Seminar Automatentheorie der Presburger Arithmetik

Vorgetragen von Viktor Rach

Vortragsgliederung:

Kapitel 1
Presburger Arithmetik

1.1 Grundlagen der Presburger Arithmetik

Mojzesz Presburger, 1904–1943

Die Presburger Arithmetik entspricht der Peano - Arithmetik mit einer wichtigen Einschränkung: die beiden Axiome zur Festlegung der Multiplikation wurden weggelassen. Die Presburger Arithmetik beschäftigt sich also mit natürlichen Zahlen und deren Addition, nicht aber mit deren Multiplikation. Daher ist z.B. der Begriff Primzahl in der Presburger Arithmetik nicht definierbar.

Axiome der Presburger Arithmetik:

$$\begin{array}{llll} (P1) & S(x) \neq 0, & (P2) & S(x) = S(y) \rightarrow x = y, \\ (P3) & x + 0 = x, & (P4) & x + S(y) = S(x + y), \\ (P5) & \neg(x < 0), & (P6) & x < S(y) \leftrightarrow x < y \vee x = y, \\ & & (P7) & x < y \vee x = y \vee y < x, \end{array}$$

Weiterhin wird das Induktionsaxiom:

$$\phi(0) \wedge \forall x(\phi(x) \rightarrow \phi(S(x))) \rightarrow \phi$$

für jede Formel gefordert. Die Presburger Arithmetik axiomatisiert also die Eigenschaften der Addition von natürlichen Zahlen.
Presburger Arithmetik ist die Menge aller Formeln, die aus den Axiomen der Presburger Arithmetik ableitbar sind.

Wenn wir auch das Prädikat

$x \cdot y = z$

zulassen, erhalten wir die Formeln der Peano Arithmetik. Die Axiome der Peano Arithmetik bestehen aus den Axiomen der Presburger Arithmetik und zusätzlich aus den beiden folgenden Axiomen:

$$(P8) \quad x \cdot 0 = 0, \quad (P9) \quad x \cdot S(y) = (x \cdot y) + x.$$

Peano Arithmetik ist die Menge aller Formeln, die aus den Axiomen der Peano Arithmetik ableitbar sind.

1.2 Einführung in Presburger Arithmetik

Die Presburger Arithmetik ist nicht in der Lage, alle berechenbaren Funktionen der natürlichen Zahlen darzustellen, d.h. das Representation Theorem ist in ihr nicht gültig. Daher ist Gödels Unvollständigkeitssatz hier nicht anwendbar.

Representations Theorem:

Letztlich bedeutet es, dass sich jede Rechenvorschrift für natürliche Zahlen im formalen System der Peano Arithmetik ausdrücken lässt. Alles, was sich durch Berechnung in den natürlichen Zahlen erreichen lässt, lässt sich in der Peano Arithmetik ausdrücken und beweisen. Aufgrund des Representation Theorems sagt man, dass die Peano Arithmetik ein hinreichend starkes formales System ist.

Betrachten wir als Beispiel die Beschreibung von Eigenschaften natürlicher Zahlen. Beispielsweise sind einige Zahlen **Primzahlen**, andere nicht. Eine Eigenschaft natürlicher Zahlen wird dadurch definiert, dass man zu jeder natürlichen Zahl angibt, ob sie die Eigenschaft besitzt oder nicht. Die Eigenschaft wird also z.B. durch eine überall definierte Funktion f beschrieben, wobei man folgendes festlegen kann:

f(n) = 1, wenn die Zahl n die Eigenschaft besitzt
f(n) ungleich 1, wenn die Zahl n die nicht Eigenschaft besitzt

Beschränken wir uns auf die **berechenbaren Eigenschaften**, d.h. die zugehörige (überall definierte) Funktion f ist berechenbar. Das Representation Theorem sagt dann, dass sich jede berechenbare Eigenschaft durch eine Aussage A(n,1) der Peano-Arithmetik darstellen lässt. Wenn die natürliche Zahl n die Eigenschaft besitzt (d.h. f(n)=1), so kann man A(n,1) aus den Axiomen ableiten. Andernfalls lässt sich "NICHT A(n,1)" ableiten. Da die Aussage A(n,1) nur noch eine freie Variable besitzt, wollen wir sie mit A(n) bezeichnen.

Ein Beispiel für eine berechenbare Eigenschaft ist die **Primzahl-Eigenschaft**. Eine Zahl n ist eine Primzahl, wenn sie nur durch Eins oder durch sich selbst teilbar ist. Ein Programm, dass eine Zahl n daraufhin überprüft, müsste einfach nur diese Zahl der Reihe nach durch alle Zahlen m von 2 bis n-1 teilen und überprüfen, ob die Division ohne Rest aufgeht. Ein solches Programm würde für jedes n nach endlich vielen Schritten ermitteln, ob n eine Primzahl ist oder nicht. Anschließend könnte man dieses Programm in einen wohlgeformten Ausdruck der Peano-Arithmetik umwandeln. Das Ergebnis wäre ein ziemlich komplizierter Ausdruck, der den kompletten Algorithmus enthält.

Es gibt jedoch i.A. mehrere gleichwertige (d.h. ineinander umformbare) Möglichkeiten, eine Eigenschaft durch einen wohlgeformten Ausdruck zu charakterisieren, genauso wie es viele gleichwertige Algorithmen zur Überprüfung der Primzahleigenschaft gibt. Eine einfache Möglichkeit ist die folgende:

NICHT {ES_GIBT m: ES_GIBT n: [p = m*n] UND [(NICHT (m = NACHFOLGER(0))) UND (NICHT (m=p))] }

Diese Aussage können wir als wohlgeformte Zeichenkette A(p) abkürzen. In lesbares Deutsch übersetzt bedeutet sie: Eine Primzahl p kann nicht als Produkt zweier natürlicher Zahlen m und n geschrieben werden (p = m*n), wobei m ungleich 1 und ungleich p ist.

<u>Gödels Unvollständigkeitssatz:</u>

Jede rekursiv aufzählbare Axiomatisierung der Zahlentheorie besitzt wahre, aber nicht beweisbare Aussagen.

Lemma 2.15 *Sei M eine Turingmaschine und sei $w \in \{0,1\}^*$ eine Zeichenkette. Dann kann man eine Formel $\phi_{M,w}$ der Peano-Arithmetik konstruieren, so dass*

$$M \text{ akzeptiert } w \Leftrightarrow \phi_{M,w} \text{ ist wahr.}$$

Die Konstruktion von $\phi_{M,w}$ gelingt mit einer stets haltenden Turingmaschine in polynomieller Zeit (in der Länge von w).

Beweis des Gödelschen Unvollständigkeitssatzes. Angenommen, es gibt eine Axiomatisierung A der Zahlentheorie mit der jede wahre Aussage auch beweisbar ist.
Sei L eine beliebige rekursiv aufzählbare Sprache, die von der Turingmaschine M akzeptiert werde. Mit Lemma 2.15 können wir für jede Zeichenkette w eine Aussage $\phi_{M,w}$ der Peano-Arithmetik bestimmen, so dass

$$w \in L \;\Leftrightarrow\; \phi_{M,w} \text{ ist wahr.}$$

Wenn Wahrheit und Beweisbarkeit (bzgl. der Axiomatisierung A) übereinstimmen, folgt also

$$w \in L \Leftrightarrow \phi_{M,w} \text{ ist aus } A \text{ ableitbar}$$

und ebenso natürlich

$$\begin{aligned} w \notin L &\;\Leftrightarrow\; \neg\phi_{M,w} \text{ ist wahr.} \\ &\;\Leftrightarrow\; (\neg\phi_{M,w}) \text{ ist aus } A \text{ ableitbar.} \end{aligned}$$

Also ist

$$L = \{w \mid \phi_{M,w} \text{ ist aus } A \text{ ableitbar}\} \text{ und } \bar{L} = \{w \mid (\neg\phi_{M,w}) \text{ ist aus } A \text{ ableitbar}\}$$

und damit ist L wie auch $\bar{L}$ rekursiv aufzählbar, denn die Axiomatisierung A ist rekursiv aufzählbar. Also ist für jede rekursiv aufzählbare Sprache L auch ihr Komplement rekursiv aufzählbar und damit ist jede rekursiv aufzählbare Sprache auch rekursiv. Und dies ist offensichtlich unmöglich. □

Es zeigt sich jedoch, dass die Presburger Arithmetik nicht in der Lage ist, alle berechenbaren Funktionen darzustellen. Sie ist zu schwach dafür. Das Representations Theorem ist in ihr nicht gültig.

Es gelang im Jahr 1929 dem Mathematiker Mojzesz Presburger tatsächlich, zu beweisen, dass die Presburger Arithmetik vollständig und widerspruchsfrei ist. Jede wohlgeformte Aussage der Presburger Arithmetik lässt sich in endlich vielen Schritten entweder beweisen oder widerlegen. Die Frage nach der Beweisbarkeit eines vorgegebenen Ausdrucks ist in diesem formalen System also entscheidbar.

Allerdings kann es extrem schwierig sein, die Lösung für etwas längere Ausdrücke allgemein zu finden, denn im Jahr 1974 gelang es Michael J.Fischer und Michael O.Rabin, folgendes zu beweisen:

- Jeder allgemeine Algorithmus, der für einen beliebigen geschlossenen (d.h. ohne freie Variable) Ausdruck A der Länge n entscheiden kann, ob er aus den Axiomen der Presburger Arithmetik ableitbar ist oder nicht, benötigt dafür mindestens 2**(2**(C*n)) Schritte (dabei ist C ist eine Konstante und ** steht für "hoch", also Exponentieren).

Der Rechenaufwand steigt also mindestens doppelt exponentiell mit der Länge des zu untersuchenden Ausdrucks. Bereits für etwas längere Ausdrücke kann dies zu astronomisch langen Laufzeiten führen. Das Beweisbarkeitsproblem der Presburger Arithmetik ist also

zumindest für bestimmte Worst-Case-Ausdrücke extrem schwierig, wenn man einen allgemeinen Entscheidungsalgorithmus einsetzen will.

1.3 Praktische Beispiele

Die Basis-Terme in Presburger Arithmetik bestehen aus Variablen (x, x1, x2, x3,…), Konstanten 0 und 1 und Summen von Basis-Termen.

Zum Beispiel *ist* $x + x + 1 + 1 + 1$ *ein Basis-Term den man auch als 2x + 3 schreiben kann.*

Die Atomformeln sind Gleichheiten und Ungleichheiten zwischen den Basis-Termen.

Zum Beispiel *wäre* $x + 2y = 3z + 1$ *eine Atomformel.*

Die Formel der Logik sind die Formeln, welche auf den Atomformeln errichtet werden. Man benutzt die Verbindungen wie: Konjunktion (und), Disjunktion (oder), Negation (nicht), Existenz Quantor und All Quantor.

Zum Beispiel *wäre:* $\forall x, \exists y.(x = 2y \vee x = 2y + 1)$ *eine Formel.*

Die freien Variablen einer Formel ϕ sind üblicherweise folgend definiert:

Zum Beispiel:

$$FV(\phi_1 \vee \phi_2) = FV(\phi_1) \cup FV(\phi_2) \text{ and } FV(\exists x.\phi) = FV(\phi) \setminus \{x\}.$$

Der Interpretationsbereich von den Formeln ist die Menge der Natürlichen Zahlen, in welcher 0, 1, 2, +, =, ≤ , ihre übliche Bedeutung haben.

Eine Lösung einer Formel $\phi(x_1, \ldots, x_n)$ ist eine Anweisung von $x_1, \ldots, x_n$ in Natürlichen Zahlen die die Formel erfüllt.

Zum Beispiel: $\{x \mapsto 0; y \mapsto 2; z \mapsto 1\}$ *ist die Lösung von* $x + 2y = 3z + 1$

Kapitel 2
Endliche Automaten und Reguläre Ausdrücke

2.1 Endliche Zustandssysteme

Endlicher Automat ist ein mathematisches Modell von einem System, mit eindeutigen Eingaben und Ausgaben. Das System kann sich in beliebigen endlichen Konfigurationen oder Zuständen befinden. Der Zustand eines Systems fasst die Informationen voriger Eingaben zusammen, welche dafür gebraucht werden um das Verhalten des Systems für anschließende Eingaben festzulegen.

Der Kontrollmechanismus eines Aufzuges ist ein gutes Beispiel für ein endliches Zustandssystem. Dieser Mechanismus erinnert sich nicht an alle vorher gehenden Anfragen auf Dienste, sondern nur an momentanen Stock, Richtung der Bewegung (hoch oder runter) und an die Menge noch nicht erfüllter Anfragen auf Dienste.

Meistens sind Texteditoren und lexikalische Analyse Programme als endliche Zustandsysteme konstruiert. Zum Beispiel, ein lexikalisches Analyse Programm scannt die Symbole eines Programms, um Strings, numerische Konstanten, reservierte Wörter und so weiter zu, erkennen. In diesem Prozess braucht der lexikalischer Analyser sich nur an endliche Menge von Informationen zu erinnern.

2.2 Endlicher Automat

Definition (endlicher Automat). *Ein* **endlicher Automat (e.a.)** *oder eine* **endliche sequentielle Maschine (finite sequential machine, fsm)** *ist ein Tupel* $A = (K, \Sigma, \delta, s_0, F)$. *Dabei ist*

- K *eine endliche Menge von Zuständen,*
- Σ *ein endliches Alphabet (aus dessen Buchstaben die Eingabewörter bestehen können),*
- $\delta : K \times \Sigma \to K$ *die Übergangsfunktion,*
- $s_0 \in K$ *der Startzustand, und*
- $F \subseteq K$ *die Menge der finalen Zustände.*

$\delta(q, a) = q'$ *bedeutet: Der Automat liest im Zustand* q *ein* a *und geht in den Zustand* q' *über. Um mehrere Übergangsschritte auf einmal betrachten zu können, erweitern wir* δ *zu* δ^*, *einer Mehrfachanwendung der Übergangsfunktion.* $\delta^* : K \times \Sigma^* \to K$ *ist induktiv über* Σ^* *definiert wie folgt:*

$$\delta^*(q, \varepsilon) := q$$
$$\delta^*(q, wa) := \delta(\delta^*(q, w), a)$$

Wenn klar ist, was gemeint ist, wird δ^* *auch einfach als* δ *geschrieben.*

Zum Beispiel:

Der Automat A'' in Abb. 5.3 akzeptiert $L(A'') = \{w \in \{a,b\}^* \mid \exists n : \#_a(w) = 2n\}$. Dabei ist $\#_a(w)$ die Anzahl der ‚a' in w. Zum Beispiel würde A'' das Wort *bbaabaab* wie folgt akzeptieren:

$$s_0 \xrightarrow{b} s_0 \xrightarrow{b} s_0 \xrightarrow{a} s_1 \xrightarrow{a} s_0 \xrightarrow{b} s_0 \xrightarrow{a} s_1 \xrightarrow{a} s_0 \xrightarrow{b} s_0$$

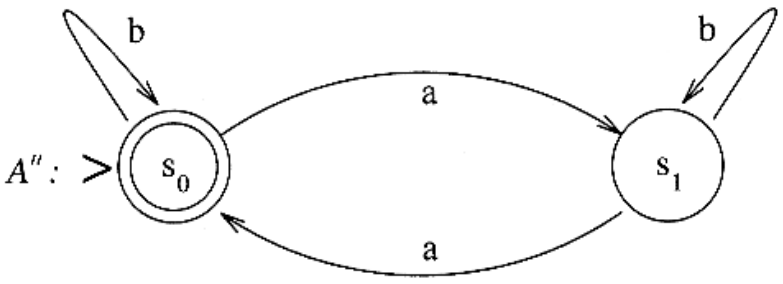


Abb. 5.3. A'' akzeptiert Wörter, die eine gerade Anzahl von ‚a' haben

2.3 Nichtdeterministischer endlicher Automat

Die endlichen Automaten, die bisher vorgestellt wurden, waren determiniert: Ein endlicher Automat, der im Zustand q ein q liest, hat genau einen nächsten Zustand q', festgelegt durch die Übergangsfunktion δ. Ein indeterminierter Automat kann zum Zustand q und dem gelesenen Zeichen a mehrere mögliche Folgezustände haben – oder gar keinen.

Definition (indeterminierter endlicher Automat). *Ein* **indeterminierter endlicher Automat (nd e.a.)** *A ist ein Tupel $A = (K, \Sigma, \Delta, I, F)$. Dabei ist*

- *K eine endliche Menge von Zuständen,*
- *Σ ein endliches Alphabet,*
- *$\Delta \subseteq (K \times \Sigma) \times K$ eine Übergangsrelation,*
- *$I \subseteq K$ eine Menge von Startzuständen und*
- *$F \subseteq K$ eine Menge von finalen Zuständen.*

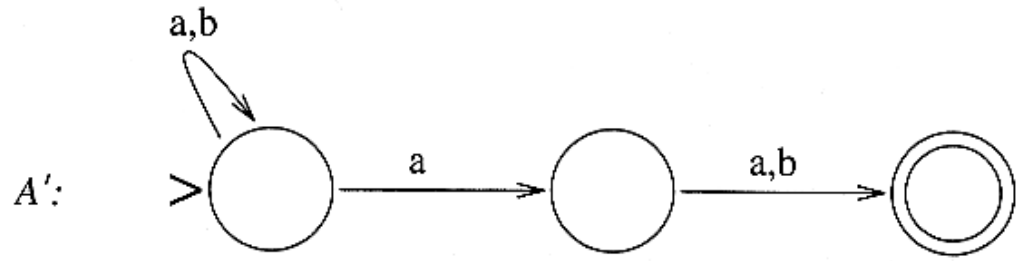
$\Delta^* \subseteq (K \times \Sigma^*) \times K$ ist definiert wie folgt:

(q, ε) Δ^* q' gdw. $q' = q$

(q, wa) Δ^* q' gdw. $\exists q'' \in K$ $\big((q, w)$ Δ^* q'' *und* (q'', a) Δ $q'\big)$

Statt (q, w) Δ^* q' schreibt man auch $\big((q, w), q'\big) \in \Delta^*$ oder $q' \in \Delta^*(q, w)$. Δ^* wird auch als Δ abgekürzt, wenn es im Kontext nicht zu Verwechslungen führt.

Zum Beispiel: Die Sprache L = {a,b}* {a} {a,b} ist die Sprache aller Wörter über {a,b}, deren zweitletzter Buchstabe ein a ist.



Ein indeterminierter Automat für $L = \{a, b\}^*\{a\}\{a, b\}$

Kapitel 3
Umsetzung von Presburger Formeln zur endlichen Automaten

3.1 Menge der Integer, welche von einem Automaten erkannt werden

Jede natürliche Zahl kann auch als Wort gesehen werden, geschrieben als Basis k mit (k>=1) über dem Alphabet A={0, …, k-1}.
Wegen der vorliegender Literatur wird im Anschluss eine andere Schreibweise benutzt, und zwar von rechts nach links, diese soll anscheinend bequemer sein.

Zum Beispiel*: kann die Zahl 13 zur Basis 2 als 1011 geschrieben werden.*

Es gibt mehr als eine Möglichkeit eine Zahl darzustellen, wir können das Wort vervollständigen indem wir an der rechten Seite Nullen anfügen: so wäre 101100 eine andere Darstellung der Zahl 13 zu Basis 2.

N - Tupel von natürliche Zahlen zur Basis k können als ein einziges Wort über dem Alphabet $\{0, \ldots, k-1\}^n$ dargestellt werden und zwar durch das stapeln der Darstellungen jeder einzelner Zahl.

Zum Beispiel*: das Paar (13,6) zur Basis 10 kann in Basis 2 über das Alphabet {0,1}² als* $\begin{smallmatrix}1011\\0110\end{smallmatrix}$ *Wieder Gibt es mehr als eine Darstellungsmöglichkeit, das Wort kann durch das Einfügen von* $\begin{smallmatrix}0\\0\end{smallmatrix}$ *an der rechten Seite vervollständigt werden.*

3.2 Automaten und Gleichheiten

In der Umsetzung, Logik auf Automaten, fangen wir erst damit an, einen Automaten mit jeder Atomaren Formel zu verbinden.

Für jede Basis Formel: $a_1x_1 + \ldots + a_nx_n = b \text{ (where } a_1, \ldots, a_n, b \in \mathbf{Z})$ konstruieren wir einen Automaten, durch das Verändern von Regeln und Zuständen und ursprünglichen Mengen {qb}, durch das benutzen der Abschlussregeln:

$$\frac{q_c \in Q}{q_d \in Q, \quad (q_c, \theta, q_d) \in \delta} \text{ if } \begin{cases} a_1\theta_1 + \ldots + a_n\theta_n =_2 c \\ d = \frac{c - a_1\theta_1 \ldots - a_n\theta_n}{2} \\ \theta \in \{0,1\}^n \text{encodes } (\theta_1, \ldots, \theta_n) \end{cases}$$

mit anderen Worten: für jeden Zustand $q_c \in Q$, ist die Berechnung der Lösungen $(\theta_1, \ldots, \theta_n)$ von $a_1x_1 + \ldots + a_nx_n = c \text{ modulo } 2$ und der Zustand q_d und die Regel $q_c \xrightarrow{\theta} q_d$ wo $d = \frac{c - a_1\theta_1 \ldots - a_n\theta_n}{2}$ gilt.

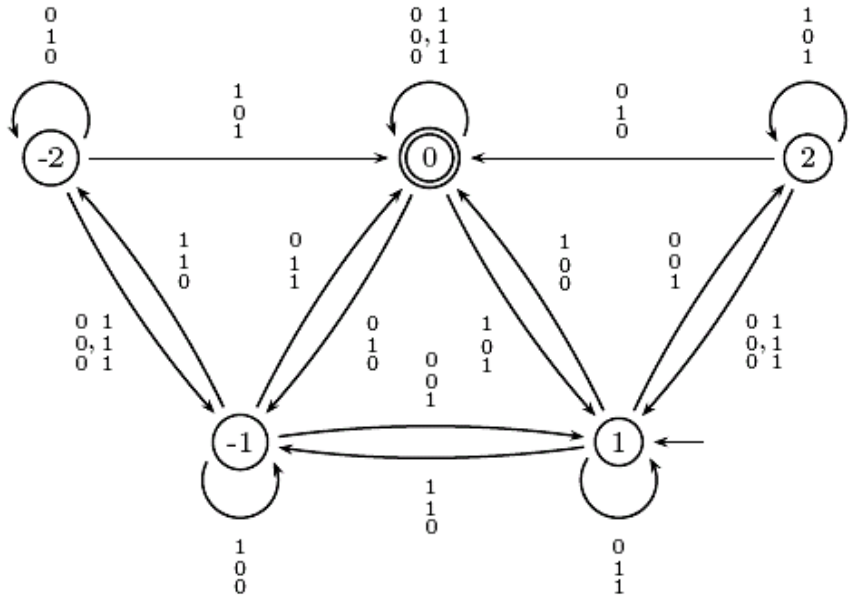
Der Anfangszustand ist q_b und wenn $q_0 \in Q$ gilt, dann ist das der einzigster finaler Zustand. Anderenfalls gibt es keinen finalen Zustand.

Zum Beispiel: *Betrachten wir die Gleichung: x + 2y = 3z + 1*

Sei b=1, dann ist $q_1 \in Q$. *Wir berechnen die Lösung des modulo 2 von x + 2y = 3z + 1, dann kriegen wir: {(0,0,1),(0,1,1),(1,0,0),(1,1,0)}. Nun berechnen wir die neuen Zustände:* $\frac{1-0-0+3}{2} = 2, \frac{1-0-2+3}{2} = 1, \frac{1-1-0+0}{2} = 0, \frac{1-1-2+0}{2} = -1,$ *daraus Folgen* $q_2, q_0, q_{-1} \in Q$ *neue Zustände und folgende neue Übergänge:*

$$q_1 \xrightarrow{\substack{0\\0\\1}} q_2, q_1 \xrightarrow{\substack{0\\1\\1}} q_1, q_1 \xrightarrow{\substack{1\\0\\0}} q_0, q_1 \xrightarrow{\substack{1\\1\\0}} q_{-1}.$$

Beim Durchgehen von diesen Zuständen kriegt man folgenden Automaten:



Betrachten wir einen mehr praktischen Beispiel: Betrachtet man den Weg vom Anfangszustand q_1 *zum finalen Zustand* q_0 *und zwar den Weg* $q_{-1}q_{-2}q_0q_1q_2$. *Das Wort*

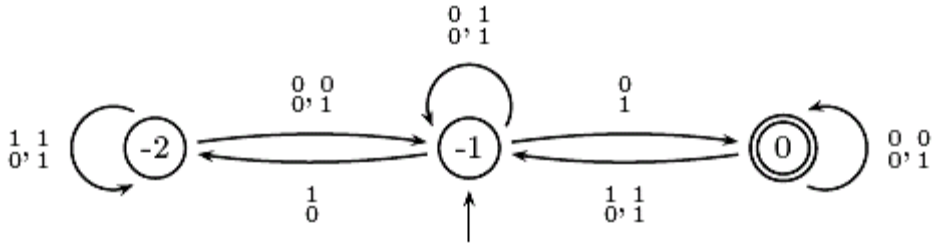
111100
110001
001110 *wird vom Automaten akzeptiert, welcher dem Tripel zu Basis 10 entspricht und zwar: x = 15, y = 35, z = 28 und man kann 15 + 2 * 35 = 3 * 28 + 1 überprüfen.*

3.3 Automaten und Ungleichheiten

Als aller erstes berechnen wir einen Automaten für die Ungleichheiten: $a_1x_1 + \ldots + a_nx_n \leq b$. Die Berechnung ist ähnlich der oberen, wir starten mit qb von einem Zustand qc, die Berechnung von Zuständen und Übergängen erfolgt wie folgt: für jeden $(\theta_1, \ldots \theta_n),\ q_c \xrightarrow{\theta_1 \ldots \theta_n} q_d$ mit $d = \lfloor \frac{c - \sum_{i=1}^{n} a_i\theta_i}{2} \rfloor$

$Q_f = \{q_c \mid c \geq 0\}.$

Zum Beispiel*: Betrachten wir die Ungleichung 2x – y ≤ -1. Diese wir durch folgenden Automaten dargestellt:*



Schließlich können wir durch die Induktion der Presburger Formel ϕ den Automaten berechnen, der die Lösungsmenge von ϕ annimmt.

Quellenverzeichnis

J.E. Hopcroft and J.D. Ullmann. *Introduction to Automata Theory, Languages, and Computation*

H. Comon and C. Kirchner. Presburger arithmetic and classical word automata. In H. Comon, C. Marché and R. Treinen, editors, *Constraints in Computational Logic: Theory and Applications*

Erk Priese *Theoretische Informatik, Eine umfassende Einführung*

http://www.cse.unsw.edu.au/~achim/Research/Philosophie/node53.html

http://www.joergresag.privat.t-online.de/mybk3htm/chap51.htm

http://www.joergresag.privat.t-online.de/

http://www.tcs.informatik.uni-muenchen.de/lehre/SS04/Automat/lecturenotes/Seite12-19.txt

http://www.yotor.com/wiki/ru/%C0%F0/%C0%F0%E8%F4%EC%E5%F2%E8%EA%E0%20Presburger.htm

http://www.ubka.uni-karlsruhe.de/cgi-bin/psview?document=1999/informatik/2&search=presburger&format=1&page=57

http://www.ti.informatik.uni-kiel.de/Veranstaltungen/Kanon/Automaten/Skript/Skript.html