SUDOC2 – Grundlagen zum Sudoku-Programm

Arnold Schönhage

Januar 2007

Vorwort. Nach mancherlei praktischer Übung beim Lösen zahlreicher Sudoku-Probleme "von Hand" (meist im Format $(3 \times 3) \times (3 \times 3)$ alias 9×9, zuweilen aber auch 12×12) sollen die dabei gelernten Techniken hier nun formalisiert beschrieben und danach auch programmiert werden. Neben dem Lösen vorgegebener Probleme wird damit dann auch die Generierung neuer eigener Probleme ermöglicht.

1 Datenstrukturen, Kodierungen

Das Spielfeld (hier am Beispiel von 9×9 *Stellen* erläutert) wird mit Zeilennummern ("r" für *rows*) $r'' = 0, \ldots, 8$ – ternär $r'' = r + 3{\cdot}r'$ $(r < 3)$ – und Spaltennummern $s'' = 0, \ldots, 8$ – ternär $s'' = s + 3{\cdot}s'$ $(s < 3)$ – koordinatisiert. Entsprechend haben die *Felder* $F(r', s')$ – hier im Format von je 3×3 Stellen – die Paare (r, s) als lokale Koordinaten. Einzutragende Ziffern (wie auch Zeichen anderer Art) werden hier mit $n = 1, \ldots, 9$ numeriert, die aktuellen Zeichen werden $z(n)$ genannt, z. B. die Standardsymbole $z(1) = \boxed{1}, \ldots, z(9) = \boxed{9}$.

Zu jedem Feld $F(r', s')$ und jedem n werden (jeweils abhängig vom Wissensstand, also von der Programmlaufzeit) die Mengen $Q_n(r', s')$ der Paare (r, s) gespeichert, für die Eintragung von $z(n)$ an der Stelle $(r'', s'') = (r + 3{\cdot}r', s + 3{\cdot}s')$ derzeit (noch) nicht ausgeschlossen werden kann; wir kodieren diese Mengen $Q_n(r', s')$ durch Binärworte

$$q_n(r', s') = \sum \{2^{r+3s} \mid (r, s) \in Q_n(r', s')\}, \tag{1}$$

also mit Einser-Bits als Indikatoren. Die Elementezahl solcher Mengen ist dann durch die binäre Quersumme $|w|$ ihrer kodierenden Worte $w = q_n(r', s')$ gegeben.

Ein Beispiel: Im Südfeld $F(2, 1)$ sei schon bekannt, daß die 7 (alias $z(7)$) im rechts gezeigten Spielfeld nur noch an den mittels $*$ markierten Stellen stehen kann, und $5 = z(5)$ nur noch an den mit $+$ markierten Stellen. Dann sind $q_7(2, 1) = 1 + 2 + 4 + 32 = 39$ und $q_5(2, 1) = 8 + 128 = 136$ die dafür nach Regel (1) zu speichernden Binärworte. Die hier schon *fixierten* Ziffern 1 und 3 sind so mit $q_1(2, 1) = 16$ und $q_3(2, 1) = 256$ kodiert.

	$s' = 0$	$s' = 1$	$s' = 2$
$r' = 0$	· · · / · · · / · · ·	· · · / · · · / · · ·	· · · / · · · / · · ·
$r' = 1$	· · · / · · · / · · ·	· · · / · · · / · · ·	· · · / · · · / · · ·
$r' = 2$	· · · / · · · / · · ·	$*$ + · / $*$ 1 + / $*$ $*$ 3	· · · / · · · / · · ·

Die in den $q_n(r', s')$ enthaltene Information könnte man auch zeilen- oder spaltenweise kodieren, ganz analog in Binärworten für entsprechend gedachte Mengen. In einer ersten Version habe ich solches mit passenden Umrechnungsformeln probiert. Weil sich diese Versuche in den Anwendungen aber als wenig effizient erwiesen, soll diese Richtung hier jetzt nicht mehr weiter verfolgt werden.

2 Inferenzen

Ohne jedes Vorwissen sind anfangs alle $q_n(r',s') = 511 = \sum_{t<9} 2^t$ zu setzen, da jedes $z(n)$ ja noch überall stehen könnte. Diesen Anfangszustand nennen wir das *leere Muster*. Bei fixierten Vorgaben einzelner Ziffern sind diese Werte dann jedoch entsprechend zu ändern. Soll in der Südwestecke (im Feld $F(2,0)$ links unten) z. B. eine $7 = z(7)$ stehen, dann impliziert das zuerst einmal $q_7(2,0) = 4$. Dieser neue Wert erlaubt dann aber (nach den Sudoku-Regeln und mit unseren Kodierungen) auch eine Reihe von Folgerungen, wobei solche *Inferenzen* zu weiteren Änderungen der q-Werte führen können, diese ihrerseits zu neuen Werten usw. rekursiv wie bei umfallenden Dominosteinen. In jenem Fall erhält man so $q_n(2,0) = 511 - 4 = 507$ für alle $n \neq 7$; diesen Schluß wollen wir *Verdrängung* nennen: an Stellen, wo eindeutig $z(7)$ steht, kann kein anderes Zeichen stehen.

Ferner folgt hier, daß in der letzten Zeile keine weitere 7 stehen kann, man also in $q_7(2,1)$ wie auch in $q_7(2,2)$ Einsen in den Bitpositionen 2,5,8 löschen sollte, was hier die neuen Werte $q_7(2,1) = q_7(2,2) = 511 - 4 - 32 - 256 = 219$ liefert. Diese Inferenz nennen wir *Zeilung,* den analogen Schluß für Spalten werden wir *Spaltung* nennen, hier die erste Spalte betreffend: weil in dieser keine weitere 7 stehen kann, führt das entsprechend zu den neuen Werten $q_7(0,0) = q_7(1,0) = 511 - 7 = 504$.

Zur formalen Behandlung solcher Schritte werden auf diese Bitmuster q_n Boole'sche Operationen *and* und *or* bitweise angewandt, die wir im Hinblick auf die entsprechenden Mengenoperationen als $\cap$ und $\cup$ schreiben. Im Einklang mit der in (1) vereinbarten (rechts angedeuteten) Numerierung der Binärstellen je Feld $F(r',s')$ verwenden wir "Zeilenmasken" rm_i und "Spaltenmasken" sm_i, die mit Bitfolgen (*from low to high,* so auch H xx in hex) so definiert werden,

0	3	6
1	4	7
2	5	8

$$
\begin{aligned}
rm_0 &= \texttt{1oo1oo1oo}\ (\texttt{H 940}), & sm_0 &= \texttt{111ooooo}\ (\texttt{H 700}),\\
rm_1 &= \texttt{o1oo1oo1o}\ (\texttt{H 290}), & sm_1 &= \texttt{ooo111ooo}\ (\texttt{H 830}),\\
rm_2 &= \texttt{oo1oo1oo1}\ (\texttt{H 421}), & sm_2 &= \texttt{oooooo111}\ (\texttt{H 0C1}),
\end{aligned}
\tag{2}
$$

2.1 Verdrängungen

Verdrängung durch eine fixierte Ziffer ($q_n(r',s')$ ist Zweierpotenz) wie zuvor im Beispiel beschrieben ist nur ein Spezialfall. Allgemein können auch zwei oder drei (oder gar mehr) Stellen eines Feldes $F(r',s')$, einer Zeile oder einer Spalte als *voll* erkannt werden, mit zwei bzw. drei der Ziffern (in noch unbekannter Reihenfolge) besetzt, so daß andere Ziffern dort nicht mehr eingetragen werden können.

Im Fall von zwei Stellen eines $F(r',s')$ wird das nun so formalisiert: für Indizes $k \neq n$ gelte $q_k(r',s') = q_n(r',s') = b$ mit Quersumme $|b| = 2$. Dann kann für alle j mit $j \neq k,n$ $u = q_j(r',s')$ durch $v = u \setminus b$ ersetzt werden, wobei $a \setminus b = a - (a \cap b)$ die Bildung von Differenzmengen kodiert.

Im Fall von drei Stellen eines $F(r',s')$ wird das etwas komplizierter. Für 3 Indizes $k \neq m$, $n \neq k,m$ sei $b := q_k(r',s') \cup q_m(r',s') \cup q_n(r',s')$ und $|b| = 3$. Dann kann für alle anderen Indizes j $u = q_j(r',s')$ durch $v = u \setminus b$ ersetzt werden.

Wenn Möglichkeiten der ein- und zweistelligen Verdrängungen vorher schon ausgelotet wurden, kann man sich bei der Suche nach diesen dreistelligen Möglichkeiten auf Paare $k \neq n$ mit $|q_i(r',s')| = 2$ oder $= 3$ für $i = k,n$, bei denen ferner $|b| = 3$ schon für $b = q_k(r',s') \cup q_n(r',s')$ gilt, beschränken. — Mehr als dreistellig werden wir dieses 'voll'-Argument nicht benutzen.

2.2 Zeilungen

(1) Falls $(r,s) \in Q_n(r',s')$ den Index r schon eindeutig bestimmt, nennen wir $z(n)$ (oder auch n) im Feld $F(r',s')$ *r-fixiert* (das umfaßt dann auch den Spezialfall $|q_n(r',s')| = 1$). Technisch gesehen ist das mit $b = rm_r$ äquivalent zu $b \cap q_n(r',s') = q_n(r',s')$. Damit ist Eintragung von $z(n)$ in der übrigen Zeile (r,r') ausgeschlossen, also kann für alle $t \neq s'$ (in den zeilig benachbarten Feldern) $u = q_j(r',t)$ durch $v = u \smallsetminus b$ ersetzt werden.

(2) Das Vorangehende läßt sich so verallgemeinern: falls $(r,s) \in Q_n(r',s') \cup Q_n(r',t)$ für ein $t \neq s'$ mit nur zwei r-Werten r und $\hat{r}$ gilt, dann kann damit Eintragung von $z(n)$ in den übrigen Teilen der beiden Zeilen (r,r') und $(\hat{r},r')$ ausgeschlossen werden.

2.3 Spaltungen

(1) Falls $(r,s) \in Q_n(r',s')$ den Index s schon eindeutig bestimmt, nennen wir $z(n)$ (oder auch n) im Feld $F(r',s')$ *s-fixiert* (das umfaßt wieder auch den Fall $|q_n(r',s')| = 1$, bei dem n auch *rs-fixiert* genannt werden könnte). Technisch ist das mit $b = sm_s$ äquivalent zu $b \cap q_n(r',s') = q_n(r',s')$. Damit ist Eintragung von $z(n)$ in den übrigen Stellen der Spalte (s,s') unmöglich, also kann man für alle $t \neq r'$ (in den Feldern darüber/darunter) $u = q_j(t,s')$ durch $v = u \smallsetminus b$ ersetzen.

(2) Auch hier entsprechend verallgemeinert: falls $(r,r') \in Q_n(r',s') \cup Q_n(t,s')$ für ein $t \neq r'$ mit nur zwei s-Werten s und $\hat{s}$ gilt, dann kann damit Eintragung von $z(n)$ in den übrigen Teilen der beiden Spalten (s,s') und $(\hat{s},s')$ ausgeschlossen werden.

2.4 Inferenz-Items

Datentechnisch organisieren wir den am Anfang dieses Abschnitts angedeuteten Domino-Effekt als *Schlange* D (FIFO, first in—first out), die temporär *Items* $I = (d,n,w)$ enthält. Darin bezeichnet d gekoppelt mit dem Zeichenindex n die Datenadresse eines $q_n(r',s')$ und w das bei Entstehung dieses Items dort neu gespeicherte Binärwort. Anfangs wird das leere Muster generiert und der (anfangs leeren) Schlange D für jede Ziffern-Vorgabe "$z(n)$ in Position (r,r',s,s')" ein Item mit (d,n) für $q_n(r',s')$ und $w = 2^{r+3s}$ zugefügt und $q_n(r',s') = w$ im Muster gespeichert.

Dann wird die Hauptschleife gestartet, in der die Bearbeitung des nächsten Items $I = (d,n,w)$ vom Kopf der Schlange weitere neue Items erzeugen kann, die wieder (hinten an D) anzufügen sind. Die Adresse (d,n) zeigt dabei auf ein $q_n(r',s')$, und dessen derzeit aktueller Wert sei $q_n(r',s') = a$. Dann ist vorab zu prüfen, ob immer noch $a = w$ gilt; sollte nämlich das bei Entstehung dieses Items I dort gespeicherte w inzwischen durch ein $a < w$ (wobei zudem immer auch $|a| < |w|$ gelten wird) ersetzt worden sein, dann wird D noch ein späteres, diesbezüglich "besseres" Item $J = (d,n,a)$ enthalten, so daß die weitere Bearbeitung des jetzt anstehenden I ohne Nachteil ausgelassen werden kann.

3 Bearbeitung der Items

Falls beim zur Bearbeitung anstehenden Item $|w| = 1$ gilt, erlaubt das die Verdrängung, für alle $j \neq n$ jeweils $u = q_j(r',s')$ durch $v := u \smallsetminus w$ zu ersetzen, und (d,j,v) als neues Item anzufügen, das aber nur, sofern $v < u$ gilt !! – Ferner erlaubt $|w| = 1$ auch die Zeilung nach 2.2(1), dann für alle $t \neq s'$ $u = q_n(r',t)$ durch $v = u \smallsetminus b$ mit passendem $b = rm_r$ zu ersetzen, und im Falle von $v < u$ als neues Item (d',n,v) anzufügen, worin (d',n) dann auf $q_n(r',t)$ zeigt, und zudem auch die Spaltung nach 2.3(1), für die Indizes $t \neq r'$ $u = q_n(t,s')$ durch $v = u \smallsetminus b$ (mit passendem $b = sm_s$) zu ersetzen, und im Falle von $v < u$ ein entsprechendes neues Items (d',n,v) anzufügen.

Falls beim anstehenden Item $|w|=2$ gilt, wird zuerst geprüft, ob damit eine zweistellige Verdrängung (wie in 2.1) möglich ist; es wird also nach einem $k \neq n$ mit $q_k(r',s') = w$ gesucht und dann gegebenenfalls für die anderen Indizes $j \neq k,n$ $u = q_j(r',s')$ durch $v := u \setminus w$ ersetzt, im Falle $v < u$ mit jeweils neuem Item (d,j,v). Falls es kein solches k gibt, wird dann nach vielleicht möglicher dreistelliger Verdrängung gesucht, wie am Ende von 2.1 schon angedeutet, und im Erfolgsfall wieder ganz ähnlich verfahren. Ferner erlaubt $|w|=2$ vielleicht auch die Zeilung nach 2.2(1) oder andernfalls vielleicht die Spaltung nach 2.3(1), wieder mit Auswirkungen wie zuvor schon bei $|w|=1$ beschrieben.

Falls beim anstehenden Item $|w|=3$ gilt, wird zuerst geprüft, ob damit eine dreistellige Verdrängung möglich ist, deren Wirkungen dann gegebenenfalls zu erledigen sind. Ferner erlaubt auch solch w mit $|w|=3$ vielleicht die Zeilung nach 2.2(1) oder die Spaltung nach 2.3(1), ebenfalls mit Auswirkungen wie zuvor schon bei $|w|=1$ beschrieben.

Außerdem wollen wir Inferenzmöglichkeiten nach 2.2(2) – und analog auch nach 2.3(2) – in folgender Weise ausnutzen: Wenn n in $F(r',s')$ nicht r-fixiert ist, also $|w|=2$ oder $|w|=3$ gilt, aber keine Zeilung nach 2.2(1) möglich ist, oder mit $|w|>3$ dann aber

$$\exists\, r<3:\ b\cap w=0 \quad \text{für} \quad b=rm_r \tag{3}$$

gilt (das bestimmt r übrigens eindeutig), dann wird geprüft, ob $b\cap q_n(r',t)=0$ auch für eines der zwei anderen zeilig benachbarten Felder $F(r',t)$ mit $t \neq s'$ gilt. Wenn ja, dann kann im dritten Feld, für $t'=3-t-s'$, nach 2.2(2) $u=q_n(r',t')$ durch $v=b\cap u$ ersetzt werden, bei $v<u$ mit Anfügung eines entsprechenden neuen Items (d',n,v).

Für Spalten ganz analog: Ist n in $F(r',s')$ nicht s-fixiert, gilt aber

$$\exists\, s<3:\ b\cap w=0 \quad \text{für} \quad b=sm_s \tag{4}$$

dann wird geprüft, ob $b\cap q_n(t,s')=0$ auch für ein $t \neq r'$ gilt. Falls ja, dann kann nach 2.3(2) für $t'=3-t-r'$ $u=q_n(t',s')$ durch $v=b\cap u$ ersetzt werden, mit Anfügung eines entsprechenden neuen Items (d',n,v), sofern $v<u$.

Da (3) und (4) für $|w|>6$ nicht gelten, werden nach dem Katalog der vorstehend beschriebenen Aktionen alle Items mit $|w|>6$ ganz ohne Wirkung bleiben. Man könnte deshalb das Anfügen neuer Items (d',j,v) mit $|v|>6$ sogleich unterlassen, ebenso auch (vorab noch genauer prüfend), wenn bei $|v|>3$ für $w=v$ weder (3) noch (4) gelten. Wir können bei unserer Implementierung auf diese Verfeinerung jedoch verzichten, denn allein die Bedingung $|v|<|u|$ garantiert ja schon, daß die Zahl der neu hinzuzufügenden Items insgesamt $< 9^3 = 729$ bleiben wird.

4 Die Hauptroutine INFER

Im Zentrum unserer Algorithmen steht eine häufig auch rekursiv anzuwendende Routine *INFER*. Deren Eingabeparameter bestehen aus einem Muster M_0, einer Schlange D_0 diesbezüglicher Inferenz-Items und einer hilfsweise hinzugefügten Anzahl $f=f(M_0)$ der in M_0 bisher schon **f**ixierten Zeichen, also der Tripel (n,r',s') mit $|q_n(r',s')|=1$. Ausgehend von dieser *Konfiguration* $K_0=(M_0,D_0)$, auch Anfangskonfiguration genannt, soll *INFER* ggf. "die" nach den Sudoku-Regeln eindeutig komplettierte *Lösung* als finales Muster M' mit $f(M')=81$ liefern, dann mit dem Ergebnis $INFER(K_0)=e=1$, und sonst (mit Ergebnis $e=0$) sagen, daß es keine Lösung gibt, oder mit Ergebnis $e=2$, daß es mehrere zulässige Lösungen gibt, und das dann mit Spezifikation einer Stelle (r'',s'') nebst darin möglicher Ziffern $k \neq n$ belegen.

Soll z. B. eine der üblichen Aufgaben im 9×9 Format mit g vorgegebenen Ziffern gelöst werden, dann wird man *INFER* mit einer Anfangskonfiguration aufrufen, wie sie im ersten Absatz von 3.4 beschriebenen wurde – und bei der dann jedenfalls $f = g$ gelten wird. – Die Arbeit von *INFER* beginnt damit, die anfangs gegebene Schlange $D = D_0$ abzuarbeiten, wie in Abschnitt 4 beschrieben. Technisch gesehen wird dabei das jeweils aktuelle Wort $q_n(r', s')$ unter der Adresse

$$dn = r' + 4{\cdot}s' + 16{\cdot}n \qquad (16 \le dn < 160 \ \text{ für } \ n \le 9) \tag{5}$$

gespeichert, so speziell günstig für TP-Implementierung gewählt, und r', s', n lassen sich daraus leicht durch Shifts gewinnen. Erzeugt eine Inferenz ein neues Item (dn, v) mit $v = 0$, dann impliziert das die Antwort, daß es zur Anfangskonfiguration $K_0 = (M_0, D_0)$ keine Lösung gibt, $e = e(K_0) = 0$. Bei jedem neuen Item mit $1 = |v| < |u|$ wird f um 1 erhöht. Ist die Schlange D schließlich leer und $f = 81$, dann hat K_0 genau eine Lösung, die mit dem nun erreichten Endmuster M' explizit gegeben ist.

4.1 Rekursive Anwendung von INFER

Ist D zwar leer, aber immer noch $f < 81$, dann wendet *INFER* auf das bisher erreichte Muster M die Methode der Fallunterscheidung an: es gibt dann ja r', s' und n, so daß $u = q_n(r', s')$ Quersumme $|u| > 1$ hat, und hierin sei v die dem höchsten Bit von u entsprechende Zweierpotenz. Für das weitere mag es hinsichtlich Rechenzeitbedarf günstig sein, r', s', n im übrigen so zu wählen, daß $|u| \ge 2$ minimal wird, so vielleicht $|u| = 2$.

Änderung von u zu $q_n(r', s') = v$ macht aus M ein Muster M_1, und dazu gehöre die Schlange D_1, die nur das eine Item (d, n, v) enthält. Analog zu dieser neuen Anfangskonfiguration $K_1 = (M_1, D_1)$ mit $f_1 = f + 1$ führt $v' := u \setminus v$ anstelle von v zur Bildung einer alternativen Konfiguration $K_2 = (M_2, D_2)$, dann mit $f_2 = f + 1$, falls auch $|v'| = 1$, sonst $f_2 = f$. Die gewünschte Entscheidung wird nun erreicht, indem sich *INFER* ein- oder zweifach selbst aufruft: Falls $e_1 = e(K_1) = 2$ mit möglichen Ziffern $k \ne n$ in Stelle (r'', s''), dann passen diese auch zum Ergebnis $e = 2$. Sonst wird (nach zweitem Aufruf) bei $e_1 = 0$ einfach $e := e_2 = e(K_2)$ weitergereicht, mit Endmuster $M' = M'_2$ bei $e_2 = 1$ oder möglichen Ziffern $k \ne n$ in Stelle (r'', s'') mit $e_2 = 2$, letzteres auch im Falle $e_1 = 1$. Übrig bleiben die Fälle $e_2 = 0$ mit $e := e_1 = 1$ nebst zugehörigem Endmuster $M' = M'_1$ sowie schließlich noch $e_1 = e_2 = 1$, dann mit dem Ergebnis $e = 2$, wozu die Darstellung $q_n(r', s') = v = 2^{r+3s}$ des in K_1 benutzten Items die gewünschte Mehrdeutigkeits-Stelle vermöge $r'' = r + 3{\cdot}r'$, $s'' = s + 3{\cdot}s'$ liefert und die Bedingung $q_k(r', s') = v$ in dem zu K_2 gefundenen Endmuster M'_2 jenes $k \ne n$ bestimmt.

Natürlich würde diese *brute force* Methode auch bei ganz einfachen Inferenz-Techniken immer schon zum Ziel führen, dann allerdings vermutlich mit stark wachsender Rechenzeit, während mit unseren feineren Techniken die Rekursionstiefe solcher Baumsuche relativ klein zu bleiben verspricht. Um solches zu dokumentieren, speichert *INFER* zusätzliche Daten für Diagnosezwecke ab, die global zur Ausgabe des Baums der rekursiven Aufrufe benutzt werden können. – Derartige Beispiele folgen in Abschnitt 5.

4.2 Weitere Implementierungs-Details

Items (d, n, w) werden mit Kopplung zu dn wie in (5) als je ein Wort $dn + 256{\cdot}w$ gespeichert, mit dem Anfang der Schlange D als Stapel auf Band T_1 und dem Ende von D umgekehrt auf Band T_2, beide mit Stopper 0 unterlegt.

Um für die Mengen kodierenden Binärworte $u = q_n(r', s')$ deren Quersumme $|u|$ und sonstige Eigenschaften (z. B. r-fixiert oder s-fixiert zu sein) und damit mögliche Aktionen

schnell zu ermitteln, benutzt *INFER* eine Tabelle QA von 512 Worten, die mit u als Index gelesen wird. Diese Worte $A(u)$ sind nach folgendem Schema in Bitfelder untergliedert,

(6) $$A(u) = \boxed{\quad h:8 \quad | \quad rm:9 \quad |rx:3| \quad sm:9 \quad |sx:3}$$

wobei $:\ell$ deren Längen nennt, mit $h = |u|$ im unteren Byte, darüber ein 12-bit Feld zur Angabe möglicher Zeilungen, sofern $rm \neq 0$ gilt, was dann die jeweils passende Zeilenmaske $b = rm_r$ angibt, bei extra Flag $rx = 4$ passend in (3) für den Modus 2.2(2) oder andernfalls $rx = 0$ für Zeilungen nach 2.2(1). Analog im oberen 12-bit Feld die entsprechenden Daten zu möglichen Spaltungen, falls $sm \neq 0$ gilt, mit passender Spaltenmaske $b = sm_s$, bei $sx = 4$ für den Modus 2.3(2), bei $sx = 0$ für Spaltungen nach 2.3(1).

Bei rekursivem Aufruf wird M_1 oberhalb von M_0 (auf Band T_0) gespeichert und im Falle von $e(K_1) = 1$ die Lösung M_1' auf Band T_3 zwischengelagert, um gegebenenfalls später, sofern $e(K_2) = 0$ (mit untauglichem M_2') folgt, schließlich an die Stelle des Anfangsmusters M_0 (wieder auf T_0) gespeichert zu werden.

Vor seinem Rücksprung speichert *INFER* jeweils auf Band T_5 in acht 32-bit Worten folgende Diagnosedaten ab. Das erste Wort ist $j + 2h$ (kurz *jh* genannt), das *INFER* bei jedem Aufruf als Parameter mit auf den Weg gegeben wird (ganz außen mit $jh = 1$), um so die Lesbarkeit der hier dokumentierten Baumstruktur der rekursiven Aufrufe zu erleichtern: bei den Aufrufen nach Abschnitt 4.1 wird *INFER* dann $jh_1 = 2(h+1)$ bzw. $jh_2 = 1 + 2(h+1)$ an seine Söhne weiterreichen. So dokumentiert h die "Höhe" und $j = 0, 1$ jeweils eine der Verzweigungs-Alternativen – das jedoch nur *lokal!* – Es folgen das Ergebnis e, anstelle von $e = 2$ jedoch r'', s'', k, n als vier Bytes im gleichen Wort und in zwei weiteren Worten Anfangs- und Endwert von f, danach im Falle rekursiver Verzweigung die Größen *dn* als 2 hex Ziffern gemäß (5) und das in v und v' gespaltene Binärwort u, dessen 9 Bits als *oct* mit drei Oktalziffern angegeben werden. Im Beispiel auf Seite 1 (mit $n = 7$ für 4-fach $*$) wären das also $dn = 67$ (hex) und $oct =$ `740`. – Bei Endknoten des Baums sind $dn =$ E und $oct =$ E als Endemarken zu lesen.

Schließlich möchte man auch die (interne) Verteilung der Rechenzeiten sehen, die hier bei unserer Implementierung in TP time units 'tu' gemessen werden. Für jeden Knoten des Baums wird deshalb in den letzten zwei Diagnoseworten die 'totale Zeit' (kurz *tt*) für das entsprechende Teilproblem (von Aufruf bis Rücksprung) und zusätzlich auch die *Nettozeit* mitgeteilt, wobei letztere (bei Verzweigungen) als *tt* abzüglich der totalen Zeiten *tt1* und *tt2* für die Subprobleme ermittelt wird.

Zur Umrechnung solcher Zeiten in Sekunden (für heutige PC's) sei hier erwähnt, daß mein Anfang 2005 gekaufter Pentium 4 (zu 2400 MHz) ca. 1200 Mtu/sec leistet, bei dieser TP Simulation also 1 tu etwas weniger als eine Nanosekunde dauert.

5 Einige Beispiele

Im letzten November schickte mir mein Sohn Bodo ein kniffliges Sudoku-Problem, wie es hier rechts gezeigt ist. Beim Lösen von Hand kam ich nach wenigen Inferenz-Schritten der üblichen Art, die das anfängliche $f_0 = 24$ auf $f = 30$ fixierte Ziffern erhöhten und auch schon allerlei andere Einengungen lieferten, an einen "toten Punkt", wo nur noch die Methode der Fallunterscheidung zu helfen schien, die dann doch recht zäh verlief.

```
| - - - | - - - | 5 - 1 |
| - - - | - 4 - | - 6 - |
| 6 2 5 | 3 - - | - - - |
|-------+-------+-------|
| - - 1 | 7 - 2 | - - - |
| - - - | 4 - 6 | - 8 3 |
| - - 7 | - - - | - - - |
|-------+-------+-------|
| - 8 3 | - - - | - - 9 |
| 7 - - | - 5 - | 4 - - |
| - - - | 9 - - | 1 - - |
```

Anwendung von *INFER* auf dieses Problem, nebst entsprechend gestaltetem Rahmenprogramm für Ein- und Ausgabe, zeigt nun, weshalb das so mühevoll war. Das folgende Protokoll beschreibt einen Baum der Höhe 5 mit 11 Aufrufen, versehen mit all den Daten und Rechenzeiten wie zuvor erklärt, und rechts daneben ist die beim neunten Aufruf ($jh = 8$) schließlich erreichte eindeutige Lösung angegeben.

jh	*e*	$f_0 \to f$		*dn*	*oct*	*tt*	*Netto*
1	1	24	30	99	500	142909	42545
3	1	31	31	89	050	86698	8169
5	0	32	49	E	E	11803	11803
4	1	32	37	59	044	66726	11100
7	1	38	38	49	003	48845	8596
9	0	39	39	29	022	22290	5520
11	0	40	51	E	E	6405	6405
10	0	40	62	E	E	10365	10365
8	1	39	81	E	E	17959	17959
6	0	38	54	E	E	6781	6781
2	0	31	61	E	E	13666	13666

```
| 9 3 4 | 2 6 8 | 5 7 1 |
| 1 7 8 | 5 4 9 | 3 6 2 |
| 6 2 5 | 3 7 1 | 8 9 4 |
|-------+-------+-------|
| 8 6 1 | 7 3 2 | 9 4 5 |
| 5 9 2 | 4 1 6 | 7 8 3 |
| 3 4 7 | 8 9 5 | 2 1 6 |
|-------+-------+-------|
| 4 8 3 | 1 2 7 | 6 5 9 |
| 7 1 9 | 6 5 3 | 4 2 8 |
| 2 5 6 | 9 8 4 | 1 3 7 |
```

Illustrierend erläutern wir die vierte dieser Zeilen, mit $jh = 4$, in der $dn =$ `59` die Ziffer 9 im Mittelquadrat $F(1,1)$ betrifft. Deren hier noch mögliche Positionen sind wegen $q_9(1,1) = oct =$ `ooo oo1 oo1` nach der am Anfang von Abschnitt 2 gezeigten Numerierung die zwei (in der Lösung) schließlich mit 9 und 5 besetzten Stellen im Südosten von $F(1,1)$. Zuerst versucht es *INFER* mit der Einschränkung auf die (für die Ziffer 9 *falsche*) Position der 5. Weil dieses Protokoll früher beendete Läufe von *INFER* später aufführt, finden wir den entsprechenden Mißerfolg ($e = 0$) erst in der vorletzten Zeile, mit $jh = 6$. Danach wird die andere (richtige) Variante versucht, hier in der fünften Zeile mit $jh = 7$ protokolliert. – Daß in diesem Beispiel alle Verzweigungen nur die Ziffer 9 betreffen, hat seinen (rein technischen) Grund in der Suchreihenfolge nach einer noch nicht fixierten Ziffer.

Bedeutsamer ist die Tatsache, daß dieses so noch kein "minimales" Problem ist: Weglassen der 5 im Südosteck von $F(0,0)$ ergibt ein ebenfalls noch eindeutig lösbares Problem ähnlicher Art, wieder mit Tiefe 5 und 11 Knoten, bei dem dann aber keine der nun noch 23 Vorgaben weggelassen werden kann. Ebenso liefert Weglassen der 3 im Südwesteck von $F(0,1)$ ein (dann) minimales Problem mit $f = 23$, und das scheint dann sogar noch etwas schwieriger zu sein, denn darauf angewandt arbeitet sich *INFER* durch einen etwas größeren Baum mit nunmehr 15 Knoten (wieder der Höhe 5) und benötigt dafür auch mehr Zeit, ca. 225 Ktu statt der vorherigen 143 Ktu. – Freilich sind beides nur Bruchteile einer Millisekunde!

Wesentlich größere Rechenzeiten werden jedoch erforderlich sein, wenn man mittels dieser Routine *INFER* neue (natürlich minimale) Probleme generieren will. Dazu kommt einem diese und jene Strategie rasch in den Sinn, bis zu konkreter Implementierung ist solches bei mir aber bisher noch nicht gediehen.

Jetzt noch einige Beispiele ähnlicher Art, die in ihrer Schwierigkeit deutlich über dem z. B. in Zeitschriften üblichen Niveau liegen und so geeignet sein mögen, dem Leser auch beim Lösen von Hand einige Freude zu bereiten. Das Problem 5, das aus einem Aufsatz von *V. Kaibel* und *T. Koch* [DMV-Mitteilungen 14, Heft 2 (2006), 93–96] stammt, wird zwar von Routine *INFER* ohne Rekursion (in nur 56 μs) recht schnell gelöst, ist aber in anderer Weise extremal: jene Autoren (`kaibel@zib.de` und `koch@zib.de`) merken an, daß diese Aufgabe die (bisher bekannte) Minimalzahl von nur 17 Vorgaben realisiert.

Problem 2

```
| - - - | - 6 - | - 8 - |
| 7 - - | 5 - - | - - - |
| - - - | 8 - 1 | - - - |
|-------+-------+-------|
| - - 9 | - - 6 | - - - |
| - - - | - - - | 2 - 1 |
| - - - | - 2 4 | 6 - - |
|-------+-------+-------|
| - 3 - | - - - | - 5 2 |
| 1 - 5 | - - 3 | - - - |
| - 2 - | 9 - - | 4 - 8 |
```

Problem 3

```
| - 6 - | 7 - - | - 8 - |
| - 4 - | - 5 - | 9 - - |
| - - - | - - 1 | - - 3 |
|-------+-------+-------|
| - - 2 | - - - | - - 1 |
| - 7 - | - - 4 | - 6 - |
| 8 - - | - - 3 | 5 - - |
|-------+-------+-------|
| 9 - - | 8 - - | - - - |
| - - 1 | - 2 - | - 7 - |
| - 5 - | - - 6 | - - - |
```

Problem 4

```
| - 7 - | - 8 - | - 5 - |
| 9 - - | - - - | - - 2 |
| - - - | 4 - 5 | - - - |
|-------+-------+-------|
| - - 6 | - 4 - | 7 - - |
| 3 - - | 7 - 2 | - - 4 |
| - - - | - 5 - | 9 - - |
|-------+-------+-------|
| - - 4 | 8 - 3 | - - - |
| 7 - - | - - - | - - 9 |
| - 3 - | - 9 - | - 1 - |
```

Problem 5

```
| - 8 - | 2 - 6 | - - - |
| - - - | - - - | - - - |
| 1 6 - | - - - | - - - |
|-------+-------+-------|
| 6 1 - | - - 4 | - - - |
| - 9 - | - - - | 3 - - |
| - - - | - - - | 7 - 5 |
|-------+-------+-------|
| - - - | - - - | - 1 - |
| - - - | - 7 - | - 8 - |
| - - 7 | - 3 - | - - - |
```

Nachtrag. Am 27. März 2007 habe ich in Dagstuhl (im Rahmen einer Klausurtagung unserer Informatik III) kurz über diese Ideen zu meinem Sudoku-Programm erzählt.

Gleich nach dem Vortrag fragte mich mein Kollege A.B. Cremers nach der Maximalzahl von Vorgaben bei Aufgaben, die noch *mehrere* Lösungen zulassen. Nach kurzem Besinnen vermutete ich, das müsse (beim 9×9 Format) wohl 77 sein. In der Tat wird das z. B. mit der Lösung von Problem 5 bestätigt, in der man von den 81 Ziffern lediglich jene zwei Ziffernpaare `1 6` am Anfang der dritten Zeile und `6 1` am Anfang der vierten Zeile wegläßt: deren Austausch liefert dann ja auch eine korrekte Ergänzung!

Zum weiteren Knobeln hier schließlich noch

Problem 6

```
| 2 - - | 8 - - | - - - |
| 4 - 9 | - 7 3 | - - - |
| - - - | - 4 6 | - - 5 |
|-------+-------+-------|
| - 7 - | - - - | 1 9 - |
| - - - | - - - | - 2 4 |
| 3 - - | - - 4 | - - - |
|-------+-------+-------|
| - 5 - | - - 8 | - - - |
| 1 - - | - 5 - | 7 - 8 |
| - - - | 6 3 - | - - - |
```