

LPL : EINE MATHEMATISCHE MODELLIERSPRACHE
EINE EINFÜHRUNG

Version 4.43

Tony Hürlimann

Working Paper

August 2003

Department of Informatics, University of Fribourg
site Regina Mundi, rue de Faucigny 2, CH-1700 Fribourg / Switzerland
tony.huerlimann@unifr.ch

Diese Forschungsarbeit wird vom Schweizerischen Nationalfond, Projekt Nr.
12-55989.98, finanziert.

LPL : Eine mathematische Modelliersprache Eine Einführung

Tony Hürlimann, Dr. lic. rer. pol.

Key-words: Modeling Language, Linear Programming, Compiler.

Summary: This paper describes the last version of the modeling language, named LPL (*Linear Programming Language* or *Logic Programming Language*). It may be used to build, modify and document (linear and nonlinear) mathematical and logical models. The LPL language has been successfully applied to formulate large real-live models. The available LPL compiler translates LPL programs to the input code of a solver, calls the solver automatically, reads the solution back to its internal representation, and the integrated Report Generator produces the user-defined reports of the model. Furthermore, an Input Generator can read the data from many formats.

Stichworte: Modelliersprachen, lineare und nicht-lineare Optimierung.

Zusammenfassung: Dieser Artikel beschreibt die neue Version der Modelliersprache LPL (*Linear Programming Language* oder *Logic Programming Language*), die sich dazu eignet, mathematische Modelle aufzubauen, zu warten und zu dokumentieren. Die LPL-Sprache wurde zum Erstellen umfangreicher mathematischer Modelle erfolgreich eingesetzt. Der LPL-Compiler übersetzt ein LPL-Programm, welches ein vollständiges Modell repräsentiert, in den Eingabecode eines Lösungsprogramms, ruft den Lösungsalgorithmus auf, liest die Lösung, und ein integrierter Tabellengenerator gibt vom Benutzer definierte Resultate-Tabellen aus. Außerdem erlaubt ein Dateneingabe-Generator, die Daten in verschiedenen Formaten zu lesen.

1. EINFÜHRUNG

Die Forschung des Operations Research hat sich in den letzten Jahrzehnten hauptsächlich auf die mathematischen Lösungsmethoden und -verfahren von Modellen konzentriert. Dadurch wurde es erst möglich – zusammen mit immer leistungsfähigeren Computern – grösseren Modellen zu lösen. Doch wurden Fragen des Modellmanagements (Modellformulierung, Modellbau und Modellwartung) lange stiefmütterlich behandelt und sind immer noch relativ unterentwickelt. Vor allem in der praktischen Anwendung des OR, wo grössere Modelle unumgänglich sind, werden Instrumente des Modellmanagements nicht nur nützlich, sondern unerlässlich. Wenn die Lösung eines Modells auf einem Grosscomputer höchstens Minuten dauert, die Modellformulierung und -wartung sich aber über Wochen, ja Monate erstreckt, wie das durchaus üblich ist, so besteht hier ein krasses Missverhältnis. Der Einsatz von OR-Methoden in der Praxis hängt damit direkt von der Investition ab, die es braucht, um solche Modelle zu erstellen und zu warten.

Es braucht für die praxisorientierte Anwendung des Operations Research mehr als nur effiziente Lösungsalgorithmen, ausserdem liegt ein Modell nicht von Anfang an in einer Form vor, die ein Lösungsalgorithmus verlangt. Solche Einsichten haben sich in der letzten Zeit nur zögernd durchgesetzt. Zwar wurden schon in den 70er Jahren effiziente und mächtige Matrix Generatoren and Report Generatoren geschaffen. Typischerweise arbeitet man mit diesen Instrumenten so, dass der Modellierer ein Computerprogramm schreibt, welches das Modell in eine vom Matrix Generator lesbare Form bringt. Wird das Modell geändert, muss auch dieses Übersetzungsprogramm neu geschrieben oder abgeändert werden. Ein solches Vorgehen verlangt vom Modellierer nicht nur Kenntnisse in der Programmierung, sondern ist auch mit einem erheblichen Aufwand verbunden. Zudem liegt das Modell in einer Form vor – nämlich oft als FORTRAN Programm –, das eher vom Programmierer als vom Modellierer interpretiert werden kann.

Eine Modelliersprache als Instrument einer Modellformulierung könnte einige dieser Nachteile umgehen, da sie eine Formulierung erlaubt, die dem Modellbauer näher steht. Die Idee einer Modelliersprache ist nicht neu, und verschiedene Sprachen wie GAMS [Brooke al. 1988], AIMMS [Bisschop 1999], AMPL [Fourer al. 2003], LINGO [Schrage 2000] sind entwickelt worden und kommerziell verfügbar. Eine Modelliersprache ist sicher nur ein, wenn auch ein wichtiger Bestandteil einer Modellumgebung. Dazu gehören nämlich auch Instrumente der Datenverwaltung eines Modells, der Modellverifikation, der Modelldokumentation, sowie Instrumente, die den gesamten Lebenszyklus und die Evolution eines Modells unterstützen [Hürlimann 1999].

Dieser Artikel stellt eine weitere Modellersprache – genannt LPL – vor, mit der es möglich ist, grössere, mathematische Modelle zu formulieren und zu warten. Die Formulierung eines Modells in der LPL-Form kann direkt dazu verwendet werden, das Modell automatisch in einen Code zu übersetzen, welcher einen Lösungsalgorithmus verlangt.

LPL unterscheidet sich von den oben genannten Sprachen folgendermassen: Die Sprache besitzt eine offene Schnittstelle zu den meisten Lösungsalgorithmen. Der Dateneingabe-Generator ist in keiner anderen Sprache vorhanden und nur GAMS besitzt auch einen Resultate-Generator. Die Syntax von LPL ist flexibler. Entitäten können mehrmals zugewiesen werden wie in GAMS, aber im Gegensatz zu GAMS kann in LPL die Modellstruktur unabhängig von den Daten verarbeitet werden. LPL besitzt zudem als einzige Sprache die Möglichkeit, Masseinheiten selber zu definieren. Erst die neue Version von AIMMS wird diese Möglichkeit auch enthalten. Die wichtigsten Bereiche, die LPL abdeckt, sind:

- Ein LPL Modell besteht aus einer einfachen Syntax - ähnlich der mathematischen Notation mit indextierten Ausdrücken.
- Die Formulierung von kleinen und grossen Modellen ist möglich dank der Aufspaltung des Modells in Modellstruktur und Modelldaten.
- Ein mächtiger Indexmechanismus ist eingebaut, mit dem die Modellstruktur sehr flexibel gestaltet und das Modell 'aufgeblasen' werden kann.
- Ein innovativer Tabellen- und Dateneingabe-Generator ist eingebaut.
- Beliebige indextierte Ausdrücke sind formulier- und sofort auswertbar.
- Mit Hilfe von verschiedene Instrumenten kann das Modell auf Fehler untersucht werden.
- Ein eingebauter Texteditor erlaubt es, ein Modell zu formulieren und zu ändern.
- Für LP/MIP Modelle wird der MPS Standard-Code schnell produziert.
- Die Schnittstelle zu einem LP/MIP Lösungsalgorithmus ist offen und kann vom Benutzer definiert werden.
- Ferner enthält LPL Elemente einer Programmiersprache.

Anhand dreier einfacher Modells sollen in §2 und §3 die Grundideen von LPL kurz vorgestellt werden. In §4 und §5 werden einige Aspekte vertieft und §6 ist dem Tabellen- und Dateneingabe-Generator gewidmet.

2. DREI MODELLBEISPIELE

Drei Modellbeispiele sollen einen kurzen Einstieg in LPL geben. Das erste Modell ist ein einfaches Produktionsmodell aus der Elektrizitätswirtschaft. Das zweite Modell ist

ein kleines Portfoliomodell zur Reinvestition von überschüssigen Liquiditätsmitteln in einer Unternehmung, und das dritte Modell ist ein Zuteilungsmodell, in dem 180 Spieler auf 15 Teams verteilt werden sollen.

2.1. BEISPIEL 1 : STROMGENERATOREN [DAY & WILLIAMS 1986]

Ein vereinfachtes Modellbeispiel aus der Elektrizitätswirtschaft soll Ausgangspunkt für die Beschreibung der Modellersprache LPL sein:

In einem Elektrizitätswerk müssen ständig Stromgeneratoren ein- und ausgeschaltet werden, um die stündlichen Schwankungen des Stromverbrauchs zu decken. Ein Tag werde in 5 Zeitzonen eingeteilt, in welchen wir einen konstanten, geschätzten Stromverbrauch annehmen (Tabelle 2-1).

Tageszeit-Zone	Stromverbrauch
Mitternacht bis 6 Uhr	15'000 Megawatt
6 - 9 Uhr	30'000 Megawatt
9 - 15 Uhr	25'000 Megawatt
15 - 18 Uhr	40'000 Megawatt
18 Uhr bis Mitternacht	27'000 Megawatt

Tabelle 2-1 : Datentabelle zum Stromverbrauch

Nehmen wir weiter vereinfachend an, dass im Elektrizitätswerk drei verschiedene Generatortypen installiert sind: Zwölf Generatoren vom Typ 1, zehn vom Typ 2 und fünf vom Typ 3 seien im Einsatz. Jeder Generatortyp, einmal in Betrieb, liefert eine minimale Strommenge, kann aber ein bestimmtes Maximum nicht übersteigen. Die Betriebskosten über dem Minimum sind höher als auf dem Minimum. Zudem müssen Anschaltkosten in Betracht gezogen werden. Die Daten sind in der Tabelle 2-2 zusammengestellt.

	Typ 1	Typ 2	Typ 3
minimale Kapazität (mW)	850	1'250	1'500
maximale Kapazität (mW)	2'000	1'750	4'000
minimale Betriebskosten (Fr/Std.)	1'000	2'600	3'000
zusätzliche Betriebskosten pro mW über dem Minimum (Fr/Std./mW)	2.--	1.30	3.--
Anschaltkosten	2'000	1'000	500
Anzahl Generatoren	12	10	5

Tabelle 2-2 : Daten zu den Stromgeneratortypen

Um die geschätzte Nachfrage jederzeit decken zu können, ist vorgesehen, dass eine plötzliche Zunahme von 15% der Nachfrage abgefangen werden kann, ohne dass neue

Generatoren in Betrieb genommen werden müssen.

Die Frage lautet nun: Welche Generatoren müssen zu jeder Tageszeit in Betrieb sein, bzw. in Betrieb genommen werden, wenn die Betriebskosten zu minimieren sind? Ausserdem interessiert uns die Frage, welches die marginalen Produktionskosten des Stroms zu jeder Tageszeit sind.

Dieses Problem kann als ein lineares Optimierungsmodell formuliert werden, welches in Abbildung 2-1 aufgelistet ist.

Gegeben		
i	Generatortypen ($i = 1, \dots, N$) mit $N = 3$	
t	Zeitzone ($t = 1, \dots, T$) mit $T = 5$ (zyklisch)	
Daten		
m_i	minimaler Betriebsoutput pro Generatortyp i (in Megawatt)	
M_i	maximale Kapazität des Generatortyps i (in Megawatt)	
C_i	minimale Betriebskosten pro Generatortyp i (in Franken/Std.)	
E_i	Extra Betriebskosten pro Megawatt über dem Minimum je Typ i (in Franken/Gigawatt/Std.)	
F_i	Anschaltkosten pro Generatortyp i (in Franken)	
L_i	Anzahl Generatoren des Typs i	
D_t	geschätzte Stromnachfrage in der Zeitzone t (in Gigawatt)	
N_t	Länge der Zeitzone t (in Stunden)	
Unbekannte		
n_{it}	Anzahl Generatoren vom Typ i in Betrieb zur Zeit t	
s_{it}	Anzahl gestartete Generatoren vom Typ i zur Zeit t	
x_{it}	Stromproduktion von Generatoren des Typs i zur Zeit t (in Gigawatt)	
Minimiere die Kosten (extra Betriebskosten über dem Minimum + minimale Betriebskosten + Anschaltkosten):		
$\sum_{i=1}^N \sum_{t=1}^T (N_i E_i (x_{it} - m_i n_{it}) + N_t C_i n_{it} + F_i s_{it})$		
mit den Restriktionen		
$\sum_{i=1}^N x_{it} \geq D_t \quad \text{mit } t = 1, \dots, T$	Nachfragedeckung	
$\sum_{i=1}^N M_i n_{it} \geq \frac{115}{100} D_t \quad \text{mit } t = 1, \dots, T$	15% extra Produktionsmarge	
$m_i n_{it} \leq x_{it} \leq M_i n_{it} \quad \text{mit } i = 1, \dots, N, \quad t = 1, \dots, T$		Strommengenproduktion
$s_{i,t} \geq n_{i,t} - n_{i,t-1} \quad \text{mit } i = 1, \dots, N, \quad t = 1, \dots, T$		Anzahl gestartete Generatoren vom Typ i in der Periode t .
$n_{it} \leq L_i \quad \text{mit } i = 1, \dots, N, \quad t = 1, \dots, T$		maximale Anzahl Generatoren
$x_{it} \geq 0, s_{it} \geq 0, n_{it} \geq 0 \quad \text{und ganzzahlig}$		

Abbildung 2-1 : Mathematische Formulierung des Stromgeneratoren-

Modells

Da die Zeitzonen zyklisch definiert sind, d.h. die Vorperiode der ersten Zeitzone die letzte Zeitzone des Vortages ist, und die nachfolgende Periode der letzten Zeitzone die erste Zeitzone des nächsten Tages ist, gilt die Restriktion $s_{i,t} \geq n_{i,t} - n_{i,t-1}$ auch für $t = 1$, die gegeben ist als: $s_{i1} \geq n_{i1} - n_{iT}$.

Die Struktur dieses Modells kann direkt in der LPL-Sprache wiedergegeben werden. Das LPL Modell, welches dieses Modell repräsentiert, ist in Abbildung 2-2 aufgelistet.

```

MODEL Will15D "Stromproduktion";

SET
  i          "Generatorentypen";
  t STRING tName  "Zeitzone";
UNIT
  sFR        "Geldeinheit";
  gW         "Gigawatt";
  mW         "Megawatt"      := gW/1000;
  hour       "Stunden";
PARAMETER
  m{i} UNIT [mW]          "minimaler Betriebsmenge pro Generatortyp i";
  M{i} UNIT [mW]          "maximale Kapazitaet des Generatortyps t";
  C{i} UNIT [sFR/hour]    "min. Betriebskosten/Std pro Generatortyp i";
  E{i} UNIT [sFR/gW/hour] "Extra Betriebskosten/gW/Std. ueber dem Minimum";
  F{i} UNIT [sFR]         "Anschaltkosten pro Generatortyp i";
  L{i}         "Anzahl von Generatoren des Typs i";
  D{t} UNIT [gW]         "geschaetzte Stromnachfrage zur Zeit t";
  N{t} UNIT [hour]       "Laenge der Zeitzone t (in Stunden)";

VARIABLE
  x{i,t} UNIT [gW]       "Stromproduktion des Typs i zur Zeit t";
  INTEGER n{i,t}         "Anzahl Generatoren vom Typ i in Betrieb zur Zeit t";
  INTEGER s{i,t}         "Anzahl gestartete Generatoren vom Typ i zur Zeit t";

CONSTRAINT (*Modellrestriktionen*)
  Nachfrage{t} UNIT [gW] : SUM{i} x >= D;
  Extrakapazitaet{t} UNIT [gW] : SUM{i} M*n >= 1.15*D;
  Output{i,t} UNIT [gW] : m*n <= x <= M*n;
  Gestartet{i,t}: s >= n - n[i,(#t+t-2)%#t+1];
  ObereSchranke{i,t}: n <= L >= s;
MINIMIZE Kosten UNIT [sFR] : SUM{i,t} (N*E*(x-m*n) + N*C*n + F*s);
WRITE n, s, x, Kosten;

MODEL DATA aData "Daten für fünf Tagesperiode und drei Generatorentypen";
  READ FROM 'Will15D.dat' '%:Tabelle:Tabelle';
  READ '%1' : ROW{t} (t , tName , D, N);
  READ '%2' : ROW{i} (i , m , M , C , E , F , L);
END

END

```

Abbildung 2-2 : LPL-Code zum Stromgeneratoren-Modells

Das Modell in Abbildung 2-1 besteht aus einer Liste von Deklarationen: Indexmengen, Datentabellen, Unbekannte, eine minimierende Funktion und lineare Restriktionen. Keine Daten sind definiert. Die Beschreibung ist sogar dimensionsunabhängig und

repräsentiert die *Modellstruktur*. Die LPL Formulierung in Abbildung 2-2 ist ähnlich aufgebaut. Die verschiedenen Deklarationen werden durch ein reserviertes Wort SET, UNIT, PARAMETER VARIABLE, CONSTRAINT, etc. eingeleitet und durch einen Strichpunkt abgeschlossen. Es ist wichtig zu sehen, dass diese Formulierung ebenfalls ganz unabhängig von den Daten ist. Ein Datensatz wird definiert in der Datei *Will15D.dat* (siehe Abbildung 2-3 für unser konkretes Beispiel). Das ausformulierte Modell wird umso grösser, je mehr die Datenmenge in den Dateien wächst. Dies berührt jedoch die Formulierung in Abbildung 2-2 (und 2-1) in keiner Weise.

```
(* Datenfile Will15D.DAT zum Modell Will15D.LPL *)
```

Tabelle 1 : Stromverbrauchskurve
 (* Zeitzonen geschätzte Nachfrage Anzahl
 (in Gigawatt) Stunden
 ----- *)

t1	'Mitternacht bis 6 Uhr'	15	6
t2	'6 - 9 Uhr'	30	3
t3	'9 - 15 Uhr'	25	6
t4	'15 - 18 Uhr'	40	3
t5	'18 Uhr bis Mitternacht'	27	6

Tabelle 2 : Daten zu den Generatorentypen
 (*Typ minimale maximale Kosten bei extra Kosten pro Start Anzahl
 Kapa.(mW) Kapa.(mW) min. Betrieb produzierte gW kosten
 Generatoren
 ----- *)

G1	850	2000	1.0	2.0	2.0	12
G2	1250	1750	2.6	1.3	1.0	10
G3	1500	4000	3.0	3.0	0.5	5

Tabelle 2-3 : LPL Daten zum Stromgeneratoren-Modell

Die LPL Formulierung zeigt noch einige weiteren Merkmale gegenüber der mathematischen Formulierung in Abbildung 2-1: Kommentare stehen in Hochkommata ("...") oder zwischen (*...*); weitere ein-zeilige Kommentare sind möglich und werden durch zwei aufeinanderfolgende Striche (--) eingeleitet. Die Indexe sind geklammert statt tiefgestellt, Zeichen wie \sum sind durch reservierte Wörter wie SUM ersetzt. Zudem wird die LPL-Formulierung durch drei weitere Anweisungen abgeschlossen:

- *READ* ... die Modelldaten werden von externen Dateien eingelesen
- *MINIMIZE* ... das Modell wird gelöst
- *WRITE* ... die entsprechenden Resultate-Tabellen werden generiert.

Nach Aufruf des LPL Compilers und des Lösungsalgorithmus werden die Resultatetabellen der drei Variablen n , s , x sowie die *Kosten* wie folgt in eine Datei geschrieben (Tabelle 2-4).

$n\{i,t\}$					
	t1	t2	t3	t4	t5
G1	12	12	12	12	12
G2	3	8	8	9	9
G3	0	0	0	2	0
$s\{i,t\}$					
	t1	t2	t3	t4	t5
G1	0	0	0	0	0
G2	0	5	0	1	0
G3	0	0	0	2	0
$x\{i,t\}$					
	t1	t2	t3	t4	t5
G1	10.2000	16.0000	11.0000	21.2500	11.2500
G2	4.8000	14.0000	14.0000	15.7500	15.7500
G3	0.0000	0.0000	0.0000	3.0000	0.0000
Kosten					
	988.5400				

Tabelle 2-4 : Ausgabe und Resultat des Stromgeneratoren-Modells

Das Resultat in Tabelle 2-4 besagt, dass in allen Zeitzeonen alle 12 Generatoren des Typs 1 in Betrieb sind. Auf Grund der Nachfrage müssen in der zweiten Zeitzone 5 Generatoren des zweiten Typ hinzugeschaltet werden und einer in der Zeitzone 4. Ebenso müssen zwei Generatoren des Typs 3 in der Zeitzone 4 hinzugeschaltet werden. Der Output x in jeder Zeitzone entsprechen gerade der Nachfrage, wobei die Generatoren des Typs 1 in der vierten Zeitzone eine grösste Kapazität erreichen, die aber noch 2.5MW unter der maximalen Kapazität der 12 Generatoren liegt. Die Gesamtkosten betragen 988.54.

2.2. BEISPIEL2 : PORTFOLIOAUSSWAHL

Problem: Ueberschüssige Liquidität soll so angelegt werden, dass in jeder Periode die Verbindlichkeiten erfüllt werden können. Dabei wird eine kostenminimierende Kaufstrategie der Anlagepapiere angestrebt. In welche Anlagepapiere ist die Liquidität in jeder Periode anzulegen?

Abbildung 2-3 gibt eine algebraische Formulierung dieses einfachen Problems zur optimalen Portfolioauswahl wieder [Shapiro 1988, S. 589ff]. Auch diese Formulierung kann direkt in ein LPL-Modell übersetzt werden (Abbildung 2-4).

Die algebraische Formulierung besteht aus verschiedenen Sektionen: Der Reihe nach werden die Indexmengen, die numerischen Datentabellen, die Unbekannten, die Zielfunktion und die linearen Beschränkungen deklariert. Man beachte auch hier, dass die Formulierung in Abbildung 2-3 noch kein bestimmtes, *instanziertes* Modell

definiert, da die Daten fehlen: Vielmehr wird eine Modellklasse oder Modellstruktur deklariert.

Die Abbildung 2-4 zeigt die Modellstruktur in LPL Syntax. Auch diese Formulierung besteht aus den Sektionen Indexmengen, Daten, Unbekannte, Modellrestriktionen und Optimierungsfunktion, die jeweils durch die reservierten Wörter SET, PARAMETER, VARIABLE, CONSTRAINT und MINIMIZE eingeleitet werden.

Gegeben	
j	Anlagepapiere ($j = 1, \dots, N$)
t	Zeithorizont ($t = 1, \dots, T$) mit $T \leq 50$
Daten	
c_j	gegenwärtiger Marktpreis des Anlagepapiers j (in sFR)
f_{jt}	Coupon des Anlagepapiers j in der Periode t (in sFR/Stück)
L_t	Liquiditätsforderungen in Periode t (in sFR)
q_j	minimaler Kauf einer Anlage j (in Stückzahl)
Q_j	maximal erlaubbarer Kauf einer Anlage j (in Stückzahl)
a_j	Reinvestitionsrate in der Periode t (in Prozent)
Unbekannte	
x_j	Kaufmenge eines Anlagepapiers j (in Stückzahl)
s_t	akkumulierter Liquiditätsüberschuss am Ende der Periode t (in sFR)
d_j	ist 1, wenn die Anlage j in das Anlagepaket eingehen soll, sonst 0.
Minimiere	
$\sum_{j=1}^N c_j x_j + s_0$ die Anlagekaufsumme plus die Anfangsliquidität	
mit den Restriktionen	
$\sum_{j=1}^N f_{jt} x_j + a_t s_{t-1} - s_t = L_t$ mit $t = 2, \dots, T$ die Couponsumme plus die Liquidität der Vorperiode muss gleich der Liquiditätsforderung der Periode t sein.	
$q_j d_j \leq x_j \leq Q_j d_j$ mit $j = 1, \dots, N$ entweder $x = 0$ oder $q \leq x \leq Q$	
$x_j \geq 0, s_t \geq 0$ und $d_j \in \{0, 1\}$	

Abbildung 2-3 : Mathematische Formulierung des Portfolio-Modells

Masseinheiten können durch die UNIT Anweisung definiert werden. Kommentare sind überall zwischen Apostrophen und (* ... *), so wie nach '--' eingestreut. Kommentare zwischen (* ... *) und nach '--' gehören nicht zum formalen Teil des Modells, sie dokumentieren bloss den Modelltext. Kommentare zwischen Apostrophen ("...") sind jedoch Teil der formalen Dokumentation und können in den Resultate-Tabellen verwendet werden. Insbesondere werden sie zur automatischen Generierung einer

TEX-Datei verwendet, um das Modell zu dokumentieren.

Da ein LPL-Modell direkt vom Computer verarbeitet werden kann, unterscheidet es sich von der algebraischen Form: Das Summenzeichen \sum wird durch das reservierte Wort SUM ersetzt, tiefgestellte Indexe werden durch eine Indexliste in Klammern geschrieben, und jede Deklaration ist durch einen Strichpunkt abzuschliessen. Statt einzelne Buchstaben als Namen zu verwenden, können auch Wörter als Namen verwendet werden. So könnte z.B. $c\{j\}$ durch *Marktpreis{Anlage}* ersetzt werden.

```

MODEL Portfold "Portfolioanlage der überschüssigen Liquidität";

SET
  j          "Anlagepapiere";
  t          "Zeithorizont";
UNIT
  sFR       "Geldeinheit";
  Prozent   "Prozent (%)"      := 1/100;
  Anzahl    "Stueckzahl";
  StkPreis  "Preis/Stueck"     := sFR/Anzahl;
PARAMETER
  c{j} UNIT [StkPreis]         "gegenwaertiger Marktpreis des Anlagepapiers j";
  f{j,t} UNIT [StkPreis]      "Coupon des Anlagepapiers j in der Periode t";
  q{j} UNIT [Anzahl]          "minimaler Kauf einer Anlage j";
  Q{j} UNIT [Anzahl]          "maximal erlaubbarer Kauf einer Anlage j";
  a{t} UNIT [Prozent]         "Reinvestitionsrate in der Periode t";
  L{t} UNIT [sFR]             "Liquiditaetsforderungen in Periode t";
VARIABLE
  x{j} UNIT [Anzahl]          "Kaufmenge eines Anlagepapiers";
  s{t} UNIT [sFR]             "akkumulierter Liquiditaetsueberschuss Ende t" ;
  BINARY d{j}                 "=1, wenn Anlage j selektioniert wird, sonst 0";
CONSTRAINT
  Balance{t|t>1} UNIT [sFR]   "Couponsumme plus Liquidität der
                               Vorperiode = Liquidit&tsforderungen"
    : SUM{j} f*x + a*s[t-1] - s[t] = L[t];
  C{j} UNIT [Anzahl]          "entweder x=0 oder q<=x<=Q"
    : q*d <= x <= Q*d;
  initS UNIT [sFR]            "Anfangsliquidit&t" : s[1] = 0[sFR];
MINIMIZE Invest UNIT [sFR]    "Anlagekaufssumme plus Anfangsliquidität"
  : SUM{j} c*x + s[1];
WRITE Invest, x, s, d;

MODEL DATA aDataSet "liest die Daten von Dateien";
PARAMETER TMAX = 50;
BEGIN
  READ FROM 'portfold.dat' '%1:Table:Table';
  READ '%1': ROW{j} (j,c,q,Q);
  READ '%2': ROW{t} (t,L,a);
  READ '%3': COL{t} t, ROW{j} (j, COL{t} f);
  CHECK This{j} UNIT [Anzahl] : q < Q ;
  CHECK This: #t <= TMAX;
END
END

```

Abbildung 2-4 : LPL-Code des Portfolio-Modells

Vier weitere Anweisungen sind in Abbildung 2-4 zu finden:

- *MODEL DATA* : definiert ein Daten-Untermmodell und eine Datei, in welcher die Daten zum Modell zu finden sind (Abbildung 2-5),
- *MINIMIZE...* übergibt das Modell dem Lösungsalgorithmus und übernimmt anschliessend die Lösungsdaten,
- *WRITE...* schreibt die gewünschten Daten als Tabellen in Dateien.
- *UNIT ...* wird verwendet, um Masseinheiten zu definieren (siehe §3.7).

```
(* Datenfile zum Modell PORTFOLD.LPL *)
Table 1
(* j      c      q      Q  *)
A1    200    10    500
A2    230     .    700
A3    400    20    700
A4    100    15    800
A5    240    20    900

Table 2
(* t      L      a      *)
T1     .      90
T2    1200    90
T3    1400    80
T4     500    80

Table 3
(* f *)
      T1  T2  T3  T4
A1    .   4   4   4
A2    .  5.5 5   4
A3    .   5   3   6
A4    .   6   6   .
A5    .   4   5   6
```

Abbildung 2-5 : LPL-Daten zum Portfolio-Modells

Die Ausführung dieses Modell generiert eine Datei *PortFold.NOM* (Abbildung 2-6), die besagt, dass 272.36 Einheiten des Anlagepapiers A4 zu kaufen sind.

```
Invest
27236.8420

x{j}
      A1      A2      A3      A4      A5
0.0000  0.0000  0.0000  272.3684  0.0000

s{t}
      T1      T2      T3      T4
0.0000  434.2105  625.0000  0.0000

d{j}
      A1      A2      A3      A4      A5
0          0          0          1          0
```

Abbildung 2-6

Wir benötigen ein Investitionsvolumen von 27236.84, um diese Anlagepapiere zu kaufen und der Rückfluss dieser Anlage garantiert, dass die Verbindlichkeiten in jeder Periode eingehalten werden können. Liquidität muss in der zweiten und der dritten Periode gehalten werden.

(Das LPL Modell zusammen mit den Daten befindet sich im Verzeichnis Models\german als Dateien PORTFOLD.LPL und PORTFOLD.DAT.)

2.3. BEISPIEL 3 : EIN ZUORDNUNGSPROBLEM

Mengen	
t	Teams (Gruppen) ($t = \{1, \dots, 15\}$)
p	Personen (Spieler) ($p = \{1, \dots, 180\}$)
Daten	
$M_{p,t}$	Spieler p muss in Team t sein
$R_{p,t}$	Spieler p darf nicht in Team t sein
$G_{p,p}$	Spieler, die zusammen im selben Team sein müssen
$N_{p,p}$	Spieler, die nie zusammen im selben Team sein dürfen
S_p	Die Spielstärke eines Spielers p (Rang zwischen 3 und 8)
A_p	Das Alter eines Spielers p (zwischen 10 und 12)
Variablen	
$w_{p,t}$	Eine Boolesche Variable, die angibt, ob ein Spieler p dem Team t zugeordnet ist oder nicht
Maximize	
	$\sum_{p=1}^{180} \sum_{t=1}^{15} w_{p,t}$ alle Spieler müssen zugeordnet werden
subject to	
	$\sum_{t=1}^{15} w_{p,t} = 1$ for $p = 1, \dots, 180$ Bedingung (1)
	$\sum_{p=1}^{180} w_{p,t} = 12$ for $t = 1, \dots, 15$ Bedingung (2)
	$\sum_{p=1}^{180} S_p \cdot w_{p,t} \geq 59$ for $t = 1, \dots, 15$ Bedingung (3)
	$\sum_{p=1}^{180} A_p \cdot w_{p,t} \geq 124$ for $t = 1, \dots, 15$ Bedingung (4)
	$w_{p,t} = 1$ for $(p,t) \in M$ Bedingung (5)
	$w_{p,t} = 0$ for $(p,t) \in R$ Bedingung (6)
	$w_{i,t} - w_{j,t} = 0$ for $t \in \{1, \dots, 15\}, (i,j) \in G$ Bedingung (7)
	$\sum_{j \in p \exists j \in p N_{i,j}} w_{j,t} \leq 1$ for $t \in \{1, \dots, 15\}, i \in p \exists j \in p N_{i,j}$ Bedingung (8)

Abbildung 2-7 : Mathematische Formulierung des Soccer-Modells

Das dritte Beispiel ist ein Zuordnungsproblem: Eine Anzahl Personen sollen so in

```

MODEL Soccer "An assignment problem of 180 player to 15 teams";
SET
  t          "the list of teams";
  p          "the list of players";
  mustBeIn{p,t} "player p must be in team t";
  rejectFrom{p,t} "player p is rejected from a team t";
  tGroups{p,p} "groups of players who must be together in a team";
  nGroups{p,p} "groups of players who must never be together";

PARAMETER Skill{p} "skill of player p";
          Age{p}   "age of player p";

BINARY VARIABLE work{p,t} "=1 if player p is in team t else =0";

CONSTRAINT
  Bounds{p}          : SUM{t} work = 1;
  Heads{t}           : SUM{p} work = 12;
  SkillLevel{t}      : SUM{p} work * Skill >= 59;
  TeamAge{t}         : SUM{p} work * Age >= 124;
  Must{i=mustBeIn}   : work[i] = 1;
  Reject{i=rejectFrom} : work[i] = 0;
  Same{t,tGroups[i,j]} : work[i,t] - work[j,t] = 0;
  Never{t,i=p | exist{j=p}nGroups[i,j]} : SUM{j=p}nGroups[i,j] work[j,t]<=1;
MAXIMIZE obj "maximize the assignment" : SUM{p,t} work;

WRITE "
  RESULTS PER PLAYER
  *****
  player Skill Age in Team
  -----
  $$$$$$ ## ## $$$$\n
  RESULTS PER TEAM
  *****
  team skill age Av.Skill Av.Age
  -----
  $$$$$$ ##### ### ###.### ###.###\n
  team | players in the team
  -----
  $$$ $ | $$$$\n"
  : ROW{p} ( p , Skill , Age , COL{t|work} t ) ,
    ROW{t} ( t , SUM{p|work} Skill , SUM{p|work} Age ,
      (SUM{p|work} Skill)/12 , (SUM{p|work} Age)/12 ) ,
    ROW{t} ( t , COL{p|work} p );

MODEL DATA data1; BEGIN -- data is here
  OPTION randomSeed:=1;
  t = /T1 T2 T3 T4 T5 T6 T7 T8 T9 T10 T11 T12 T13 T14 T15/;
  p = /1:180/;
  mustBeIn{p,t} = / 1 T2 , 2 T6 , 34 T7 /;
  rejectFrom{p,t} = / 10 T1 , 20 T2 , [166,*] T1 T3 T4 T5 T6 T7 T8 T9 ,
    [64,*] T1 T12 /;
  tGroups{p,p} = / 2 3 , 112 76 , 89 9 , 34 135 , [4,*] 35 47 81 98 /;
  nGroups{p,p} = / 21 22 , 55 56 , [11,*] 35 45 56 67 78 89 90 21 /;
  Skill{p} = trunc(rnd(3,8));
  Age{p} = trunc(rnd(10,12));
  (* check that a player p can be only in one team *)
  CHECK This{p}: SUM{t | mustBeIn}1 <= 1;
END

END

```

Abbildung 2-8 : LPL Formulierung des Fussball-Zuordnungsproblems

einzelne Gruppen verteilt werden, dass verschiedenen Bedingungen erfüllt sind [Byer J., 1990]. Die Formulierung des Zuweisungsproblems sei wie folgend:

Verteile 180 Spieler in 15 Teams für ein Fussballturnier (Soccer), sodass folgende Bedingungen erfüllt sind:

- 1 Ein Spieler darf nur in einem Team sein,
- 2 Jedes Team muss aus 12 Spielern bestehen,
- 3 Die gesamte Teamstärke muss mindestens 59 Einheiten betragen,
- 4 Das gesamte Teamalter muss mindestens 124 Jahre sein,
- 5 Gewisse Spieler müssen in bestimmten Teams sein,
- 6 Gewisse Spieler dürfen nicht in bestimmten Teams sein,
- 7 Gewisse Spieler müssen zusammen im selben Team sein,
- 8 Gewisse Spieler dürfen nicht zusammen im selben Team sein.

Das Modell kann mathematisch wie in Abbildung 2-7 formuliert werden.

Die Datentabellen $M_{p,t}$, $R_{p,t}$, $G_{p,p}$ und $N_{p,p}$ sind keine numerischen Datentabellen, sondern sie definieren Tupelmengen.

Beispielsweise bedeutet: $M_{p,t} = \{(p1,t2),(p45,t12)\}$, dass Spieler 1 ($p1$) in Team 2 ($t2$) sein muss, und dass Spieler 45 ($p45$) im Team 12 ($t12$) sein muss. Die Beziehung $G_{p,p} = \{(p1,p2),(p45,p89)\}$ bedeutet, dass Spieler 1 ($p1$) und Spieler 2 ($p2$) zusammen in derselben Mannschaft sein müssen, und dass ebenfalls Spieler 45 ($p45$) und Spieler 89 ($p89$) zusammen sind. Solche Relationen können direkt (auch in LPL) als Tupellisten codiert werden (siehe 2-8).

Abbildung 2-8 gibt eine vollständige Implementierung des Zuordnungsmodells in LPL.

Die drei Modellbeispiele zeigen einige Aspekte der Modellierung in LPL, die wir jetzt im einzelnen erläutern wollen.

3. EIN KURZER ÜBERBLICK ÜBER DIE LPL SPRACHE

Anhand der Modellbeispiele sollen nun die einzelnen Elemente der LPL Sprache kurz vorgestellt werden.

3.1. INDEXMENGEN

Eine Indexmenge ist eine ungeordnete oder geordnete Menge von Objekten. Die Menge der Anlagepapiere j in unserem Beispiel ist eine solche Menge. Sie wird in LPL deklariert als

```
SET j "Anlagepapiere" ;
```

Die Elemente der Menge sind damit noch nicht festgelegt. Diese werden als Teil der Modelldaten (Abbildung 2-5) betrachtet. Dem Modellierer steht es allerdings frei, bei der Deklaration auch gleich die Elemente zu definieren. So hätte man j auch als

```
SET j := /1:5/; -- j ist eine Menge von fünf Anlagepapieren
```

deklarieren können. Die beiden Elemente 1 und 5 , getrennt durch einen Doppelpunkt, definieren einen Bereich zwischen einer unteren und oberen Grenze. Die explizite Deklaration

```
SET j = /1 2 3 4 5/;
```

ist dazu äquivalent. Für die Modelldokumentation noch besser wäre es, die Kürzel $1, \dots, 5$ durch sprechende Namen zu ersetzen, wie in

```
SET j := /WeltBank_Obligation88 IBM_Aktie Sandoz_Inhaber EFF_Fribourg ATT87/;
```

Der Text in Hochkommata (" ... ") ist ein qualifizierter Kommentar, der von LPL mitgeführt wird.

Indexmengen können selber indexiert sein. In diesem Fall besteht die Indexmenge aus einer Tupelliste, und die Indexmenge heisst *indexiert*. Wenn p und t einfache Mengen repräsentieren (Abb. 2-8), die definiert sind als :

```
SET p = / 1:180 /; -- 180 Spieler
SET t = /T1 T2 T3 T4 T5 T6 T7 T8 T9 T10 T11 T12 T13 T14 T15/; -- 15 Gruppen
```

dann können wir eine indexierte Tupelmengemenge *MustBeIn* deklarieren als

```
mustBeIn{p,t} = / 1 T2 , 2 T6 , 34 T7 /;
```

Diese Tupelliste definiert eine Relation zwischen den Spielern p und den Gruppen t . Sie besagt, dass Spieler 1 der Gruppe 2 ($T2$), Spieler 2 der Gruppe 6 ($T6$) und Spieler 3 der Gruppe 7 ($T7$) angehören soll. Wichtig ist, dass damit nicht die gesamte Tupelliste, sondern nur eine kleine Untermenge definiert wird. Ein weiteres Beispiel definiert zwei Untermengen einer einfachen Liste aus Spielern als

```
SET Sp1{p} = / 1 45 56 67 78 122 /;
SET Sp2{p} = / 2 67 123 145 12 178 /;
```

Die Menge $Sp1$ ist beispielsweise eine Untermenge von p und enthält die Elemente (1, 45, 56, 67, 78, 122). Statt eine explizite Liste anzugeben, können die indexierten Mengen auch aus mathematischen und logischen Ausdrücken gebildet werden. Die Vereinigungs-, Schnitt- und Differenzmenge von $Sp1$ und $Sp2$ werden folgendermassen generiert:

```
SET Vereinigung{p} = Sp1 or Sp2;
SET Schnitt{p}     = Sp1 and Sp2;
SET Differenz{p}   = Sp1 and ~Sp2; -- ~ ist der NOT-operator
```

Index-Mengen sind wichtige Bestandteile eines jeden grösseren Modells. LPL bietet eine breite Palette von Operationen und Manipulationsmöglichkeiten von Index-Mengen.

3.2. NUMERISCHE DATEN

Unter Parametern werden numerische Werte verstanden, die in das Modell eingehen. Die einfachsten Parameter besitzen nur einen Wert, wie in:

```
INTEGER PARAMETER TMAX [0,100];
```

Parameter können durch Bedingungen eingeschränkt werden. So bestimmt *INTEGER* und *[0,100]*, dass *TMAX* nur einen ganzzahligen Wert zwischen 0 und 100 annehmen darf. Der LPL Compiler wird einen Wert ausserhalb dieses Bereichs als Fehler erkennen. LPL besitzt auch eine *CHECK* Anweisung, mit welcher komplexere Bedingungen getestet und verschiedene Modellkomponenten auf ihre Konsistenz hin geprüft werden können. So prüft z.B. die Anweisung

```
CHECK This{j} UNIT [Anzahl]: q < Q;
```

ob q für jedes Anlagepapier strikte kleiner als Q ist.

Parameter können auch durch Indexmengen in Tabellen zusammengefasst werden. $c\{j\}$ ist ein solcher Parameter in unserem zweiten Modellbeispiel. Er deklariert für jedes Anlagepapier einen Marktpreis. Auch die minimale, sowie die maximale Kaufmenge (q und Q) einer Anlage sind Tabellen (Vektoren), die über die Menge der Anlagen definiert sind. Beachte: LPL unterscheidet Klein- und Grossbuchstaben, sodass q und Q als zwei verschiedene Namen erkannt werden. In den Datentabellen in Abbildung 2-5 sind diese drei Parameter direkt zusammen mit den Elementen (den einzelnen Anlagepapieren) definiert worden. Dies ist eine bequeme Art, eindimensionale Tabellen (Tabellen mit nur einem Index), die über denselben Index laufen, in einer Tabellenform zusammenzufassen. Der Modellierer ist aber auch hier frei, die Parameter in der Modellstruktur selbst und getrennt voneinander einzugeben. Die erste Tabellenform in Abbildung 2-5 könnte somit auch codiert werden wie folgt :

```
SET j = /1:5/;
PARAMETER c{j} = [ 200  230  400  100  240 ];
PARAMETER q{j} = [  10   .   20   15   20 ];
PARAMETER Q{j} = [  50   70   70   80   90 ];
```

LPL besitzt mehrere Standardformate, welche für die Eingabe von Daten verwendet werden können. Gewöhnlich werden die Daten jedoch von externen Dateien mit Hilfe des Dateneingabe-Generators gelesen und müssen nicht in LPL Form vorliegen.

3.3. UNBEKANNTE (VARIABLEN)

Die Unbekannten werden gleich definiert wie die Parameter. Auch sie besitzen einen numerischen Wert, der allerdings von einem Lösungsalgorithmus bestimmt werden soll. Eine typische Deklaration einer Unbekannten ist

```
VARIABLE x{j} UNIT [Anzahl] "Kaufmenge eines Anlagepapiers";
```

Auch hier ist die Unbekannte x für jede Anlage deklariert und kann daher auch als eine Tabelle aufgefasst werden. Nichts hindert den Modellierer daran, den Unbekannten auch Werte zuzuweisen, wie das für Parameter möglich ist. Diese werden, zum Beispiel für nicht-lineare Solver, dazu verwendet, eine Startposition zur Suche anzugeben. Dadurch hat die Unbekannte einen bestimmten Wert, bis der Lösungsalgorithmus diesen Wert unter Umständen überschreibt. Auch die Werte der Unbekannten können durch Restriktionen eingeschränkt werden. Geläufig sind für Unbekannte eine obere oder untere Schranke. Häufig findet man auch die Beschränkung, dass eine Variable nur ganzzahlige Werte annehmen darf. Die Deklaration

```
BINARY VARIABLE d{j};
```

bestimmt, dass alle Variablen d nur den Wert Null oder Eins annehmen dürfen. Wird das reservierte Wort BINARY durch INTEGER ersetzt, so darf d nur ganzzahlige Werte annehmen. Der LPL Compiler übergibt diese Informationen für lineare Modelle dem LP/MIP Lösungsalgorithmus über die BOUND-Section und über INT-MARKERS in der COLUMN-Section im MPS-Code, ein Standard-Inputformat für lineare Modelle (ein gemischt ganzzahliges Modell).

3.4. DIE BESCHRÄNKUNGEN

Die Beschränkungen werden durch die CONSTRAINT Anweisung definiert. Jede Restriktion beginnt mit einem Namen, die Zielfunktion(en) eingeschlossen. Gefolgt wird der Name von einem Doppelpunkt und dem Ausdruck, welcher die Beschränkung definiert. Die Restriktionen können gleich wie die Parameter oder die Unbekannten über mehrere Indize laufen. Dadurch wird nicht nur eine einzige, sondern eine ganze Menge von Restriktionen definiert. Die Restriktion

```
CONSTRAINT C{j} UNIT [Anzahl] : q*d <= x <= Q*d;
```

ist definiert für jedes Element der Menge j . Beliebige algebraische oder logische Ausdrücke sind erlaubt. Indexierte Summen werden mit dem reservierten Wort SUM eingeleitet. Der Ausdruck

```
... SUM{j} c[j]*x[j] ...
```

summiert die Multiplikation $c*x$ der einzelnen Anlagepapiere auf und modelliert den mathematischen Ausdruck $\sum_j c_j \cdot x_j$. Die Indexe nach c und x können bei

Unzweideutigkeiten sogar weggelassen werden, was bei einfachen Ausdrücken wie in diesem Falle die Lesbarkeit erhöht, wie folgt :

```
... SUM{j} c*x ...
```

Die Zielfunktion wird genau gleich definiert wie die Beschränkungen, mit der Ausnahme, dass die Definition mit dem reservierten Wort MINIMIZE (bzw. MAXIMIZE) eingeleitet wird. Eine solche Instruktion instanziiert das Modell, schickt es einem vorkonfigurierten Solver (Lösungsalgorithmus) und holt das Resultat zurück in die interne Datenspeicherung von LPL.

Beschränkungen können auch inaktiviert, und Variablenwerte können fixiert werden, indem das reservierte Wort FREEZE der Definition hinzugefügt wird :

```
VARIABLE x FREEZE := 10;
CONSTRAINT R FREEZE: a*x + b*y <= 19;
```

Inaktive Beschränkungen sind unwirksam und werden nicht zum Solver geschickt. Fixierte Variablen werden wie Parameter behandelt. Natürlich machen inaktive Beschränkungen oder fixierte Variablen nur Sinn, wenn sie im Verlaufe der Abarbeitung eines Modells auch wieder aktiviert werden können (zum Beispiel für eine zweite Optimierung). Dies geschieht durch das reservierte Wort UNFREEZE. Das Modell *library.lpl* in der LPL Modellbibliothek gibt ein Beispiel.

3.5. DER LÖSUNGALGORITHMUS

LPL enthält einen einfachen LP-Lösungsalgorithmus (Simplex), um kleinere Modelle (Modellmatrix: 200x200) zu lösen. Für echte und grosse Probleme sollten jedoch kommerzielle Solver verwendet werden. In LPL können solche Solver vorkonfiguriert und dann automatisch aufgerufen werden. Die Konfiguration wird ausführlich im Manual [Hürlimann 2001] beschrieben. Verschiedene Solver sind in der Datei *LPLCFG.LPL* vordefiniert. CPLEX kann auch als dynamische Linkbibliothek (DLL) aufgerufen werden. In diesem Falle werden die Daten zwischen LPL und CPLEX intern im Speicher effizient übergeben. LPL enthält auch einen heuristischen Solver (Tabu Search), welcher für gewisse Reihenfolgeprobleme (scheduling problems) gute Lösungen liefert. Für andere nicht-lineare Modelle muss der Benutzer ein Interface schreiben.

3.6. RESULTATE

Mit LPL kann nicht nur ein Modell formuliert, sondern es können auch die entsprechenden Resultatetabellen produziert werden. Das reservierte Wort WRITE leitet die Tabellen-Generierung ein. Die einfachsten Tabellen liefert LPL, indem WRITE gefolgt wird von den Namen der Tabellen, die der Modellierer ausgeben möchte, wie z.B. im Portfolio-Modell :

```
WRITE Invest, x, s, d;
```

Diese Instruktion produziert vier Tabellen in einem vordefinierten Format (Abb. 2-6).

Die Tabellen-Generator wird in §6 ausführlicher besprochen.

3.7. MASSEINHEITEN

Numerische Daten werden meistens in einer Masseinheit (Meter, Kilogramm usw.) angegeben. In LPL kann die Deklaration jeder numerischen Entität erweitert werden mit der Angabe der Masseinheit. Die Masseinheiten kann der Modellierer selber definieren. Dies erhöht die Lesbarkeit eines Modells. Ausserdem erlaubt dies, Ausdrücke automatisch auf Masseinheitsverträglichkeit hin zu überprüfen. Eine fehlerhafte Formel kann dadurch vom Compiler besser entdeckt werden.

Masseinheiten werden in der UNIT Anweisung deklariert, welche durch das reservierte Wort UNIT eingeleitet wird. Basismasse (wie 'Meter') werden durch den blossen Namen deklariert. Bei abgeleiteten Masseinheiten muss der entsprechende Ausdruck angegeben werden :

```
UNIT
  meter;           (* Basismass der Länge *)
  kilo = 1000;    (* Mass in 1000 Einheiten *)
  km = kilo*meter; (* ein von Meter abgeleitetes Mass *)
  cm = m/100;     (* ein weiteres von Meter abgeleitetes Mass *)
  speed = meter/sec; (* Mass der Geschwindigkeit *)
```

Die Verwendung der Masseinheiten in LPL ist einfach: die Deklaration muss um die Masseinheitsangabe erweitert werden. Eine Zahl innerhalb eines Ausdrucks wird von der Angabe [*Masseinheit*] gefolgt. Auch der Tabellen-Generator akzeptiert Masseinheiten. So möchte man beispielsweise die Tabelle *Invest* in 1000 sFR statt in sFR ausgeben. Folgende Anweisung führt dies aus:

```
WRITE invest UNIT [1000*sFR];
```

Zu bemerken bleibt, dass die Angabe der Masseinheiten in LPL nicht obligatorisch ist.

3.8. ALGORITHMISCHER TEIL

Ein LPL Modell besteht entweder aus einem rein deklarativen Teil oder einem rein algorithmischen Teil oder aus beiden. Die Uebersichtsstruktur ist :

```
MODEL <ModelHeader>
  <deklarativer Teil>
BEGIN
  <algorithmischer Teil>
END
```

Alle drei Modellbeispiele in diesem Paper bestehen nur aus einem deklarativen Teil. Ein Modellbeispiel, welches auch einen algorithmischen Teil enthält, ist *library.lpl*. Der algorithmische Teil wird als eine Sequenz von Instruktionen abgearbeitet wie in einer imperativen Programmiersprache. Der algorithmische Teil kann aus folgenden Instruktionsblöcken gebildet werden:

- Schleifen (FOR, WHILE),
- Bedingte Ausführung (IF ... THEN ... ELSE ...),

- FREEZE/UNFREEZE Instruktion (um Variablenwerte zu (ent)fixieren oder Beschränkungen zu (des)aktivieren),
- EMPTY Instruktion (um Daten zu löschen),
- MINIMIZE/MAXIMIZE/WRITE/READ Instruktionen,
- Zuweisungsinstruktion (um Mengen und Daten zuzuweisen),
- Modell-Aufruf (um ein anderes Sub-Modell auszuführen).

Damit wurden einzelne Elemente der LPL Modelliersprache kurz vorgestellt. LPL besitzt noch eine ganze Reihe weiterer Besonderheiten, z.B. logische Ausdrücke, auf die hier nicht näher eingegangen werden kann. Einige grundlegenden Aspekte der LPL Sprache sollen nun eingehender behandelt werden. Beispiele des Soccer-Modells werden eingestreut.

4. DER GEBRAUCH VON INDEXMENGEN

Die Indexmengen werden dazu verwendet, mehr-dimensionale (indexierte) Objekte wie Parameter, Unbekannte, Restriktionen oder indexierte Mengen zu definieren.

```
PARAMETER a{i,j} := 1;
VARIABLE x{i,j,k,l};
CONSTRAINT r{i} : SUM{j} x[i,j] = 1;
SET s{i,j};
```

a deklariert eine zwei-dimensionale, numerische Tabelle und weist jedem Element den Wert 1 zu, x ist eine vier-dimensionale Unbekannte, r ein Restriktions-Vektor und s eine zwei-dimensionale Indexmenge. In derselben Weise werden die Indexmengen für verschiedene *Indexoperatoren* (SUM, PROD,...) verwendet.

```
....SUM{i,j} a[i,j]...
....PROD{i,j,k,l} x[i,j,k,l]...
```

Im ersten Ausdruck werden alle numerischen Werte der Tabelle $a[i,j]$ addiert, im zweiten wird das Produkt aus allen $x[i,j,k,l]$ Werten gebildet.

In diesen beiden Verwendungsarten von Mengen spielen Indexnamen eine *aktive* Rolle. Im Ausdruck *PARAMETER a{i,j}...* bestimmen die Indexmengen i und j die Grösse der Tabelle a , im Ausdruck *SUM{i,j} a* bestimmen die Mengen i und j , über wieviele und welche Elemente die Summation ausgedehnt wird. Die aktive Rolle besteht darin, dass in einem Ausdruck wie

```
PARAMETER a{i,j} := 1;
```

ein Mechanismus in Gang gesetzt wird, welcher der gesamten Tabelle a – d.h. jeder (i,j)-Kombination von a – die Zahl 1 zuweist. Dieser Vorgang könnte in den imperativen Programmiersprachen mit einer zweifach-geschachtelten Schlaufe (Loop) verglichen werden. Da es sich bei LPL um eine deklarative Modelliersprache handelt,

ist diese Betrachtungsweise jedoch nicht ganz korrekt. Eher entspricht diese Anweisung der Idee, dass alle Elemente der Matrix a Eins sind.

Indexmengen werden aber auch in algebraischen Ausdrücken verwendet (in Ausdrücken wie $[...]$), wo sie eine rein *passive* Rolle spielen. Eine Anwendung findet sich im obigen Beispiel in eckigen Klammern. Diese Indexe sollen *passiv* genannt werden, im Gegensatz zu den ersteren, die *aktiv* heissen. Im Beispiel

```
PARAMETER a{i,j} = b[i,j] + 1;
```

wird jedem Tabellenwert von a der entsprechende Wert plus Eins der Tabelle b zugewiesen. Dabei spielen die Indizes in $[i,j]$ eine rein passive Rolle. Jeder passive Index muss an einen aktiven Index gebunden werden können. Im letzten Beispiel wird der Index i in $[i,j]$ an i in $\{i,j\}$ gebunden; Analoges gilt für j . Die Bindung gewährleistet eine eindeutige Auswertung der Ausdrucks. In der Zuweisung

```
PARAMETER a{i,i,j} = b[i,j]; (* kein Fehler, aber ... *)
```

kann eine Bindung nicht eindeutig erfolgen. Dies führt zwar nicht zu einem Fehler, es kann aber nicht entschieden werden, an welches i in $\{i,i,j\}$ das i in $[i,j]$ gebunden werden soll.

LPL ist relativ flexibel in der Bindung. Es ist jedoch wichtig, dass der bindende und der gebundene Index dieselbe Menge bezeichnen müssen. Die Zuweisung

```
PARAMETER a{i,j} = b[i,k]; (* Fehler! *)
```

produziert einen Fehler, da k nicht gebunden werden kann. Die Instruktion muss ersetzt werden durch

```
PARAMETER a{i,j} = b[i,j IN k]; (* korrekt *)
```

j IN k gibt die Position von einem spezifischen Element der Menge j innerhalb der Menge k zurück. (k wird in diesem Fall als eine *geordnete* Menge betrachtet). Wenn ein bestimmtes Element j in der Menge k nicht vorhanden ist, so ist der ganze Ausdruck $b[i,j$ IN $k]$ Null. Ist die Schnittmenge von j und k jedoch eine echte Teilmenge von k , so wird die Tabelle a durch diese Zuweisung nur partiell aufgefüllt, der Rest bleibt unberührt.

Im allgemeinen führt der LPL-Compiler die Indexbindungen automatisch durch. Der Modellierer kann aber jederzeit eine Bindung durch Hilfsindexe erzwingen. Dies geschieht dadurch, dass dem aktiven Index ein Hilfsindex zugeordnet wird. Dabei geht der Hilfsindex dem aktiven Index voraus und wird von diesem durch ein Gleichheitszeichen oder das reservierte Wort IN getrennt.

```
PARAMETER a{h1=i,h2=j} = b[h1,h2];
PARAMETER a{h1 IN i,h2 IN j} = b[h1,h2]; (* ist dasselbe *)
```

Die Hilfsindexe $h1$ und $h2$ können dann an der Stelle der passiven, zu bindenden

Indexe stehen. Dadurch wird die Bindung von $h1$ an i und von $h2$ an j gewährleistet. Im LPL kann jedem aktiven Index ein Hilfsindex zugeordnet werden. Dieselben Hilfsindexe können in verschiedenen Anweisungen immer wieder verwendet werden, da ein Hilfsindex nur lokale Wirkung hat, vergleichbar einer lokalen Variablen in einer Programmiersprache. Dies kann die Lesbarkeit eines Ausdrucks verbessern und ist für die Wartung eines Modells von Vorteil. In vielen einfacheren Ausdrücken sind Hilfsindexe jedoch nicht nötig. Die 'LPL-Philosophie' geht sogar soweit, dass alle passiven Indexe, die eindeutig gebunden werden können, weggelassen werden können. Diese Praxis steht dem Modellierer jedoch völlig frei. Er kann den LPL-Compiler sogar zwingen, einen solch laschen Umgang mit Indexen nicht zu tolerieren. Detaillierte Angaben dazu finden sich im Manual [siehe Hürlimann 2001].

Indizes können auch in Ausdrücken vorkommen. In diesem Falle wird eine ganze positive Zahl zurückgegeben, welche die Position eines Elementes innerhalb der Indexmenge ist. Der Ausdruck beispielsweise

```
PARAMETER a{i} = i;
```

weist der Tabelle a die Werte 1, 2, 3, ... und n zu, wenn n die Anzahl der Elemente der Indexmenge i ist. Wird der Index jedoch in einer WRITE Anweisung verwendet wie in:

```
WRITE{i} : i , a[i];
```

in welcher i und a als Vektoren ausgegeben werden, so wird nicht die Position von i , sondern der Name des i -ten Elements ausgegeben.

5. ARITHMETISCHE UND LOGISCHE AUSDRÜCKE

Die arithmetischen und logischen Ausdrücke sind ein wichtiger Bestandteil eines jeden LPL Modells. Sie werden dazu verwendet,

- Restriktionen zu definieren
- numerische und logische Ausdrücke zu evaluieren und auszugeben
- Tabellen durch Bedingungen einzugrenzen.

Die Ausdrücke werden in einer dem Mathematiker gewohnten Art aus arithmetischen (+ - * / ^) und logischen (*and*, *or*, \sim) Operatoren, Parameter, Zahlen und Funktoren (Tabelle 5-1) gebildet. Ausdrücke können auch Indexoperatoren (Tabelle 5-3) enthalten, die über Indexmengen iteriert werden. Ein Beispiel aus der Arithmetik ist der Summenoperator Σ , der in LPL durch das Wort SUM wiedergegeben wird. LPL unterscheidet – wie die Programmiersprache C – nicht zwischen arithmetischen und logischen Ausdrücken. In logischen Ausdrücken wird die Zahl 0 als FALSE und jede

andere Zahl als TRUE interpretiert. Tabelle 5-1 gibt eine Übersicht der in LPL erlaubten Operatoren mit absteigendem Vorrang. Der Operatorvorrang kann in einem Ausdruck durch die runden Klammern verändert werden. Ein Ausdruck gibt gewöhnlich einen numerischen Wert zurück. Ausdrücke, die Strings generieren, sind allerdings auch möglich.

Werden Parameter oder Unbekannte in einem Ausdruck verwendet, so wird der entsprechende Tabelleneintrag gesucht und in den Ausdruck eingebunden. Ist dieser Wert noch nicht definiert, so wird der Basiswert genommen. Dieser ist Null, wenn er vom Modellierer nicht explizit definiert wurde. Der Modellierer kann aber den Basiswert selber vorgeben. Dies geschieht durch das reservierte Wort DEFAULT:

```
PARAMETER a{i} DEFAULT 2;
```

Für nicht definierte Werte erhält die Tabelle *a* jetzt den Wert 2.

Beispiele von Ausdrücken sind

```
4+7^9 - SIN(879.8)
(((4+5)*4)^2 - 12 ) + IF(a>0,3,7)
a OR b AND NOT (3*4)
SUM{i} (a[i]+b[i])/2
PROD{i} x[i,j]
```

Der letzte Ausdruck besitzt noch einen ungebundenen Index *j*, daher liefert dieser Ausdruck nicht einen *einzelnen* Wert sondern eine eindimensionale Tabelle über *j*. Der Index kann dadurch gebunden werden, indem der Ausdruck einer Tabelle *b{j}* zugewiesen oder durch einen weiteren Indexoperator gebunden wird:

```
PARAMETER b{j} = PROD{i} x[i,j];
SUM{j} (PROD{i} x[i,j]);
```

Es ist wichtig zu sehen, dass LPL nicht nur Ausdrücke auswertet, die einen *einzigsten* Wert zurückgeben, sondern Listen von Ausdrücken zurückgeben kann. Insofern ist LPL mehr ein Tabellenmanipulator als eine Modelliersprache.

SIN LOG ...	Funktoren
+ - ~ # IN	einstellige Operatoren
^	Potenzieren
* / %	Multiplikation, Division, Modulo
SUM PROD,...	Indexoperatoren
+ -	Addition, Subtraktion
= <> <> <= >=	relationale Vergleichsoperatoren
AND NAND	logischer Und-Operator
OR NOR	logischer Oder-Operator
XOR	exklusives Oder
, (oder)	Aufzählungsoperator
:=	Zuweisungsoperator

Tabelle 5-1 : Operatoren in LPL

ABS(x)	Absolutwert von x
CEIL(x)	nächst grössere ganze Zahl
FLOOR(x)	nächst kleinere ganze Zahl
TRUNC(x)	löscht die Dezimalstellen
SIN(x)	sinus von x
COS(x)	cosinus von x
LOG(x)	nat. logarithmus von x
SQRT(x)	Wurzel von x
ARCTAN(x)	der Arc-tangens von x
RND(x,y)	uniform verteilte Zufallszahl zwischen x und y
RNDN(x,y)	normal verteilte Zufallszahl
IF(x,y,z)	liefert y, wenn x TRUE ist, sonst z

Tabelle 5-2 : Funktionen in LPL

SUM{i} a[i]	summiert alle a über i
PROD{i} a[i]	multipliziert alle a über i
MIN{i} a[i]	das kleinste aller a[i]
MAX{i} a[i]	das grösste aller a[i]
EXIST{i} a[i]	testet, ob ein a[i] TRUE ist
FORALL{i} a[i]	testet, ob alle a[i] TRUE sind
XOR ...	weitere logische Operatoren
PMIN{i} a[i]	Position mit dem min. Wert
PMAX{i} a[i]	Position mit dem max. Wert
COL{i} ...	horizontaler Tabellen-Expander
ROW{i} ...	vertikaler Tabellen-Expander

Tabelle 5-3 : Index-Operatoren in LPL

5.1. BESCHRÄNKUNGEN

Ausdrücke erlauben es, Beschränkungen zu definieren. Die Syntax von Restriktionen ist kaum eingeschränkt: eine Restriktion kann in Bezug auf die definierten Unbekannten linear oder nicht-linear sein. Sogar logische Operatoren sind erlaubt. LPL enthält einen komplizierten Algorithmus, der logische Beschränkungen in lineare Beschränkungen, die 0–1 Variablen enthalten, standardmässig umwandelt. Restriktionen dürfen auch als Bereich (range) definiert werden. Dieselbe Unbekannte kann mehrmals im Ausdruck verwendet werden. Unbekannte mit einem Parameter Null werden automatisch aus der Restriktion eliminiert. Beschränkungen, die nur eine (lineare) Variable enthalten, werden automatisch in entsprechende Bounds umgewandelt.

```

CONSTRAINT R: x + y = SUM{i} z[i]  (* definiert eine Restriktion R *)
CONSTRAINT B: a <= x+y <= b;      (* definiert einen Bereich (range) *)
CONSTRAINT L{i}: x[i] + y = 5;    (* definiert Restriktionen über i *)
CONSTRAINT S: x + 3*y = 4*(x+y);  (* x und y kommen zweimal vor *)
CONSTRAINT T: x + z -y = (4-4)*x +x

```

Die zweitletzte Restriktion (S) liefert automatisch den Ausdruck $-3*x - y = 0$ und in der letzten (T) wird die Unbekannte x eliminiert.

5.2. TABELLEN

Ausdrücke werden auch dazu verwendet, den Tabellen Werte zuzuweisen und Zwischenberechnungen anzustellen oder auszudrucken.

```
PARAMETER a{i,j} = b{j,i};
PARAMETER d{i,j} = a[i,j] + b[i,j];
CONSTRAINT FREEZE e{i,k} = SUM{j} a[i,j]*c[j,k];
WRITE{i,j} : SUM{k} c[j,k] + a[i,j];
```

Der erste Ausdruck kopiert die transponierte Tabelle b in die Tabelle a . Der zweite Ausdruck weist die Matrixaddition von a und b der Tabelle d zu. Der dritte Ausdruck definiert eine Matrixmultiplikation, führt diese aber nicht unmittelbar aus. Erst wenn e seinerseits in einem Ausdruck verwendet wird, wird der Ausdruck auf der rechten Seite evaluiert (delayed evaluation). Der vierte Ausdruck gibt die Berechnung als zweidimensionale Tabelle aus.

5.3. DÜNN-BESETZTE TABELLEN

Eine weitere wichtige Anwendung von Ausdrücken ist das Eingrenzen von indexierten Tabellen durch Bedingungen. Dies soll an einem Beispiel erläutert werden. Angenommen, in einem Transportmodell seien folgende Komponenten vorhanden:

```
SET i ALIAS j; (* eine Liste von geographischen Orten *)
SET Verbindungen{i,j}; (* alle Strassenverbindungen von i nach j *)
PARAMETER Kosten{i,j}; (* die Transportkosten auf den Verbindungen *)
VARIABLE x{i,j}; (* die unbekanntes Transportmengen *)
```

Beachte zunächst, dass die Indexmenge i zwei verschiedene Namen (i und j) hat. Dies kann durch die Option ALIAS erwirkt werden. Die indexierte Indexmenge *Verbindungen* definiert, welche Transportverbindungen zwischen den einzelnen Orten i (und j) existiert. Nicht jede mögliche Verbindung ist auch sinnvoll. Die Liste der Tupel *Verbindungen* wird also in der Praxis aus einer Teilliste aller möglichen Verbindungen bestehen. Dies hat aber Folgen für die weiteren Modellkomponenten. Sicher ist *Kosten* auch nur sinnvoll für Verbindungen, die tatsächlich existieren. Nehmen wir weiter an, dass eine Menge nur auf Verbindungen verschoben werden soll, deren Kosten nicht über 1000 liegen. Dann ist eine Transportmenge x offenbar nur sinnvoll auf tatsächlich bestehenden Verbindungen, deren Kosten 1000 nicht übersteigen. Für alle andern Verbindungen macht eine Mengenangabe keinen Sinn. LPL besitzt einen Mechanismus, solche dünn besetzten Tabellen zu definieren als:

```
PARAMETER Kosten{i,j | Verbindungen[i,j]};
VARIABLE x{i,j | Verbindungen[i,j] and Kosten[i,j]<=1000};
```

-- oder besser und kürzer:

```
PARAMETER Kosten{Verbindungen};
VARIABLE x{Verbindungen | Kosten<=1000};
```

Wenn die Bedingung - eingeleitet durch '|' - für eine bestimmte Kombination der Verbindungswege TRUE zurückgibt, dann ist die Kombination definiert, sonst nicht. Unbekannte, die diese Bedingung nicht erfüllen, werden in der Folge gar nicht mehr produziert. So liefert der Ausdruck

```
SUM{i,j} x[i,j]
```

beispielsweise nicht die Summe *aller möglichen* $x[i,j]$ -Verbindungen, sondern die Summe nur derjenigen, welche bei der Definition durch die Bedingung eingegrenzt worden sind. Dieselbe Syntax kann auch für die Iteration von Indexoperatoren verwendet werden. Sollen von einer Tabelle a nur die ersten 5 Werte aufsummiert werden, so könnte dies folgendermassen definiert werden:

```
SUM{i | i<=5} a[i];
```

Ein weiteres Beispiel ist die Restriktion *Must* im Modell *soccer.lpl*.

```
Must{p,t | MustBeIn}: Work = 1;
```

Diese Restriktion würde ohne die Bedingung *MustBeIn* ganze 2700 (=180x15) einzelne Restriktionen produzieren, generiert aber mit diesen Daten nur gerade drei, nämlich

```
Must1: Work[1,T2] = 1
Must2: Work[2,T6] = 1
Must3: Work[34,T7] = 1
```

da *MustBeIn* als die Menge bestehend aus den drei Tupeln definiert ist:

```
SET MustBeIn{p,t} = / 1 T2 , 2 T6 , 34 T7 /;
```

Noch besser ist die im Modell verwendete Methode, um diese Restriktion zu definieren:

```
Must{i=MustBeIn} : work[i] = 1;
```

welche nicht über die gesamte Tupelmenge $\{p,t\}$ definiert ist und die meisten (2697 von 2700) Restriktionen dann doch nicht generiert. Diese letztere Formulierung geht nur über drei Restriktionen, die alle generiert werden. Der Geschwindigkeitsunterschied in der Generierung ist erheblich.

6. DER TABELLEN- UND DATENEINGABE-GENERATOR

Die WRITE-Anweisung erlaubt, Tabellen auszugeben. Diese werden bei der Compilation eines Modells in eine Datei geschrieben, die vom Modellierer mit seinem bevorzugten Textverarbeitungssystem weiter verarbeitet werden kann. Einfache Beispiele haben wir bereits gesehen, wobei LPL über das Format des Ausdrucks

entscheidet.

```
WRITE a;                (* druckt die Tabelle a im Standardformat aus *)
WRITE b '%2';          (* Tabelle b, mit 2-stelligen Werten *)
WRITE c '%10.2';      (* Tabelle c, 10-stellig mit 2 Dezimalstellen *)
WRITE a, b, c;        (* alle drei Tabellen drucken *)
WRITE a '%6:10:10';   (* Tabelle wird geschrieben in 10x10 Blöcke *)
```

6.1. TABELLEN VON AUSDRÜCKEN

Auch Tabellen von beliebigen Ausdrücke können ausgegeben werden. In diesem Falle wird das reservierte Wort `WRITE` durch einen Doppelpunkt vom auszugebenden Ausdruck getrennt. Auch bei dieser `WRITE`-Anweisung wird das Ausgabeformat vom LPL-Compiler selber gesteuert.

```
WRITE '%12': a*b;      (* ein Wert 12stellig ausgeben *)
WRITE{j,i} : a[i,j];  (* transponierte Matrix a *)
WRITE      : 'dieser Text'; (* eine Textzeile ausgeben *)
WRITE{j}   : SUM{i} a[i,j]; (* die Kolonnen totale von a ausgeben *)
```

Bei indexierten Ausdrücken muss das reservierte Wort `WRITE` von einer Indexliste gefolgt werden.

6.2. TABELLEN MIT MASKENFORMATEN

Für komplexere Resultate-Tabellen kann der Modellierer mit Hilfe einer Masken-Anweisung das Ausgabeformat vorgeben. Die Maske wird in der `WRITE`-Anweisung zwischen Hochkommata gesetzt. Der Maskeninhalte besteht aus beliebigem Text und wird vom Tabellen-Generator übernommen. Die beiden Zeichen `#` und `$` innerhalb des Maskeninhalts werden als Platzhalter bezeichnet und haben eine spezielle Bedeutung. Sie werden von einer `WRITE`-Anweisung mit numerischen und alphanumerischen Daten aufgefüllt.

Beispiel:

```
WRITE
" Monat: $$$$$$$$ : ####.##%  ####.#### Gramm\n"
: 'April' , 17.15*2 , 2378.567321 ;
```

Diese Anweisungen produzieren folgende Zeile

```
Monat:   April   :   34.30%   2378.5673 Gramm
```

Die `WRITE` Anweisung besteht aus dem reservierten Wort `WRITE` gefolgt von der Maske in Hochkommata, gefolgt von einem Doppelpunkt und einer Ausdrucksliste. Die Ausdrücke werden durch Kommata voneinander getrennt. Diese werden von links nach rechts an Stelle der Platzhalter aufgefüllt.

```

RESULTS PER PLAYER
*****
player Skill Age in Team
-----
1          3  10   T2
2          7  10   T6
3          4  10   T6
4          4  10   T1
5          6  11   T15
6          4  11   T6
7          3  10   T8
..... lines have been cut here .....
171         6  10   T8
172         3  11   T3
173         6  10   T5
174         3  10   T4
175         4  11   T10
176         7  11   T2
177         5  10   T9
178         4  11   T15
179         5  11   T9
180         7  11   T5

RESULTS PER TEAM
*****
team  skill  age  Av.Skill  Av.Age
-----
T1      60  127   5.000   10.583
T2      59  127   4.917   10.583
T3      59  125   4.917   10.417
T4      59  126   4.917   10.500
T5      59  124   4.917   10.333
T6      59  124   4.917   10.333
T7      59  126   4.917   10.500
T8      59  126   4.917   10.500
T9      59  127   4.917   10.583
T10     59  129   4.917   10.750
T11     59  125   4.917   10.417
T12     59  125   4.917   10.417
T13     59  126   4.917   10.500
T14     59  124   4.917   10.333
T15     59  126   4.917   10.500

team | players in the team
-----
T1 | 4  19  25  35  47  74  81  98  120  151  156  160
T2 | 1  41  63  64  71  80  84  92  131  138  166  176
T3 | 13  21  27  31  37  39  43  82  95  113  141  172
T4 | 15  36  46  59  60  72  83  128  134  147  154  174
T5 | 12  16  20  45  87  94  101  105  137  164  173  180
T6 | 2  3  6  8  17  56  110  119  130  136  146  148
T7 | 34  48  70  75  85  102  103  121  135  142  145  163
T8 | 7  77  86  100  117  122  144  153  159  167  169  171
T9 | 49  55  68  78  111  123  132  155  165  168  177  179
T10 | 33  76  90  91  106  107  112  124  125  162  170  175
T11 | 30  32  57  62  66  79  93  96  116  139  143  157
T12 | 26  38  42  50  51  53  65  73  108  127  133  150
T13 | 10  28  40  52  67  88  104  126  129  140  149  158
T14 | 11  18  23  54  58  61  97  109  114  115  118  161
T15 | 5  9  14  22  24  29  44  69  89  99  152  178

```

Abbildung 6-1 : Ausgabe und Resultat des Soccer-Modells

Doch erst durch die beiden Indexoperatoren ROW und COL wird der Tabellen-

Generator zu einem mächtigen Instrument. *Mit Hilfe des Indexoperator ROW kann ein Platzhalter über Indexmengen vertikal und mit COL horizontal ausgedehnt werden.*

Die Kombination beider Indexoperatoren erlaubt es dem Modellierer, komplexe Tabellen auf einfache Weise in zweidimensionaler Form zu produzieren. Die Syntax von ROW und COL ist genau dieselbe wie für andere Indexoperatoren. Abbildung 6-1 zeigt das Resultat der WRITE-Anweisung des Modells *soccer.lpl* (Abb. 2-8).

Die Maske (in Abbildung 2-8) besteht aus 11 Platzhaltern. Die ersten vier befinden sich auf Zeile 5 der Maskendefinition, welche hier wiederholt wird:

```
$$$$$ ## ## $$$
```

Diese werden aufgefüllt durch den Ausdruck

```
(p , skill , age , COL{t|work} t)
```

in der WRITE Anweisung: An die Stelle des ersten Platzhalter tritt der Spielername *p*. Es folgt der Geschicklichkeitsgrad *skill* und das Alter *age* als numerisches Datum. Durch die Instruktion *COL{t|work} t* wird angegeben, dass horizontal alle Gruppen aufgeführt werden sollen, in welche der Spieler *p* eingeteilt ist. Diese Liste ist natürlich stets auf einen Spieler reduziert, da ein Spieler gemäss der Modellrestriktion *Bounds* nur in einer Gruppe sein darf. Da in der WRITE Anweisung der ganze Ausdruck zudem über *ROW{p}* definiert ist, wird diese Zeile für alle Spieler wiederholt, was dazu führt, dass 180 Zeilen produziert werden.

Die nächsten fünf Platzhalter sind in Zeile 10 der Maske wie folgt definiert:

```
$$$$$ ##### ### ###.### ###.###\n
```

Diese werden vom Ausdruck

```
ROW{t} (t , SUM{p|work} Skill , SUM{p|work} Age ,  
(SUM{p|work} Skill)/12 , (SUM{p|work} Age)/12 ) ,
```

in der WRITE Anweisung aufgefüllt. Der erste Platzhalter wird durch *t*, dem Gruppennamen, ersetzt, der nächste durch die aufaddierten Gruppengeschicklichkeit, welche durch den Ausdruck *SUM{p|work} skill* berechnet wird. Die weiteren werden ähnlich berechnet.

Die letzten zwei Platzhalter befinden sich auf Zeile 13 (der letzten Maskenzeile) :

```
$$$ | $$$\n"
```

Der zugehörige Ausdruck ist

```
ROW{t} (t , COL{p|work} p);
```

Diese Zeile produziert eine zweidimensionale Tabelle: die Spielernamen pro Gruppe

werden auf einer Zeile zusammengefasst. Man beachte, dass die Iteration der beiden Indexoperatoren durch eine Bedingung eingeschränkt werden kann. So wird in $COL\{t/work\} p$ die Kolonnenausdehnung über t durch die Bedingung $Work$ eingeschränkt. $work$ ist genau dann TRUE, wenn es nicht Null ist. Damit ist die obige Ausgabe vollständig beschrieben.

6.3. DER DATENEINGABE-GENERATOR

Daten müssen nicht in LPL-Form vorliegen. Sie können durch den Dateneingabe-Generator, in LPL repräsentiert durch die READ Anweisung, eingelesen werden. Die READ Anweisung ist ähnlich wie die WRITE Anweisung aufgebaut. Als Beispiel soll das Portfolio-Modell gewählt werden, bei welchem die Daten in Form einer Textdatei vorliegen (Abbildung 2-5). Die Daten werden von dieser Datei durch folgende Instruktionen eingelesen (Abbildung 2-4):

```
READ FROM 'portfol.dat' ':Table:Table';
READ '%1' : ROW{j} ( j , c , q , Q );
READ '%2' : ROW{t} ( t , L , a );
READ '%3' : ROW{j} ( j , COL{t} f[j,t] );
```

Die erste READ Anweisung legt den Namen der zu lesenden Daten fest. Ausserdem werden die *Leseblock-Begrenzer*, welche angeben, zwischen welchen Abschnitten innerhalb der Datei gelesen werden soll. Die zweite READ Anweisung platziert den Lesekopf an das erste Vorkommen des Textes 'Table' – beginnend auf einer neuen Zeile innerhalb der Lesedatei, und liest Daten von der Datei bis zum nächsten 'Table' Zeichenkette, beginnend ebenfalls am Beginn einer Zeile. Auf jeder Zeile werden – falls vorhanden – vier Worte (Token) gelesen. $ROW\{\}$ übernimmt zwei Funktionen: Erstens kann dadurch das Zeilenlesen wiederholt werden, zweitens wird das Zeilenlesen synchronisiert. Dies bedeutet folgendes: Falls eine Zeile aus mehr als vier Worten besteht, so werden nur die ersten vier gelesen, falls hingegen eine Zeile aus weniger als vier Worten besteht, so werden weniger gelesen. In jedem Falle wird eine neue Vierergruppe auf einer *neuen* Zeile gelesen. Der Operator $COL\{\}$ hat eine ähnliche Funktion, allerdings werden dadurch Worte auf derselben Zeile gelesen, bis diese zu Ende gelesen ist.

Zu beachten ist bei beiden Indexoperatoren, dass die aktive Indexmenge noch gar nicht vorhanden ist. Die Anweisung:

```
READ '%1' : ROW{j} ( j , c , q , Q );
```

bedeutet also nicht: “lese die vier Worte j , c , q , und Q so oft, wie j Elemente enthält.” Die Menge zum Index j ist ja eben gerade leer und wird durch die READ Anweisung *während* des Lesens generiert. Das Ende des Lesens wird durch das Blockende 'Table' oder das Dateiende in der Textdatei bestimmt.

Es ist ebenfalls möglich, von verschiedenen Dateien zu lesen. Wenn die obige Datei *PortfolD.dat* in drei Dateien entsprechend den einzelnen Tabellen verteilt würde, so wäre die Angabe der Leseblock-Begrenzer überflüssig, da der Dateneingabe-Generator automatisch die ganze Datei liest, wenn keine Block-Begrenzer angegeben werden.

Experimente mit dem Dateneingabe-Generator sind vielversprechend: Für ein konkretes LP-Modell von 1300 Restriktionen und 1500 Variablen, musste ein Pascal Programm von 32 Seiten geschrieben werden, welches die Modelldaten manipulierte und in die richtige Form brachte. Unter Benützung des neuen Dateneingabe-Generators konnte dieses Programm durch einen LPL-Code von 2 Seiten ersetzt werden.

7. ZUSAMMENFASSUNG

Dieser Artikel ist eine notwendigerweise unvollständige Zusammenfassung der Modelliersprache LPL. Das Benutzerhandbuch [Hürlimann 2001] gibt eine detaillierte Beschreibung der Sprache. Es wurden ferner etwa 200 Modellbeispiele in LPL geschrieben, die einen vertieften Einblick geben.

Die Entwicklung von LPL war von Anfang an motiviert durch den praktischen Einsatz von grossen Modellen. Verschiedene LP Modelle mit 8000 Beschränkungen, 20'000 Unbekannten und einer Matrixbesetzung von 70'000 Elementen werden am Departement für Informatik an der Universität Freiburg im Auftrage des Bundes gewartet. Sie sind in LPL formuliert worden. Da diese Modelle noch Mitte der 80-iger Jahren einem Grossrechner zur Lösung übergeben werden mussten, stand für die LPL Sprache die automatische Produktion des MPS-Codes im Vordergrund. Dank der raschen Entwicklung der Personal Computer und den darauf implementierten Lösungsalgorithmen (CPLEX, Xpress, XA, u.a.) ist es heute möglich, alle diese Modelle lokal auf dem PC zu lösen. Damit rückt der Modellierungszyklus (Modell ändern – lösen – Resultate generieren) immer mehr in den Vordergrund. Das Soccer-Modell, welches aus 2700 0–1 Variablen besteht, wurde vor acht Jahren unter MS/DOS mit einer auf einem 80386 Prozessor von Intel basierten PC in etwa 2 Minuten von LPL generiert. XA braucht damals etwa 30 Minuten, um eine optimale Lösung zu finden. Heute wird das Problem auf einem Pentium 4 basierten Computer mit der CPLEX, Version 6.5, unter Windows XP in etwa 5 Sekunden generiert und in 15 Sekunden gelöst. Daraus mag man den immensen Fortschritt, der es erlaubt, grössere Modelle zu lösen, abschätzen. Umso wichtiger werden Modelliertools, die es dem Benutzer erlauben, grosse Modelle zu manipulieren, zu warten und zu dokumentieren.

Die Produktion und Selektion von Resultate-Tabellen wird daher ebenso wichtig wie die Generierung des Eingabecodes für den Lösungsalgorithmus. LPL unterstützt diesen

Zyklus voll, indem es automatisch einen externen Lösungsalgorithmus aufrufen und die Lösung zur weiteren Verarbeitung übernehmen kann. Der eingebaute Tabellengenerator produziert die gewünschten Resultate-Tabellen. Der Modellierer muss sich daher nicht mehr mit den vielen technischen Details befassen, sondern kann sich der eigentlichen Modellierung widmen. Die praktische Erfahrung mit den genannten Modellen hat gezeigt, dass sich der Modellierungszyklus (Modellmodifikationen-Lösen-Resultate) von einigen Stunden, ja sogar Tagen, auf einige Minuten reduziert hat.

Der LPL-Compiler wurde mit Delphi 7 von Inprise Corp. entwickelt und läuft somit unter dem Betriebssystem MS/DOS und Windows 2000/XP. Er kann auch mit freePascal (www.freePascal.com), einem freien Pascal-Compiler kompiliert werden. Die ältere Version 4.34 des LPL-Compilers sowie einer ausführlichen Dokumentation ist erhältlich bei der LPL-Site. Eine beschränkte, neuste Version von LPL kann an derselben Adresse heruntergeladen werden :

www.virtual-optima.com

und ausprobiert werden.

LITERATUR

- BEILBY M.H., MOTT T.H. [1983], Academic Library Acquisitions Allocation based on Multiple Collection Development Goals, *Comput. & Ops. Res.* Vol. 10, No. 4, pp 335-343.
- BISSCHOP J. & ROELOFS M., [1999], AIMMS, the Language Reference, see [www.paragon.nl].
- BROOKE A., KENDRICK D., and MEERAUS A. [1988], GAMS, A User's Guide, The Scientific Press.
- BYER J. [1990], Sunset Software California, personal communication.
- DAY R.E., WILLIAMS H.P., [1986], MAGIC: The Design and Use of an Interactive Modelling Language for Mathematical Programming, *IMA Journal of Mathematics in Management* Vol. 1, No:1, p.53-65.
- FOURER R., GAY D.M., KERNIGHAN B.W., [2003], AMPL, A Modeling Language For Mathematical Programming, Duxbury Press, Belmont CA.
- HÜRLIMANN T., [2003], Reference Manual for the LPL Modeling Language, file *manual.pdf* at the LPL-site.
- HÜRLIMANN T., [2003a], LPL : A mathematical modeling language, file *intro.pdf* at the LPL-site, (englische Version dieses Papers).
- HÜRLIMANN T., [1999], Mathematical Modeling and Optimization, An Essay for the Design of Coputer-Based Modeling Tools, Kluwer Academic Publ., (Applied Optimization 31).
- LPL-Site : HOMEPAGE at: www.virtual-optima.com
- SHAPIRO J.F. [1988], Stochastic Programming Models for Dedicated Portfolio Selection, NATO ASI Series, Vol. F48, Mathematical Models for Decision Support, Edited by G. Mitra, Springer Verlag, Berlin, S.587-611.
- SCHRAGE L., [2000], Optimization Modeling with LINGO, Lindo Systems, Inc., (see WWW at: <http://www.lindo.com>).