Lösungen mit dem MNI Proposition Analyzer MPA

Logeleien, Kartenfärbung, Sudoku, Variabilitätsmodelle

Technischer Bericht – Fachhochschule Gießen-Friedberg

Burkhardt Renz

Fachhochschule Gießen-Friedberg, Fachbereich MNI,
Wiesenstr. 14, D-35390 Gießen
Burkhardt.Renz@mni.fh-giessen.de

20. November 2008

Der MNI Proposition Analyzer mpa ist ein einfaches Werkzeug für die Analyse von Aussagen der Aussagenlogik. Es kann Aussagen in die konjunktive oder disjunktive Normalform transformieren, die Erfüllbarkeit und die Allgemeingültigkeit von Aussagen überprüfen und einiges mehr (siehe [8]).

In diesem Bericht werden einige Beispiele mit dem mpa untersucht, in der Absicht zu demonstrieren, was man mit dem Werkzeug machen kann. Wir beginnen in Abschnitt 1 mit ganz einfachen Logeleien und erweitern unsere Möglichkeiten über Kartenfärbungen (Abschnitt 2) zu einem zweiten Typ von Logeleien (Abschnitt 3). Ein Höhepunkt des Rätsellösens ist das Sudoku in Abschnitt 4. Aber mein eigentliches Interesse an der Aussagenlogik und der Überprüfung der Erfüllbarkeit von Aussagen durch sogenannte *SAT solver* zielt auf Anwendungen in der Softwaretechnik. Im letzten Abschnitt 5 wird gezeigt, wie man Variabilitätsmodelle für Softwareproduktlinien aussagenlogisch betrachten und analysieren kann.

1 Logelei I

Eine typische Logelei, wie man sie etwa in der Wochenzeitschrift „Die Zeit“ findet, sieht so aus:

Anna sagt: „Bettina lügt“.
Bettina sagt: „Claudia lügt“.
Claudia sagt: „Anne und Bettina lügen“.
Wer lügt denn nun?

Diese Aufgabe „kodiert“ man in der Aussagenlogik, in dem man für jede Person eine Boolesche Variable einführt. Ist diese Variable „true“, dann sagt die Person die Wahrheit, ist sie „false“, lügt sie. Aussagen, die eine Person macht, sind logisch äquivalent zum Wahrheitswert der Person.

In unserem Beispiel ergibt sich in der Syntax von mpa mit den Variablen `a` für Anna, `b` für Bettina und `c` für Claudia:

```
(a <-> !b) &
(b <-> !c) &
(c <-> !a&!b)
```

mpa kann nun die Erfüllbarkeit dieser Aussage prüfen. Ist sie erfüllbar, gibt mpa auch eine Bewertung der Variablen aus, unter der die Aussage wahr ist, ein *Modell*.

In mpa kann man zwischen drei SAT-Solvern wählen: „DefaultSAT4J“, „LightSAT4J“ und „NaiveDPLL“.

Die ersten beiden SAT-Solver stammen aus dem Projekt „SAT4J“ [10], das von Daniel Le Berre von der Université d'Artois in Lens geleitet wird. Die SAT4J-Bibliothek ist eine Bibliothek von *effizienten* SAT-Solvern in Java, die auf dem Konzept von „MiniSAT“ von Niklas Eén und Niklas Sörensson [6] basieren.

„NaiveDPLL“ ist eine *naive* Implementierung des DPLL-Algorithmus nach Davis, Putnam, Logemann und Loveland [3], [4]. Diese Implementierung ist Teil des Projekts mpa und dient der Demonstration der Grundidee des DPLL-Algorithmus. Insbesondere will „NaiveDPLL“, wie der Name andeutet, *nicht* effizient sein.

Das Ergebnis der Prüfung der Erfüllbarkeit der obigen Aussage sieht in mpa so aus:

```
The proposition is satisfiable.

A model is:
Valuation

    a       b       c
    F       T       F
```

Eine Lösung des Rätsels ist also:

Nur Bettina sagt die Wahrheit.

Das Ergebnis des SAT-Solvers gibt ein Modell, aber dieses muss selbstverständlich nicht das einzig mögliche sein. Im konkreten Fall unseres Rätsel

können wir mpa dazu verwenden, eine Wahrheitstafel für die Formel auszugeben, an der man ablesen kann, dass die gefundene Lösung tatsächlich die einzige Lösung ist.

Die Berechnung der Wahrheitstafel in mpa ist auf Formeln mit höchstens 16 Variablen beschränkt, denn eine Wahrheitstafel mit mehr als 2^{16} Zeilen dürfte kaum sinnvoll sein. Es gibt SAT-Solver, die alle möglichen Modelle zu einer Formel sukzessive berechnen, SAT4J etwa bietet diese Möglichkeit[1], in mpa ist sie jedoch noch nicht verfügbar.

Weitere Beispiele für diese erste Klasse von Logeleien, die sich direkt in eine Aussage übersetzen lassen, finden sich im Listing im Anhang auf Seite 28.

2 Kartenfärbung

Jetzt wollen wir uns einer spannenderen Frage zuwenden: der Färbung ebener Karten. Bekanntermaßen haben Appel und Haken 1976 den berühmten Vierfarben-Satz bewiesen. Der Beweis war damals insofern ein Novum, und auch etwas umstritten, weil in dem Beweis ein Teil per Computer und nicht durch ein intelligentes Wesen überprüft wurde. Aber das ist eine andere Geschichte. Wir möchten gerne das Problem der Kartenfärbung in der Aussagenlogik „kodieren“ und dann Beispiele von Karten ansehen.

Beginnen wir mit einem Ausschnitt Europas rund um Luxemburg, siehe Abb. 1.

Abbildung 1: Luxemburg und Umgebung

[1] Das Vorgehen ist recht naheliegend: Wird ein Modell gefunden, fügt man eine Klausel mit der Negation der Literale im Modell hinzu und sucht nach einer weiteren Lösung – dies solange, wie die Aussage noch erfüllbar ist.

Es gibt sicherlich Karten, bei denen man mit drei Farben auskommt. Ist das für Luxemburg und Umgebung auch der Fall? Man kann sich leicht überlegen, dass in dieser Situation drei Farben nicht ausreichen. Wenn ein Land von einer ungeraden Anzahl von Ländern umgeben ist, braucht man bereits drei Farben für die umgebenden Länder, insgesamt also vier. Dies ist auch bei Luxemburg der Fall, denn Luxemburg ist von den drei Ländern Frankreich, Belgien und Deutschland umgeben.

Aber wir wollen ja Lösungen mit dem mpa finden. Wie stellt man also die Fragestellung aussagenlogisch dar?

Zunächst definieren wir für jedes Land drei Boolesche Variablen für die drei Farben der Karte. In unserem Beispiel steht lu_r, lu_g, lu_b für: Luxemburg wird rot, grün bzw. blau gefärbt. Da wir 4 Länder und drei Farben haben, ergeben sich 12 Variablen.

Ein Land kann natürlich nur eine Farbe haben, ganz egal, wie die Länder benachbart sind. Diese Eigenschaft kann man so ausdrücken: (1) ein Land kann keine zwei Farben haben und (2) ein Land hat mindestens eine Farbe. Das ergibt in unserem Beispiel für Luxemburg:

$$(1) \quad (\neg lu_r \vee \neg lu_g) \wedge (\neg lu_r \vee \neg lu_b) \wedge (\neg lu_g \vee \neg lu_b)$$

$$(2) \quad lu_r \vee lu_g \vee lu_b$$

Allgemein gilt:

Gegeben n Variablen $\{x_1, x_2, \ldots x_n\}$ $(1 \leq n)$ und ein k mit $0 \leq k \leq n$, dann bedeutet folgende Aussage: Höchstens k der Variablen sind wahr[2]:

$$\bigwedge_{\substack{S \subseteq \{1,\ldots,n\} \\ \text{mit } |S| = k+1}} \bigvee_{i \in S} \neg x_i$$

Für den Fall $k = 1$ ergibt dies

$$(1) \quad \bigwedge_{S = \{i,j\} \subseteq \{1,\ldots,n\}} \neg x_i \vee \neg x_j$$

$$(2) \quad x_1 \vee \cdots \vee x_n$$

Das ergibt $\binom{n}{2} + 1$ Klauseln.

Also hat man pro Land 4 Klauseln, insgesamt 16 Klauseln für die vier beteiligten Ländern – eine ganze Menge zu schreiben.

Man sieht an diesem Beispiel, dass schon recht einfache Fragestellungen recht umfangreiche Aussagen erfordern. Das führt in der Verwendung von

[2] Für $k = n$ ergibt die Formel die leere Aussage. In der Tat: Höchstens n von n Variablen als wahr auszuwählen, ist bei jeder beliebigen Wahl erfüllt, also trivialerweise „true“.

mpa schnell dazu, dass man immer wieder Gleiches in analoger Situation wiederholen müsste.

Damit man es hier leichter hat, hat mpa einen optionalen Makroprozessor: mpa verwendet den MNI Macro Processor mmp (siehe [9]), der in Java geschrieben und kompatibel ist zum „klassischen“ Makroprozessor m4. mmp und m4 sind ein eigenes Thema, eine ausgezeichnete Anleitung für die Verwendung von m4 findet man in [11]. Wir werden hier nur an den Beispielen sehen, wie man mmp innerhalb des mpa verwendet.

Zunächst brauchen wir eine Direktive, um den mmp zu aktivieren. Diese Direktive muss am Beginn der ersten Zeile einer mmp-Eingabe stehen und sie sieht so aus[3]:

```
dnl needs mmp
```

Um in m4 auszudrücken, dass aus einer Menge von Variablen genau eine wahr sein soll, kann man ein Makro definieren, nennen wir es `oneOf`, das man so verwendt:

```
oneOf(‘lu_r’,‘lu_g’,‘lu_b’)
```

Dieses Makro expandiert zur oben angegebenen Aussage, die sicher stellt, dass Luxemburg mit genau einer Farbe gefärbt wird.

Es ist nicht schwer in m4 ein solches Makro für eine feste Anzahl von Argumenten zu schreiben, deutlich schwieriger jedoch für eine beliebige Zahl von Argumenten. Da wir jedoch diese Art von Kardinalitätsbeschränkung in vielen Situationen benötigen, ist es sehr hilfreich ein solches Makro mit variabler Zahl von Argumenten zu haben. Hier kommt uns nun eine spezielle Eigenschaft von mmp zupass. Makros in mmp sind Java-Objekte, die eine bestimmte Methode implementieren müssen. Und in der eigentlichen *Engine* des mmp, die die Transformation durchführt, können solche Java-Objekte registriert werden.[4]

Für mpa habe ich eine kleine Bibliothek `logic.mml` von Makros geschrieben, die speziell Makros für die Aussagenlogik in der Syntax des mpa bereit stellt. Ein solches Makro ist `oneOf`.

Man muss natürlich die Aussage, dass ein Land nur *eine* Farbe haben kann, für jedes der Länder formulieren. Dabei hilft es, wenn man Schleifenkonstrukte wie „for“ oder „foreach“ verwenden kann. Es ist möglich, solche Makros in m4 zu schreiben, wie dies etwa in [11, S. 44ff] beschrieben ist. Für mmp hat man aber wieder die Möglichkeit, diese Makros in Java zu schreiben. Die Bibliothek `util.mml` enthält Makros für Schleifen, die wir einfach durch die Verwendung in mpa kennenlernen.

Zunächst müssen wir die beiden Bibiotheken inkludieren:

[3] Da „dnl“ das Kommentarzeichen für den m4 ist, wird diese Zeile nach der Transformation durch den Makroprozessor nicht mehr vorhanden sein.

[4] Diese Eigenschaft der Erweiterbarkeit ist einer der Gründe, weshalb es mir sinnvoll erschienen ist, m4 in die Sprache Java zu portieren. Dadurch, dass man Makros für mmp in Java schreiben kann, hat man ganz neue Möglichkeiten. Man kann z.B. iterativ vorgehen, wo man im m4 gezwungen ist, rekursiv zu arbeiten.

```
dnl needs mmp
divert(-1)
include(‘util.mml’)
include(‘logic.mml’)
divert
```

Dabei wird eine typische m4-Mimik verwendet: In m4 kann man die Ausgabe in nummerierte Puffer, sogenannte *diversion buffers* umlenken und später ausgeben. Mit `divert(-1)` wird die Ausgabe in einen Puffer mit negativer Nummer umgelenkt, was nichts anderes bedeutet, als dass die Ausgabe „verschluckt“ wird. Mit `divert` wird wieder auf den normalen Ausgabekanal mit der *diversion number* 0 umgeschaltet. So kann man *diversions* verwenden, um eine Ausgabe zu unterdrücken.

In unserem Beispiel registrieren die Include-Dateien die Makros der jeweiligen Bibliotheken beim Makroprozessor mmp. Die Umlenkung der Ausgabe dient nur dazu, dass Kommentare in den Include-Dateien nicht in unsere Ausgabe übernommen werden.

Nun sind die Makros registriert und wir können sie verwenden. Wir müssen ausdrücken, dass jedes Land genau eine der drei Farben hat. Dies können wir durch die Kombination von `foreach` und `oneOf` tun:

```
// Jedes Land hat genau eine Farbe:
foreach(‘c’,‘lu’,‘de’,‘fr’,‘be’,‘oneOf(c‘’_r, c‘’_g, c‘’_b) &
’)
```

Bemerkenswert ist: m4 erkennt Makros einfach durch eine Zeichenfolge im Text. In der Foreach-Schleife muss somit das Makro `c` erkannt werden. Würde man nun in `oneOf` einfach `c_r` schreiben, könnte das Makro nicht erkannt werden, weil das komplette Token gelesen wird und dann kein Makro ist. Deshalb muss man die *Quotes* von m4 einfügen, um diese Zeichenfolge in zwei Tokens zu trennen: `c‘’_r`. Nun erhält m4 zuerst das Token `c` und expandiert es zum jeweiligen Land und hängt dann das zweite Token `_r` an, wie gewünscht.

Das Makro produziert nun jeweils eine Zeile pro Land mit der gewünschten Aussage, dass es nur eine der Farben haben kann. Jede dieser Zeilen endet mit einem `&`. Dies ist in Ordnung, denn es folgen ja noch weitere Bedingungen. Man sieht jedoch an diesem Beispiel, dass es leicht vorkommen kann, dass in einer Schleife das letzte Element anders behandelt werden muss als die vorherigen. Wir werden gleich sehen, wie man das mit `foreach` tun kann.

Nun wollen wir formulieren, was es bedeutet, dass zwei Länder benachbart sind. Zwei benachbarte Länder dürfen nicht die gleiche Farbe haben. Dazu wollen wir ein Makro `benachbart($1,$2)` verwenden, das die Aussage ergibt, dass die beiden als Argumente übergebenen Länder verschiedene Farben haben.

Definition des Makros „benachbart“:

```
define(‘benachbart’,‘(!$1_r | !$2_r) & (!$1_g | !$2_g ) & (!$1_b | ↩
```

```
    !$2_b)')
```

Auch hier können wir wieder mit einer Schleife arbeiten:

```
define(‘benachbart’,‘(foreach(‘c’,‘r’,‘g’,‘b’,‘(!$1_‘’c|!$2_‘’c)
    ifdef(@lastc,‘’,‘ & ’)’))’)
```

In diesem Fall wird nun der letzte Durchgang der Schleife anders behandelt als die vorherigen, es wird kein `&` ans Ende gedruckt. `foreach` definiert im letzten Durchgang der Schleife das Makro `@lastc`, wobei `c` das erste Argument von `foreach` ist. Man beachte, dass diese Mimik mit einer gewissen Vorsicht zu genießen ist, denn man muss die „Schleifenvariable“ `x` von `foreach` so wählen, dass das Makro `@lastx` einen bisher nicht definierten Makronamen ergibt. m4 kennt keine Gültigkeitsbereiche für Definitionen von Makros!

Nun können wir die Fragestellung, ob Luxemburg und Umgebung mit 3 Farben gefärbt werden kann, vollständig formulieren:

```
dnl needs mmp

divert(-1)
include(‘util.mml’)
include(‘logic.mml’)
divert

// Jedes Land hat genau eine Farbe:
foreach(‘c’,‘lu’,‘de’,‘fr’,‘be’,‘oneOf(c‘’_r, c‘’_g, c‘’_b) &
’)

// Benachbarte Länder haben verschiedene Farben:
define(‘benachbart’,‘(foreach(‘c’,‘r’,‘g’,‘b’,‘(!$1_‘’c|!$2_‘’c)
    ifdef(@lastc,‘’,‘ & ’)’))’) dnl
benachbart(lu, de) &
benachbart(lu, fr) &
benachbart(lu, be) &
benachbart(de, fr) &
benachbart(fr, be) &
benachbart(be, de)
```

Wenden wir einen SAT-Solver an, erhalten wir das (nicht überraschende) Ergebnis:

```
The proposition is not satisfiable.
```

Werfen wir noch einen Blick auf einige andere Beispiele:

Tschechien ist von vier Staaten umgeben, nämlich Polen, Slowakei, Österreich und Deutschland. Stellt man hier die gleiche Frage, nämlich, ob die Karte mit Tschechien und Umgebung mit drei Farben färbbar ist, liefert der SAT-Solver eine Färbung, zum Beispiel wie in Abb. 2.

Australien kann mit drei Farben gefärbt werden, die USA nicht, hier brauchen wir vier Farben – siehe Anhang Seite 32.

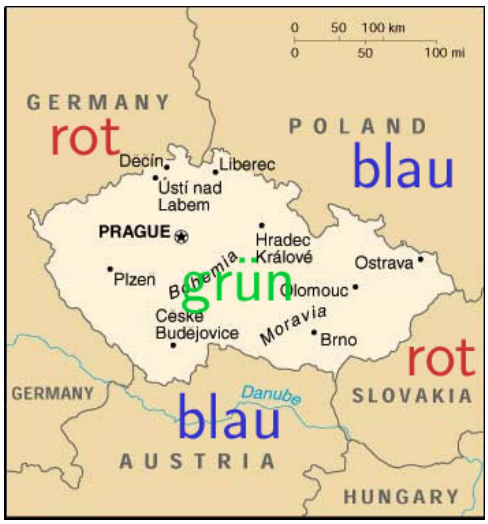

Abbildung 2: Tschechien und Umgebung – mit Färbung

Am Beispiel der Karte der USA hat man es schon mit einer nicht mehr ganz kleinen Aussage zu tun. Die SAT-Solver von SAT4J liefern innerhalb von 1 bis 2 Sekunden eine Antwort, anders „NaiveDPLL“: um eine Färbung der Karte mit 4 Farben zu finden, benötigt dieser SAT-Solver 40 Sekunden.

3 Logelei II

Ein zweiter Typ von Logeleien, wie sie zum Beispiel in der Zeit veröffentlicht werden, besteht darin, dass man verschiedene Objekte hat, deren Eigenschaften man korrekt zuordnen muss.

Ein Beispiel für diesen Typ ist die folgende Logelei, die Einstein zugeschrieben wird (ich konnte jedoch keinerlei Beleg dafür finden):

> Albert Einstein verfasste (angeblich) dieses Rätsel, von dem er behauptete, dass 98% der Weltbevölkerung nicht in der Lage seien, es zu lösen. Gehören Sie zu den 2%? Viel Spaß beim Ausprobieren!
>
> Es gibt fünf Häuser mit je einer anderen Farbe. In jedem Haus wohnt eine Person einer anderen Nationalität. Jeder Hausbewohner bevorzugt ein bestimmtes Getränk, raucht eine bestimmte Zigarettenmarke und hält ein bestimmtes Haustier. Keine der 5 Personen trinkt das gleiche Getränk, raucht die gleichen Zigaretten oder hält das gleiche Tier wie einer seiner Nachbarn.
>
> Frage: Wer besitzt einen Fisch?
>
> Bekannt sind folgende Tatsachen:
>
> – Der Brite lebt im roten Haus.

- Der Schwede hält einen Hund.
- Der Däne trinkt gerne Tee.
- Das grüne Haus steht links vom weißen Haus.
- Der Besitzer des grünen Hauses trinkt Kaffee.
- Die Person, die Pall Mall raucht, hält einen Vogel.
- Der Mann, der im mittleren Haus wohnt, trinkt Milch.
- Der Besitzer des gelben Hauses raucht Dunhill.
- Der Norweger wohnt im ersten Haus.
- Der Marlboro-Raucher wohnt neben dem, der eine Katze hält.
- Der Mann, der ein Pferd hält, wohnt neben dem, der Dunhill raucht.
- Der Winfield-Raucher trinkt gerne Bier.
- Der Norweger wohnt neben dem blauen Haus.
- Der Deutsche raucht Rothmanns.
- Der Marlboro-Raucher hat einen Nachbarn, der Wasser trinkt

Die eigentliche Aufgabe besteht darin, das Rätsel in der Aussagenlogik zu „kodieren“. Wir haben 5 Personen mit 6 Attributen: Nationalität, Haustier, Farbe und Position des Hauses, Zigarettenmarke und Getränk.

Wir benennen die Person in Haus mit der Position i pi und betrachten dann die Ausprägungen der Attribute, zum Beispiel $p2.n\_brite$. Diese Variable ist wahr, wenn die Person in Haus 2 britischer Nationalität ist. (Die Wahl, die Person mit der Position des Hauses zu identifizieren ist natürlich willkürlich, man hätte genauso gut die Person mit der Nationalität identifizieren und sagen können, dass $brite.h\_2$ bedeutet, der Brite wohne im zweiten Haus.)

Zunächst müssen wir nun ausdrücken, dass jede Person nur jeweils genau eine Ausprägung der restlichen fünf Attribute hat. Nach der Kartenfärbung ist dies nichts wirklich Neues:

```
define(‘einAttr’,‘for(‘i’,‘1’,‘5’,‘oneOf(p‘’i.$1,p‘’i.$2,p‘’i.$3,p‘’i.$4,p‘’i.$5)ifdef(@lasti,‘’,‘ & ’) ’)’) dnl
einAttr(‘f_rot’,‘f_gruen’,‘f_gelb’,‘f_blau’,‘f_weiß’) &
einAttr(‘n_brite’,‘n_daene’,‘n_deutscher’,‘n_norweger’,‘n_schwede’) &
einAttr(‘t_fisch’,‘t_hund’,‘t_katze’,‘t_pferd’,‘t_vogel’) &
einAttr(‘z_dunhill’,‘z_marlboro’,‘z_pallmall’,‘z_rothmanns’,‘z_winfield’) &
einAttr(‘g_bier’,‘g_kaffee’,‘g_milch’,‘g_tee’,‘g_wasser’)
```

Weiter müssen wir aussagen, dass die Werte eines Attributs für alle Personen paarweise verschieden sind. Dazu benutzen wir `maxOneOf` aus der Bibliothek `logic.mml`:

```
define(‘different’,‘foreach(‘a’,$1,$2,$3,$4,$5,‘maxOneOf(p1.a,p2.a,
    p3.a,p4.a,p5.a)ifdef(@lasta,‘’,‘& ’) ’)’)
different(‘f_rot’,‘f_gruen’,‘f_gelb’,‘f_blau’,‘f_weiß’) &
different(‘n_brite’,‘n_daene’,‘n_deutscher’,‘n_norweger’,‘n_schwede
    ’) &
different(‘t_fisch’,‘t_hund’,‘t_katze’,‘t_pferd’,‘t_vogel’) &
different(‘z_dunhill’,‘z_marlboro’,‘z_pallmall’,‘z_rothmanns’,‘
    z_winfield’) &
different(‘g_bier’,‘g_kaffee’,‘g_milch’,‘g_tee’,‘g_wasser’)
```

Schließlich benötigen wir noch einige Hilfsmakros: `gleich($1,$2)` gibt an, dass ein und dieselbe Person die beiden Eigenschaften `$1` und `$2` hat. `benachbart($1,$2)` gibt an, dass benachbarte Personen die beiden Eigenschaften haben und `links($1,$2)` ist wahr, wenn eine Person mit der ersten Eigenschaft links der Person mit der zweiten Eigenschaft wohnt:

```
define(‘gleich’,‘for(‘i’,‘1’,‘5’,‘(p‘’i.$1 -> p‘’i.$2)ifdef(@lasti
    ,‘’,‘&’)’)’)
define(‘benachbart’,‘(p1.$1 -> p2.$2) & (p2.$1 -> (p1.$2|p3.$2)) &
(p3.$1 -> (p2.$2|p3.$2)) & (p4.$1 -> (p3.$2|p5.$2)) & (p5.$1 -> (p4
    .$2))’)
define(‘links’,‘(p1.$1 -> (p2.$2|p3.$2|p4.$2)) &
(p2.$1 -> (p3.$2|p4.$2|p5.$2)) & (p3.$1 -> (p4.$2|p5.$2)) & (p4.$1
    -> p5.$2)’)
```

Jetzt kann man die Aussagen formulieren, die den Vorgaben aus dem Rätsel entsprechen:

```
gleich(‘n_brite’,‘f_rot’) &
    // Der Brite lebt im roten Haus
gleich(‘n_schwede’,‘t_hund’) &
    // Der Schwede hält einen Hund
gleich(‘n_daene’,‘g_tee’) &
    // Der Däne trinkt gerne Tee
links(f_gruen,f_weiß) &
    // Das grüne Haus steht ‘links’ vom weißen Haus
gleich(‘f_gruen’,‘g_kaffee’) &
    // Der Besitzer des grünen Hauses trinkt Kaffee
gleich(‘z_pallmall’,‘t_vogel’) &
    // Die Person, die Pall Mall raucht, hält einen Vogel
p3.g_milch &
    // Der Mann, der im mittleren Haus wohnt, trinkt Milch
gleich(‘f_gelb’,‘z_dunhill’) &
    // Der Besitzer des gelben Hauses raucht Dunhill
p1.n_norweger &
    // Der Norweger wohnt im ersten Haus
benachbart(‘z_marlboro’,‘t_katze’) &
    // Der Marlboro-Raucher wohnt neben dem, der eine Katze hält
benachbart(‘t_pferd’,‘z_dunhill’) &
    // Der Mann, der ein Pferd hält, wohnt nebem dem, der Dunhill
        raucht
gleich(‘z_winfield’,‘g_bier’) &
```

```
    // Der Winfield-Raucher trinkt gerne Bier
benachbart(‘n_norweger’,‘f_blau’) &
    // Der Norweger wohnt neben dem blauen Haus
gleich(‘n_deutscher’,‘z_rothmanns’) &
    // Der Deutsche raucht Rothmanns
benachbart(‘z_marlboro’,‘g_wasser’)
    // der Marlboro-Raucher hat einen Nachbarn, der Wasser trinkt
```

Unser SAT-Solver löst das Rätsel nun in einer Sekunde (aber nur, wenn wir nicht den „NaiveDPLL“ verwenden, dieser braucht 68 Sekunden):

Der Deutsche im vierten Haus hat einen Fisch als Haustier.

Um uns ein Bild von der Größe der Formel zu machen: Die Aussage wird zunächst durch die sogenannte Tseitin-Transformation in eine Formel in konjunktiver Normalform transformiert, die in Bezug auf die Erfüllbarkeit äquivalent ist, aber mehr Variablen hat[5]. Nach der Tseitin-Transformation hat die Formel 2396 Variablen und 5814 Klauseln.

Mit den bisher erläuterten Techniken, kann man diesen Typ Rätsel recht leicht lösen – Beispiele in Anhang Seite 37.

Es gibt übrigens noch einen dritten Typ von Logeleien in der Zeit, in dem man lineare Gleichungen lösen muss. Es handelt sich um Rätsel des Typs „Seine Schwester ist doppelt so alt, als er war, als er halb so alt war wie sein Bruder...“. Solche Rätsel lassen sich mit mpa nicht elegant lösen, da man Unmengen Formeln braucht, um die benötigte Arithmetik in der Aussagenlogik nachzubilden.

4 Sudoku

Sudoku ist eine Variante eines lateinisches Quadrates und hat in den letzten Jahren Kultstatus als Rätsel bekommen. Die Regeln sind recht einfach: In einem Quadrat von 9×9 Zellen sind die Zahlen $1 - 9$ so unterzubringen, dass in jeder Zeile und jeder Spalte jede Zahl genau einmal vorkommt. Außerdem darf jede Zahl nur einmal in den 9 Subquadraten mit 3×3 Zellen vorkommen.

Ein korrekt gestelltes Rätsel gibt einige Belegungen vor und darf dann nur eine Lösung haben. Ein Beispiel in Abb. 3.

Sudoku kann man folgendermaßen in der Aussagenlogik ausdrücken: Wir definieren Boolesche Variablen $c_{i,j,k}$ für $i, j, k \in \{1, \ldots, 9\}$, wobei $c_{i,j,k}$ genau dann wahr sein soll, wenn die Zelle $c_{i,j}$ den Wert k hat. Wir haben somit 9^3 Variablen.

Die Regeln kann man nun leicht ausdrücken:

(1) In jeder Zelle $c_{i,j}$ muss genau *ein* Wert k sein.

[5] Dafür ist die Tseitin-Transformation linear im Unterschied zur üblichen Transformation in die konjunktive Normalform, die im schlechtesten Falle exponentiell ist

	2	4				3	8	
6		7	2		9	1		4
8	1		7		3		9	6
	4	8				9	7	
	6	9				5	1	
7	5		9		8		4	1
4		1	6		5	7		9
	9	6				8	3	

Abbildung 3: Beispiel Sudoku aus der ZEIT 32/2008

(2) In jeder Zeile, jeder Spalte und jedem Subquadrat muss jede der Zahlen $1-9$ einmal vorkommen.

Die Mittel, diese Regeln für mpa zu formulieren, haben wir bereits bei den Logeleien zweiten Typs verwendet:

Steht oneOf$(x_1, \ldots, x_9)$ für die Aussage, dass genau eine der Variablen x_i wahr ist, dann lautet Regel (1):

$$(1) \qquad \bigwedge_{i\in\{1,\ldots,9\}} \bigwedge_{j\in\{1,\ldots,9\}} \text{oneOf}(c_{i,j,1}, \ldots c_{i,j,9})$$

Regel (2) können wir formulieren, wenn wir für jede Zeile, jede Spalte und jedes Subquadrat fordern, dass die 9 Zellen verschiedene Werte enthalten. Seien $z_1, \ldots, z_9$ 9 Zellen und steht maxOneOf$(x_1, \ldots, x_9)$ für die Aussage, dass höchstens eine der Variablen x_i wahr ist, fordert folgende Aussage die Verschiedenheit der Werte in den neun Zellen:

(2) Für die Zellen $z_1, \ldots, z_9$ jeder Zeile, jeder Spalte und jedes Subquadrats gilt:

$$\bigwedge_{k\in\{1,\ldots,9\}} \text{maxOneOf}(z_{i,k}, \ldots z_{i,k})$$

wobei $z_{i,k}$ bedeutet, dass die Zelle z_i den Wert k hat.

Dies ausgedrückt im mpa ergibt:

```
// Regel 1
for(‘i’,‘1’,‘9’,‘for(‘j’,‘1’,‘9’,‘ifdef(‘@lastj’,‘oneOf(for(‘k
    ’,‘1’,‘9’,‘c‘’i‘’j‘’k‘’ifdef(‘@lastk’,‘’,‘,’)’))
’,‘oneOf(for(‘k’,‘1’,‘9’,‘c‘’i‘’j‘’k‘’ifdef(‘@lastk’,‘’,‘,’)’)) &
’)’) &’)
```

```
// Regel 2
define(‘different’,‘for(‘x’,‘1’,‘9’,‘maxOneOf($1‘’x,$2‘’x,$3‘’x,$4‘’x,$5‘’x,$6‘’x,$7‘’x,$8‘’x,$9‘’x)ifdef(‘@lastx’,‘’, &‘’)’)’)
// rows
for(‘i’,‘1’,‘9’,‘different(for(‘j’,‘1’,‘9’,‘c‘’i‘’j‘’,’)) &
’)
// columns
for(‘j’,‘1’,‘9’,‘different(for(‘i’,‘1’,‘9’,‘c‘’i‘’j‘’,’)) &
’)
// regions
different(‘c11’,‘c12’,‘c13’,‘c21’,‘c22’,‘c23’,‘c31’,‘c32’,‘c33’) &
different(‘c41’,‘c42’,‘c43’,‘c51’,‘c52’,‘c53’,‘c61’,‘c62’,‘c63’) &
different(‘c71’,‘c72’,‘c73’,‘c81’,‘c82’,‘c83’,‘c91’,‘c92’,‘c93’) &
different(‘c14’,‘c15’,‘c16’,‘c24’,‘c25’,‘c26’,‘c34’,‘c35’,‘c36’) &
different(‘c44’,‘c45’,‘c46’,‘c54’,‘c55’,‘c56’,‘c64’,‘c65’,‘c66’) &
different(‘c74’,‘c75’,‘c76’,‘c84’,‘c85’,‘c86’,‘c94’,‘c95’,‘c96’) &
different(‘c17’,‘c18’,‘c19’,‘c27’,‘c28’,‘c29’,‘c37’,‘c38’,‘c39’) &
different(‘c47’,‘c48’,‘c49’,‘c57’,‘c58’,‘c59’,‘c67’,‘c68’,‘c69’) &
different(‘c77’,‘c78’,‘c79’,‘c87’,‘c88’,‘c89’,‘c97’,‘c98’,‘c99’)
```

Diese Regeln sind natürlich für jedes Sudoku gleich, wir lagern sie daher in eine eigene Datei `Sudoku.m4` aus. Um ein Rätsel vollständig zu formulieren, müssen wir dann nur noch für die vorgegeben Werte die entsprechenden Booleschen Variablen als wahr fordern. Das obige Rätsel ergibt dann:

```
dnl needs mmp
/*
 * Sudoku aus der Zeit Nr. 32 2008
 */
// Regeln
include(‘sudoku.m4’)
// und vorgegebene Werte
& c122 & c134 & c173 & c188
& c216 & c237 & c242 & c269 & c271 & c294
& c318 & c321 & c347 & c363 & c389 & c396
& c424 & c438 & c479 & c487
& c626 & c639 & c675 & c681
& c717 & c725 & c749 & c768 & c784 & c791
& c814 & c831 & c846 & c865 & c877 & c899
& c929 & c936 & c978 & c983
```

mpa kann nun die Erfüllbarkeit prüfen und liefert in der Tat ein Modell mit der Lösung in Abb. 4.

Für die Berechnung des Modells wird viel Hauptspeicher benötigt, deshalb ist es leicht möglich, dass die Ausnahme

```
java.lang.OutOfMemoryError: Java heap space
```

auftritt. Dagegen hilft der Aufruf von mpa mit der Option `-Xmx512m` für die Java Virtual Machine.

9	2	4	1	5	6	3	8	7
6	3	7	2	8	9	1	5	4
8	1	5	7	4	3	2	9	6
1	4	8	5	6	2	9	7	3
5	7	2	3	9	1	4	6	8
3	6	9	8	7	4	5	1	2
7	5	3	9	2	8	6	4	1
4	8	1	6	3	5	7	2	9
2	9	6	4	1	7	8	3	5

Abbildung 4: Lösung Sudoku aus der Zeit 32/2008

Nach der Tseitin-Transformation haben wir 118945 Klauseln (von denen aber einige Duplikate sind) und 48153 Variablen.[6]

Die Berechnung dauert auf meinem Rechner mit dem „DefaultSAT4J“ als SAT-Solver 12 Sekunden. Davon entfallen etwa 11 Sekunden auf den Makroprozessor mmp und die Tseitin-Transformation. Von der Verwendung von „NaiveDPLL“ als SAT-Solver bei Sudoku ist abzuraten, es sei denn, man hat sehr viel Zeit, so etwa 40 Minuten nämlich.

Die International Herald Tribune hat eine Variante von Sudoku in ihrem Blatt, in der vier weitere Subquadrate die Zahlen $1-9$ enthalten müssen. Diese Subquadrate sind in Abb. 5 grau unterlegt.

					6			
9								2
			8	4			5	
						6		
		4						
	3					5		
		2	4					1
	5				7	4		
				2		3		

8	2	5	7	9	6	1	4	3
9	4	6	5	3	1	7	8	2
1	7	3	8	4	2	9	5	6
5	1	9	2	7	4	6	3	8
6	8	4	3	1	5	2	9	7
2	3	7	6	8	9	5	1	4
7	9	2	4	5	3	8	6	1
3	5	8	1	6	7	4	2	9
4	6	1	9	2	8	3	7	5

Abbildung 5: Sudoku der International Herald Tribune samt Lösung

[6] mpa macht die Tseitin-Transformation bei jeder Formel, auch wenn diese bereits in konjunktiver Normalform ist. Am Beispiel von Sudoku hat die ursprüngliche Formel $9^3 = 729$ Variablen und 11745 Klauseln. Man sieht, dass die Tseitin-Transformation die Formel doch erheblich vergrößert.

Es ist in mpa einfach, diesen Typ Sudoku zu lösen, wir fügen einfach die Bedingungen für die erweiterten Regeln hinzu:

```
// Die Sudoku in der International Herald Tribune haben
// vier weitere Regionen, in denen 1 - 9 vorkommen sollen
& different(‘c22’,‘c23’,‘c24’,‘c32’,‘c33’,‘c34’,‘c42’,‘c43’,‘c44’)
& different(‘c62’,‘c63’,‘c64’,‘c72’,‘c73’,‘c74’,‘c82’,‘c83’,‘c84’)
& different(‘c26’,‘c27’,‘c28’,‘c36’,‘c37’,‘c38’,‘c46’,‘c47’,‘c48’)
& different(‘c66’,‘c67’,‘c68’,‘c76’,‘c77’,‘c78’,‘c86’,‘c87’,‘c88’)
```

Interessant ist die Frage, ob das Rätsel in der IHT auch ohne die zusätzlichen Regeln eine eindeutige Lösung hat. Dies ist in diesem Beispiel nicht der Fall; mpa liefert ein anderes Modell, wenn man die zusätzlichen Regeln weglässt.

5 Variabilitätsmodelle

In diesem Abschnitt kommen wir nun endlich zu einer Anwendung der Aussagenlogik in einem softwaretechnischen Thema, dem *Variabilitätsmodell*.

In Softwareproduktlinien geht es darum, geplante Wiederverwendung dadurch zu betreiben, dass Softwareprodukte in vielfältigen Varianten auf Basis einer gemeinsamen Plattform entwickelt werden. Dazu ist es unerlässlich, zu definieren, welche Gemeinsamkeiten diese Produkte einer Plattform teilen und wodurch sie sich unterscheiden können, kurz: welche Variabilität möglich ist. Deshalb ist die Spezifikation von Variabilität ein wichtiger Bestandteil der Softwareproduktlinienentwicklung.

Es haben sich sogenannte Feature-Modelle eingebürgert, mit den Gemeinsamkeiten und Unterschiede von Softwareprodukten innerhalb einer Softwareproduktlinie beschrieben werden. In den letzten Jahren hat sich daraus das Variabilitätsmodell entwickelt. Eine Ausprägung des Variabilitätsmodells wird in [7] beschrieben.

Im Folgenden wird gezeigt, dass man ein solches Variabilitätsmodell auf naheliegende Weise in der Aussagenlogik darstellen kann. Einige wenige Makros für den mpa erlauben die Formulierung von Variabilitätsmodellen als Formeln der Aussagenlogik. In mpa kann man dann prüfen, ob eine bestimmte Auswahl von Varianten, eine *Produktkonfiguration* zulässig ist. Wir werden auch unter dem Begriff der *Konfliktfreiheit* formulieren, was man von einem „ordentlich“ konstruierten Variabilitätsmodell erwarten wird.

5.1 Das Variabilitätsmodell

Bei der Softwareentwicklung entstehen diverse sogenannte *Artefakte*, wie Anforderungsspezifikationen, Entwürfe, Testpläne usw. bis hin zum Programmcode. Entwickelt man verschiedene Produkte auf Basis einer gemeinsamen Plattform, dann enthalten diese Artefakte vorgeplante Stellen, an denen Variabilität möglich ist. Diese Entscheidungsmöglichkeiten samt ihren möglichen Ausprägungen werden in einem *Variabilitätsmodell* beschreiben. Es

kann für allen Phasen des Entwicklungsprozesses und für alle Arten von Artefakten eingesetzt werden.

Das Variabilitätsmodell umfasst Variationspunkte und Varianten sowie Abhängigkeiten und Bedingungen zwischen diesen. Im Detail sieht das Variabilitätsmodell in [7] vor:

Ein *Variationspunkt* ist eine Stelle in der Plattform, an der eine Entscheidung offen gehalten wurde. Eine *Variante* ist eine spezifische Ausprägung der Eigenschaft eines Variationspunktes, also eine mögliche Entscheidung. So kann etwa ein Authentifizierungsmechanismus ein Variationspunkt sein, für den innerhalb der Softwareproduktlinie die Varianten Benutzerkennwort, Zertifikat und Fingerabdruck vorgesehen sind.

Ein Variationspunkt wird *gebunden*, wenn eine oder mehrere der im Variabilitätsmodell beschrieben Varianten ausgewählt wird. Dies kann je nach Verwendung und Absicht des Modells in jeder Phase der Entwicklung geschehen. Welche Entscheidung beim Binden von Varianten getroffen werden können, wird durch *Variabilitätsabhängigkeiten* spezifiziert, mit denen Varianten Variationspunkten zugeordnet werden. Dabei gilt:

1. Jeder Variationspunkt hat $1 - n$ Varianten.
2. Jede Variante gehört zu $1 - m$ Variationspunkten.
3. Eine Variante kann obligatorisch (*mandatory*) für einen Variationspunkt sein, d.h. ist der Variationspunkt Teil eines Produkts, dann *muss* auch die Variante Bestandteil sein.
4. Eine Variante kann optional für einen Variationspunkt sein.
5. Aus einer Menge von Varianten zu einem Variationspunkt können $min - max$ Varianten erforderlich sein (*alternative choice*[7]).

Man sieht leicht, dass die Einteilung der Variationsabhängigkeiten in die drei Arten (*mandatory*, *optional* und *alternative choice*) auch so beschrieben werden kann:

Die Menge der Varianten zu einem Variationspunkt kann disjunkt zerlegt werden in Teilmengen und zu jeder dieser Teilmengen gibt es eine Bedingung der Form $[min, max]$, die angibt, wie viele der Varianten dieser Teilmenge gewählt werden müssen. Eine Teilmenge T der Mächtigkeit $|T| = n$ und der Bedingung $[n, n]$ besteht aus obligatorischen Varianten, eine Teilmenge T der Mächtigkeit m und der Bedingung $[0, m]$ enthält nur optionale Varianten.

Es ergibt sich damit auch eine *Normalform* für die Darstellung von Variationspunkten, Varianten und Variationsabhängigkeiten:

Die Menge der Varianten zu einem Variationspunkt zerfällt disjunkt in

– eine Teilmenge T_{man} von obligatorischen Varianten,

[7] In [7] wird die Bezeichnung *alternative choice* verwendet, obgleich einem *multiple choice* als angemessener erscheint.

- eine Teilmenge T_{opt} von optionalen Varianten und
- weitere Teilmengen T_i der Mächtigkeit n_i mit den Bedingungen $[min_i, max_i]$, wobei $min_i >= 0$, $max_i <= n_i$ und $[min_i, max_i]$ ist weder $[n_i, n_i]$ noch $[0, n_i]$.

Darüberhinaus enthält das Variabilitätsmodell *Variabilitätsbedingungen*, die angeben,

- ob Variationspunkte andere Variationspunkte benötigen (*requires*),
- ob Varianten andere Varianten oder Variationspunkte benötigen (*requires*),
- ob Variationspunkte andere Variationspunkte ausschließen (*excludes*),
- ob Varianten andere Varianten oder Variationspunkte ausschließen (*excludes*).

In [7] werden diese Eigenschaften in einem UML-Klassendiagramm dargestellt[8].

Es gibt für das Variabilitätsmodell eine graphische Notation [7, Abb. 4-11 auf S. 85]. In Abb. 6 wird ein Beispiel aus [2] in dieser Notation dargestellt.[9]

5.2 Eine Sprache für das Variabilitätsmodell

Wir wollen eine Sprache zur Spezifikation von Variabilitätsmodellen entwerfen, aus der dann innerhalb von mpa mit dem Makroprozessor mmp eine Formel der Aussagenlogik erzeugt werden kann. Wir beschreiben die Sprache zunächst informell.

- Variationspunkte und Varianten werden eindeutig identifiziert durch ihre Bezeichnung. Diese Bezeichnungen müssen für das gesamte Modell eindeutig sein, sie beginnen mit einem Buchstaben, gefolgt von Buchstaben, Ziffern oder dem Zeichen `_`. Schlüsselwörter der Sprache dürfen nicht als Bezeichner verwendet werden.
- Schlüsselworte der Sprache sind
  - `vm` (*variability model*),
  - `vp` (*variation point*),
  - `man` (*group of mandatory variants*),

[8] Nicht alle Eigenschaften des Variabilitätsmodells können in diesem Klassendiagramm ohne zusätzliche Kommentare dargestellt werden, siehe die Bemerkung in [7, S. 79 oben].

[9] Unklar bleibt, wie der Fall in der graphischen Darstellung untergebracht wird, dass eine Variante mehreren Variationspunkten zugeordnet ist.

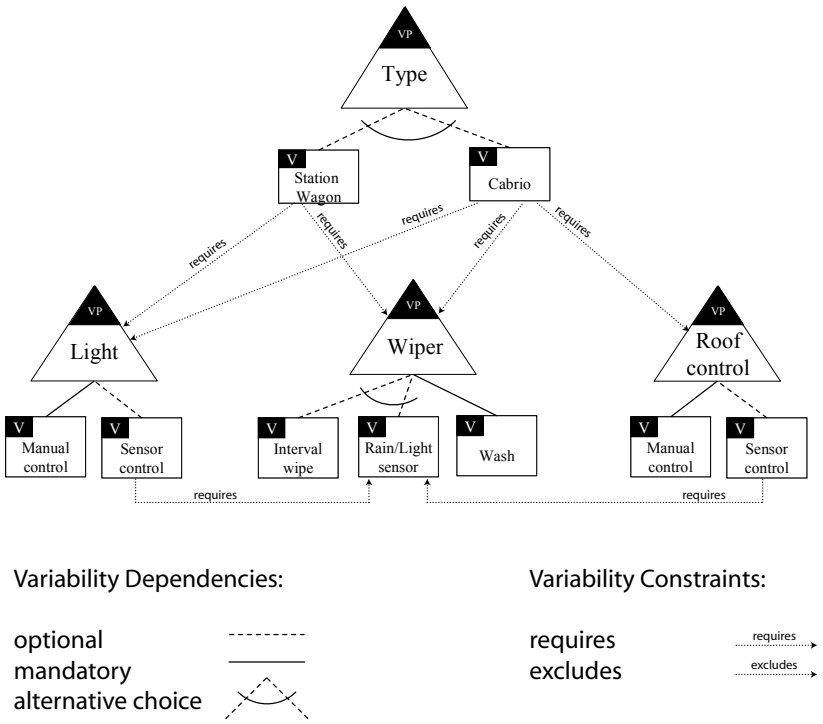

Abbildung 6: Beispiel für ein Variabilitätsmodell aus [2] in der Notation von [7]

  - `opt` (*group of optional variants*),
  - `alt` (*group of alternative variants*),
  - `req` (*variability constraint requires*),
  - `excl` (*variability constraint excludes*).

- Ein Variabilitätsmodell wird definiert durch den Ausdruck `vm()`, der durch Komma getrennte Ausdrücke `vp()`, `req()` oder `excl()` als Argumente enthält.

- Variationspunkte und ihnen zugeordnete Varianten werden in einem Ausdruck `vp()` spezifiziert: Zuerst wird der Variationspunkt angeben, gefolgt von mindestens einem der Unterausdrücke `man()`, `opt()` oder `alt()`; alle getrennt durch Komma.

- Der Unterausdruck `man()` enthält eine Menge von Varianten, getrennt durch Komma, die für den Variationspunkt obligatorisch sind.

- Der Unterausdruck `opt()` enthält eine Menge von Varianten, getrennt durch Komma, die für den Variationspunkt optional sind.
- Der Unterausdruck `alt()` enthält zunächst zwei nicht-negative Zahlen, gefolgt von einer Menge von Varianten, jeweils getrennt durch Komma. Die Ziffern geben die Kardinalitätsbedingung für die danach folgenden Varianten an, die erste Ziffer die minimale, die zweite Ziffer die maximale Zahl der erlaubten Varianten aus der Gruppe.

  Die Angabe der Kardinalitätsbedingung ist optional, wird sie weggelassen, ist genau eine der Varianten erlaubt.
- Alle Varianten zu einem Variationspunkt, die in den Unterausdrücken vorkommen, sind voneinander verschieden.
- Variabilitätsbedingungen werden durch Ausdrücke `req()` für „requires“ und `excl()` für „excludes“ angegeben. Jeder solche Ausdruck enthält mindestens zwei durch Komma getrennte Variationspunkte bzw. Varianten, wobei der erste die folgenden benötigt bzw. ausschließt.

In dieser Sprache wird Beispiel aus der Abb. 6 folgendermaßen formuliert[10]:

```
vm(
vp(Type, alt(StationWagon,Cabrio)),
vp(Light, man(LightManualControl), opt(LightSensorControl)),
vp(Wiper,man(Wash),alt(IntervalWipe,RainLightSensor)),
vp(RoofControl,man(RoofManualControl),opt(RoofSensorControl)),
req(StationWagon,Light,Wiper),
req(Cabrio,Light,Wiper,RoofControl),
req(LightSensorControl,RainLightSensor),
req(RoofSensorControl,RainLightSensor)
)
```

Formal kann man die Grammatik dieser Sprache in der Backus-Naur-Form BNF beschrieben (hier so wie dies in javacc geschieht):

```
TOKEN : {
  <VM: "vm"> |
  <VP: "vp"> |
  <MAN: "man"> |
  <OPT: "opt"> |
  <ALT: "alt"> |
  <REQ: "req"> |
  <EXCL: "excl">
```

[10] Bei dieser Formulierung fällt auf, dass das Diagramm aus der Vorlage interpretationsbedürftig ist: Wird eine Variante durch ihre Bezeichnung oder durch das Kästchen als Symbol eindeutig identifiziert? Ich vermute, dass das zweite gemeint ist, denn „Manual-Control“, bzw. „SensorControl“ für Licht und für Dach werden wohl jeweils zwei verschiedene Varianten sein. Infolgedessen werden die Bezeichnungen in der Umsetzung eindeutig gemacht.

```
}

TOKEN : {
    <IDENTIFIER: <LETTER> (<LETTER> | <DIGIT>)*>
|   <NUMBER: (<DIGIT>)+ >
|   <#LETTER: ["A"-"Z","_","a"-"z"]>
|   <#DIGIT: ["0"-"9"]>
}

NONTERMINALS:
  VMSpecification :=  ( VMDefinition )
  VMDefinition  :=  <VM> "(" VPDefinition ( ( "," VPDefinition ) | ↩
      ( "," ReqConstraint ) | ( "," ExclConstraint ) )* ")"
  VPDefinition  :=  <VP> "(" <IDENTIFIER> ( ( "," ManExpression ) | ↩
       ( "," OptExpression ) | ( "," AltExpression ) )+ ")"
  ReqConstraint :=  <REQ> "(" <IDENTIFIER> ( "," <IDENTIFIER> )+ ") ↩
      "
  ExclConstraint  :=  <EXCL> "(" <IDENTIFIER> ( "," <IDENTIFIER> )+ ↩
       ")"
  ManExpression :=  <MAN> "(" <IDENTIFIER> ( "," <IDENTIFIER> )+ ") ↩
      "
  OptExpression :=  <OPT> "(" <IDENTIFIER> ( "," <IDENTIFIER> )+ ") ↩
      "
  AltExpression :=  <ALT> "(" <NUMBER> "," <NUMBER> "," <IDENTIFIER ↩
      > ( "," <IDENTIFIER> )+ ")"
    | <ALT> "(" <IDENTIFIER> ( "," <IDENTIFIER> )+ ")"
```

Bemerkung: Da die Ausdrücke mit mmp verarbeitet werden, müssen die Ausdrücke und Bezeichner mit den „Quotes“ von m4, also ‘ ’ umrahmt werden. Dies ist aber nur wegen der Art der Weiterverarbeitung erforderlich und keine notwendige Eigenschaft der Sprache. In den Beispielen werden die „Quotes“ der Lesbarkeit halber weggelassen. Ferner müssen in m4 Klammern unmittelbar auf den Namen eines Makros folgen, auch dies ist natürlich nur wegen der spezifischen Weiterverarbeitung zu beachten.

5.3 Umsetzung der Sprache in die Aussagenlogik

Diese Sprache kann auf folgende Weise in die Aussagenlogik „kodiert“ werden:

Die aus dem Variabilitätsmodell entstehende Formel ist die Konjunktion der im Folgenden beschriebenen Subformeln.

- Ist v eine Variante zu den Variationspunkten $vp_1, \ldots, vp_n$, dann gilt $v \rightarrow (vp_1 \vee \cdots \vee vp_n)$.
- Sind $v_1, \ldots, v_n$ obligatorische Varianten des Variationspunkts vp, dann gilt $vp \rightarrow (v_1 \wedge \cdots \wedge v_n)$.
- Sind $v_1, \ldots, v_n$ optionale Varianten des Variationspunkts vp, dann besteht keine weitere Bedingung.

– Wird für die Varianten $v_1, \ldots, v_n$ im Variationspunkt vp $alt_{[k_1,k_2]}(v_1 \ldots, v_n)$ verlangt (mit $k_1, k_2 \geq 0$, $k_1, k_2 \leq n$), dann gilt $vp \rightarrow (min_{[k_1]}(v_1, \ldots, v_n) \wedge max_{[k_2]}(v_1, \ldots . v_n))$ wobei $min_{[k_1]}(\ldots)$ und $max_{[k_2]}(\ldots)$ für die Aussagen stehen, dass mindestens bzw. höchstens k_1 bzw. k_2 der Varianten zutreffen.

– Wird die Beziehung „requires“ zwischen v_1 und v_2 vorgegeben, gilt $v_1 \rightarrow v_2$.

– Wird die Beziehung „excludes“ zwischen v_1 und v_2 vorgegeben, gilt $v_1 \rightarrow \neg v_2$.

Wie man sieht, kann man das Variabilitätsmodell ganz geradlinig in die Aussagenlogik umsetzen.

Die Ausdrücke der Sprache lassen sich weitgehend direkt in die entsprechenden Aussagen übertragen. Unsere Makro-Bibliothek `logic.mml` enthält Makros, die Aussagen erzeugen, dass mindestens oder höchstens k von n Booleschen Variablen wahr sein müssen. Da im Variabilitätsmodell eine Variante mehreren Variationspunkten zugeordnet werden kann, kann die dafür notwendige Aussage jedoch nicht „lokal“ entschieden werden, sondern erst, wenn alle Variationspunkte spezifiziert sind. Hier erweist es sich als sehr nützlich, dass in mmp Makros in Java geschrieben werden können. Zu Beginn der Verarbeitung der Anweisung `vm` wird eine statische Datenstruktur initialisiert, die alle Variabilitätsabhängigkeiten aufnehmen kann. Bei der Verarbeitung der Definitionen der Variationspunkte werden alle Variabilitätsabhängigkeiten in diese Datenstruktur eingetragen, so dass am Ende der Verarbeitung von `vm` die benötigen Bedingungen erzeugt werden können.

5.4 Auswahl von Variationspunkten und Varianten – Produktkonfiguration

Eine *Produktkonfiguration* auf Basis eines Variabilitätsmodells ist eine Wahl von Variationspunkten und Varianten. Es ist dann mit dem mpa möglich, zu überprüfen, ob eine solche Produktkonfiguration mit dem Variabilitätsmodell konform ist, d.h. ob ein so spezifiziertes Produkt im Rahmen des Variabilitätsmodells „baubar“ ist. Dazu setzt man einfach die Booleschen Variablen für die ausgewählten Variationspunkte und Varianten konjunktiv mit der Formel für das Variabilitätsmodell zusammen.

Am Beispiel aus Abb. 6 ergibt etwa die Auswahl von „Cabrio“ und „RoofSensorControl“ und die Überprüfung der Erfüllbarkeit in mpa eine mögliche Konfiguration:

```
The proposition is satisfiable.

A model is:
Valuation
 true  Cabrio
```

```
false  IntervalWipe
 true  Light
 true  LightManualControl
false  LightSensorControl
 true  RainLightSensor
 true  RoofControl
 true  RoofManualControl
 true  RoofSensorControl
false  StationWagon
 true  Type
 true  Wash
 true  Wiper
```

Auch die Auswahl von „StationWagon“ und „RoofControlSensor“ ergibt auf Basis des Variabilitätsmodells eine mögliche Produktkonfiguration. Es geht aus dem Beispiel in [2] nicht hervor, ob ein Kombi (*station wagon*) tatsächlich eine Steuereinheit für das Dach besitzen darf oder nicht.

Man sieht an diesem Beispiel, dass die Transformation des Variabilitätsmodells in die Aussagenlogik und die Überprüfung von Produktkonfigurationen durch mpa auf eventuelle Unklarheiten im Variabilitätsmodell hinweisen kann.

In unserem Beispiel können wir etwa unser Variabilitätsmodell ergänzen, indem wir ausschließen, dass ein Kombi eine Steuerung für das Dach hat. Wir ergänzen das Variabilitätsmodell um die Zeile

```
excl(StationWagon,RoofControl)
```

Dann führt die Wahl von „StationWagon“ und „RoofControl“ zum Ergebnis:

```
The proposition is not satisfiable.
```

Beobachtung: Die Formel ϕ_{VM} zu einem Variabilitätsmodell VM selbst, d.h. ohne Auswahl eines Variationspunktes oder einer Variante ist stets erfüllbar, trivialerweise dadurch, dass alle Variablen „false“ sind. Dies entspricht der „leeren“ Produktkonfiguration.

Deshalb scheint es sinnvoll zu sein, das Variabilitätsmodell zu erweitern, indem man auch vorschreiben kann, dass bestimmte Variationspunkte in jedem Fall vorhanden sein müssen. Eine dafür denkbare graphische Notation dafür wird am Beispiel in Abb. 7 dargestellt. Die Umsetzung in die Aussagenlogik ist klar: Solcherart gekennzeichnete Variationspunkte werden mit der Formel für das Variabilitätsmodell mit $\wedge$ verbunden.

Wir betrachten im Folgenden Variabilitätsmodelle *ohne* solche „vorausgewählte“ Variationspunkte.

5.5 Konfliktfreiheit von Variabilitätsmodellen

Es ist leicht, Variabilitätsmodelle zu konstruieren, die nicht konfliktfrei sind, in dem intuitiven Sinn, dass die Wahl einer Variante nicht zu einem sinnvol-

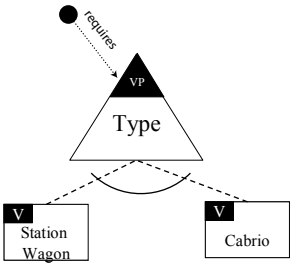

Abbildung 7: Obligatorischer Variationspunkt

len Ergebnis führt. In folgendem Beispiel etwa hat die Auswahl der Variante v_1 zur Folge, dass die resultierende Aussage nicht erfüllbar ist. Also kann es keine Produktkonfiguration geben, in der diese Variante „verbaut“ ist.

```
vm(
vp(vp1,alt(v1,v2)),
req(v1,v2)
)
```

Dieses Beispiel führt uns zur Frage, wie man die *Konfliktfreiheit* von Variabilitätsmodellen definieren kann. Sei ϕ eine Formel über den Variablen $x_1, \ldots, x_n$.

Eine *partielle Bewertung* von ϕ ist eine partielle Abbildung $v : \{x_1, \ldots, x_n\} \rightarrow \{0, 1\}$. Zu jeder partiellen Bewertung von ϕ gibt es ein *partielles Modell*: Es besteht aus der Menge $\mathcal{M}$ von Literalen mit

$$\mathcal{M} = \{x_i \mid v(x_i) = 1\} \cup \{\neg x_i \mid v(x_i) = 0\}.$$

Unter einer *Teilauswertung* einer Formel ϕ wollen wir nun Folgendes verstehen:

Sei $\mathcal{V} = \{x_1, \ldots, x_n\}$ eine Menge von Variablen, die alle Variablen in ϕ enthält und l ein Literal zu einer dieser Variablen.

Wenn $l \wedge \phi$ erfüllbar ist, können wir $\mathcal{M}'$, $\mathcal{V}'$ und ϕ' konstruieren:

$$\mathcal{M}' = \{k \text{ Literal über } \mathcal{V} \mid l \wedge \phi \rightarrow k\}$$

$\mathcal{M}'$ ist das maximale partielle Modell zu $l \wedge \phi$, in dem Sinne, dass $\mathcal{M}'$ aus genau den Literalen besteht, die aus l in der Formel ϕ folgen. Es gilt dann natürlich $l \in \mathcal{M}'$.

$$\mathcal{V}' = \{x \in \mathcal{V} \mid x \notin \mathcal{M}' \text{ und } \neg x \notin \mathcal{M}'\}$$

$\mathcal{V}'$ ist die Menge der Variablen, die nicht durch das partielle Modell $\mathcal{M}'$ festgelegt sind.

$$\phi' = \phi[t_1, \ldots, t_m / k_1, \ldots, k_m]$$

wobei $k_i \in \mathcal{M}'$ und $t_i = \top$ bzw. $t_i = \bot$, je nach Wert von k_i.

ϕ' ist die Formel, die durch die Substitution aller Literale aus $\mathcal{M}'$ in ϕ durch ihren Wert entsteht. Die Variablen in ϕ' sind dann alle in $\mathcal{V}'$ enthalten und es gilt:

$$l \wedge \phi \equiv \bigwedge_{k \in \mathcal{M}'} k \wedge \phi'$$

Man kann Teilauswertungen *sukzessive* ausführen, beginnend mit einer Formel $\phi = \phi_0$, der Menge $\mathcal{V}_0$ der Variablen von ϕ und $\mathcal{M}_0$ der leeren Menge von Literalen. In jedem Schritt wählt man ein beliebiges Literal über $\mathcal{V}_i$ und konstruiert $\mathcal{M}_{i+1}$, $\mathcal{V}_{i+1}$ und ϕ_{i+1}.

Eine solche Folge von Teilauswertungen führt zu einem *Konflikt*, wenn für ein i gilt: $l_i \wedge \phi_i$ ist *nicht* erfüllbar.

Führt eine Folge von Teilauswertungen zu einer leeren Variablenmenge $\mathcal{V}_i$, dann ist $\mathcal{M}_1 \cup \cdots \cup \mathcal{M}_i$ ein (vollständiges) Modell für ϕ.

Definition: Ein Variabilitätsmodell VM heißt *konfliktfrei*, wenn für die zugehörige Formel ϕ_{VM} jede beliebige Folge von Teilauswertungen zu einem Modell führt.

Diese Definition bedeutet für das Variabilitätsmodell, dass jede Wahl bzw. jeder Ausschluss einer Variante oder eines Variationspunkts zu einer Situation führt, bei der alle aus der Wahl folgenden Entscheidungen gebunden sind und jede weitere Wahl nicht zu einer unmöglichen Produktkonfiguration führt. Sukzessive Wahl oder Ausschluss von Variabilität führt unter Berücksichtigung der daraus folgenden Konsequenzen stets zu einer gültigen Produktkonfiguration.

Bemerkungen:

- Diese Definition ist insofern unhandlich, da man zu einem Variabilitätsmodell alle möglichen Folgen von Teilauswertungen betrachten muss. Es stellt sich deshalb die Frage: Gibt es ein einfacheres Kriterium für diese Eigenschaft?

- Man kann diese Eigenschaft natürlich für eine beliebige Aussage der Aussagenlogik formulieren. Sie ist dann identisch mit der Eigenschaft, dass ein Algorithmus, der bei der Suche nach einem Modell sukzessive Teilauswertungen mit beliebiger Wahl einer Variablen macht, niemals auf einen Widerspruch stößt, also stets ohne *backtracking* ein Modell findet. Frage: Gibt es eine andere Charakterisierung von Aussagen, die diese Eigenschaft haben?

5.6 Einsatzmöglichkeiten

Die „Kodierung“ des Variabilitätsmodells in der Aussagenlogik kann mit Gewinn dadurch eingesetzt werden, dass in Werkzeugen für Variabilitätsmodelle „on the fly“ Modelle überprüft werden:

- In einem Editor für Variabilitätsmodelle kann die Konfliktfreiheit von Modellen während der Entwicklung überprüft werden.
- Man kann erlaubte Konfigurationen durch den SAT-Solver erzeugen lassen und an ihnen überprüfen, ob das Variabilitätsmodell tatsächlich alle für die Softwarerproduktlinie erforderlichen Abhängigkeiten und Einschränkungen enthält.
- Bei der Auswahl einer Produktkonfiguration kann man überprüfen, ob eine Auswahl von Variationspunkte bzw. Varianten zu einer erlaubten Konfiguration führt. Es ist dann auch leicht möglich, die noch verbleibenden Wahlmöglichkeiten zu berechnen und unmittelbar bei der Konfiguration sichtbar zu machen.
- Man kann einen SAT-Solver dazu verwenden, einige oder alle zulässigen Produktkonfigurationen zu ermitteln, die dann etwa für den Test der Softwareproduktlinie eingesetzt werden können.

5.7 Verwandte Arbeiten

Merijn de Jonge und Joost Visser haben Feature-Modelle durch eine Sprache beschrieben [5] und Don Batory hat Feature-Modelle in die Aussagenlogik „kodiert“ [1].

Eine industrielle Anwendung eines SAT-Solvers zur Konfiguration von Automobilen bei der Daimler AG haben Carsten Sinz, Andreas Kaiser und Wolfgang Küchlin [12] entwickelt.

6 Fazit

Wir haben Beispiele für die „Kodierung“ von Fragestellungen in der Aussagenlogik vorgestellt und gezeigt, wie man den MNI Proposition Analyzer mpa für die Lösung verwenden kann.

Einfache Logeleien lassen in mpa leicht formulieren und lösen, solche Logeleien entstammen selbst dem Feld der Aussagenlogik.

Schon bei der *Kartenfärbung* stellt es sich heraus, dass der in mpa eingebaute Makroprozessor mmp mit Gewinn verwendet werden kann, um bestimmte Eigenschaften darzustellen. Erst recht bemerkt man das dann bei *Logeleien vom Typ 2* und bei *Sudoku*. Diese Problemstellungen entstammen der relationalen Logik. Der Makroprozessor von mpa wird verwendet, um die Prädikate der Formulierung der Fragestellung in der relationalen Logik

in die Aussagenlogik zu transformieren und dann durch den SAT-Solver von mpa zu lösen.

Schließlich können wir mpa auch auf eine softwaretechnische Fragestellung, das *Variabilitätsmodell* anwenden. Die Transformation des Variabilitätsmodells in die Aussagenlogik erlaubt die Überprüfung von Modellen und die Berechnung erlaubter Konfigurationen. Es ist ohne weiteres denkbar, einen SAT-Solver im Rahmen eines Editors für Variabilitätsmodelle oder eines Produkt-Konfigurators einzusetzen.

Literaturverzeichnis

[1] Batory, D.: *Feature Models, Grammars, and Propositional Formulas*, in: Software Product Lines, Proceedings of the 9th International Conference, SPLC 2005, LNCS 3714, S.7–20, 2005

[2] Bühne, S.; Lauenroth, K.; Pohl, K.: *Why is it not Sufficient to Model Requirement Variability with Feature Models*, in: Proceedings of Workshop Automotive Requirements Engineering (AUREA04), 2004.

[3] Davis, M.; Punam, H.: *A Computing Procedure for Quantification Theory*, in: J. ACM 7(3) S.201–215, 1960.

[4] Davis, M.; Logemann, G.; Loveland, D.: *A machine programm for theorem-proving*, in: Commun. ACM, 5(7) S.394–397, 1962.

[5] de Jonge, M; Visser, J.: *Grammars As Feature Diagrams*, Generative Programming Workshop Austin, Texas, 2002. http://wiki.di.uminho.pt/twiki/bin/view/Personal/Joost/PublicationList

[6] Eén, N.; Sörensson, N.: *An Extensible SAT-solver*, in: Proceedings of the Sixth International Conference on Theory and Applications of Satisfiability Testing, LNCS 2919, pp 502-518, Springer:2003.

[7] Pohl, K.; Böckle, G.; van der Linden, F.: *Software Product Line Engineering: Foundations, Principles, and Techniques*, Springer:2005.

[8] Renz, B.: *The MNI Proposition Analyzer* mpa Technischer Bericht, Fachhochschule Gießen-Friedberg, in Vorbereitung.

[9] Renz, B.: *The MNI Macro Processor* mmp*: An Embeddable, Extensible, M4-Compatible Macro Processor Written in Java* Technischer Bericht, Fachhochschule Gießen-Friedberg, in Vorbereitung.

[10] Le Berre, D. et al.: *SAT4J: Bringing the power of SAT technology to the Java platform*, http://www.sat4j.org

[11] Seindal, René; Pinard, François; Vaughan, Gary V.; Blake, Eric: *GNU M4, version 1.4.11: A powerful macro processor* http://www.gnu.org/software/m4/, 2008.

[12] Sinz, C.; Kaiser, A.; Küchlin,W.: *Formal methods for the validation of automotive product configuration data*, in: Artificial Intelligence for Engineering Design, Analysis and Manufacturing, 17(1): S.75–97, 2003.

Anhang 1: Logelei I

```
/* ## Last written: Fri Aug 08 13:23:12 CEST 2008 */
/*
 * Copyright (c) 2008 by Burkhardt Renz.
 *
 * Logelei1
 * Logeleien des ersten Typs -- direkte Übersetzung in
     Aussagenlogik
 *
 * $Id: Logelei1.mpf 1.1 2008/08/08 11:39:26Z br Exp $
 *
 */
phi::Anne-Bettina-Claudia
/*
Einfache Logelei, aus der ZEIT:

Anna sagt: "Bettina lügt".
Bettina sagt: "Claudia lügt".
Claudia sagt: "Anne und Bettina lügen".
 Wer lügt denn nun?
 */

(a <-> !b) &
(b <-> !c) &
(c <-> !a&!b)

phi::Kuckucksuhren
/*
Logelei aus dem ZEIT Magazin Nr.9/2008 21.Februar 2008

Nach § 3 der Kuckucksuhrenverkaufsverordnung (KucWO) ist an jeder
echten Kuckucksuhr eine wahre Aussage und an jeder unechten
Kuckucksuhr eine falsche Aussage anzubringen.

Herr Huggelhuber hat seine Kuckucksuhren deshalb wie folgt
beschriftet:

Kuckucksuhr Nr. 1:
  »Wenn Kuckucksuhr Nr. 3 echt ist, dann ist auch
  Kuckucksuhr Nr. 2 echt.«
Kuckucksuhr Nr. 2:
  »Wenn Kuckucksuhr Nr. 4 unecht ist, dann ist auch
  Kuckucksuhr Nr. 7 unecht.«
Kuckucksuhr Nr. 3:
  »Wenn Kuckucksuhr Nr. 2 unecht ist, dann ist
  Kuckucksuhr Nr. 7 echt.«
Kuckucksuhr Nr. 4:
  »Wenn Kuckucksuhr Nr. 6 unecht ist, dann ist
  Kuckucksuhr Nr. 7 echt.«
```

```
Kuckucksuhr Nr. 5:
  »Wenn Kuckucksuhr Nr. l echt ist, dann ist
  Kuckucksuhr Nr. 3 unecht.«
Kuckucksuhr Nr. 6:
  »Wenn Kuckucksuhr Nr. 7 unecht ist, dann ist
  Kuckucksuhr Nr. 2 echt.«
Kuckucksuhr Nr. 7:
  »Wenn Kuckucksuhr Nr. 3 unecht ist, dann ist
  Kuckucksuhr Nr. 5 echt.«

Welche Kuckucksuhren sind echt?
*/

[ k1 <-> (k3 -> k2)   ] &
[ k2 <-> (!k4 -> !k7) ] &
[ k3 <-> (!k2 -> k7)  ] &
[ k4 <-> (!k6 -> k7)  ] &
[ k5 <-> (k1 -> !k3)  ] &
[ k6 <-> (!k7 -> k2)  ] &
[ k7 <-> (!k3 -> k5)  ]

phi::Lausbuben
/*
Logelei aus dem ZEIT Magazin Nr.48/2007 22.November 2007

Eines Tages kommt die Katze des Dorflehrers mit einem rosa Hut
auf dem Kopf nach Hause.
Der Lehrer empört: "Das waren bestimmt wieder einige der acht
Lausbuben in meiner Klasse!"
Im Dorf läuft ihm Simon über den Weg, von dem er erfahrt:
"Wenn Detlev an der Aktion beteiligt war, dann auch Rolf!"
Als er gerade den Laden betreten will, sieht er Michel.
Der beteuert: "Wenn Friedl mitgemacht hat, dann hat aber Detlev
nichts mit der Sache zu tun."
Im Laden trifft er Detlev, welcher bekundet: "Das hat Simon
angezettelt."
Auf dem Heimweg begegnet er schließlich Tobias und erfahrt von
diesem: "Klaus hat nichts mit der Sache zu tun!"
Am nächsten Morgen, bevor der Unterricht anfängt, begegnet der
Lehrer im Schulhof Friedl, der sagt:
"Wenn Tobias unschuldig ist, dann war Michel daran beteiligt!"
In der großen Pause nimmt er sich der Reihe nach Klaus und
Rolf vor.
Klaus: "Wenn Rolf dabei war, dann auch Simon!"
Rolf: "Torsten war's!"
Fehlt nur noch Torsten, der muss ohnehin heute nachsitzen.
Eine gute Gelegenheit, um auch ihn zu befragen.
Ergebnis: "Soweit ich weiß, war Klaus einer von denen, die
der Katze den Hut aufgebunden haben."
```

```
Die Schuldigen haben natürlich gelogen, die anderen nicht.
Wer war's?
*/

/*
 lausbub = Lausbub unschuldig,
 !lausbub = lausbub schuldig und Lügner
*/

[ simon <-> (!detlev -> !rolf) ] &
[ michel <-> (!friedl -> detlev) ] &
[ detlev <-> !simon ] &
[ tobias <-> klaus ] &
[ friedl <-> tobias -> !michel ] &
[ klaus <-> !rolf -> !simon ] &
[ rolf <-> !torsten ] &
[ torsten <-> !klaus ]

phi::Unsen
/*
Ein Zeit-Rätsel:

Von den Unsen ist bekannt, dass sie entweder immer lügen oder
immer die Wahrheit sagen.
Ansen, der Magier: "Wenn Honsen, der Medizinmann, lügt, dann
  sagt genau einer von Donsen, dem Wächter, und Insen, dem
  Späher, die Wahrheit."
Bonsen, der Häuptling: "Wenn Consen, der Krieger, und Ensen,
  der Stallbursche, lügen, dann sagt Konsen, der Älteste,
  die Wahrheit."
Consen, der Krieger: "Wenn Konsen, der Älteste, lügt, dann
  lügt auch Donsen, der Wächter."
Donsen, der Wächter: "Wenn Insen, der Späher, die Wahrheit
  sagt, dann lügt entweder Gonsen oder Ensen."
Ensen, der Stallbursche: "Wenn Ansen, der Magier, lügt, dann
  sagt Insen die Wahrheit."
Fonsen, der Jäger: "Wenn Insen, der Späher, und Ansen, der
  Magier, beide die Wahrheit sagen, dann lügt Gonsen, der
  Gärtner."
Gonsen, der Gärtner: "Wenn Insen lügt, dann lügt auch Fonsen."
Honsen, der Medizinmann: "Wenn Ensen die Wahrheit sagt, dann
  sagt auch Donsen,der Wächter, die Wahrheit."
Insen, der Späher: "Wenn Honsen, der Medizinmann, lügt, dann
  sagt Bonsen, der Häuptling, die Wahrheit."
Konsen, der Älteste: "Wenn Ensen, der Stallbursche, lügt,
  dann sagt von Insen, dem Späher, und Ansen, dem Magier,
  genau einer die Wahrheit."
*/

( a <-> (!h -> ((d&!i) | (!d&i))) ) &
```

```
( b <-> ((!c&!e)->k) ) &
( c <-> (!k -> !d) ) &
( d <-> (i -> ((!g&e)|(g&!e))) ) &
( e <-> (!a -> i) ) &
( f <-> ((i&a) -> !g) ) &
( g <-> (!i->!f) ) &
( h <-> (e->d) ) &
( i <-> (!h->b) ) &
( k <-> (!e -> ((i&!a) | (!i&a))) )


phi::Philosophen
/*
Noch ein Zeit-Rätsel:

Von den turbetischen Philosophen ist bekannt, dass sie entweder
immer lügen oder immer die Wahrheit sagen. Zudem nehmen sie an,
dass bei einer »Oder-Aussage« niemals beides der Fall sein kann.

Phil: »Max oder Elke sagt die Wahrheit.«
Tom: »Sven lügt, oder Elke sagt die Wahrheit.«
Mani: »Phil lügt, oder Gert sagt die Wahrheit.«
Chris: »Hol lügt, oder Lu sagt die Wahrheit.«
Elke: »Mani lügt, oder Chris sagt die Wahrheit.«
Bat: »Tanja lügt, oder Hol sagt die Wahrheit.«
Sven: »Tanja oder Chris sagt die Wahrheit.«
Gert: »Hol oder Elke sagt die Wahrheit.«
Tanja: »Bat oder Phil lügt.«
Hol: »Tom oder Chris lügt.«
Max: »Elke oder Chris sagt die Wahrheit.«
Lu: »Hol lügt, oder Elke sagt die Wahrheit.«
*/

( phil <-> (max xor elke) ) &
( tom <-> (!sven xor elke) ) &
( mani <-> (!phil xor gert) ) &
( chris <-> (!hol xor lu) ) &
( elke <-> (!mani xor chris) ) &
( bat <-> (!tanja xor hol) ) &
( sven <-> (tanja xor chris) ) &
( gert <-> (hol xor elke) ) &
( tanja <-> (!bat xor !phil) ) &
( hol <-> (!tom xor !chris) ) &
( max <-> (elke xor chris) ) &
( lu <-> (!hol xor elke) )
```

Anhang 2: Kartenfärbung

```
/* ## Last written: Mon Aug 11 10:26:53 CEST 2008 */
/*
 * Examples of coloring problem
 * $Id: Coloring.mpf 1.1 2008/08/11 08:57:34Z br Exp $
 */
phi::Luxemburg
dnl needs mmp
/*
  Kartenfärbung Luxemburg
  Luxemburg (LU) ist im Uhrzeigersinn umgeben
  von Deutschland (DE), Frankreich (FR) und Belgien (BE),
  die in dieser Reihenfolge miteinander benachbart sind.

  Frage: Kann dieser Kartenauschnitt mit drei Farben gefärbt
  werden?
*/
divert(-1)
include(‘util.mml’)
include(‘logic.mml’)
divert

// Jedes Land hat genau eine Farbe:
foreach(‘c’,‘lu’,‘de’,‘fr’,‘be’,‘oneOf(c‘’_r, c‘’_g, c‘’_b) &
’)

// Benachbarte Länder haben verschiedene Farben:
define(‘benachbart’,‘(foreach(‘c’,‘r’,‘g’,‘b’,‘(!$1_‘’c|!$2_‘’c)
    ifdef(@lastc,‘’,‘ & ’)’))’)
benachbart(lu, de) &
benachbart(lu, fr) &
benachbart(lu, be) &
benachbart(de, fr) &
benachbart(fr, be) &
benachbart(be, de)


phi::Tschechien
dnl needs mmp
/*
  Kartenfärbung Tschechische Republik
  Die Tschechische Republik (CZ) ist im Uhrzeigersinn umgeben
  von Polen (PL), Slowakei (SK), Österreich (AT) und Deutschland (
      DE),
  die in dieser Reihenfolge miteinander benachbart sind.

  Frage: Kann dieser Kartenauschnitt mit drei Farben gefärbt
  werden?
*/
```

```
divert(-1)
include(‘util.mml’)
include(‘logic.mml’)
divert

// Jedes Land hat genau eine Farbe:
foreach(‘c’,‘cz’,‘pl’,‘sk’,‘at’,‘de’,‘oneOf(c‘’_r, c‘’_g, c‘’_b) &
’)

// Benachbarte Länder haben verschiedene Farben:
define(‘benachbart’,‘(foreach(‘c’,‘r’,‘g’,‘b’,‘(!$1_‘’c|!$2_‘’c)
    ifdef(@lastc,‘’,‘ & ’)’))’)
benachbart(cz, pl) &
benachbart(cz, pl) &
benachbart(cz, sk) &
benachbart(cz, at) &
benachbart(cz, de) &
benachbart(pl, sk) &
benachbart(sk, at) &
benachbart(at, de) &
benachbart(de, pl)


phi::Australia
dnl needs mmp
/*
  Coloring Australia map

  Australia has the following states and territories:
  Western Australia (wa)
  Queensland (qld)
  New South Wales (nsw)
  Victoria (vic)
  South Australia (sa)
  Tasmania (tas)
  Northern Territory (nt)
  Australien Capital Territory (act)
  Jevis Bay Territory (jbt) (a peninsula adjacent to nsw)

  Question: can a map of australia be colored with 3 colors?
  (The ocean has a fixed color different from all states.)
*/
divert(-1)
include(‘util.mml’)
include(‘logic.mml’)
divert

// Each state/territory has exactly one color out of r,g,b:
foreach(‘c’,‘wa’,‘qld’,‘nsw’,‘vic’,‘sa’,‘tas’,‘nt’,‘act’,‘jbt’,‘
    oneOf(c‘’_r, c‘’_g, c‘’_b) &
```

```
')

// Adjacent states have different colors:
define(‘adjacent’,‘(foreach(‘c’,‘r’,‘g’,‘b’,‘(!$1_‘’c|!$2_‘’c)ifdef
    (@lastc,‘’,‘ & ’)’))’)

adjacent(‘wa’,‘nt’) &
adjacent(‘wa’,‘sa’) &
adjacent(‘nt’,‘sa’) &
adjacent(‘nt’,‘qld’) &
adjacent(‘qld’,‘sa’) &
adjacent(‘qld’,‘nsw’) &
adjacent(‘nsw’,‘sa’) &
adjacent(‘nsw’,‘act’) &
adjacent(‘nsw’,‘jbt’) &
adjacent(‘vic’,‘sa’) &
adjacent(‘vic’,‘nsw’)

phi::US-3
dnl needs mmp
/*
  Coloring U.S. map

  Question: can a map of the U.S. be colored with 3 colors?
  (We only consider North America without Alaska.)

*/
divert(-1)
include(‘util.mml’)
include(‘logic.mml’)
divert

// Each state has exactly one color out of r, g, b:
foreach(‘c’,‘al’,‘ak’,‘az’,‘ar’,‘ca’,‘co’,‘ct’,‘de’,‘fl’,‘ga’,‘id
    ’,‘il’,‘in’,‘ia’,‘ks’,‘ky’,‘la’,‘me’,‘md’,‘ma’,‘mi’,‘mn’,‘ms’,‘
    mo’,‘mt’,‘ne’,‘nv’,‘nh’,‘nj’,‘nm’,‘ny’,‘nc’,‘nd’,‘oh’,‘ok’,‘or
    ’,‘pa’,‘ri’,‘sc’,‘tn’,‘tx’,‘ut’,‘vt’,‘va’,‘wa’,‘wv’,‘wi’,‘wy’,‘
    oneOf(c‘’_r, c‘’_g, c‘’_b) &
’)

// Adjacent states have different colors:
define(‘adjacent’,‘(foreach(‘c’,‘r’,‘g’,‘b’,‘(!$1_‘’c|!$2_‘’c)ifdef
    (@lastc,‘’,‘ & ’)’))’)

adjacent(‘wa’,‘or’) & adjacent(‘wa’,‘id’) & adjacent(‘or’,‘id’) &
adjacent(‘or’,‘nv’) & adjacent(‘or’,‘ca’) & adjacent(‘ca’,‘nv’) &
adjacent(‘ca’,‘az’) & adjacent(‘az’,‘nm’) & adjacent(‘az’,‘nv’) &
adjacent(‘az’,‘ut’) & adjacent(‘nv’,‘ut’) & adjacent(‘nv’,‘id’) &
adjacent(‘id’,‘mt’) & adjacent(‘id’,‘ut’) & adjacent(‘id’,‘wy’) &
adjacent(‘ut’,‘wy’) & adjacent(‘ut’,‘co’) & adjacent(‘nm’,‘tx’) &
```

```
adjacent(‘nm’,‘ok’) & adjacent(‘nm’,‘co’) & adjacent(‘co’,‘ok’) &
adjacent(‘co’,‘ks’) & adjacent(‘co’,‘ne’) & adjacent(‘co’,‘wy’) &
adjacent(‘wy’,‘ne’) & adjacent(‘wy’,‘sd’) & adjacent(‘wy’,‘mt’) &
adjacent(‘mt’,‘sd’) & adjacent(‘mt’,‘nd’) & adjacent(‘nd’,‘sd’) &
adjacent(‘nd’,‘mn’) & adjacent(‘sd’,‘ne’) & adjacent(‘sd’,‘ia’) &
adjacent(‘sd’,‘mn’) & adjacent(‘ne’,‘ks’) & adjacent(‘ne’,‘mo’) &
adjacent(‘ne’,‘ia’) & adjacent(‘ks’,‘ok’) & adjacent(‘ks’,‘mo’) &
adjacent(‘ok’,‘tx’) & adjacent(‘ok’,‘ar’) & adjacent(‘ok’,‘mo’) &
adjacent(‘tx’,‘la’) & adjacent(‘la’,‘ms’) & adjacent(‘la’,‘ar’) &
adjacent(‘ar’,‘ms’) & adjacent(‘ar’,‘tn’) & adjacent(‘ar’,‘mo’) &
adjacent(‘mo’,‘tn’) & adjacent(‘mo’,‘ky’) & adjacent(‘mo’,‘il’) &
adjacent(‘mo’,‘ia’) & adjacent(‘ia’,‘il’) & adjacent(‘ia’,‘wi’) &
adjacent(‘ia’,‘mn’) & adjacent(‘mn’,‘wi’) & adjacent(‘wi’,‘il’) &
adjacent(‘wi’,‘mi’) & adjacent(‘il’,‘ky’) & adjacent(‘il’,‘in’) &
adjacent(‘ms’,‘al’) & adjacent(‘ms’,‘tn’) & adjacent(‘al’,‘fl’) &
adjacent(‘al’,‘ga’) & adjacent(‘tn’,‘ga’) & adjacent(‘tn’,‘nc’) &
adjacent(‘tn’,‘va’) & adjacent(‘tn’,‘ky’) & adjacent(‘ky’,‘va’) &
adjacent(‘ky’,‘wv’) & adjacent(‘ky’,‘oh’) & adjacent(‘ky’,‘in’) &
adjacent(‘in’,‘oh’) & adjacent(‘in’,‘mi’) & adjacent(‘oh’,‘pa’) &
adjacent(‘oh’,‘wv’) & adjacent(‘wv’,‘pa’) & adjacent(‘wv’,‘md’) &
adjacent(‘wv’,‘va’) & adjacent(‘va’,‘md’) & adjacent(‘va’,‘dc’) &
adjacent(‘vc’,‘nc’) & adjacent(‘nc’,‘sc’) & adjacent(‘sc’,‘ga’) &
adjacent(‘ga’,‘fl’) & adjacent(‘dc’,‘md’) & adjacent(‘md’,‘de’) &
adjacent(‘md’,‘pa’) & adjacent(‘de’,‘pa’) & adjacent(‘de’,‘nj’) &
adjacent(‘nj’,‘pa’) & adjacent(‘nj’,‘ny’) & adjacent(‘pa’,‘ny’) &
adjacent(‘ny’,‘ct’) & adjacent(‘ny’,‘ma’) & adjacent(‘ny’,‘vt’) &
adjacent(‘ct’,‘ma’) & adjacent(‘ct’,‘ri’) & adjacent(‘ma’,‘ri’) &
adjacent(‘ma’,‘vt’) & adjacent(‘ma’,‘nh’) & adjacent(‘vt’,‘nh’) &
adjacent(‘nh’,‘me’)


phi::US-4
dnl needs mmp
/*
  Coloring U.S. map

  Question: can a map of the U.S. be colored with 4 colors?
  (We only consider north america without alaska.)
  Surely -- the 4-color theorem says so!
*/
divert(-1)
include(‘util.mml’)
include(‘logic.mml’)
divert

// Each state has exactly one color out of r, g, b, w:
foreach(‘c’,‘al’,‘ak’,‘az’,‘ar’,‘ca’,‘co’,‘ct’,‘de’,‘fl’,‘ga’,‘id
    ’,‘il’,‘in’,‘ia’,‘ks’,‘ky’,‘la’,‘me’,‘md’,‘ma’,‘mi’,‘mn’,‘ms’,‘
    mo’,‘mt’,‘ne’,‘nv’,‘nh’,‘nj’,‘nm’,‘ny’,‘nc’,‘nd’,‘oh’,‘ok’,‘or
    ’,‘pa’,‘ri’,‘sc’,‘tn’,‘tx’,‘ut’,‘vt’,‘va’,‘wa’,‘wv’,‘wi’,‘wy’,‘
```

```
    oneOf(c‘’_r, c‘’_g, c‘’_b,c‘’_w) &
’)

// Adjacent states have different colors:
define(‘adjacent’,‘(foreach(‘c’,‘r’,‘g’,‘b’,‘w’,‘(!$1_‘’c|!$2_‘’c)
    ifdef(@lastc,‘’,‘ & ’)’))’)

adjacent(‘wa’,‘or’) & adjacent(‘wa’,‘id’) & adjacent(‘or’,‘id’) &
adjacent(‘or’,‘nv’) & adjacent(‘or’,‘ca’) & adjacent(‘ca’,‘nv’) &
adjacent(‘ca’,‘az’) & adjacent(‘az’,‘nm’) & adjacent(‘az’,‘nv’) &
adjacent(‘az’,‘ut’) & adjacent(‘nv’,‘ut’) & adjacent(‘nv’,‘id’) &
adjacent(‘id’,‘mt’) & adjacent(‘id’,‘ut’) & adjacent(‘id’,‘wy’) &
adjacent(‘ut’,‘wy’) & adjacent(‘ut’,‘co’) & adjacent(‘nm’,‘tx’) &
adjacent(‘nm’,‘ok’) & adjacent(‘nm’,‘co’) & adjacent(‘co’,‘ok’) &
adjacent(‘co’,‘ks’) & adjacent(‘co’,‘ne’) & adjacent(‘co’,‘wy’) &
adjacent(‘wy’,‘ne’) & adjacent(‘wy’,‘sd’) & adjacent(‘wy’,‘mt’) &
adjacent(‘mt’,‘sd’) & adjacent(‘mt’,‘nd’) & adjacent(‘nd’,‘sd’) &
adjacent(‘nd’,‘mn’) & adjacent(‘sd’,‘ne’) & adjacent(‘sd’,‘ia’) &
adjacent(‘sd’,‘mn’) & adjacent(‘ne’,‘ks’) & adjacent(‘ne’,‘mo’) &
adjacent(‘ne’,‘ia’) & adjacent(‘ks’,‘ok’) & adjacent(‘ks’,‘mo’) &
adjacent(‘ok’,‘tx’) & adjacent(‘ok’,‘ar’) & adjacent(‘ok’,‘mo’) &
adjacent(‘tx’,‘la’) & adjacent(‘la’,‘ms’) & adjacent(‘la’,‘ar’) &
adjacent(‘ar’,‘ms’) & adjacent(‘ar’,‘tn’) & adjacent(‘ar’,‘mo’) &
adjacent(‘mo’,‘tn’) & adjacent(‘mo’,‘ky’) & adjacent(‘mo’,‘il’) &
adjacent(‘mo’,‘ia’) & adjacent(‘ia’,‘il’) & adjacent(‘ia’,‘wi’) &
adjacent(‘ia’,‘mn’) & adjacent(‘mn’,‘wi’) & adjacent(‘wi’,‘il’) &
adjacent(‘wi’,‘mi’) & adjacent(‘il’,‘ky’) & adjacent(‘il’,‘in’) &
adjacent(‘ms’,‘al’) & adjacent(‘ms’,‘tn’) & adjacent(‘al’,‘fl’) &
adjacent(‘al’,‘ga’) & adjacent(‘tn’,‘ga’) & adjacent(‘tn’,‘nc’) &
adjacent(‘tn’,‘va’) & adjacent(‘tn’,‘ky’) & adjacent(‘ky’,‘va’) &
adjacent(‘ky’,‘wv’) & adjacent(‘ky’,‘oh’) & adjacent(‘ky’,‘in’) &
adjacent(‘in’,‘oh’) & adjacent(‘in’,‘mi’) & adjacent(‘oh’,‘pa’) &
adjacent(‘oh’,‘wv’) & adjacent(‘wv’,‘pa’) & adjacent(‘wv’,‘md’) &
adjacent(‘wv’,‘va’) & adjacent(‘va’,‘md’) & adjacent(‘va’,‘dc’) &
adjacent(‘vc’,‘nc’) & adjacent(‘nc’,‘sc’) & adjacent(‘sc’,‘ga’) &
adjacent(‘ga’,‘fl’) & adjacent(‘dc’,‘md’) & adjacent(‘md’,‘de’) &
adjacent(‘md’,‘pa’) & adjacent(‘de’,‘pa’) & adjacent(‘de’,‘nj’) &
adjacent(‘nj’,‘pa’) & adjacent(‘nj’,‘ny’) & adjacent(‘pa’,‘ny’) &
adjacent(‘ny’,‘ct’) & adjacent(‘ny’,‘ma’) & adjacent(‘ny’,‘vt’) &
adjacent(‘ct’,‘ma’) & adjacent(‘ct’,‘ri’) & adjacent(‘ma’,‘ri’) &
adjacent(‘ma’,‘vt’) & adjacent(‘ma’,‘nh’) & adjacent(‘vt’,‘nh’) &
adjacent(‘nh’,‘me’)
```

Anhang 3: Logelei II

```
/* ## Last written: Mon Aug 11 14:53:13 CEST 2008 */


phi::Einstein
dnl needs mmp
divert(-1)
/*

Albert Einstein verfasste (angeblich) dieses Rätsel, von dem er ↩
    behauptete,
dass 98% der Weltbevölkerung nicht in der Lage seien, es zu lösen.
Gehören Sie zu den 2%? Viel Spaß beim Ausprobieren!

Es gibt fünf Häuser mit je einer anderen Farbe.
In jedem Haus wohnt eine Person einer anderen Nationalität.
Jeder Hausbewohner bevorzugt ein bestimmtes Getränk,
raucht eine bestimmte Zigarettenmarke und hält ein bestimmtes ↩
    Haustier.
Keine der 5 Personen trinkt das gleiche Getränk,
raucht die gleichen Zigaretten oder hält das gleiche Tier
wie einer seiner Nachbarn.

Frage: Wer besitzt einen Fisch?

Bekannt sind folgende Tatsachen:

- Der Brite lebt im roten Haus.
- Der Schwede hält einen Hund.
- Der Däne trinkt gerne Tee.
- Das grüne Haus steht links vom weißen Haus.
- Der Besitzer des grünen Hauses trinkt Kaffee.
- Die Person, die Pall Mall raucht, hält einen Vogel.
- Der Mann, der im mittleren Haus wohnt, trinkt Milch.
- Der Besitzer des gelben Hauses raucht Dunhill.
- Der Norweger wohnt im ersten Haus.
- Der Marlboro-Raucher wohnt neben dem, der eine Katze hält.
- Der Mann, der ein Pferd hält, wohnt neben dem, der Dunhill raucht ↩
    .
- Der Winfield-Raucher trinkt gerne Bier.
- Der Norweger wohnt neben dem blauen Haus.
- Der Deutsche raucht Rothmanns.
- Der Marlboro-Raucher hat einen Nachbarn, der Wasser trinkt

*/
divert(-1)
include(‘util.mml’)dnl
include(‘logic.mml’)dnl
```

```
/* Kodierung der Situation:
Wir nehmen fünf Personen pi im Haus mit der Position i
und jeweils die 5 Ausprägungen der
Attribute: Nationalität, Haustier, Farbe des Hauses,
Zigarettenmarke, Getränk. Wir schreiben zum Beispiel p1.brite. Es ist
also p1.brite genau dann wahr, wenn die Person im ersten Haus Brite ist.
*/

/* Zunächst brauchen  wir Formeln, die ausdrücken, dass jede Person nur jeweils
genau eine Ausprägung der sechs Attribute hat.
*/
divert
define(‘einAttr’,‘for(‘i’,‘1’,‘5’,‘oneOf(p‘’i.$1,p‘’i.$2,p‘’i.$3,p‘’i.$4,p‘’i.$5)ifdef(@lasti,‘’,‘ & ’) ’)’) dnl
einAttr(‘f_rot’,‘f_gruen’,‘f_gelb’,‘f_blau’,‘f_weiß’) &
einAttr(‘n_brite’,‘n_daene’,‘n_deutscher’,‘n_norweger’,‘n_schwede’) &
einAttr(‘t_fisch’,‘t_hund’,‘t_katze’,‘t_pferd’,‘t_vogel’) &
einAttr(‘z_dunhill’,‘z_marlboro’,‘z_pallmall’,‘z_rothmanns’,‘z_winfield’) &
einAttr(‘g_bier’,‘g_kaffee’,‘g_milch’,‘g_tee’,‘g_wasser’)

divert(-1)
/* Nun müssen wir ausdrücken, dass die Werte eines Attributs für alle Personen
paarweise verschieden sind.
*/
divert
&
define(‘different’,‘foreach(‘a’,$1,$2,$3,$4,$5,‘maxOneOf(p1.a,p2.a,p3.a,p4.a,p5.a)ifdef(@lasta,‘’,‘& ’) ’)’)
different(‘f_rot’,‘f_gruen’,‘f_gelb’,‘f_blau’,‘f_weiß’) &
different(‘n_brite’,‘n_daene’,‘n_deutscher’,‘n_norweger’,‘n_schwede’) &
different(‘t_fisch’,‘t_hund’,‘t_katze’,‘t_pferd’,‘t_vogel’) &
different(‘z_dunhill’,‘z_marlboro’,‘z_pallmall’,‘z_rothmanns’,‘z_winfield’) &
different(‘g_bier’,‘g_kaffee’,‘g_milch’,‘g_tee’,‘g_wasser’)

/* Hilfsmakros:
   Ein und dieselbe Person hat zwei Eigenschaften
   ‘gleich’(a1,a2)
   Benachbarte Personen haben zwei Eigenschaften
   ‘benachbart’(a1,a2)
   Eine Person in einem Haus links dem einer anderen haben zwei Eigenschaften
```

```
   ‘links’(a1,a2)
*/
define(‘gleich’,‘for(‘i’,‘1’,‘5’,‘(p‘’i.$1 -> p‘’i.$2)ifdef(@lasti
    ,‘’,‘&’)’)’)
define(‘benachbart’,‘(p1.$1 -> p2.$2) & (p2.$1 -> (p1.$2|p3.$2)) &
(p3.$1 -> (p2.$2|p3.$2)) & (p4.$1 -> (p3.$2|p5.$2)) & (p5.$1 -> (p4
    .$2))’)
define(‘links’,‘(p1.$1 -> (p2.$2|p3.$2|p4.$2)) &
(p2.$1 -> (p3.$2|p4.$2|p5.$2)) & (p3.$1 -> (p4.$2|p5.$2)) & (p4.$1
    -> p5.$2)’)

/* Nun kommen die Vorgaben aus der Aufgabenstellung
*/
&
// Der Brite lebt im roten Haus
gleich(‘n_brite’,‘f_rot’) &
// Der Schwede hält einen Hund
gleich(‘n_schwede’,‘t_hund’) &
// Der Däne trinkt gerne Tee
gleich(‘n_daene’,‘g_tee’) &
// Das grüne Haus steht ‘links’ vom weißen Haus
links(f_gruen,f_weiß) &
// Der Besitzer des grünen Hauses trinkt Kaffee
gleich(‘f_gruen’,‘g_kaffee’) &
// Die Person, die Pall Mall raucht, hält einen Vogel
gleich(‘z_pallmall’,‘t_vogel’) &
// Der Mann, der im mittleren Haus wohnt, trinkt Milch
p3.g_milch &
// Der Besitzer des gelben Hauses raucht Dunhill
gleich(‘f_gelb’,‘z_dunhill’) &
// Der Norweger wohnt im ersten Haus
p1.n_norweger &
// Der Marlboro-Raucher wohnt neben dem, der eine Katze hält
benachbart(‘z_marlboro’,‘t_katze’) &
// Der Mann, der ein Pferd hält, wohnt nebem dem, der Dunhill
    raucht
benachbart(‘t_pferd’,‘z_dunhill’) &
// Der Winfield-Raucher trinkt gerne Bier
gleich(‘z_winfield’,‘g_bier’) &
// Der Norweger wohnt neben dem blauen Haus
benachbart(‘n_norweger’,‘f_blau’) &
// Der Deutsche raucht Rothmanns
gleich(‘n_deutscher’,‘z_rothmanns’) &
// der Marlboro-Raucher hat einen Nachbarn, der Wasser trinkt
benachbart(‘z_marlboro’,‘g_wasser’)

/*
Ergebnis:
The proposition is satisfiable.
```

```
A model is:
Valuation
 true  p1.f_gelb
 true  p1.g_wasser
 true  p1.n_norweger
 true  p1.t_katze
 true  p1.z_dunhill

 true  p2.f_blau
 true  p2.g_tee
 true  p2.n_daene
 true  p2.t_pferd
 true  p2.z_marlboro

 true  p3.f_rot
 true  p3.g_milch
 true  p3.n_brite
 true  p3.t_vogel
 true  p3.z_pallmall

 true  p4.f_gruen
 true  p4.g_kaffee
 true  p4.n_deutscher
 true  p4.t_fisch
 true  p4.z_rothmanns

 true  p5.f_weiß
 true  p5.g_bier
 true  p5.n_schwede
 true  p5.t_hund
 true  p5.z_winfield
*/



phi::Dschungel
dnl needs mmp
/*
Logelei aus dem ZEIT Magazin Nr.39/2007 20.September 2007

Fabian ist ein neuer Wildpfleger im Dschungel.
Er unterhält sich gerade mit seinem Mentor Hans-Friederich,
einem erfahrenen Wildhüter: "Woran erkennt man eigentlich
die unterschiedlichen Wildtiere im Dschungel?"
"Jedes macht ein ganz bestimmtes Geräusch. Das ist ganz einfach.
Pass auf:
Falls Fiedertiger surren, miauen Beutelaffen.
Falls Fiedertiger miauen, zischen Tentakelläufer.
Falls Fiedertiger kratzen, oinken Beutelaffen.
Falls Fiedertiger zischen, kratzen Ohrhörner.
```

```
Falls Fiedertiger pfeifen, miauen Rüsselratten.
Falls Fiedertiger oinken, pfeifen Tentakelläufer."

"Und was ist mit den anderen Tieren?", will Fabian wissen.
"Da ist es so:
Falls Ohrhörner miauen, kratzen Tentakelläufer.
Falls Beutelaffen miauen, kratzen Fiedertiger.
Falls Rüsselratten surren, dann zischen Ohrhörner.
Falls Nagekühe miauen, pfeifen Ohrhörner.
Falls Beutelaffen oinken, kratzen Fiedertiger.
Falls Ohrhörner surren, kratzen Tentakelläufer.
Falls Beutelaffen surren, zischen Nagekühe.
Falls Nagekühe surren, kratzen Fiedertiger.
Falls Rüsselratten miauen, dann surren Ohrhörner.
Falls Beutelaffen oinken, surren Ohrhörner."

Welches Tier macht welches Geräusch?
*/
divert(-1)
include(‘logic.mml’)
divert

define(genauEins,‘oneOf($1.surren,$1.miauen,$1.kratzen,$1.zischen,
    $1.pfeifen,$1.oinken)’)dnl

genauEins(Fiedertiger) &
genauEins(Beutelaffen) &
genauEins(Tentakelläufer) &
genauEins(Ohrhörner) &
genauEins(Rüsselratten) &
genauEins(Nagekühe)
&

(Fiedertiger.surren ->   Beutelaffen.miauen) &
(Fiedertiger.miauen ->   Tentakelläufer.zischen) &
(Fiedertiger.kratzen ->  Beutelaffen.oinken) &
(Fiedertiger.zischen ->  Ohrhörner.kratzen) &
(Fiedertiger.pfeifen ->  Rüsselratten.miauen) &
(Fiedertiger.oinken ->   Tentakelläufer.pfeifen) &
(Ohrhörner.miauen ->     Tentakelläufer.kratzen) &
(Beutelaffen.miauen ->   Fiedertiger.kratzen) &
(Rüsselratten.surren ->  Ohrhörner.zischen) &
(Nagekühe.miauen ->      Ohrhörner.pfeifen) &
(Beutelaffen.oinken ->   Fiedertiger.kratzen) &
(Ohrhörner.surren ->     Tentakelläufer.kratzen) &
(Beutelaffen.surren ->   Nagekühe.zischen) &
(Nagekühe.surren ->      Fiedertiger.kratzen) &
(Rüsselratten.miauen ->  Ohrhörner.surren) &
(Beutelaffen.oinken ->   Ohrhörner.surren)
```

```
phi::Filme
dnl needs mmp
/*
Logelei aus dem ZEIT Magazin Nr.44/2007 25.Oktober 2007

Vier Nachwuchsregisseure haben kürzlich je einen Film gedreht.
Folgendes ist bekannt:
Entweder spielt der Film von Garsten im Himalaya, oder Garsten
  heißt mit Nachnamen Guckenbauer.
Herr Fliegemond drehte entweder keinen Dokumentarfilm oder keinen
  Film, der in den Rheinauen spielt.
Wenn Roland mit Nachnamen Hammerschmidt heißt, dann heißt Martin
  mit Nachnamen Fliegemond.
Entweder wurde die Komödie von Martin gedreht oder der Thriller
  von Roland.
Entweder spielt die Komödie nicht in den Rheinauen, oder
  Fliegemond hat seinen Film nicht in der Innenstadt von Godia
  gedreht.
Wenn Roland mit Nachnamen Fliegemond heißt, dann heißt Peter
  Guckenbauer.
Der Science-Fiction-Film spielt nicht in den Rheinauen.
Entweder Herr Hammerschmidt oder Herr Bochmann hat die Komödie
  gedreht.
Entweder hat Garsten den Film gedreht, der in der Sierra Nevada
  spielt, oder Martin hat nicht die Komödie gedreht.
Wenn Martin mit Nachnamen Hammerschmidt heißt, dann heißt Peter mit
  Nachnamen Fliegemond.
Entweder spielt der Film von Martin nicht in den Rheinauen, oder
  aber er heißt mit Nachnamen nicht Hammerschmidt.
*/
divert(-1)
include(‘util.mml’)
include(‘logic.mml’)
divert

/* Kodierung: Wir haben vier Personen mit vier Attributen und
   je vier Werten: p1.genre.dokumentar z.B. bedeutet, dass
   Person p1 das Genre Dokumentarfilm hat.
*/

/* Zunächst muss man sicherstellen, dass jeweils nur eine
   Eigenschaft wahr ist.
*/
define(‘oneOfFour’,‘for(‘i’,‘1’,‘4’,‘oneOf(p‘’i.$1,p‘’i.$2,p‘’i.$3,
    p‘’i.$4) &’)’) dnl
oneOfFour(genre.dokumentar, genre.komödie, genre.scifi, genre.
    thriller)
oneOfFour(nachname.bochmann, nachname.fliegemond, nachname.
    guckenbauer, nachname.hammerschmidt)
```

```
oneOfFour(ort.godia, ort.himalya, ort.rheinauen, ort.sierra)
oneOfFour(vorname.garsten, vorname.martin, vorname.peter, vorname.
    roland)

/* Dann müssen die Personen paarweise verschiedene Eigenschaften
    haben
*/
define(‘different’,‘foreach(‘a’,$1,$2,$3,$4,‘maxOneOf(p1.a,p2.a,p3.
    a,p4.a) & ’)’)
different(genre.dokumentar, genre.komödie, genre.scifi, genre.
    thriller)
different(nachname.bochmann, nachname.fliegemond, nachname.
    guckenbauer, nachname.hammerschmidt)
different(ort.godia, ort.himalya, ort.rheinauen, ort.sierra)
different(vorname.garsten, vorname.martin, vorname.peter, vorname.
    roland)

/* Hilfsmakro: Zwei Eigenschaften zu derselben Person
*/
define(gleich,‘((p1.$1 <-> p1.$2) & (p2.$1 <-> p2.$2) & (p3.$1 <->
    p3.$2) &
(p4.$1 <-> p4.$2))’)dnl

[ gleich(vorname.garsten, ort.himalya) xor
  gleich(vorname.garsten, nachname.guckenbauer) ] &
[ !gleich(nachname.fliegemond, genre.dokumentar) xor
  !gleich(nachname.fliegemond, ort.rheinauen) ] &
[ gleich(vorname.roland, nachname.hammerschmidt) ->
  gleich(vorname.martin, nachname.fliegemond) ] &
[ gleich(vorname.martin, genre.komödie) xor
  gleich(vorname.roland, genre.thriller) ] &
[ !gleich(genre.komödie, ort.rheinauen) xor
  !gleich(nachname.fliegemond, ort.godia) ] &
[ gleich(vorname.roland, nachname.fliegemond) ->
  gleich(vorname.peter, nachname.guckenbauer) ] &
[ !gleich(genre.scifi, ort.rheinauen) ] &
[ gleich(nachname.hammerschmidt, genre.komödie) xor
  gleich(nachname.bochmann, genre.komödie) ] &
[ gleich(vorname.garsten, ort.sierra) xor
  !gleich(vorname.martin, genre.komödie) ] &
[ gleich(vorname.martin, nachname.hammerschmidt) ->
  gleich(vorname.peter, nachname.fliegemond) ] &
[ gleich(vorname.peter, ort.rheinauen) xor
  !gleich(vorname.peter, nachname.hammerschmidt) ]

/* Ergebnis:
   Roland Bachmann dreht Thriller im Himalaya.
   Garsten Guckenbauer dreht Science Fiction in Godia.
   Peter Hammerschmidt dreht Komöodie in den Rheinauen.
   Martin Fliegemond dreht Dokumentarfilm in der Sierra.
```

```
*/

phi::Spielkarten
dnl needs mmp
divert(-1)
include(‘util.mml’)
include(‘logic.mml’)
divert
/*
Rätsel aus dem ZEIT-Magazin Nr.15 3.April 2008

Professor Knusi ist zu Gast in der Welt der Spielkarten.  Kürzlich
lagen einige Bildkarten (Asse, Könige, Damen und Buben) am Strand.
Bei einer Party erzählen einige der Karten Professor Flusi davon.
Aber nur diejenigen Karten, die damals am Strand dabei waren,
sagen die Wahrheit, alle anderen lügen.

Kreuzass: »Weder Pikbube noch Karobube waren dabei.«
Pikass: »Von jeder der vier Farben war mindestens eine Karte am
  Strand.«
Herzass: »Am Strand lagen genauso viele Asse wie Buben.«
Pikkönig: »Mindestens eine der fünf Karten war eine Herz-Karte und
  hatte ein Ass und eine Dame als direkte Nachbarn.«
Herzkönig: »Die mittlere der fünf Karten war ein Bube.«
Kreuzdame: »Genau eine der fünf Karten hatte die Farbe Kreuz.«
Herzdame: »Die beiden Karten links und rechts außen gehörten zur
  Karo-Farbe.«
Karodame: »Unter den fünf Karten fanden sich zwei, die direkt
  nebeneinanderlagen, welche die gleiche Farbe hatten.«
Kreuzbube: »Die zweite Karte von links war der Pikkönig.«
Pikbube: »Herzkönig und Herzdame lagen beide am Strand, direkt
  nebeneinander.«
Karobube: »Sowohl Kreuzass als auch Kreuzkönig waren am Strand.«

Welche fünf Karten lagen in welcher Reihenfolge am Strand?
*/

/* Kodierung:
   pos.farbe und pos.wert
*/

// Regel1: jede Position ist von genau einer Farbe und einem Wert ↩
    belegt

// Jede Position hat genau eine Farbe
for(‘i’,1,5,‘oneOf(p‘’i.kreuz,p‘’i.pik,p‘’i.herz,p‘’i.karo) &’)

// Jede Position hat genau einen Wert
for(‘i’,1,5,‘oneOf(p‘’i.ass,p‘’i.koenig,p‘’i.dame,p‘’i.bube) &’)
```

```
// Regel2: eine Kombination von Farbe und Wert kann nur einmal
    auftreten

dnl posHatKarte(pos, farbe, wert)
define(posHatKarte,‘($1.$2 & $1.$3)’) dnl

dnl karteEinmal(farbe, wert)
define(karteEinmal,( (posHatKarte(p1, $1, $2) -> (!posHatKarte(p2,
    $1, $2) & !posHatKarte(p3, $1, $2) & !posHatKarte(p4, $1, $2) &
     !posHatKarte(p5, $1, $2))) &
(posHatKarte(p2, $1, $2) -> (!posHatKarte(p1, $1, $2) & !
    posHatKarte(p3, $1, $2) & !posHatKarte(p4, $1, $2) & !
    posHatKarte(p5, $1, $2) )) &
(posHatKarte(p3, $1, $2) -> (!posHatKarte(p1, $1, $2) & !
    posHatKarte(p2, $1, $2) & !posHatKarte(p4, $1, $2) & !
    posHatKarte(p5, $1, $2) )) &
(posHatKarte(p4, $1, $2) -> (!posHatKarte(p1, $1, $2) & !
    posHatKarte(p2, $1, $2) & !posHatKarte(p3, $1, $2) & !
    posHatKarte(p5, $1, $2) )) &
(posHatKarte(p5, $1, $2) -> (!posHatKarte(p1, $1, $2) & !
    posHatKarte(p2, $1,
$2) & !posHatKarte(p3, $1, $2) & !posHatKarte(p4, $1, $2) )) )) dnl

foreach(‘f’,‘kreuz’,‘pik’,‘herz’,‘karo’,‘foreach(‘w’,‘ass’,‘koenig
    ’,‘dame’,‘bube’,‘karteEinmal(f,w) &’)’)

// Hilfsmakro
dnl dabei( farbe, wert )
define(dabei,‘((p1.$1 & p1.$2) | (p2.$1 & p2.$2) | (p3.$1 & p3.$2 )
     | (p4.$1
& p4.$2) | (p5.$1 & p5.$2))’)dnl

// Kreuzass: »Weder Pikbube noch Karobube waren ‘dabei’.«
[ dabei(kreuz, ass) <-> !dabei(pik, bube) & !dabei(karo, bube)] &

// Pikass: »Von jeder der vier Farben war mindestens eine Karte am
    Strand.«
define(minEine,‘(dabei($1, ass) | dabei($1, koenig) | dabei($1,
    dame) |
dabei($1, bube))’)dnl
[ dabei(pik, ass) <-> minEine(kreuz) & minEine(pik) & minEine(herz)
     & minEine(karo)] &

define(‘genauEin’,‘oneOf(p1.$1,p2.$1,p3.$1,p4.$1,p5.$1)’)
define(‘genauZwei’,‘[(p1.$1 & p2.$1 & !p3.$1 & !p4.$1 & !p5.$1) |
(p1.$1 & !p2.$1 & p3.$1 & !p4.$1 & !p5.$1) |
(p1.$1 & !p2.$1 & !p3.$1 & p4.$1 & !p5.$1) |
(p1.$1 & !p2.$1 & !p3.$1 & !p4.$1 & p5.$1) |
(!p1.$1 & p2.$1 & p3.$1 & !p4.$1 & !p5.$1) |
(!p1.$1 & p2.$1 & !p3.$1 & p4.$1 & !p5.$1) |
```

```
(!p1.$1 & p2.$1 & !p3.$1 & !p4.$1 & p5.$1) |
(!p1.$1 & !p2.$1 & p3.$1 & p4.$1 & !p5.$1) |
(!p1.$1 & !p2.$1 & p3.$1 & !p4.$1 & p5.$1) |
(!p1.$1 & !p2.$1 & !p3.$1 & p4.$1 & p5.$1)]') dnl

// Herzass: »Am Strand lagen genauso viele Asse wie Buben.«
[ dabei(herz, ass) <-> ( (genauEin(ass) & genauEin(bube)) | (
    genauZwei(ass) &
genauZwei(bube)) ) ] &

dnl Hilfsmakro
define(inMitte,`($1.ass & $2.herz & $3.dame)') dnl
// Pikkönig: »Mindestens eine der fünf Karten war eine Herz-Karte
    und
//            hatte ein Ass und eine Dame als direkte Nachbarn.«
[ dabei(pik, koenig) <-> ( inMitte(p1, p2, p3) | inMitte(p2, p3, p4
    ) |
inMitte(p3, p4, p5)) ] &

// Herzkönig: »Die mittlere der fünf Karten war ein Bube.«
[ dabei(herz, koenig) <-> (p3.bube) ] &


// Kreuzdame: »Genau eine der fünf Karten hatte die Farbe Kreuz.«
[ dabei(kreuz, dame) <-> genauEin(kreuz) ] &

// Herzdame: »Die beiden Karten links und rechts außen gehörten zur
     Karo-Farbe.«
[ dabei(herz, dame) <-> (p1.karo & p5.karo) ] &

dnl gleicheFarbe(pos1, pos2)
define(gleicheFarbe,`(($1.kreuz & $2.kreuz) | ($1.pik & $2.pik) |
($1.herz & $2.herz) | ($1.karo & $2.karo))') dnl
// Karodame: »Unter den fünf Karten fanden sich zwei, die direkt
//            nebeneinanderlagen, welche die gleiche Farbe hatten.«
[ dabei(karo, dame) <-> ( gleicheFarbe(p1, p2) | gleicheFarbe(p2,
    p3) |
  gleicheFarbe(p3, p4) | gleicheFarbe(p4, p5) )] &

// Kreuzbube: »Die zweite Karte von links war der Pikkönig.«
[  dabei(kreuz, bube) <-> (p2.pik & p2.koenig) ] &

dnl Hilfsmakro
define(aufPos,`(($1.herz & $1.koenig & $2.herz & $2.dame) |
 ($2.herz & $2.koenig & $1.herz & $1.dame))') dnl
// Pikbube: »Herzkönig und Herzdame lagen beide am Strand, direkt
    nebeneinander.«
[ dabei(pik, bube) <-> ( aufPos(p1, p2) | aufPos(p2, p3) | aufPos(
    p3, p4) | aufPos(p4, p5) ) ] &
```

```
// Karobube: »Sowohl Kreuzass als auch Kreuzkönig waren am Strand.«
[  dabei(karo, bube) <-> dabei(kreuz, ass) & dabei(kreuz, koenig)
   ]

/* Ergebnis:
   Kreuzbube, Pikkönig, Kreuzass, Herzbube, Pikdame
*/
```

Anhang 4: Sudoku

Die Regeln für Sudoku:

```
divert(-1)
/*
 * Rules of sudoku
 * $Id: Sudoku.m4 1.1 2008/08/13 06:54:44Z br Exp $
 */
include(‘util.mml’)dnl
include(‘logic.mml’)dnl
/*
 * Encoding of sudoku:
 * 1. cijk means that cell (i,j) has value k
 * 2. Each cell has exactly one value (Rule1)
 * 3. The values of each row, each column and each region
 *    are pairwise different (Rule2)
 */
divert
// Rule1
for(‘i’,‘1’,‘9’,‘for(‘j’,‘1’,‘9’,‘ifdef(‘@lastj’,‘oneOf(for(‘k
    ’,‘1’,‘9’,‘c‘’i‘’j‘’k‘’ifdef(‘@lastk’,‘’,‘,’)’))
’,‘oneOf(for(‘k’,‘1’,‘9’,‘c‘’i‘’j‘’k‘’ifdef(‘@lastk’,‘’,‘,’)’)) &
’)’) &’)
// Rule2
define(‘different’,‘for(‘x’,‘1’,‘9’,‘maxOneOf($1‘’x,$2‘’x,$3‘’x,$4
    ‘’x,$5‘’x,$6‘’x,$7‘’x,$8‘’x,$9‘’x)ifdef(‘@lastx’,‘’, &‘’)’)’)
// rows
for(‘i’,‘1’,‘9’,‘different(for(‘j’,‘1’,‘9’,‘c‘’i‘’j‘’,’)) &
’)
// columns
for(‘j’,‘1’,‘9’,‘different(for(‘i’,‘1’,‘9’,‘c‘’i‘’j‘’,’)) &
’)
// regions
different(‘c11’,‘c12’,‘c13’,‘c21’,‘c22’,‘c23’,‘c31’,‘c32’,‘c33’) &
different(‘c41’,‘c42’,‘c43’,‘c51’,‘c52’,‘c53’,‘c61’,‘c62’,‘c63’) &
different(‘c71’,‘c72’,‘c73’,‘c81’,‘c82’,‘c83’,‘c91’,‘c92’,‘c93’) &

different(‘c14’,‘c15’,‘c16’,‘c24’,‘c25’,‘c26’,‘c34’,‘c35’,‘c36’) &
different(‘c44’,‘c45’,‘c46’,‘c54’,‘c55’,‘c56’,‘c64’,‘c65’,‘c66’) &
different(‘c74’,‘c75’,‘c76’,‘c84’,‘c85’,‘c86’,‘c94’,‘c95’,‘c96’) &

different(‘c17’,‘c18’,‘c19’,‘c27’,‘c28’,‘c29’,‘c37’,‘c38’,‘c39’) &
different(‘c47’,‘c48’,‘c49’,‘c57’,‘c58’,‘c59’,‘c67’,‘c68’,‘c69’) &
different(‘c77’,‘c78’,‘c79’,‘c87’,‘c88’,‘c89’,‘c97’,‘c98’,‘c99’)
```

Beispiele:

```
/* ## Last written: Thu Aug 07 11:35:44 CEST 2008 */
/*
 * This is the default header of a new MpaFile
 */
```

```
phi::Zeit-32-2008
dnl needs mmp
/*
 * Sudoku aus der Zeit Nr. 32 2008
 */
// Regeln
include(‘sudoku.m4’)
// und vorgegebene Werte
& c122 & c134 & c173 & c188
& c216 & c237 & c242 & c269 & c271 & c294
& c318 & c321 & c347 & c363 & c389 & c396
& c424 & c438 & c479 & c487
& c626 & c639 & c675 & c681
& c717 & c725 & c749 & c768 & c784 & c791
& c814 & c831 & c846 & c865 & c877 & c899
& c929 & c936 & c978 & c983

phi::iht-2008-08-02
dnl needs mmp
/*
 * Sudoku aus der International Herald Tribune vom 2. August 2008
 */
// Regeln
include(‘sudoku.m4’)
// Die Sudoku in der International Herald Tribune haben
// vier weitere Regionen, in denen 1 - 9 vorkommen sollen
& different(‘c22’,‘c23’,‘c24’,‘c32’,‘c33’,‘c34’,‘c42’,‘c43’,‘c44’)
& different(‘c62’,‘c63’,‘c64’,‘c72’,‘c73’,‘c74’,‘c82’,‘c83’,‘c84’)
& different(‘c26’,‘c27’,‘c28’,‘c36’,‘c37’,‘c38’,‘c46’,‘c47’,‘c48’)
& different(‘c66’,‘c67’,‘c68’,‘c76’,‘c77’,‘c78’,‘c86’,‘c87’,‘c88’)

// und vorgegebene Werte
& c166
& c219 & c292
& c348 & c354 & c385
& c476
& c534
& c623 & c675
& c732 & c744 & c791
& c825 & c867 & c874
& c952 & c973
```

Anhang 5: Variabilitätsmodell

```
/* ## Last written: Thu Aug 21 10:41:17 CEST 2008 */
/*
 * Beispiele für Variabilitätsmodelle
 */

phi::Automobil
dnl needs mmp
/* Example from
   Bühne, Lauenroth, Pohl 2004
*/
divert(-1)
include(‘util.mml’)
include(‘logic.mml’)
include(‘varmod.mml’)
divert

vm(
‘vp(‘Type’,‘alt(‘StationWagon’,‘Cabrio’)’)’,
‘vp(‘Light’,‘man(‘LightManualControl’)’,‘opt(‘LightSensorControl’)
    ’)’,
‘vp(‘Wiper’,‘man(‘Wash’)’,‘alt(‘IntervalWipe’,‘RainLightSensor’)’)
    ’,
‘vp(‘RoofControl’,‘man(‘RoofManualControl’)’,‘opt(‘
   RoofSensorControl’)’)’,
‘req(‘StationWagon’,‘Light’,‘Wiper’)’,
‘req(‘Cabrio’,‘Light’,‘Wiper’,‘RoofControl’)’,
‘req(‘LightSensorControl’,‘RainLightSensor’)’,
‘req(‘RoofSensorControl’,‘RainLightSensor’)’,
)

phi::Automobil2
dnl needs mmp
/* Example from
   Bühne, Lauenroth, Pohl 2004 -- extended
*/
divert(-1)
include(‘util.mml’)
include(‘logic.mml’)
include(‘varmod.mml’)
divert

vm(
‘vp(‘Type’,‘alt(‘StationWagon’,‘Cabrio’)’)’,
‘vp(‘Light’,‘man(‘LightManualControl’)’,‘opt(‘LightSensorControl’)
    ’)’,
‘vp(‘Wiper’,‘man(‘Wash’)’,‘alt(‘IntervalWipe’,‘RainLightSensor’)’)
    ’,
```

```
‘vp(‘RoofControl’,‘man(‘RoofManualControl’)’,‘opt(‘ ↩
    RoofSensorControl’)’)’,
‘req(‘StationWagon’,‘Light’,‘Wiper’)’,
‘excl(‘StationWagon’,‘RoofControl’)’,
‘req(‘Cabrio’,‘Light’,‘Wiper’,‘RoofControl’)’,
‘req(‘LightSensorControl’,‘RainLightSensor’)’,
‘req(‘RoofSensorControl’,‘RainLightSensor’)’,
)

phi::Ungültiges Modell
dnl needs mmp
/* Modell, das offensichtlich ungültig ist
*/
divert(-1)
include(‘util.mml’)
include(‘logic.mml’)
include(‘varmod.mml’)
divert

vm(
‘vp(‘vp1’,‘alt(v1,v2)’)’,
‘req(v1,v2)’
)
&
v1
```