (:name "T0"
   :meaning "Bezugstemperatur innen"
  )
  (:name "T0zu"
   :meaning "Bezugstemperatur Zulauf"
  )
) ;; end of help-variables



;;; 6. Process Values (grouping of data output):
;;; --------------------------------------------------------------------------
(define-process-values
  (:name "TANK_"
   :meaning "Tank innen"
   :components
   ((:index ("i" 1 "NC")
     :comp-scope
     ( "x[i]" )) ; comp-scope/index i end
    "n"
    "T"
    "Vol"
    "pzu"
    "Lab"
   )
  )
  (:name "IN___"
   :meaning "Eingangsgroessen"
   :components
   ("pzu"
    "Tzu"
    "Lzu"
    (:index ("i" 1 "NC")
     :comp-scope
     ( "xzu[i]" )) ; comp-scope/index i end
   )
  )
  (:name "OUT__"
   :meaning "Ausgangsgroessen"
   :components
   ("pab"
    "T"
    "Lab"
    (:index ("i" 1 "NC")
     :comp-scope
     ( "x[i]" )) ; comp-scope/index i end
   )
  )
```

```
  (:name "INTRN"
   :meaning "Druck intern"
   :components ( "pab" )
  )
  (:name "VOL__"
   :meaning "Volumen"
   :components ( "Vol" )
  )
) ;; end of process-values



;;; 7. Model Equations
;;; -----------------------------------------------------------------------------
(define-equations

  ;;
  ;; Hilfsgroessenberechnungen:
  ;;
  (:assignment-name
   ;; -------------------------------------------
      "Bezugstemperatur T0"
   ;; -------------------------------------------
   :meaning "wird bei Stoffdatenberechnung benoetigt"
   :help-variable "T0"
   :h-value "T - 273.15"
  )
  (:assignment-name
   ;; -------------------------------------------
      "Bezugstemperatur T0zu"
   ;; -------------------------------------------
   :meaning "wird bei Stoffdatenberechnung benoetigt"
   :help-variable "T0zu"
   :h-value "Tzu - 273.15"
  )
    ;; Indices are automatically assumed !!!
    ;;  (:index ("i" 1 "NC")
    ;;   :eq-scope
    ;;   (
  (:procedure-name
   ;; -------------------------------------------
      "CGP_Enthalpie"
   ;; -------------------------------------------
   :compute "h[:all]"
   :meaning "Man beachte, dass H0[i] nicht vorkommt
     und dass ein Einheiten-Korrekturfaktor 4.186 benutzt wird."
   :args ("A[:all]" "B[:all]" "C[:all]" "T0")
   :subroutine-names ("H" "H1LEN" "A" "A1LEN" "B" "B1LEN" "C" "C1LEN" "TEMP")
   :fortran-code "
C
C      L   Deklaration der Laufvariablen i fuer die Berechnungsschleife :
C
           INTEGER          i
C
       DO 100 i = 1, h1LEN
             H(i) =  4.186 * ( A(i) + (B(i) + C(i) * TEMP )) * TEMP ** 2
```

```
  100  CONTINUE
"
  )
    ;;   )) ; eq-scope/index i end
  (:procedure-name
   ;; --------------------------------------------
      "CGP_Enthalpie"
   ;; --------------------------------------------
   :compute "hzu[:all]"
   :meaning "Verwendung des gleichen Unterprogramms wie bei h."
   :args ("A[:all]" "B[:all]" "C[:all]" "T0zu")
  )
  (:assignment-name
   ;; --------------------------------------------
      "Enthalpie-Berechnung h_ge"
   ;; --------------------------------------------
   :meaning "Verwendung einer Summe ueber alle Komponenten"
   :help-variable "h_ge"
   :h-value ":sum<i,1,NC>(h[i]*x[i])"
  )
  (:assignment-name
   ;; --------------------------------------------
      "Enthalpie-Berechnung hzu_ge"
   ;; --------------------------------------------
   :meaning "Verwendung einer Summe ueber alle Komponenten"
   :help-variable "hzu_ge"
   :h-value ":sum<i,1,NC>(hzu[i]*xzu[i])"
  )
  (:assignment-name
   ;; --------------------------------------------
      "Molmasse des Gemischs M_ge"
   ;; --------------------------------------------
   :meaning "Verwendung einer Summe ueber alle Komponenten"
   :help-variable "M_ge"
   :h-value ":sum<i,1,NC>(M[i]*x[i])"
  )
  (:assignment-name
   ;; --------------------------------------------
      "molares Volumen v[i]"
   ;; --------------------------------------------
   :meaning "Berechnung nach Cavett"
   :help-variable "v[i]"
   :h-value "vb[i]/(5.7 + (3.0*Tb[i])/Tkr[i])*(5.7 + 3.0*(T/Tkr[i]))"
  )

;; alternative Berechnung mit externem FORTRAN-Unterprogramm:
;  (:procedure-name
   ;; --------------------------------------------
      "CGP_Cavett"
   ;; --------------------------------------------
;    :compute "v[:all]"
;    :meaning "Berechnung nach Cavett"
;    :args ("vb[:all]" "Tb[:all]" "Tkr[:all]" "T")
;    :subroutine-names ("V" "V1LEN" "VB" "VB1LEN" "TB" "TB1LEN"
;    "TKR" "TKR1LEN" "TEMP")
;    :fortran-code "
```

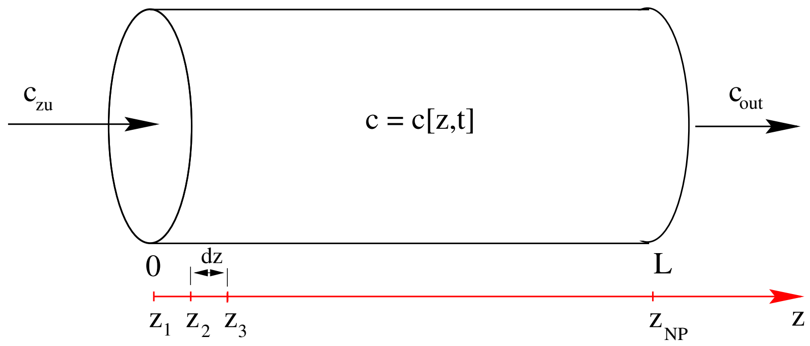
```
;C
;C      L   Deklaration der Laufvariablen i fuer die Berechnungsschleife :
;C
;           INTEGER          i
;C
;       DO 100 i = 1, V1LEN
;             V(i) = VB(i) / (5.7 + (3.0 * TB(i))/TKR(i)) *
;     #              (5.7 + 3.0 * ( TEMP/TKR(i) ))
;  100  CONTINUE
;"
;  )

  (:assignment-name
   ;; --------------------------------------------
      "molares Volumen v_ge"
   ;; --------------------------------------------
   :meaning "Verwendung einer Summe ueber alle Komponenten"
   :help-variable "v_ge"
   :h-value ":sum<i,1,NC>(v[i]*x[i])"
  )
  (:assignment-name
   ;; --------------------------------------------
      "molare Waermekapazitaet (Fluessigkeit) cp[i]"
   ;; --------------------------------------------
   :help-variable "cp[i]"
   :meaning "Berechnung nach Polynomansatz (Zeitableitung von h[i])"
   :h-value "(A[i] + (2.0*B[i] + 3.0*C[i]*T0)*T0)*4.186"
  )
  (:assignment-name
   ;; --------------------------------------------
      "molare Waermekapazitaet (Fluessigkeit) cp_ge"
   ;; --------------------------------------------
   :help-variable "cp_ge"
   :meaning "aehnliche Berechnung wie h_ge"
   :h-value ":sum<i,1,NC>(cp[i]*x[i])"
  )

  (:index ("i" 1 "NC - 1")
   :meaning "Komponentenmaterialbilanz (Teil)"
   :eq-scope
   ((:equation-name "KMB (1)"
     :meaning "Komponentenmaterialbilanz (Teil Komponente i)
       Achtung: B-Matrix-Eintr\"age in aufsteigender Reihenfolge angeben
                [ x[1] vor x[2] vor n ]"
     :f-value "Lzu*xzu[i] - Lab*x[i]"
     :b-values ((:state-variable-reference "x[i]"
         :b-value "n")
        (:state-variable-reference "n"
         :b-value "x[i]"))
    )
  )) ; eq-scope/index i end
  (:equation-name "SB (2)"
   :meaning "Schliessbedingung
      1.0 statt 1 benutzen !"
   :f-value "1.0 - :sum<i,1,NC>(x[i])" ; 1.0 instead of 1 !!!
   :b-values ()
```

```
  )
  (:equation-name "GMB (3)"
   :meaning "Gesamtmaterialbilanz"
   :f-value "Lzu - Lab"
   :b-values ((:state-variable-reference "n"
       :b-value 1))
  )
  (:equation-name "EB (4)"
   :meaning "Energiebilanz
     Achtung: B-Matrix-Eintr\"age in aufsteigender Reihenfolge angeben
              [ x[1] vor x[2] vor n vor T ]
     Fuer eine ideale Mischung gilt h[i]=hp[i]. Damit wird die
       partielle Enthalpie hp[i] durch die molare Reinstoffenthalpie
       h[i] in  n*(h[i] - h[NC]) * :diff[x[i],t] ersetzt."
   :f-value "Lzu*hzu_ge - Lab*h_ge - alpha*AA*(T - Tu)"
   :b-values ((:index ("i" 1 "NC - 1")
               :b-elem-scope
                 ((:state-variable-reference "x[i]"
                   :b-value "n*(h[i] - h[NC])")
                 )) ; b-elem-scope/index i end
              (:state-variable-reference "n"
       :b-value "h_ge")
              (:state-variable-reference "T"
       :b-value "n*cp_ge")
             )
  )
  (:equation-name "Fluessigkeitsvolumen (5)"
   :meaning "Fluessigkeitsvolumen"
   :f-value "Vol - n*v_ge"
   :b-values ()
  )
  (:equation-name "Druckbeziehung (6)"
   :meaning "Druckbeziehung"
   :f-value "- pinnen + pzu"
   :b-values ()
  )
  (:equation-name "Druckverlustbeziehung (7)"
   :meaning "Druckverlustbeziehung"
   :f-value "(pzu + M_ge*n*g/AA - pab) - (xsi*Lab*Lab + xsi0*Lab)"
   :b-values ()
  )
) ;; end of equations
```

F Code-Generator Eingabe-File: Beispiel Rohrreaktor



Partielle Differentialgleichung:

$$\frac{\partial c}{\partial t} = D \cdot \frac{\partial^2 c}{\partial z^2} - v \cdot \frac{\partial c}{\partial z} - k \cdot c, \quad t > 0, \quad 0 \leq z \leq L$$

Anfangsbedingung:

$$c[z, t = 0] = 0, \quad 0 \leq z \leq L$$

Randbedingung links:

$$v \cdot c[z = 0, t] - D \cdot \left.\frac{\partial c}{\partial z}\right|_{z=0} = v \cdot c_{zu}$$

Randbedingung rechts:

$$\left.\frac{\partial c}{\partial z}\right|_{z=L} = 0 \quad (D \neq 0)$$

Ortsdiskretisierung mit

$$z_i = (i-1) \cdot \frac{L}{(NP-1)}, \quad c[z_i, t] = c_i[t], \quad i = 1 \cdots NP$$

$$\left.\frac{\partial c}{\partial z}\right|_{z=z_i} \approx \frac{c_i - c_{i-1}}{dz}, \quad \left.\frac{\partial^2 c}{\partial z^2}\right|_{z=z_i} \approx \frac{c_{i+1} - 2c_i + c_{i-1}}{dz^2}$$

ergibt die Differential-Algebra-Gleichungen

$$\begin{aligned}
0 &= v \cdot (c_{zu} - c_1) + D \cdot \frac{c_2 - c_1}{dz} \\
\dot{c}_i &= -v \cdot \frac{c_i - c_{i-1}}{dz} + D \cdot \frac{c_{i+1} - 2c_i + c_{i-1}}{dz^2} - k \cdot c_i, \quad i = 2 \cdots NP - 1 \\
0 &= c_{NP} - c_{NP-1}
\end{aligned}$$

mit $dz := \frac{L}{(NP-1)}$ und $c_i[t = 0] = 0$.

```
;;; Input-File for the Code-Generator

;;; 0.1 Documentation for head of FORTRAN programs
;;; ---------------------------------------------------------------------------
```

```
(define-header
  :program "ovs"
  :date "24.01.95"
  :filename "ovs.cg"
  :documentation "Ein oertlich verteiltes System: Instationaerer Rohrreaktor.
  (Stroemungsgleichung mit Diffusion; D=0: reine Stroemungsgleichung)

                2
    dc         d c         dc
    -- =  D * ----  -  v * --  -  k * c    ,    c = c[z,t]
    dt          2          dz
              dz

  Initial condition:  c[z,  t=0] = 0
  Left boundary condition:
                       dc |
    v * c[z=0,t] - D * -- |   = v * czu
                       dz |
                          |z=0
  Right boundary condition (if D <> 0):
    dc |
    -- |   = 0
    dz |
       |z=L

  'method of lines'"
  )

;;; 0.2 Name of time variable
;;; ------------------------------------------------------------------------
(define-time-variable :name "SimTime")



;;; 1. Parameters:
;;; ------------------------------------------------------------------------

;;; 1.1 Structure Parameters:
;;; - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
(define-structure-parameters
  (:name "NP"
   :meaning "Anzahl der Ortsstuetzstellen"
   :default-value 201
  )
) ;; end of structure-parameters

;;; 1.2 Integer Parameters (FORTRAN vector IPAR):
;;; - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
;; not needed

;;; 1.3 Real Parameters (FORTRAN vector RPAR):
;;; - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
(define-real-parameters
  (:name "L"
   :meaning "Laenge des Stroemungsabschnittes"
   :default-value 2.0
```

```
  )
  (:name "k"
   :meaning "Reaktionskonstante [1/s]"
   :default-value 1.0
  )
  (:name "v"
   :meaning "Stroemungsgeschwindigkeit [m/s]"
   :default-value 1.0
  )
  (:name "D"
   :meaning "Diffusionskoeffizent [m^2/s]"
   :default-value 0.000001
  )
) ;; end of real-parameters

;;; 1.4 Character Parameters (FORTRAN vector CPAR):
;;; - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
;; not needed



;;; 2. Model Inputs (FORTRAN vector U):
;;; ---------------------------------------------------------------------------
(define-input-variables
  (:name "czu" :meaning "Zuflaufkonzentration [kmol/m^3]")
) ;; end of input-variables



;;; 3. Model States (FORTRAN vector X):
;;; ---------------------------------------------------------------------------
(define-state-variables
  (:name "c"
   :meaning "Konzentration an der i-ten Ortsstuetzstelle [kmol/m^3]"
   :indices ((:lower-boundary 1 :upper-boundary "NP"))
   :default-state-value 0                         ; Anfangsbedingung
   :default-min-value 0
   :default-max-value 100
   :default-numerical-error-value 0.1
  )
) ;; end of state-variables



;;; 4. Model Outputs (FORTRAN vector IOUT):
;;; ---------------------------------------------------------------------------
(define-output-variables
  (:name "cout"
   :meaning "Konzentration am Rohrausgang [kmol/m^3]"
   :is-state-variable-with-name "c[NP]"
  )
) ;; end of output-variables



;;; 5. Help Variables:
```

```
;;; --------------------------------------------------------------------------
(define-help-variables
  (:name "dz"
   :meaning "Ortsabstand"
  )
  (:name "z"
   :meaning "Ortst\"utsstellen"
   :indices ((:lower-boundary 1 :upper-boundary "NP"))
   :index-names ("i")
   :value "(i-1) * L/(NP-1)"
 )
) ;; end of help-variables



;;; 6. Process Values (grouping of data output):
;;; --------------------------------------------------------------------------
(define-process-values
  (:name "KONZ_"
   :meaning "Konzentrationen [kmol/m^3]"
   :x-scope (
     (:index ("i" 1 "NP" 2)
      :meaning "Konzentration an jeder zweiten Ortsst\"utzstelle"
      :components ( "z[i]" "c[i]" )
     )
   )
  )
) ;; end of process-values



;;; 7. Model Equations
;;; --------------------------------------------------------------------------
(define-equations
  (:assignment-name
   ;; --------------------------------------------
      "Ortsabstand"
   ;; --------------------------------------------
   :meaning "Hilfsgr\"o\"senberechnung:
             Abstand der Ortsst\"utzstellen: dz := L/(NP-1)"
   :help-variable "dz"
   :h-value "L/(NP-1)"
  )
  (:equation-name
   ;; --------------------------------------------
      "RB links"
   ;; --------------------------------------------
   :meaning "Randbedingung links bei c[1]  (speziell fuer D=0: c[1] = czu)"
   :f-value "v*(czu - c[1]) + D*(c[2] - c[1])/dz"
   :b-values ()
  )
  (:index ("i" 2 "NP-1")
   :meaning
   ;; --------------------------------------------
     "Index f\"ur den Innenbereich (i = 2,3,...,NP-1)"
   ;; --------------------------------------------
```

```
  :eq-scope
  ((:equation-name "Diskretisierte Pdgl."
    :meaning "Gleichungen f\"ur den Innenbereich (i = 2 ... NP-1)
      Konzentrations\"anderung aufgrund Str\"omung, Diffusion
      und Reaktion
      :diff(c[i],t) = -v*(c[i] - c[i-1])/dz
                     + D*(c[i+1] -2*c[i] + c[i-1])/dz^2 - k*c[i]"
   :f-value
      " -v*(c[i] - c[i-1])/dz
       + D*(c[i+1] - 2*c[i] + c[i-1])/dz^2
       - k*c[i]"
    :b-values ((:state-variable-reference "c[i]" :b-value 1))
   )
 ))
 (:equation-name
  ;; ---------------------------------------------
     "RB rechts"
  ;; ---------------------------------------------
  :meaning "Randbedingung rechts bei c[NP]:  :diff(c[NP],z) = 0, d.h.
          0 = (c[NP] - c[NP-1])/dz oder c[NP]=c[NP-1]"
  :f-value "c[NP] - c[NP-1]"
  :b-values ()
 )
) ;; end of equations
```

G CG-Eingabe-File: Beispiel Sättigungskennlinie

Einfaches Beispiel für ein Petri-Netz zur Darstellung gemischt kontinuierlich-ereignisdiskreter Modelle. Die nichtlineare Sättigungskennlinie zeigt Unstetigkeiten in der ersten Zeitableitung von x.

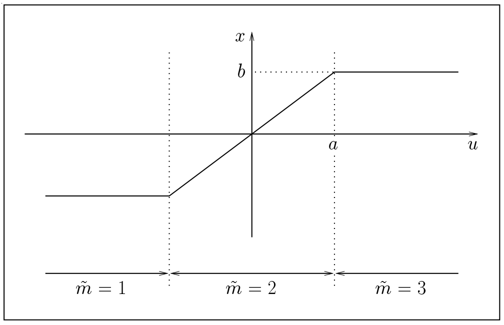


Abbildung 9: Sättigungskennlinie mit drei Kennlinienbereichen $\tilde{m} = 1, 2, 3$

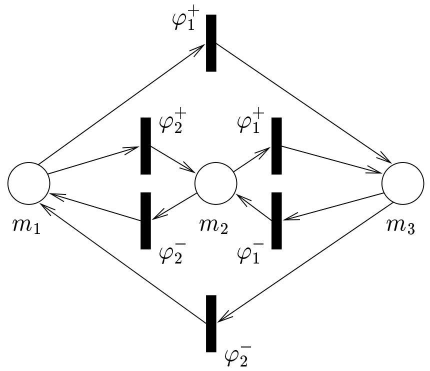


Abbildung 10: Petri-Netz der Sättigungskennlinie. Die Stellen repräsentieren die drei binären Komponenten des Schaltparametervektors $\boldsymbol{m}$ und entsprechen jeweils einem Bereich der Sättigungskennlinie. Die Anfangsmarkierung hängt vom Anfangswert des Kennlinieneingangs $u(t_0)$ ab und ist deshalb hier nicht mit abgebildet.

Die drei Modellgleichungen werden in einer algebraischen Gleichung zusammengefasst. Dabei wird die rechte Seite als unstetige Funktion in Abhängigkeit der Schaltparameter $\boldsymbol{m}$ formuliert:

$$0 = \begin{cases} x + b & : m_1 = 1 \\ ax - bu & : m_2 = 1 \\ x - b & : m_3 = 1 \end{cases} \tag{60}$$

Es wird vorausgesetzt, dass im Petri-Netz des Sättigungskennlinie jeweils immer nur eine Stelle markiert ist. Die Ereigniszeitpunkte werden durch folgende φ-Funktionen definiert:

$$\varphi_1 = u - a \tag{61}$$

$$\varphi_2 = u + a \tag{62}$$

Für jeden der drei Kennlinienbereiche $u < -a$, $u \in [-a, a]$ und $u > a$ wird jeweils eine Stelle definiert. Die Sättigungskennlinie wird also in ein äquivalentes Petri-Netz umgewandelt, welches in der nachfolgenden Abbildung dargestellt ist.

Die Anfangsmarkierung des Petri-Netzes ergibt sich aus

$$\begin{array}{lll} m_1 = 1 & \text{falls} & (\varphi_1(t_0) < 0 \text{ und } \varphi_2(t_0) < 0) \\ m_2 = 1 & \text{falls} & (\varphi_1(t_0) \leq 0 \text{ und } \varphi_2(t_0) \geq 0) \\ m_3 = 1 & \text{falls} & (\varphi_1(t_0) > 0 \text{ und } \varphi_2(t_0) > 0) \end{array} \tag{63}$$

wobei in allen drei Fällen die jeweils anderen Stellen den Wert 0 erhalten.

Die entsprechende Eingabedatei sieht folgendermaßen aus:

```
;;; Input-File for the Code-Generator

;;; 0. Documentation for head of FORTRAN programs
;;; ---------------------------------------------------------------------------
(define-header
  :program "saett"
  :documentation "Saettigungskennlinie"
  :date "26.06.00"
  :filename "saett.cg")

;;; 1. Parameters:
;;; ---------------------------------------------------------------------------

;;; 1.1 Structure Parameters:
;;; - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
;;  not needed

;;; 1.2 Integer Parameters (FORTRAN vector IPAR):
;;; - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
;;  not needed

;;; 1.3 Real Parameters (FORTRAN vector RPAR):
;;; - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
(define-real-parameters
  (:name "a"
   :meaning "Eintritt der Saettigung"
   :default-value 10.0)
  (:name "b"
   :meaning "maximaler Betrag fuer x"
   :default-value 10.0)
)


;;; 2. Model Inputs (FORTRAN vector U):
;;; ---------------------------------------------------------------------------
(define-input-variables
  (:name "u"
   :meaning "Einganggroesse u der Saettigungskennlinie" )
)

;;; 3. Model States (FORTRAN vector X):
;;; ---------------------------------------------------------------------------
(define-state-variables
  (:name "x"
   :meaning "Zustand der Saettigungskennlinie, ist auch Ausgangsgroesse"
   :default-state-value 0.0)
  (:name "dummy"
   :meaning "Dynamischer Zustand, wird nicht weiter verwendet"
   :default-state-value 10.0)
)

;;; 4. Model Outputs (FORTRAN vector IOUT):
;;; ---------------------------------------------------------------------------
(define-output-variables
  (:name "x_out"
```

```
   :meaning "Ausgangsgroesse der Saettigungskennlinie"
   :is-state-variable-with-name "x")
)

;;; 5. Help Variables:
;;; ----------------------------------------------------------------------------
;; not needed

;;; 6. Events:
;;; ----------------------------------------------------------------------------
(define-event-schedule
  (:petrinet-name "Saettigungsautomat"
   :meaning "Petri-Netz des Zustandsautomaten der Saettigungskennlinie"
   :places
   ((:place-name "m_1"  :meaning "linker Bereich der Kennlinie")
    (:place-name "m_2"  :meaning "mittlerer Bereich der Kennlinie")
    (:place-name "m_3"  :meaning "rechter Bereich der Kennlinie")
   )
   ;;; -----------------------------------------------------------
   ;;; der FORTRAN-Code dieses :cond muss noch von Hand codiert
   ;;; werden, siehe Beschreibung :initial-marks
   :initial-marks (":cond((phi1minus<0)  :and (phi2minus<0)  , m_1 ,
                          (phi1minus<=0) :and (phi2minus>=0) , m_2 ,
                         m_3)")
   ;;; -----------------------------------------------------------
   :event-fired-transitions
   ((:implicit-event-name "phi1plus"
     :meaning "u ueberschreitet obere Grenze a"
     :phi-value "-(u - a)"
     :type "unidirectional"
     :default-numerical-error-value 0.01
     :transitions
     ((:from ("m_1")    :to ("m_3"))
      (:from ("m_2")    :to ("m_3"))))
    (:implicit-event-name "phi2minus"
     :meaning "u unterschreitet untere Grenze -a"
     :phi-value "u + a"
     :type "unidirectional"
     :default-numerical-error-value 0.01
     :transitions
     ((:from ("m_2")    :to ("m_1"))
      (:from ("m_3")    :to ("m_1"))))
    (:implicit-event-name "phi1minus"
     :meaning "u unterschreitet obere Grenze a"
     :phi-value "u - a"
     :type "unidirectional"
     :default-numerical-error-value 0.01
     :transitions
     ((:from ("m_3")    :to ("m_2"))))
    (:implicit-event-name "phi2plus"
     :meaning "u ueberschreitet untere Grenze -a"
     :phi-value "-(u + a)"
     :type "unidirectional"
     :default-numerical-error-value 0.01
     :transitions
     ((:from ("m_1")    :to ("m_2"))))
```

```
   )
  )
)

;;; 7. Process Values (grouping of data output):
;;; ---------------------------------------------------------------------------
(define-process-values
  (:name "X"
   :meaning "Systemzustand"
   :components ( "x" ))
  (:name "Stellen"
   :meaning "Stellen des Petri-Netzes Saettigungsauomat"
   :components ( "m_1" "m_2" "m_3" ))
)

;;; 8. Model Equations:
;;; ---------------------------------------------------------------------------
(define-equations
  (:equation-name "Kennlinien-Gleichung"
   :meaning "algebraische Gleichung der Saettigungskennlinie"
   :f-value ":cond(m_1 , x + b , m_2 , a*x - b*u , x - b)")
  (:equation-name "dummy-Gleichung"
   :meaning "Dgl. fuer den dynamischen dummy-Zustand"
   :f-value "-dummy"
   :b-values ((:state-variable-reference "dummy"
               :b-value 1.0)))
)

;;; end of file
```

H Code-Generator Eingabe-File: Beispiel Ablaufsteuerung eines Batch-Prozesses

In diesem Beispiel, das [42] entnommen ist, geht es um die Ablaufsteuerung eines zyklisch betriebenen Reaktionsprozesses. Die Anlage ist in Abbildung 11 schematisch dargestellt und besteht aus zwei Vorlagebehältern VA und VB sowie einem Batchreaktor R. Aus Platzgründen wird hier auf eine Darstellung der

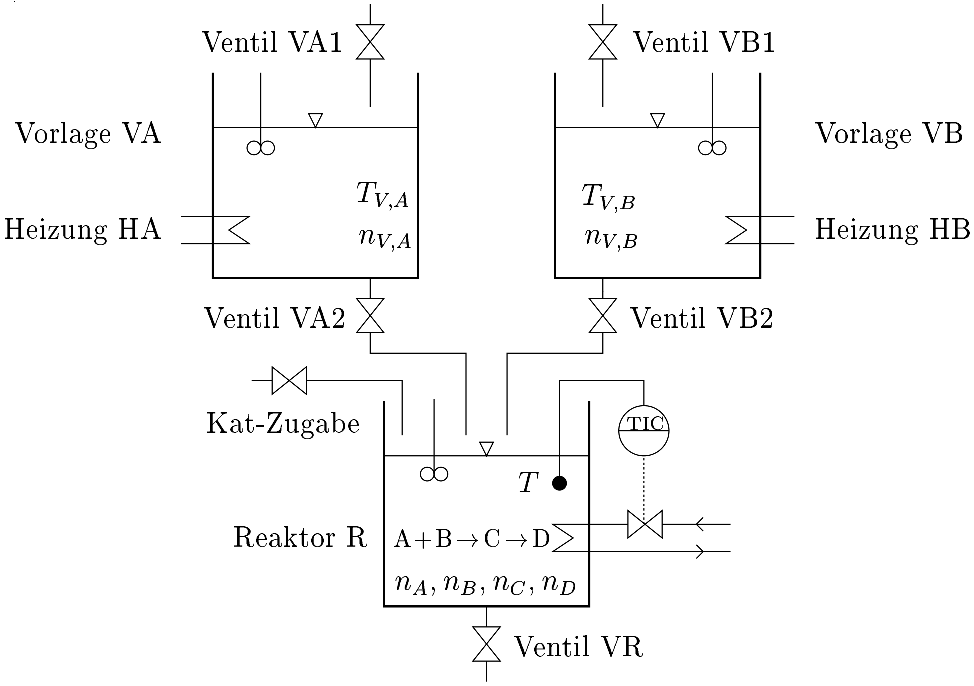


Abbildung 11: Anlagenschema des zyklisch betriebenen Batch-Prozesses mit den beiden Vorlagebehältern VA und VB und dem Batchreaktor R.

Modellgleichungen und der φ-Funktionen für dieses Beispiel verzichtet. Das Beispiel zeigt eine relativ komplexe Schaltwerk-Struktur – insbesondere mit nebenläufigen Teilprozessen, die zu einem späteren Zeitpunkt synchronisiert werden.

Abbildung 12 zeigt die Ablaufsteuerung des Prozesses als Petri-Netz mit der Anfangsmarkierung. Die zu den Transitionen gehörigen Ereignisse sind im Petri-Netz mit Bezeichnern versehen. Die Bedeutungen der einzelnen Ereignisse sowie ihre Schaltzeitpunkte bei der durchgeführten Simulation mit DIVA sind in Tabelle 8 aufgelistet. Die drei unbenannten Transitionen des Petri-Netzes sind keinen Ereignissen zugeordnet und lösen bei einer Aktivierung direkt aus.

Zu Beginn eines Zyklus ist der Reaktor leer, und die Vorlagen sind mit je einem der beiden Reaktanden A und B der Temperatur T_{hot} gefüllt. In diesem Zustand wird direkt weitergeschaltet, so dass der Inhalt der beiden Vorlagen über die Ventile VA2 und VB2 in den Reaktor abgelassen wird. Sobald die beiden Vorlagebehälter VA und VB leer sind (Ereignisse φ_1 und φ_2), laufen zwei Prozesse parallel ab: Die Steuerung zur Vorbereitung der Vorlagen VA und VB für den nächsten Zyklus und die Steuerung der Reaktion im Batchreaktor.

Die beiden Vorlagebehälter VA und VB werden über die Ventile VA1 und VB1 kalt befüllt und, sobald beide wieder voll sind (Ereignisse φ_3 und φ_4), so lange aufgeheizt, bis sie die Temperatur T_{hot} erreicht haben (Ereignisse φ_5 und φ_6). Anschließend bleibt der Zustand der Vorlagen so lange erhalten, bis der nebenläufige Reaktions-Prozess im Batch-Reaktor abgelaufen ist.

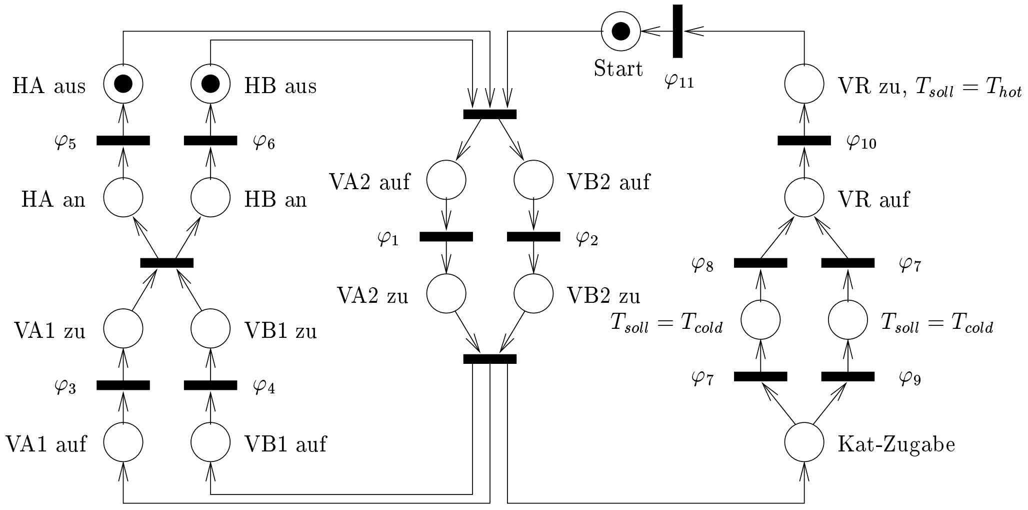


Abbildung 12: Petri-Netz der Ablaufsteuerung des zyklisch betriebenen Batch-Prozesses mit der Anfangsmarkierung. Den Transitionen sind die Namen der Ereignisse zugeordnet, mit denen sie bei einer Aktivierung schalten. Transitionen ohne Ereignisbezeichner lösen direkt aus.

φ	$t_{ev}[s]$	Ereignis	φ	$t_{ev}[s]$	Ereignis
φ_1	100	Vorlage VA leer	φ_7	1600	Reaktionszeit Δt abgelaufen
φ_2	100	Vorlage VB leer	φ_8	1637	Reaktor kalt
φ_3	2098	Vorlage VA voll	φ_9	–	Reaktor zu heiß
φ_4	2098	Vorlage VB voll	φ_{10}	1690	Reaktor leer
φ_5	2274	Vorlage VA heiß	φ_{11}	2019	Reaktor heiß
φ_6	2274	Vorlage VB heiß			

Tabelle 8: Ereignisse in der Ablaufsteuerung des zyklisch betriebenen Batch-Prozesses aus Abbildung 12 mit den Ereigniszeiten für $\Delta t = 1500s$ und $T_{hot} = 353K$.

Dem Reaktionsgemisch, das im Batch-Reaktor durch eine Regelung auf $T_{soll} = T_{hot}$ temperiert wird, wird Katalysator zugegeben, worauf die Edukte A und B gemäß einer Kinetik zweiter Ordnung in das Zielprodukt C umgesetzt werden. In einer Folgereaktion erster Ordnung kann sich aus C das unerwünschte Nebenprodukt D bilden. Für den folgenden Verlauf ergeben sich zwei Möglichkeiten:

1. Die Reaktortemperatur bleibt während der Reaktion im zulässigen Bereich. Dann wird der Reaktorinhalt nach Ablauf der Reaktionszeit Δt (Ereignis φ_7) abgekühlt. Hierfür setzt man den Sollwert für die Temperaturregelung auf $T_{soll} = T_{cold}$. Hat die Reaktortemperatur den neuen Sollwert erreicht (Ereignis φ_8), so wird der Reaktorinhalt über das Ventil VR abgelassen.
2. Die Reaktortemperatur erreicht noch vor Ablauf der Reaktionszeit Δt einen maximal zulässigen Wert T_{max} (Ereignis φ_9). In diesem Fall wird der Reaktorinhalt sofort abgekühlt ($T_{soll} = T_{cold}$) und ebenfalls nach Ablauf der Reaktionszeit (Ereignis φ_7) abgelassen.

Ist der Reaktor schließlich leer (Ereignis φ_{10}), so wird das Ablaufventil VR wieder geschlossen und der Sollwert für die Temperaturregelung auf $T_{soll} = T_{hot}$ erhöht. Nach Erreichen von T_{hot} (Ereignis φ_{11}) ist der Reaktionszyklus beendet.

Über eine direkt schaltende Transition werden die beiden nebenläufigen Prozesse zur Vorbereitung der Vorlagen bzw. zur Durchführung der Reaktion synchronisiert. Sobald beide Prozesse abgeschlossen sind, wird der nächste Zyklus gestartet.

```
;;; Input-File for the Code-Generator

;;; 0. Documentation for head of FORTRAN programs
;;; ---------------------------------------------------------------------------
(define-header
  :program "steuerung"
  :documentation "Steuerung eines zyklischen Batch-Prozesses"
  :date "07.07.00"
  :filename "steue.cg")

;;; 1. Parameters:
;;; ---------------------------------------------------------------------------

;;; 1.1 Structure Parameters:
;;; - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
;;  not needed

;;; 1.2 Integer Parameters (FORTRAN vector IPAR):
;;; - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
;;  not needed

;;; 1.3 Real Parameters (FORTRAN vector RPAR):
;;; - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
(define-real-parameters
  ;; ----------------------------------
  ;;  Stoffdaten:
  ;; ----------------------------------
  (:name "cpA"      :meaning "[J/mol/K]"         :default-value 160.0)
  (:name "cpB"      :meaning "[J/mol/K]"         :default-value 160.0)
  (:name "cpC"      :meaning "[J/mol/K]"         :default-value 160.0)
  (:name "cpD"      :meaning "[J/mol/K]"         :default-value 160.0)
  (:name "vA"       :meaning "[l/mol]"           :default-value   0.1)
  (:name "vB"       :meaning "[l/mol]"           :default-value   0.1)
  (:name "vC"       :meaning "[l/mol]"           :default-value   0.2)
  (:name "vD"       :meaning "[l/mol]"           :default-value   0.2)
  ;; ----------------------------------
  ;;  Reaktion:
  ;; ----------------------------------
  (:name "k1"       :meaning "[l/mol/s]"         :default-value 1.0E12)
  (:name "k2"       :meaning "[1/s]"             :default-value 1.0E16)
  (:name "E1"       :meaning "[J/mol]"           :default-value 1.0E5)
  (:name "E2"       :meaning "[J/mol]"           :default-value 1.3E5)
  (:name "R"        :meaning "[J/mol/K]"         :default-value 8.314)
  (:name "dhr1"     :meaning "[J/mol]"           :default-value 1.5E4)
  (:name "dhr2"     :meaning "[J/mol]"           :default-value 1.0E4)
  ;; ----------------------------------
  ;;  Stoffstroeme:
  ;; ----------------------------------
  (:name "J_Azu"    :meaning "Zulauf Vorlage A"  :default-value 0.0)
  (:name "J_Bzu"    :meaning "Zulauf Vorlage B"  :default-value 0.0)
  (:name "J_A"      :meaning "Ablauf Vorlage A"  :default-value 0.0)
  (:name "J_B"      :meaning "Ablauf Vorlage B"  :default-value 0.0)
  (:name "J_Rab"    :meaning "Ablauf Reaktor"    :default-value 0.0)
  ;; ----------------------------------
  ;;  Kuehlung:
  ;; ----------------------------------
```

```
  (:name "UA"       :meaning "Waermeuebertragung"
                    :default-value 100.0)
  (:name "T_hot"    :meaning "heisse Solltemperatur"
                    :default-value 353.0)
  (:name "T_cold"   :meaning "kalte Solltemperatur"
                    :default-value 298.0)
  (:name "T_soll"   :meaning "Solltemperatur im Reaktor"
                    :default-value "T_hot")
  (:name "TA_soll"  :meaning "Solltemperatur in Vorlage VB"
                    :default-value 353.0)
  (:name "TB_soll"  :meaning "Solltemperatur in Vorlage VA"
                    :default-value 353.0)
  (:name "TBoffset" :meaning "Offset bei Temperaturmessung in Vorlage B"
                    :default-value  10.0)
  (:name "KP"       :meaning "Eingangskonstante im Temperaturregler"
                    :default-value   0.7)
  (:name "J_QAmax"  :meaning "Waermestrom bei Heizung in Vorlage A"
                    :default-value 500.0)
  (:name "J_QBmax"  :meaning "Waermestrom bei Heizung in Vorlage B"
                    :default-value 590.909091)
  (:name "TCSPEED"  :meaning "Reglerdynamik"
                    :default-value 1.0E1 )
  ;; ---------------------------------
  ;;  Schwellwerte:
  ;; ---------------------------------
  (:name "nA_rest"  :meaning "Rest an nA"
                    :default-value   9.999)
  (:name "nB_rest"  :meaning "Rest an nB"
                    :default-value   9.999)
  (:name "V_rest"   :meaning "Restvolumen"
                    :default-value   0.02)
  (:name "T_rest"   :meaning "erlaubte Temperaturabweichung"
                    :default-value   1.0)
  (:name "schwelli" :meaning "fuer Berechnung der Reaktionskonstanten"
                    :default-value 400.0)
  (:name "dT_max"   :meaning "maximale Differenz zwischen T und T_soll"
                    :default-value  10.0)
)

;;; 2. Model Inputs (FORTRAN vector U):
;;; -------------------------------------------------------------------------
(define-input-variables
  (:name "u"        :meaning "Einganggroesse u, hier unnoetig")
)

;;; 3. Model States (FORTRAN vector X):
;;; -------------------------------------------------------------------------
(define-state-variables
  (:name "nA_v"     :meaning "Molmenge in Vorlage A"
                    :default-state-value  9.9990005)
  (:name "nB_v"     :meaning "Molmenge in Vorlage B"
                    :default-state-value  9.9990005)
  (:name "TA"       :meaning "Temperatur in Vorlage A"
                    :default-state-value  "T_hot+0.000001")
  (:name "TB"       :meaning "Temperatur in Vorlage B"
                    :default-state-value  "T_hot+0.000001")
```

```
  (:name "nA"       :meaning "Molmenge A im Reaktor"
                    :default-state-value  1.0E-5)
  (:name "nB"       :meaning "Molmenge B im Reaktor"
                    :default-state-value  1.0E-5)
  (:name "nC"       :meaning "Molmenge C im Reaktor"
                    :default-state-value  0.0)
  (:name "nD"       :meaning "Molmenge D im Reaktor"
                    :default-state-value  0.0)
  (:name "T"        :meaning "Temperatur im Reaktor"
                    :default-state-value  "T_hot")
  (:name "Tc"       :meaning "Stellgroesse der Temperaturregelung"
                    :default-state-value  "T_hot")
  (:name "Tc_help"  :meaning "Hilfsgroesse fuer Temperaturregelung"
                    :default-state-value  0.0)
  (:name "V"        :meaning "Volumen im Reaktor"
                    :default-state-value  2.0E-6)
)

;;; 4. Model Outputs (FORTRAN vector IOUT):
;;; ----------------------------------------------------------------------------
(define-output-variables
  (:name "nA_v_out" :meaning "nA_v"
                    :is-state-variable-with-name "nA_v")
  (:name "nB_v_out" :meaning "nB_v"
                    :is-state-variable-with-name "nB_v")
  (:name "TA_out"   :meaning "TA"
                    :is-state-variable-with-name "TA")
  (:name "TB_out"   :meaning "TB"
                    :is-state-variable-with-name "TB")
  (:name "nA_out"   :meaning "nA"
                    :is-state-variable-with-name "nA")
  (:name "nB_out"   :meaning "nB"
                    :is-state-variable-with-name "nB")
  (:name "nC_out"   :meaning "nC"
                    :is-state-variable-with-name "nC")
  (:name "nD_out"   :meaning "nD"
                    :is-state-variable-with-name "nD")
  (:name "T_out"    :meaning "T"
                    :is-state-variable-with-name "T")
  (:name "Tc_out"   :meaning "Tc"
                    :is-state-variable-with-name "Tc")
  (:name "V_out"    :meaning "V"
                    :is-state-variable-with-name "V")
)

;;; 5. Help Variables:
;;; ----------------------------------------------------------------------------
(define-help-variables
  (:name "nopro1"   :meaning "nopro1")
  (:name "nopro2"   :meaning "nopro2")
  (:name "r01V"     :meaning "RG1 * V [mol/s]")
  (:name "r02V"     :meaning "RG2 * V [mol/l/s]")
)

;;; 6. Events:
;;; ----------------------------------------------------------------------------
```

```
(define-event-schedule
  (:petrinet-name "Prozess-Steuerung"
   :meaning "Steuerung des zyklischen Batch-Prozesses"
   :places
   ;; ---------------------------------
   ;;  Stellen der Prozess-Steuerung:
   ;; ---------------------------------
   ((:place-name "Start"
     :meaning "Beginn eines Zyklus")
    (:place-name "VA1_auf"
     :meaning "Vorlage VA ablassen"
     :set-characteristic-values
     ((:real-parameter-name "J_Azu"      :value 0.005)))
    (:place-name "VA1_zu"
     :meaning "Ventil VA1 schliessen"
     :set-characteristic-values
     ((:real-parameter-name "J_Azu"      :value 0.0)))
    (:place-name "VB1_auf"
     :meaning "Vorlage VB ablassen"
     :set-characteristic-values
     ((:real-parameter-name "J_Bzu"      :value 0.005)))
    (:place-name "VB1_zu"
     :meaning "Ventil VB1 schliessen"
     :set-characteristic-values
     ((:real-parameter-name "J_Bzu"      :value 0.0)))
    (:place-name "VA2_auf"
     :meaning "Vorlage VA fuellen"
     :set-characteristic-values
     ((:real-parameter-name "J_A"        :value 0.1)))
    (:place-name "VA2_zu"
     :meaning "Ventil VA2 schliessen"
     :set-characteristic-values
     ((:real-parameter-name "J_A"        :value 0.0)))
    (:place-name "VB2_auf"
     :meaning "Vorlage VB fuellen"
     :set-characteristic-values
     ((:real-parameter-name "J_B"        :value 0.1)))
    (:place-name "VB2_zu"
     :meaning "Ventil VB2 schliessen"
     :set-characteristic-values
     ((:real-parameter-name "J_B"        :value 0.0)))
    (:place-name "HA_an"
     :meaning "Heizung HA anschalten")
    (:place-name "HA_aus"
     :meaning "Heizung HA ausschalten")
    (:place-name "HB_an"
     :meaning "Heizung HB anschalten")
    (:place-name "HB_aus"
     :meaning "Heizung HB ausschalten")
    (:place-name "Kat_Zugabe"
     :meaning "Kat-Zugabe anschalten")
    (:place-name "Reaktionsende"
     :meaning "Reaktion normal beenden: Reaktor abkuehlen"
     :set-characteristic-values
     ((:real-parameter-name "T_soll"     :value "T_cold")))
    (:place-name "Abkuehlen"
```

```
  :meaning "Reaktionsgemisch vor Reaktionsende abkuehlen"
  :set-characteristic-values
  ((:real-parameter-name "T_soll"     :value "T_cold")))
 (:place-name "VR_auf"
  :meaning "Reaktionsgemisch ablassen"
  :set-characteristic-values
  ((:real-parameter-name "J_Rab"      :value 0.2)))
 (:place-name "VR_zu"
  :meaning "Ventil VR schliessen, Reaktor wieder aufheizen"
  :set-characteristic-values
  ((:real-parameter-name "J_Rab"      :value 0.0)
   (:real-parameter-name "T_soll"     :value "T_hot")))
)
;; ---------------------------------
;;  Anfangsmarkierung:
;; ---------------------------------
:initial-marks ("Start" "HA_aus" "HB_aus")
:event-fired-transitions
;; ---------------------------------
;;  Ereignisse mit Transitionen:
;; ---------------------------------
((:implicit-event-name "phi1"
  :meaning "Behaelter VA leer"
  :phi-value "nA_v - 0.01"
  :type "unidirectional"
  :default-numerical-error-value 0.0001
  :transitions
  ((:from ("VA2_auf")                 :to ("VA2_zu")))
 )
 (:implicit-event-name "phi2"
  :meaning "Behaelter VB leer"
  :phi-value "nB_v - 0.01"
  :type "unidirectional"
  :default-numerical-error-value 0.0001
  :transitions
  ((:from ("VB2_auf")                 :to ("VB2_zu")))
 )
 (:implicit-event-name "phi3"
  :meaning "Behaelter VA voll"
  :phi-value "nA_rest - nA_v"
  :type "unidirectional"
  :default-numerical-error-value 0.0001
  :transitions
  ((:from ("VA1_auf")                 :to ("VA1_zu")))
 )
 (:implicit-event-name "phi4"
  :meaning "Behaelter VB voll"
  :phi-value "nB_rest - nB_v"
  :type "unidirectional"
  :default-numerical-error-value 0.0001
  :transitions
  ((:from ("VB1_auf")                 :to ("VB1_zu")))
 )
 (:implicit-event-name "phi5"
  :meaning "Solltemperatur in Vorlage VA erreicht"
  :phi-value "TA_soll - TA"
```

```
    :type "unidirectional"
    :default-numerical-error-value 0.01
    :transitions
    ((:from ("HA_an")                  :to ("HA_aus")))
   )
   (:implicit-event-name "phi6"
    :meaning "Solltemperatur in Vorlage VB erreicht"
    :phi-value "TB_soll - TB"
    :type "unidirectional"
    :default-numerical-error-value 0.01
    :transitions
    ((:from ("HB_an")                  :to ("HB_aus")))
   )
   (:explicit-event-name "phi7"
    :meaning "Reaktionszeit delta_T nach Kat-Zugabe abgelaufen"
    :default-delay-time 1500.0
    :trigger-place "Kat_Zugabe"
    :transitions
    ((:from ("Kat_Zugabe")             :to ("Reaktionsende"))
     (:from ("Abkuehlen")              :to ("VR_auf")))
   )
   (:implicit-event-name "phi8"
    :meaning "Reaktionsgemisch abgekuehlt"
    :phi-value "T - T_rest - T_cold"
    :type "unidirectional"
    :default-numerical-error-value 0.01
    :transitions
    ((:from ("Reaktionsende")          :to ("VR_auf")))
   )
   (:implicit-event-name "phi9"
    :meaning "Reaktionsgemisch zu heiss"
    :phi-value "T_hot + dT_max - T"
    :type "unidirectional"
    :default-numerical-error-value 0.01
    :transitions
    ((:from ("Kat_Zugabe")             :to ("Abkuehlen")))
   )
   (:implicit-event-name "phi10"
    :meaning "Reaktor leer"
    :phi-value "V - V_rest"
    :type "unidirectional"
    :default-numerical-error-value 0.0001
    :transitions
    ((:from ("VR_auf")                 :to ("VR_zu")))
   )
   (:implicit-event-name "phi11"
    :meaning "Reaktor wieder aufgeheizt"
    :phi-value "T_hot - T"
    :type "unidirectional"
    :default-numerical-error-value 0.01
    :transitions
    ((:from ("VR_zu")                  :to ("Start")))
   )
  )
  :directly-fired-transitions
  ;; ---------------------------------
```

```
   ;;  direkt ausloesende Transitionen:
   ;; ---------------------------------
   ((:from ("Start" "HA_aus" "HB_aus")   :to ("VA2_auf" "VB2_auf"))
    (:from ("VA2_zu" "VB2_zu")           :to ("VA1_auf" "VB1_auf" "Kat_Zugabe"))
    (:from ("VA1_zu" "VB1_zu")           :to ("HA_an" "HB_an"))
   )
  )
) ; end of event-schedule

;;; 7. Process Values (grouping of data output):
;;; ------------------------------------------------------------------------
(define-process-values
  (:name "STATES"
   :meaning "alle Zustaendsgroessen"
   :components ("r01V"
                "r02V"
                "nA_v"
                "nB_v"
                "TA"
                "TB"
                "nA"
                "nB"
                "nC"
                "nD"
                "T"
                "Tc"
                "V")
  )
)

;;; 8. Model Equations:
;;; ------------------------------------------------------------------------
(define-equations
  ;; ---------------------------------
  ;;  Hilfsgroessenzuweisungen:
  ;; ---------------------------------
  (:assignment-name "nopro1"
   :meaning "Rekationsterm"
   :help-variable "nopro1"
   :h-value ":cond( -E1/R/T > schwelli,
                schwelli + 100.0 * :tanh(( (-E1/R/T) - schwelli)/100.0),
                    -E1/R/T)"
  )
  (:assignment-name "nopro2"
   :meaning "Rekationsterm"
   :help-variable "nopro2"
   :h-value ":cond( -E2/R/T > schwelli,
                 schwelli + 100.0 * :tanh(( (-E2/R/T) - schwelli)/100.0),
                    -E2/R/T)"
  )
  (:assignment-name "RG1 * V [mol/s]"
   :meaning "Reaktionsgeschwindigkeit der ersten Reaktion"
   :help-variable "r01V"
   :h-value ":cond(Kat_Zugabe :or Abkuehlen, k1*:exp(nopro1)*nA*nB/V, 0)"
  )
  (:assignment-name "RG2 * V [mol/s]"
```

```
 :meaning "Reaktionsgeschwindigkeit der zweiten Reaktion"
 :help-variable "r02V"
 :h-value ":cond(Kat_Zugabe :or Abkuehlen, k2*:exp(nopro2)*nC, 0)"
)
;; ---------------------------------
;;  Modellgleichungen:
;; ---------------------------------
(:equation-name "nA_v"
 :meaning "Stoffbilanz Vorlage A"
 :b-values ((:state-variable-reference "nA_v" :b-value 1.0))
 :f-value "J_Azu - J_A"
)
(:equation-name "nB_v"
 :meaning "Stoffbilanz Vorlage B"
 :b-values ((:state-variable-reference "nB_v" :b-value 1.0))
 :f-value "J_Bzu - J_B"
)
(:equation-name "TA"
 :meaning "Temperaturbilanz Vorlage A"
 :b-values ((:state-variable-reference "TA" :b-value "(nA_v*cpA)"))
 :f-value "(:cond(HA_an, J_QAmax, 0) + J_Azu*cpA*(T_cold-TA))"
)
(:equation-name "TB"
 :meaning "Temperaturbilanz Vorlage B"
 :b-values ((:state-variable-reference "TB" :b-value "(nB_v*cpB)"))
 :f-value "(:cond(HB_an, J_QBmax, 0) + J_Bzu*cpB*(T_cold-TBoffset-TB))"
)
(:equation-name "nA"
 :meaning "Stoffbilanz Stoff A im Reaktor"
 :b-values ((:state-variable-reference "nA" :b-value 1.0))
 :f-value "J_A - r01V - J_Rab*nA/(nA+nB+nC+nD)"
)
(:equation-name "nB"
 :meaning "Stoffbilanz Stoff B im Reaktor"
 :b-values ((:state-variable-reference "nB" :b-value 1.0))
 :f-value "J_B - r01V - J_Rab*nB/(nA+nB+nC+nD)"
)
(:equation-name "nC"
 :meaning "Stoffbilanz Stoff C im Reaktor"
 :b-values ((:state-variable-reference "nC" :b-value 1.0))
 :f-value "r01V - r02V - J_Rab*nC/(nA+nB+nC+nD)"
)
(:equation-name "nD"
 :meaning "Stoffbilanz Stoff D im Reaktor"
 :b-values ((:state-variable-reference "nD" :b-value 1.0))
 :f-value "r02V - J_Rab*nD/(nA+nB+nC+nD)"
)
(:equation-name "T"
 :meaning "Temperaturbilanz im Reaktor"
 :b-values ((:state-variable-reference "T"
             :b-value "(nA*cpA + nB*cpB + nC*cpC + nD*cpD)"))
 :f-value "(dhr1*r01V + dhr2*r02V +
            UA*(Tc-T) +
            J_A*cpA*(TA-T) + J_B*cpB*(TB-T) )"
)
(:equation-name "Tc_help"
```

```
   :meaning "Temperaturregelung (1)"
   :b-values ((:state-variable-reference "Tc_help" :b-value "1.0/TCSPEED"))
   :f-value "-Tc_help + KP*(T_soll-T)"
  )
  (:equation-name "Tc"
   :meaning "Temperaturregelung (2)"
   :b-values ((:state-variable-reference "Tc" :b-value "1.0"))
   :f-value "Tc_help + TCSPEED*(-Tc_help + KP*(T_soll-T))"
  )
  (:equation-name "Reaktorvolumen"
   :meaning "konstantes Volumen im Reaktor"
   :f-value "V - (nA*vA + nB*vB + nC*vC + nD*vD)"
  )
) ; end of equations

;;; end of file
```

Literatur

[1] Marquardt, W.; Holl, P.; Gilles, E. D.: Diva - A dynamic process flowsheet simulator. In: III World Congress of Chemical Engineering, Tokyo 1986, 687–690.

[2] Marquardt, W.; Holl, P.; Butz, D.; Gilles, E. D.: DIVA - A flow-sheet oriented dynamic process simulator. Chemical Engineering Technology, 10, 1987, 164–173.

[3] Holl, P.; Marquardt, W.; Gilles, E. D.: DIVA - A powerful tool for dynamic process simulation. Computers chem. Engng., 12, 1988, 421–425.

[4] Kröner, A.; Holl, P.; Marquardt, W.; Gilles, E. D.: DIVA - An Open Architecture for Dynamic Simulation. Computers chem. Engng., 14, 1990, 1289–1295.

[5] Gilles, E. D.; Holl, P.; Marquardt, W.; Mahler, R.; Schneider, H.; Brinkmann, K.; Will, K. H.: Ein Trainingssimulator zur Ausbildung von Betriebspersonal in der chemischen Industrie. Automatisierungstechnische Praxis, 32, 1990, 343–350.

[6] Kurrle, H.-P.; Holl, P.; Munoz, G.: Struktur und Funktionsweise der Simulationsumgebung DIVA bei der Kopplung mit Prozeßleitsystemen. Automatisierungstechnische Praxis, 32, 1990, 565–569, 608–612.

[7] Hager, M.; Stephan, K.; Bär, M.; Zeitz, M.; Kröner, A.; Gilles, E. D.: Stoffdatenversorgung des Simulators DIVA für dynamische Prozesse und Anlagen — Parameterbeschaffung und Stoffwertberechnung. Chemie-Ingenieur-Technik, 63(3), 1991, 260–261.

[8] Bär, M.: Wissensbasierte und fließbildorientierte Bedienung eines verfahrenstechnischen Simulators, Fortschrittsberichte VDI Reihe 20 Nr. 10. VDI-Verlag, Düsseldorf, 1993.

[9] Bär, M.; Schaffner, J.; Selg, W.; Zeitz, M.: Functionality and implementation of a knowledge-based flowsheet-oriented user interface for the dynamic process simulator DIVA. SIMULATION, 61, 1993, 117–123.

[10] Holl, P.: Entwicklung und Einsatz eines dynamischen Simulators für verfahrenstechnische Prozesse, Fortschrittsberichte VDI Reihe 3 Nr. 359. VDI-Verlag, Düsseldorf, 1994.

[11] Helget, A.; Gilles, E. D.: Dynamische Prozeß– und Anlagensimulation. In: Schuler, H. (Hrsg.): Prozeßsimulation, DECHEMA, VCH, Weinheim 1995, 109–152.

[12] Scheuring, R.: Modellierung, Beobachtung und Steuerung ereignisorientierter verfahrenstechnischer Systeme, Fortschrittsberichte VDI Reihe 8 Nr. 475. VDI-Verlag, Düsseldorf, 1995.

[13] Mohl, K. D.; Spieker, A.; Köhler, R.; Gilles, E. D.; Zeitz, M.: DIVA - A simulation environment for chemical engineering applications. In: Informatics, Cybernetics and Computer Science (ICCS-97), Collected Volume of Scientific Papers, Donetsk State Technical University, Donetsk, Ukraine, 1997, 8–15.

[14] Mohl, K. D.; Spieker, A.; Stein, E.; Gilles, E. D.: DIVA - Eine Umgebung zur Simulation, Analyse und Optimierung verfahrenstechnischer Prozesse. In: Kuhn, A.; Wenzel, S. (Hrsg.): Simulationstechnik, 11. ASIM-Symposium in Dortmund, Vieweg Verlag, Braunschweig/Wiesbaden, 1997, 278–283.

[15] Rettinger, H.: Programmierung von DIVA-Grundelementen. Interner Bericht, Institut für Systemdynamik und Regelungstechnik, Universität Stuttgart, 1992.

[16] Köhler, R.; Räumschüssel, S.; Zeitz, M.: Code Generator for Implementing Differential Algebraic Models Used in the Process Simulation Tool DIVA. In: Sydow, A. (Hrsg.): 15th IMACS World Congress, volume 3, Wissenschaft und Technik Verlag, Berlin, Germany 1997, 621–626.

[17] Räumschüssel, S.; R., K.; Gerstlauer, A.; Zeitz, M.: Rechnerunterstützte Vorverarbeitung und Codierung eines Festbettreaktor-Modells für die Simulationsumgebung DIVA. In: Kuhn, A.; Wenzel, S. (Hrsg.): Simulationstechnik, 11. ASIM-Symposium in Dortmund, Vieweg Verlag, Braunschweig/Wiesbaden, 1997, 284–289.

[18] Räumschüssel, S.: Rechnerunterstützte Vorverarbeitung und Codierung verfahrenstechnischer Modelle für die Simulationsumgebung DIVA. VDI Fortschritt–Berichte Nr. 20/270, VDI-Verlag, Düsseldorf, 1998.

[19] Räumschüssel, S.; Gerstlauer, A.; Gilles, E. D.; Marquardt, W.; Raichle, B.; Zeitz, M.: An Architecture of a Knowledge-Based Process Modeling Simulation Tool. In: Proc. IMACS/IFAC Second International Symposium on Mathematical and Intelligent Models in System Simulation, Volume 2, Brussels/Belgium 1993, 242–247.

[20] Marquardt, W.: Rechnergestützte Erstellung verfahrenstechnischer Prozeßmodelle. Chemie-Ingenieur-Technik, 64, 1992, 25–40.

[21] Marquardt, W.: An Object-Oriented Representation of Structured Process Models. European Symposium on Computer Aided Process Engineering 1, ESCAPE-1 24-28 May 1992, Elsinore, Denmark. Computers chem. Engng. (Suppl.), 16, 1992, 329–336.

[22] Marquardt, W.; Gerstlauer, A.; Gilles, E. D.: Modeling and Representation of Complex Objects: A Chemical Engineering Perspective. In: 6th International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems, June 1-4, 1993, Edinburgh, Scotland 1993, 219–228.

[23] Marquardt, W.; Zeitz, M.: Rechnergestützte Modellbildung in der Verfahrenstechnik. In: Troch, I. (Hrsg.): Modellbildung für Regelung und Simulation. Methoden – Werkzeuge – Fallstudien. VDI/VDE-GMA-Aussprachetag, 25./26. März 1992 Langen. VDI-Berichte 925, VDI-Verlag, Düsseldorf, 1992, 307–341.

[24] Marquardt, W.: Dynamic Process Simulation – Recent Progress and Future Challenges. In: Ray, W. H.; Arkun, Y. (Hrsg.): Chemical Process Control – CPC IV, CACHE, South Padre Island, 1991, 131–180.

[25] Räumschüssel, S.; Gerstlauer, A.; Gilles, E. D.; Zeitz, M.: Ein Präprozessor für den verfahrenstechnischen Simulator DIVA. In: Kampe, G.; Zeitz, M. (Hrsg.): Simulationstechnik, 9. ASIM-Symposium in Stuttgart, Vieweg Verlag, Braunschweig/Wiesbaden, 1994, 177–182.

[26] Wolfram, S.: Mathematica: A System for Doing Mathematics by Computer. Addison-Wesley, 2. Auflage, 1991.

[27] Tränkle, F.: Rechnerunterstützte Modellierung verfahrenstechnischer Prozesse für die Simulationsumgebung DIVA. VDI Fortschritt–Berichte Nr. 20/309, VDI-Verlag, Düsseldorf, 2000.

[28] Tränkle, F.; Zeitz, M.; Ginkel, M.; Gilles, E.: A modeling tool for chemical processes. Mathematical and Computer Modeling of Dynamical Systems, 2000.

[29] Tränkle, F.: Modelldefinitionssprache MDL von PROMOT: Modellierungsanleitung. Interner Bericht 98-5, Institut für Systemdynamik und Regelungstechnik, Universität Stuttgart, Germany, 1998.

[30] Tränkle, F.; Gerstlauer, A.; Zeitz, M.; Gilles, E. D.: PROMOT/DIVA: A Prototype of a Process Modeling and Simulation Environment. In: Troch, I.; Breitenecker, F. (Hrsg.): IMACS Symposium on Mathematical Modelling, 2nd MATHMOD, ARGESIM Report, Vienna, Austria 1997, 341–346.

[31] Tränkle, F.; Gerstlauer, A.; Zeitz, M.; Gilles, E. D.: Application of the Modeling and Simulation Environment PROMOT/DIVA to the Modeling of Distillation Processes. volume 21, 1997, 841–846.

[32] Köhler, R.; Gerstlauer, A.; Zeitz, M.: Symbolic Preprocessing for Simulation of PDE Models of Chemical Processes. Journal of Math. and Comp. in Sim., special issue Method of Lines, 2000 (submitted).

[33] Köhler, R.: Manual SyPProT Version Beta.3 Symbolic PreProcessing Tool for the Simulation Environment DIVA. Interner Bericht 00-1, Institut für Systemdynamik und Regelungstechnik, Universität Stuttgart, 2000.

[34] Steele Jr., G. L.: Common Lisp: The Language. Digital Press, Bedford, Mass., 2. Auflage, 1990.

[35] Keene, S. E.: Object-oriented programming in Common Lisp : a programmer's guide to CLOS. Addison-Wesley, Reading [u.a.], 1989.

[36] Küpper, S.: Automatische Codierung gemischt-kontinuierlich-ereignisdiskreter Prozessmodelle für die Simulationsumgebung DIVA. Studienarbeit, Institut für Systemdynamik und Regelungstechnik, Universität Stuttgart, 2000.

[37] Opertopp, T.: Behandlung von Unstetigkeiten im Anlagensimulator DIVA am Beispiel der Mehr–Wege–Ventile. Studienarbeit, Institut für Systemdynamik und Regelungstechnik, Universität Stuttgart, 1993.

[38] Barton, P. I.: The modelling and simulation of combined discrete/continuous processes. Ph. D. Thesis, Department of Chemical Engineering, Imperial College, University of London. 1992.

[39] Cellier, F. E.: Combined Continuous/Discrete System Simulation by Use of Digital Computers: Techniques and Tools. Ph. D. Thesis, Swiss Federal Institute of Technology, ETH Zürich. 1979.

[40] Baumgarten, B.: Petri-Netze: Grundlagen und Anwendungen. Spektrum Akademischer Verlag, 2nd Auflage, 1996.

[41] Marquardt, W.; unter Mitarbeit von Gerstlauer, A.; Räumschüssel, S.; Raichle, B.: Objekt-orientierte Datenmodellierung für die Repräsentation von verfahrenstechnischen Prozeßmodellen. Interner Bericht, Institut für Systemdynamik und Regelungstechnik, Universität Stuttgart, 1993.

[42] Obertopp, T.; Spieker, A.; Gilles, E. D.: Plant-Wide Optimization of Parallel Batch Recipes. AIChE'98, Miami Beach, Florida/USA, November 15-20, 1998 (unveröffentlicht).

Index

:abs, 76
:all, 60
:and, 77
:arccos, 76
:arcosh, 76
:arcsin, 76
:arctan, 76
:arsinh, 76
:artanh, 76
:assignment-name, 53
:b-elem-scope, 50
:b-value, 50
:b-values, 49
:comp-scope, 25
:components, 24
:compute, 60
:cond, 47, 76
:cos, 76
:cosh, 76
:default-cg-name, 66
:default-dat-name, 65
:default-delay-time, 45
:default-latex-name, 66
:default-max-value, 16
:default-min-value, 16
:default-numerical-error-value, 16, 44
:default-output-pathname, 67
:default-output-subname, 65
:default-state-value, 16
:default-value, 14, 16, 19, 44
:default-values
    Array-Listen, 8
    voreingestellter Wert, 19
:diff, 77
:directly-fired-transitions, 42
:div, 72, 76, 77
:do-overwrite, 67
:end, 22, 23
:eq, 77
:eq-scope, 52
:equation-name, 49
:event-fired-transitions, 42, 44, 45
:event-initiated-status, 42, 44, 45
:exp, 76
:explicit-event-name, 44, 45
:f, 74
:f-value, 49
:false, 74
:from, 42
:ge, 77
:gt, 77
:h-accumulate, 58
:h-remember, 58
:h-value, 58
:help-var-accumulate, 52, 57
:help-var-remember, 52, 56
:help-variable, 54
:help-variable-name, 42
:implicit-event-name, 43
:index, 25, 50, 52
:index-names, 16, 17
:indices, 15
:initial-marks, 42
:input-file, 67
:int, 77
:integer-parameter-name, 41
:is-state-variable-with-name, 16, 17
:le, 77
:ln, 76
:local-library-name, 67
:log, 76
:lower-boundary, 15
:lt, 77
:marks, 44, 45
:max, 77
:meaning, 14
:min, 77
:mod, 72, 76, 77
:name, 14, 24
:neq, 77
:not, 77
:or, 77
:pattern-opt-level, 65
:petrinet-name, 40
:phi-value, 44
:place-name, 41
:places, 41
:procedure-name, 59
:prod, 46, 72, 77
:provided-pattern, 49, 50, 54, 60
:real-parameter-name, 42
:sequentially-in, 56
:set-characteristic-values, 41
:sin, 48, 76
:sinh, 76
:sqrt, 76
:start, 22, 23
:state-variable-reference, 49
:subroutine-names, 60
:sum, 46, 72, 77
:t, 74
:tan, 76
:tanh, 76
:to, 42
:transitions, 44, 45
:trigger-place, 45

:true, 74
:type, 43
:upper-boundary, 15
:value, 18, 41, 42
:values, 22, 23
:verbose-level, 65
:with-init, 56, 57
:write-as-infix, 66
:x-scope, 24, 25
:y-scope, 24, 26
:z-scope, 24

Alternativen, 32, 33
Anfangs- und Parameterwerte
  function-Ausdruck, 21
  spalten/zeilenweise Zuweisung mehrdimensionaler, 21
  Spline-Ausdruck, 22
Anfangs- und Parameterwerte
  lambda-Ausdruck, 19
Anfangs- und Parameterwerte, 19
Anfangswerte, 1, 3, 19
  konsistente, 31
Array-Liste, 8, 20, 21
Ausdruck
  arithmetischer, 3, 7, 20, 69, 71
    Beispiel, 71
  bedinungsabhängiger, 72
  konditionaler, 47, 72
  konstanter, 53
  logischer, 3, 71, 77
Ausgangsvariablen, 13, 16
  Änderung der Indizierungsdefinition, 9
  indizierte, 17

B-Matrix, 49
  einfache Einträge, 49
  indizierte Einträge, 49, 50
  Reihenfolge der Einträge, 51
  stetige Anteile, 28
  unstetige Anteile, 28
  Zuweisungen an Hilfsvariablen, 49
Backus-Naur-Form, 70, 71
Benutzungsoberfläche, 3, 66
Bereichsuberschreitungen in DIVA, 66

Charakteristische Werte, 39, 41
CLOS, 3
Code-Generator
  Aufruf, 3, 78
  Initialisierung, 65
  Starten, 3, 78
  Steuerbefehle, 65
  Version 1.0, 8
  Version 1.1, 8
  Version 1.2, 9
  Version 1.3, 9
  Version 1.3.1, 9
  Version 2.0, 9
  Version Beta.4, 8
  Version Beta.5, 8
  Version Beta.6, 8

DAE, 48
  linear-implizite Form, 1
  Standardform, 1
Datenfile, i, 3, 4, 14, 16, 24, 40, 65
Datenstruktur, 6
Definitionsfunktionen, 10, 13, 15
  überschreiben, 15
  Allgemeine Angaben (define-header), 12
  Ausgangs-Variablen (define-output-variables), 16, 17
  Character-Parameter (define-character-parameters), 14, 15
  Gleichungen (define-equations), 48
  Hilfs-Variablen (define-help-variables), 18
  Integer-Parameter (define-integer-parameters), 14, 15
  Petri-Netze (define-event-schedule), 10, 40
  Prozeßgrößen (define-process-values), 23
  Real-Parameter (define-real-parameters), 14, 15
  Struktur-Parameter (define-structure-parameters), 14
DIVA, i, 1–9, 13, 14, 23, 27–33, 37, 39–41, 43, 49–51, 54, 59, 60, 62, 65, 81, 100, 115
  Ereignisbehandlung, 30
  Eventverwaltung, 32
  Grundelement, i, 2–4, 8
    Datenfile, 3
    Erzeugung, 65
    Name, 65
Double-Format, 9

Eingangsvariablen, 13, 16
Ereignisdefinition, 27
Ereignisdiskrete Modelle, 28, 34–36
Ereignisse, siehe Events, 2
Ereigniszeitpunkt, 29–32, 43
Events, 9, 27
  Änderung der Definitionsfunktion, 9
  Beispiel, 45
  Erkennen von, 30
  explizite, 29, 31
    endogene, 30, 39, 44
    exogene, 30, 44
    indizierte, 44
    Verzogerungszeit, 39, 45
  Hilfsvariablenberechnung, 40
    EVLHV-Variable, 40
  implizite, 29, 30, 43
  Schalt- (φ-) Funktionen, 29, 44

Schalteingangs- (φ-) Funktionen, 29–32, 37, 38, 43, 96, 100
Schalteingangs- (φ-) Funktionen
bidirektional, 29, 43
unidirektional, 29, 43
Schaltfunktionen, 2
Schaltfunktionen S, 30, 40
Schaltschranke ϵ_φ, 30, 40, 44
Schaltvorgang ausfuhren, 30
Schaltzeitpunkt, 29–32, 43
bestimmen, 30
Stellen
charakteristische Werte, 39, 41
Triggern endogener expliziter Events, 39

f-Vektor, 49
stetige Anteile, 28
unstetige Anteile, 28
FORTRAN 77 Standard, 7
FORTRAN-Unterprogramme, 2–4, 6, 65
Änderung der Datei-Endung, 9
Berechnung analytischer Jacobi-Matrizen, 3
Datenausgabe, i, 3
Ereignisdefinition, i, 2, 40
externe, 58
Argumente, 60
Aufruf, 59
Bibliothek, 59
Bibliothek, globale, 63
Bibliothek, lokale, 63
Feld-Argumente, 60
Funktionsauswertung, i, 2
Initialisierung, i, 2, 40
Include-Datei, 7
Unstetigkeitsverwaltung, 9
function-Ausdruck, 21

Gleichungen
algebraische, 51
einfache (nicht-indizierten), 48
indizierte, 47, 48, 51
invariante, 53
Zuweisungen an Hilfsvariablen, 48, 53
Grammatikschreibweise, 70, 71

Hilfsvariablen, 13, 17, 53
accumulate-help-variable, 52, 55–58
assignments, 53
Berechnung mit externen FORTRAN-Unterprogrammen, 8, 58–64, 66
Berechung mit externen FORTRAN-Unterprogrammen, 58
help-accumulate, 55, 57
help-remember, 55
indizierte, 18, 54
Patternbestimmung, 18, 55
stetige, 28
teilweise Belegung, 18
unstetige, 28, 39
zeitinvariante, 18
zeitvariante, 18, 53
Zuweisungen an, 47, 53

Include-Datei
für Ereignis- und Schaltfunktionen, 12
Indizierung, 19
Laufgrenzen, 15
lokale Indexvariable, 20
Variablen und Parameter, 14
Infixnotation, 7
Integer-Division, 49

Jacobi-Matrizen, 2, 31
Pattern, 2, 3, 6, 8

keys, 13, 69
Kommentar, 13

lambda-Ausdruck, 19, 20
LISP, i, 3, 7, 10, 13, 20, 21, 59, 60, 65, 66, 69, 72
spezifische Ausdrücke, 69
Argumente, 69
Atome, 69
Funktion, 69
Leerzeichen, 69
Liste, 69
Präfix-Notation, 69
Schlüsselwort, 69
Liste, 13, 69

MATHEMATICA, 1
Modellbibliothek, i, 1–4
Modelldokumentation, 3
Modellierungswerkzeug, 1, 3
PROMOT, 1, 2
Modellspezifikation, 10
Prozeßgrößen (define-process-values), 10
Allgemeine Angaben (define-header), 10, 12
Events, 27
Gleichungen (define-equations), 10, 46
Hilfsvariablen (define-help-variables), 17
Integrationsvariable (define-time-variable), 10, 13
Parameter, 10, 13, 14
Prozeßgrößen (define-process-values), 23
Stutzen (define-interfaces), 8
Variablen, 10, 13, 16

Nebenlaufigkeiten, 32, 33
Synchronisieren von, 36

objektorientierte Programmierung, 3, 5
Klassen, 6
Mehrfachvererbung, 6

Optionen
- :pattern-opt-level, 8
- :pattern-opt-level Patternoptimierung, 65
- :verbose-level Bildschirmausgabe, 65

Parameter
- Character-, 7, 14
- Integer-, 14
- Real-, 14
- stetige, 28
- unstetige, 28, 39

Parameterwerte, 19
Parser, 7
Pattern
- konditionaler Ausdruck
  - Fehlerquelle, 66
- konditionaler Ausdruck, 47
- siehe Jacobi-Matrizen, 2

Patternbelegung, 49
Patternoptimierung, 8, 65
- :pattern-opt-level, 8
- :provided-pattern, 8

Petri-Netz, 9
Petri-Netze, 32
- Anfangsmarkierung, 34, 42
- Beispiel, 45
- Definition, 34, 40
- Flussrelation, 32, 34, 42
- Knoten, 34
- Marken, 32
- Schaltdynamik, 37
  - Konflikte, 37
- Schaltregel, 36
- Schlichtheit von, 35, 37
- schlingungsfrei, 35
- Stellen, 32, 34
  - Nachbereich, 34
  - Vorbereich, 34
- Transitionen, 32, 34, 36, 44, 45
  - aktivierte, 36, 38
  - direkt auslosende, 36, 37, 42
  - mit Ereignissen auslosende, 36, 37, 42

Produkte, 46, 72
Programmablauf, 6
- Optionen
  - :pattern-opt-level, 8
  - Bildschirmausgabe, 65
  - Patternoptimierung, 65

PROMOT, 1, 2
Prozeßgrößenkonzept, 14
Prozeßgrößen, 23
- Aneinandersetzung örtlicher Teilbereiche, 24
- Ausgabeintervall (step-width), 25, 26
- indizierte, 25
- Komponenten, 24
- mehrdimensionale, 25
- Prozeßgrößenzustande, 24

Prozeßgrößenkonzept, 23
Präfix-Notation, 7, 20, 47, 66, 69

Real-Format, 9

Scanner, 7
Schalten
- statisches, 31

Schaltflags, 31
- SWITCH, 31
- TTFLAG, 31

Schaltfunktion, siehe Events, 2
Schaltparameter, 28, 29, 31
Schaltzeitpunkt, 29
Schlüsselworte, 13, 69
- Übersicht für Parameter und Variablen, 18

Sequenzen, 32, 33
Sparse-Matrix-Technik, 2
Spline-Ausdruck, 20, 22
Steuerbefehle
- (cg-cg) CG-Eingabedatei erzeugen, 66
- (cg-dat) Datenfile-Codierung, 65
- (cg-fortran) FORTRAN-Unterprogramm-Codierung, 65
- (cg-fortran-dat) Grundelement-Codierung, 65
- (cg-init) Initialisierung, 65
- (cg-latex) LaTeX-ModellDokumentation, 66
- (cg-lib) Konsistenzprüfung der Unterprogramm-Bibliothek, 66
- Überschreib-Modus (do-overwrite), 67
- Aufruf des Code-Generators, 78
- Ausgabepfad (default-output-pathname), 67
- Eingabedatei (input-file), 67
- Optionen
  - :pattern-opt-level, 8, 65
  - :verbose-level, 65

Steuerfile, 65
Strukturparameter, 13, 17
Stutzendefinitionen, 8
Summen, 46, 57, 72
SWITCH, 31
Symbolische Vorverarbeitung, 3
Syntax-Checker, 8
Syntax-Notation, 12
SYPPROT, 2

TTFLAG, 31

Unstetigkeiten, 27
- durch Interaktion mit der Umgebung, 27
- innere, 27
- numerische Behandlung, 27
- siehe auch Events, 2
- strukturelle, 28

Wertelisten, 21

Zustandsvariablen, 13, 16
    Anfangswerte, 1, 3
    Zeitableitung, 49