Gottfried Wilhelm Leibniz Universität Hannover
Institut für Theoretische Informatik

Bachelorarbeit

Die Komplexität der Presburger Arithmetik

Julian Alexander Gercke

Matrikelnummer:	10003079
Erstprüfer und Betreuer:	Prof. Dr. Heribert Vollmer
Zweitprüfer:	Dr. Arne Meier
Abgabedatum:	08.08.2019

Inhaltsverzeichnis

1 Einleitung

1.1 Motivation

Kurt Gödel konnte in seiner 1931 veröffentlichten Arbeit „Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme I“ beweisen, dass es in hinreichend ausdrucksstarken Systemen Sätze gibt, die nicht formal beweisbar oder widerlegbar sind. So ein System heißt unvollständig. Es zeigt sich, dass die allgemein bekannte Arithmetik mit der Addition und der Multiplikation, mit dem Namen „Peano Arithmetik“ genau solch ein unvollständiges System ist. [Göd31, Hof17]
Bereits zwei Jahre vor diesem Resultat war es „Mojżesz Presburger“ jedoch gelungen, eine vollständige Arithmetik zu finden, in der jeder Satz bewiesenermaßen wahr oder falsch ist. Zu diesem Zweck beschränkt er die Arithmetik auf die Addition, wodurch sie nach späteren Erkenntnissen von Gödel nicht mehr hinreichend ausdrucksstark sein kann. [Zyg91]
Auch heutzutage besteht weiterhin ein Interesse daran, die Korrektheit einer gegebenen arithmetischen Aussage bestimmen zu können. Gute Beispiele dafür sind Datenbank- und Computerbeweissysteme, in denen es nötig ist, Aussagen zu analysieren und zu bearbeiten. In Datenbanksystemen könnten so beispielsweise bei der Bearbeitung von arithmetisch „falschen“ Anfragen Zeit gespart werden, bei denen sonst eventuell das Durchsuchen einer ganzen Relation nötig wird (Relation Scan). Jedoch kann nur dann tatsächlich eine kürzere Laufzeit erreicht werden, wenn das Analysieren der Aussage tatsächlich schneller erfolgt, als der Relation Scan.

Beispiel 1: Anfragen und Integritätsbedingungen im Tupelkalkül als Beispiel

Es sei GERADE eine Tabelle mit den ersten 1000 geraden Zahlen und PRIM eine Tabelle mit den Primzahlen kleiner 1000. Beide Tabellen haben nur eine Spalte „Zahl“, gefüllt mit den jeweiligen Zahlen.

1. Beispielanfrage zum Bestimmen von „Primzahlzwillingen“

$$\{z : (p1, p2) \mid \exists(P1 : PRIM)\exists(P2 : PRIM)(z.p1 = P1.Zahl \wedge z.p2 = P2.Zahl \ \wedge \ P2.Zahl = P1.Zahl + 2)\}$$

2. Integritätsbedingung zum Testen der „Goldbachsche Vermutung“

$$\forall(g : GERADE)\exists(P1 : PRIM)\exists(P2 : PRIM)(g.Zahl = p1.Zahl + p2.Zahl \wedge g.Zahl > 2)$$

Mit der sich daraus ergebenen Motivation gab es bis 1997 diverse komplexitätstheoretische Betrachtungen. Diese hatten zum Ziel, obere und untere Schranken für die benötigte Zeit zu finden, in der es möglich ist, zu entscheiden, ob ein Satz der Presburger Arithmetik wahr ist. Schon früh stellt sich heraus, dass dieses Problem allgemein nicht effizient gelöst werden kann [Pau78]. Durch die Einschränkung der Quantorenwahl ist es Grädel 1989 aufbauend auf vorausgegangenen Resultaten gelungen, die Komplexität für festgelegte Teilklassen zu reduzieren [Grä88]. Ein Resultat, welches schließlich 1997 von Schöning erweitert wurde [Sch97]. Erst 2017 gelang es Nguyen und Pak ([NP17]) durch weitere Einschränkungen bezüglich der Länge des

zu entscheidenden Satzes, die Komplexität dieses Problems einen weiteren Schritt zu senken.

1.2 Aufbau der Arbeit

In dieser Arbeit sollen einige, der bis 1997 erschienen, Resultate vorgestellt werden, um dann im Ausblick kurz einen neueren Sachverhalt aus 2017 zu präsentieren und damit einen Überblick über die Komplexität der Presburger Arithmetik zu geben. Dabei lege ich besonderen Wert auf die mathematisch vollständige Abhandlung:
Begonnen bei der formellen Definition der Presburger Arithmetik über die benötigten komplexitätstheoretischen Grundlagen, bis zu zwei Erkenntnissen, die in der folgenden Arbeit als Werkzeug dienen, stelle ich in Kapitel 3 zunächst die Grundlagen vor. Darauffolgend zeige ich in Kapitel 4 das von Cooper vorgestellte Verfahren, mit dem bestimmt werden kann, ob ein Satz der Presburger Arithmetik „wahr“ ist. Anschließend gehe ich in Kapitel 5 durch eine Reduktion auf den Zusammenhang zwischen den leicht verschiedenen Strukturen der Presburger Arithmetik ein, mit dem es später möglich ist, Komplexitätsresultate zu übertragen. Motivierend für das darauffolgende Kapitel, gehe ich in Kapitel 6.1 zunächst auf die Komplexität des Problems ein, zu entscheiden, ob ein beliebiger Satz der Presburger Arithmetik wahr ist. Im folgenden Kapitel 6.2 wird die Quantorenwahl der zu entscheidenden Sätze beschränkt, wodurch sich Teilklassen wahrer Sätze der Presburger Arithmetik bilden. Die Einschränkung des Quantorenprefix sorgt für die Begrenzung der Komplexität des Entscheidungsproblems und die komplexitätstheoretische Betrachtung dieser Teilklassen führt zu der von Grädel und Schöning erarbeitete Hierarchie von Teilklassen, die vollständig für ihre Komplexitätsklasse sind. Schließlich gebe ich im Fazit ein Überblick über die zusammengefassten Resultate dieser Arbeit und ordne ein Ergebnis von Nguyen und Pak in die vorausgegangenen Resultate ein.

2 Notationen

Nachfolgend sind einige Notationen, die ich im Laufe meiner Bachelorarbeit verwenden werde, aufgelistet:
Für alle $i \in \mathbb{N}$ seien

- c_i Konstanten und α_i, β_i Konstanten > 0
- x_i Variablen; die Sprache der Variablen ist $\mathcal{L}_{\mathrm{x}} := \{x\}^*$, insofern beispielsweise $x_4 := xxxx$, dadurch bleibt ein Alphabet mit Variablen endlich, obwohl es unendlich viele Variablen gibt
- x_i^j ebenfalls Variablen aus $\mathcal{L}_{\mathrm{x}}$ nur mit alternativer Nummerierung (für alle $j \in \mathbb{N}$) (Bem.: Variablen werden niemals potenziert!)
- t_i Terme
- Q_i Quantoren d.h. $Q_i \in \{\forall, \exists\}$ (Q_i^t steht für alle $t \in \mathbb{N}$ für $\underbrace{Q_i, ..., Q_i}_{t \text{ Mal}}$)
- F_i Funktionen
- ϕ_i Formeln
- δ_i Teilformeln einer Formel ϕ_i

Falls es in dem gegebenen Kontext nur einen Term, eine Funktion etc. gibt, lasse ich das i auch weg. Des Weiteren sei

- Σ_{Name} das Alphabet von „Name“
- $\mathcal{S}_{\text{Name}}$ die Struktur von „Name“; Das in der Struktur enthaltende Universum, bildet die Menge aus der die Belegungen für freie und gebundene Variablen gewählt werden können.
- $\mathcal{L}_{\text{Name}}$ die Sprache von „Name“
- σ_{Name} die Signatur von „Name“
- **frei** eine Funktion, die für eine Formel ϕ der Prädikatenlogik die Menge der Variablen zurückgibt, die in ϕ nicht gebunden vorkommen
- $|x|$ (x eine Variable) die Betragsfunktion $|x| := \begin{cases} x & \text{falls } x \geq 0 \\ -x & \text{sonst} \end{cases}$
- $|w|$ (w ist das Wort einer Sprache oder eine Konfiguration) die Funktion, die die Länge von w zurückgibt
- $\#(M)$ die Funktion, die bei Eingabe einer Menge M die Kardinalität der Menge zurückgibt. Diese wird ausschließlich in Präfixnotation verwendet: $\#M := \#(M)$
- / das Symbol für die Substitution (y/x ist die Substitution von y durch x)
- Σ^* die Menge aller möglichen Wörter, die aus Σ gebildet werden können (inklusiv dem leeren Wort).
- Σ^+ das Alphabet Σ^* ohne das leere Wort ϵ
- 1 bzw. 0 (bei Benutzung als Primformel) das Symbol für wahr bzw. falsch

3 Grundlagen

3.1 Die Presburger Arithmetik

Die Presburger Arithmetik (PA) ist eine Prädikatenlogik erster Stufe [FR98]. Zunächst enthält das Alphabet Σ_{PA} deshalb die üblichen Symbole der Logik d.h. Quantoren $\{\forall, \exists\}$, Junktoren $\{\wedge, \vee, \rightarrow, \leftrightarrow, \neg\}$, Klammern $\{(,)\}$, das Gleichzeichen $\{=\}$ und Variablen $\mathcal{L}_{\text{x}}$. Hinzu kommt noch $\{+\}$ als alleiniges Funktionssymbol der Signatur der Presburger Arithmetik:

$$\sigma_{\text{PA}} := (+; 0, 1)$$

Somit ergibt sich das Alphabet der Presburger Arithmetik:

Definition 1: Alphabet der Presburger Arithmetik

$$\Sigma_{\text{PA}} = \{\forall, \exists, \wedge, \vee, \rightarrow, \leftrightarrow, \neg,), (, +, =, 0, 1, x\}$$

Syntaktisch fällt auf, dass es sich bei σ_{PA} um die Signatur der Peano Arithmetik mit eingeschränkter Symbolwahl handelt.

Definition 2: Sprache der Presburger Arithmetik
Es sei $\mathcal{L}_{\text{Peano}}$ die Sprache der Peano Arithmetik, dann ist $\mathcal{L}_{\text{PA}} := \{\phi \in \mathcal{L}_{\text{Peano}} \mid \phi \in \Sigma^*_{\text{PA}}\}$, nur eingeschränkt durch die Wahl des Alphabets, die Sprache der Presburger Arithmetik. Allgemeiner kann formuliert werden:

$\mathcal{L}_{\text{PA}} := \{\phi \in \mathcal{L}_{\text{Peano}} \mid \phi$ enthält keine Zeichen, die die Multiplikation benötigen [Pau78]$\}$

Semantisch werden die Symbole aus diesem Grund auch genauso behandelt wie in der Peano Arithmetik. Das Universum der Presburger Arithmetik sind die natürlichen Zahlen $\mathbb{N}$, sodass sich die Struktur der Presburger Arithmetik ergibt:

Definition 3: Struktur der Presburger Arithmetik vgl. [FR98]

$$\mathcal{S}_{\text{PA}} = (\mathbb{N}; +; 0, 1)$$

Das Aufschreiben von Formeln, wenn nur die Addition erlaubt ist, ist sehr umständlich.

Beispiel 2: Ein Ausdruck aus $\mathcal{L}_{\text{PA}}$

Die Presburger Arithmetik kann unter anderem ausdrücken, dass es eine Variable x_1 gibt, die 100 Mal zusammenaddiert, x_2 ergibt:

$$\exists x_1 \ \underbrace{x_1 + x_1 + ... + x_1}_{\text{100 Mal}} = x_2$$

Zur Einsparung dieses enormen Schreibaufwands wird die Skalarmultiplikation definiert:

Definition 4: Skalarmultiplikation in der Presburger Arithmetik
Es sei $c_1 \in \mathbb{N}$ ein fester Wert. Dann sei $+^{c_1}(x_1) := \underbrace{x_1 + x_1 + ... + x_1}_{c_1 \text{ Mal}}$, für $+^{c_1}(x_1)$ schreibe ich in Infixnotation $c_1 * x_1$.

Die Skalarmultiplikation kann uneingeschränkt in der Presburger Arithmetik benutzt werden, da sie mit größerem Schreibaufwand auch nur mit Zeichen aus Σ_{PA} hätte ausgedrückt werden können. Wenn Beispiel 2 mit der Skalarmultiplikation ausgedrückt werden soll, dann ergibt sich:

$$\exists x_1 \; 100 * x_1 = x_2$$

In der Prädikatenlogik bedeutet dieser Ausdruck aber auch, dass 100 ein Teiler von x_2 ist. Tatsächlich kann in der Presburger Arithmetik auch die Teilbarkeit eingeschränkt definiert werden:

Definition 5: Die eingeschränkte Teilbarkeitsrelation in der Presburger Arithmetik
Es sei

$$c_1 \mid x_1$$

die Abkürzung für $\exists x_2 \; c_1 * x_2 = x_1$.

Dabei ist zu beachten, dass ein Primzahltest einer beliebigen Zahl mit dieser Relation nicht möglich ist. Für diesen Zweck würde die Teilbarkeit einer Variable durch eine andere Variable benötigt werden oder eine variable Anzahl an Teilbarkeitsrelationen. Beides ist nicht in der Presburger Arithmetik ausdrückbar.

Definition 4 und 5 sind nur als Abkürzungen für das Aufschreiben von Formeln zu verstehen. Sie erweitern weder das Alphabet Σ_{PA}, noch ermöglichen sie „komplexere“ Ausdrücke in $\mathcal{L}_{\text{PA}}$.

Es zeigt sich jedoch, dass eine tatsächliche Erweiterung der Struktur $\mathcal{S}_{\text{PA}}$ um das Symbol „$\leq$“ hilfreich ist. Zu diesem Zweck wird die um „$\leq$“ erweiterte Struktur $\mathcal{S}_{\text{PA}_\leq}$ mit zugehörigem Alphabet $\Sigma_{\text{PA}_\leq}$ und Sprache $\mathcal{L}_{\text{PA}_\leq}$ definiert.

Definition 6: Die Erweiterung der Presburger Arithmetik um „$\leq$“ vgl. [Grä88]
Es sei:

1. $\Sigma_{\text{PA}_\leq} = \{\forall, \exists, \wedge, \vee, \rightarrow, \leftrightarrow, \neg,), (, +, =, 0, 1, x, \leq\}$
2. $\mathcal{L}_{\text{PA}_\leq} = \{\phi \in \mathcal{L}_{\text{Peano}} \mid \phi \in \Sigma^*_{\text{PA}_\leq}\}$
3. $\mathcal{S}_{\text{PA}_\leq} = (\mathbb{N}; +; \leq; 0, 1)$

Infolgedessen können auch die restlichen Ordnungsrelationen ausgedrückt werden:

Beispiel 3: Weitere Ordnungsrelationen in der Presburger Arithmetik

- $<: \mathbb{N} \times \mathbb{N} \to \{0,1\}$ mit $x_1 < x_2 \coloneqq \neg(x_1 \leq x_2)$
- $>: \mathbb{N} \times \mathbb{N} \to \{0,1\}$ mit $x_1 > x_2 \coloneqq \neg(x_2 \leq x_1)$
- $\geq: \mathbb{N} \times \mathbb{N} \to \{0,1\}$ mit $x_1 \geq x_2 \coloneqq (x_2 \leq x_1)$

Um weitere Anwendungsmöglichkeiten der Presburger Arithmetik zu demonstrieren, folgt ein weiteres nützliches Beispiel:

Beispiel 4: Rechnen im Restklassenring $\mathbb{Z}/c_1\mathbb{Z}$

Die folgenden Beispiele orientieren sich an Rechnungen in [Sch97].

1. $x_2 \equiv x_1 \bmod c_1$
   Ähnlich wie in Beispiel 5 kann auch der Modulo-Operator eingeschränkt übersetzt werden:

$$\begin{aligned} x_2 \equiv x_1 \bmod c_1 &\Leftrightarrow c_1 \mid (x_1 - x_2) \\ &\Leftrightarrow \exists x_3 \; c_1 * x_3 = x_1 - x_2 && \mid \text{Definition 5} \\ &\Leftrightarrow \exists x_3 \; c_1 * x_3 + x_2 = x_1 \end{aligned}$$

2. $(x_1 \bmod c_1) \in M$
   Für eine Menge $M \subsetneq \{1, ..., c_1\}$ kann überprüft werden, ob $(x_1 \bmod c_1) \in M$ gilt, indem alle möglichen Fälle getestet werden.

$$\begin{aligned} x_1 \bmod c_1 \in M &\Leftrightarrow \bigwedge_{i \notin M} \neg(i \equiv x_1 \bmod c_1) \\ &\Leftrightarrow \bigwedge_{i \notin M} \neg(\exists x_2 \; c_1 * x_2 + i = x_1) && \mid \text{Anwendung von 1.} \\ &\Leftrightarrow \bigwedge_{i \notin M} \forall x_2 \; \neg(c_1 * x_2 + i = x_1) \\ &\Leftrightarrow \forall x_2 \bigwedge_{i \notin M} \neg(c_1 * x_2 + i = x_1) \end{aligned}$$

Obwohl es sich bei der Presburger Arithmetik um die Arithmetik ohne die Multiplikation auf den natürlichen Zahlen handelt, wird in den späteren Kapiteln auch die Presburger Arithmetik auf den ganzen Zahlen benötigt. In dieser ist sowohl die in Definition 5 eingeführte eingeschränkte Teilbarkeit $|$, als auch die dazu gehörige Nichtteilbarkeit $\nmid$ eine atomare Relation.

Definition 7: Die Erweiterung der Presburger Arithmetik auf die ganzen Zahlen vgl. [Zyg91, Coo72]
Es sei:

1. $\Sigma_{\mathrm{PA}_\mathbb{Z}} = \{\forall, \exists, \wedge, \vee, \rightarrow, \leftrightarrow, \neg,), (, +, -, =, 0, 1, x, \leq, |, \nmid\}$
2. $\mathcal{L}_{\mathrm{PA}_\mathbb{Z}} = \{\phi \in \mathcal{L}_{\mathrm{Peano}} \mid \phi \in \Sigma^*_{\mathrm{PA}_\mathbb{Z}}\}$
3. $\mathcal{S}_{\mathrm{PA}_\mathbb{Z}} = (\mathbb{Z}; +; \leq, |, \nmid; 0, 1)$

Auch in dieser Struktur kann $\leq$ ähnlich wie in Beispiel 3 genutzt werden, um die übrigen üblichen Ordnungsrelationen zu definieren.

3.2 Alternierende Turingmaschinen

Eine alternierende Turingmaschine (ATM), wie von Chandra und Stockmeyer in „Alternation“ [CS76] vorgestellt, ist eine Verallgemeinerung einer nicht-deterministischen Turingmaschine (NTM). Während eine NTM akzeptiert, wenn es einen Berechnungspfad gibt, der zu einer Konfiguration mit akzeptierenden Zustand führt, ist dies bei den ATMs etwas komplizierter.

Um das Tupel der ATM zu erhalten, wird das Tupel $(Z, \Gamma, \delta, z_0, A, V)$ der NTM um eine weitere Menge $\bigwedge \supset Z$ erweitert. Nun kann zum Zweck der Erklärung hinter die Konfigurationen $\alpha_1, ..., \alpha_n$ die jeweilige Zustandsmenge des Zustands der Konfiguration geschrieben werden. Beispielsweise wird für eine Konfiguration α_i mit Zustand $z_j \in A$, $\alpha_i(A)$ geschrieben. Ein Zustand, der in keiner der im Tupel gegebenen Zustandsmengen A, V enthalten ist, sei in $\bigvee$.

Mithilfe dieser Notation kann rekursiv über den Ablauf der Konfigurationen mit einer Funktion

$$f : \text{Konfiguration} \rightarrow \{0, 1\}$$

bestimmt werden, ob eine Eingangskonfiguration zu einer Akzeptierenden wird.

Im Fall der NTM wird überprüft, ob es für die Konfiguration α_1 mit Zustand z_0 einen Berechnungspfad

$$\alpha_1(\bigvee) \vdash \alpha_2(\bigvee) \vdash \cdots \vdash \alpha_n(A)$$

gibt und die NTM damit akzeptiert (vgl. Abbildung 1). Als Funktion wäre das:

$$f(\alpha(X)) = \begin{cases} 1 & \text{falls } X = A \\ 0 & \text{falls } X = V \\ \bigvee_{\alpha \vdash \beta} f(\beta) & \text{sonst (Fall: } \bigvee) \end{cases}$$

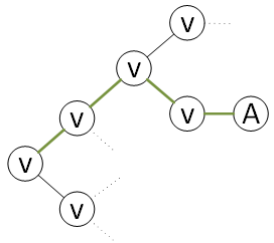

Abbildung 1: Berechnungspfad einer NTM mit Zustandsmengen

Im Fall der ATM werden die möglichen Zustandsmengen um $\bigwedge$ erweitert. Dadurch können Konfigurationen $\alpha(\bigwedge)$ erreicht werden, die nur dann akzeptieren, wenn alle möglichen Folgekonfigurationen β mit $\alpha \vdash \beta$ akzeptieren (vgl. Abbildung 2). Das ist genau dann der Fall, wenn für alle Folgekonfigurationen β $f(\beta) = 1$ ist.

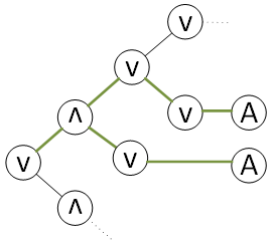

Abbildung 2: Berechnungspfad einer ATM mit Zustandsmengen

$$f(\alpha(X)) = \begin{cases} 1 & \text{falls } X = A \\ 0 & \text{falls } X = V \\ \bigwedge_{\alpha \vdash \beta} f(\beta) & \text{falls } X = \bigwedge \\ \bigvee_{\alpha \vdash \beta} f(\beta) & \text{sonst} \end{cases} \quad (1)$$

Die Zustände $z \in \bigwedge$ werden universelle Zustände genannt. Die übrigen Zustände, die in $Z \setminus (A \cap V \cap \bigwedge)$ liegen, werden in der Menge $\bigvee$ zusammengefasst und heißen existenzielle Zustände. Dann kann die „alternierende Turingmaschine“ definiert werden:

Definition 8: Alternierende Turingmaschinen Σ_k, Π_k (ähnlich der in [CS76])
Im Fall $k \geq 1$ ist eine alternierende Turingmaschine Σ_k bzw. Π_k ein Tupel $(Z, \Gamma, \delta, z_0, A, V, \bigwedge)$ (s.o.), wobei in den möglichen Berechnungspfaden

$$\alpha_1 \vdash^* \alpha_2 \vdash^* \cdots \vdash^* \alpha_{k-1} \vdash^* \alpha_k \vdash \alpha_{Ende}$$

α_1 einen Zustand aus der Zustandsmenge $\bigvee$ bzw. $\bigwedge$ hat. $\alpha_i \vdash^* \alpha_{i+1}$ steht für alle $1 \leq i \leq k-1$ für eine Konfigurationsfolge $\alpha_i(X) \vdash \alpha_{i_1}(X) \vdash \cdots \vdash \alpha_{i_y}(X) \vdash \alpha_{i+1}(Y)$, wobei $X, Y \in \{\bigvee, \bigwedge\}$ verschieden sind ($y \in \mathbb{N} \cup \{0\}$). Insgesamt gibt es demzufolge $k-1$ „Zustandsmengenwechsel“ zwischen $\bigvee$ und $\bigwedge$.
Im Fall $k = 0$ bezeichnet Σ_0 bzw. Π_0 eine deterministische Turingmaschine, die in Polynomialzeit läuft.

Nun lässt sich der übliche Zeitbegriff auf ATMs übertragen. Die Laufzeit einer ATM wird analog zu der Laufzeit einer NTM definiert. Sie ist die Anzahl der Rechenschritte, durch die alle möglichen Berechnungspfade der ATM begrenzt sind. Eine ATM Σ_k bzw. Π_k arbeitet in Zeit $f : \mathbb{N} \to \mathbb{N}$, falls für alle Eingaben w die Laufzeit von Σ_k bzw. Π_k durch $f(|w|)$ beschränkt ist. [AB09]

Dann wird Σ_k-TIME bzw. Π_k-TIME wie folgt definiert:

Definition 9: Σ_k-TIME bzw. Π_k-TIME vgl. [AB09]
Es sei $t : \mathbb{N} \to \mathbb{N}$ eine Funktion. Die Komplexitätsklasse Σ_k-TIME(t) bzw. Π_k-TIME(t) besteht aus allen Sprachen A, für die es eine ATM Σ_k bzw. Π_k gibt, die A entscheidet und in Zeit $O(t)$ arbeitet.

3.3 Die Polynomialzeithierarchie

Die Polynomialzeithierarchie ist eine rekursiv definierte Komplexitätshierarchie. Sie wird definiert, um weitere Probleme, zu denen keine explizite untere Schranke für die Laufzeit gefunden werden kann, in größere möglichst komplexitätshomogene Klassen zu teilen [Sto76]. Dieser Fall tritt schon bei Problemen aus $\mathcal{NP}$ auf, da nicht klar ist, ob $\mathcal{P} = \mathcal{NP}$ gilt oder nicht, und damit keine explizite untere Schranke für die Laufzeit dieser Probleme bekannt ist. Jedoch erstreckt sich diese Herausforderung auf weitere Probleme. Als Beispiel dafür bietet es sich an, auf die Komplexität des Entscheidungsproblems der Sprache B_m der wahren quantifizierten aussagenlogischen Formeln einzugehen.

Definition 10: Die Sprache B_m bzw. $\overline{B}_m$ vgl. [Grä88, Wra76]
B_m bzw. $\overline{B}_m$ bezeichnet die Sprache der wahren quantifizierten aussagenlogischen Formeln mit $m-1$ Quantorenwechsel beginnend mit einem existenziellen bzw. universellen Quantorenblock.

$$B_m \coloneqq \Big\{ \phi \coloneqq \overbrace{\underbrace{\exists x_1^1 ... \exists x_{i_1}^1}_{\text{existenzieller Quantorenblock}} \quad \underbrace{\forall x_1^2 ... \forall x_{i_2}^2}_{\text{universeller Quantorenblock}} \quad \cdots \quad Q x_1^m ... Q x_{i_m}^m}^{\text{m Quantorenblöcke}} \quad \underbrace{F(x_1^1, ..., x_{i_m}^m)}_{\text{aussagenlogische Formel}} \mid \phi = 1 \Big\}$$

bzw.

$$\overline{B_m} \coloneqq \Big\{ \phi \coloneqq \forall x_1^1 ... \forall x_{i_1}^1 \ \exists x_1^2 ... \exists x_{i_2}^2 \ \cdots \ Q x_1^m ... Q x_{i_m}^m F(x_1^1, ..., x_{i_m}^m) \mid \phi = 1 \Big\}$$

wobei

$$Q = \begin{cases} \exists \text{ bzw. } \forall \text{ falls } m \text{ ungerade} \\ \forall \text{ bzw. } \exists \text{ falls } m \text{ gerade} \end{cases}$$

ist und

- $\Phi \coloneqq \exists x \phi$ bedeutet, dass falls es eine Belegung mit $x \in \{0,1\}$ gibt, sodass ϕ wahr wird, dann gilt Φ.
- $\Phi \coloneqq \forall x \phi$ bedeutet, dass falls für alle $x \in \{0,1\}$ ϕ wahr ist, dann gilt Φ.

B_1 ist polynomiell-überprüfbar und es gilt außerdem $B_1 = \text{SAT}$, wodurch sich die $\mathcal{NP}$-vollständigkeit auf B_1 überträgt. Auch $\overline{B_1}$ besitzt ein äquivalentes Problem namens TAUT. TAUT ist jedoch ein Problem aus $co\mathcal{NP}$.[AB09]

Definition 11: $co\mathcal{NP}$ vgl. [AB09]
Eine Sprache $L \subseteq \{0,1\}^*$ ist in $co\mathcal{NP}$, falls es einen Algorithmus V gibt, sodass für alle Eingaben w gilt:

$$w \in L \Leftrightarrow \text{Für alle Zertifikate } x \text{ gilt, dass } V \text{ bei Eingabe } < w, x > \text{ akzeptiert}$$

Damit ist die Komplexität von B_1 und $\overline{B_1}$ geklärt, es stellt sich jedoch die Frage, wie es sich

mit B_m bzw. $\overline{B_m}$ mit $m > 1$ verhält.

Beispiel 5: Der Fall B_2

B_2 ist die Menge der wahren quantifizierten aussagenlogischen Formeln der Form:

$$\exists x_1^1 ... \exists x_{i_1}^1 \forall x_1^2 ... \forall x_{i_2}^2 F(x_1^1, ..., x_{i_1}^1, x_1^2, ..., x_{i_2}^2)$$

Für B_1 konnte einfach die erfüllende Belegung als Zertifikat für die polynomielle Überprüfbarkeit gewählt werden. Das geht im Fall von B_2 jedoch nicht. Ein Zertifikat müsste zunächst ein passende Belegung für die $x_1^1, ..., x_{i_1}^1$ liefern, dann allerdings auch enthalten, dass für alle möglichen Belegungen für $x_1^2, ..., x_{i_2}^2$ F erfüllt ist. Ein Polynomialzeitalgorithmus kann jedoch nicht 2^{i_2} Belegungen testen. (Das ist kein Beweis für $B_2 \notin \mathcal{NP}$!)

Bei dem Versuch, die Idee hinter dem Zertifikat für die Inklusion in $\mathcal{NP}$ und die Unabhängigkeit von der Eingabe im Fall von $co\mathcal{NP}$ zu kombinieren, ergibt sich die Definition der Polynomialzeithierarchie:

Definition 12: Polynomialzeithierarchie vgl. [AB09]
Für alle $k \geq 1$ ist eine Sprache L in Σ_k^P bzw. Π_k^P, wenn es einen Algorithmus V und ein Polynom q gibt, sodass für alle Eingaben w gilt:

$$w \in L \Leftrightarrow \exists u_1 \in \{0,1\}^{q(|w|)} \forall u_2 \in \{0,1\}^{q(|w|)} ... Q_k u_k \in \{0,1\}^{q(|w|)} V(w, u_1, u_2, ..., u_k) = 1$$

bzw.

$$w \in L \Leftrightarrow \forall u_1 \in \{0,1\}^{q(|w|)} \exists u_2 \in \{0,1\}^{q(|w|)} ... Q_k u_k \in \{0,1\}^{q(|w|)} V(w, u_1, u_2, ..., u_k) = 1$$

Die Laufzeit von V bei Eingabe $< w, u_1, u_2, ..., u_k >$ ist dabei durch ein Polynom in $|w|$ beschränkt und es ist:

$$Q_k = \begin{cases} \exists \text{ bzw. } \forall \text{ falls } k \text{ ungerade} \\ \forall \text{ bzw. } \exists \text{ falls } k \text{ gerade} \end{cases}$$

Im Fall von $k = 1$ zeigt sich durch die Definition von $\mathcal{NP}$, $co\mathcal{NP}$ und der Polynomialzeithierarchie, dass $\Sigma_1^P = \mathcal{NP}$ und $\Pi_1^P = co\mathcal{NP}$ gilt. B_1 bzw. $\overline{B_1}$ ist demnach vollständig für Σ_1^P bzw. für Π_1^P. Für B_2 kann, wie in Definition 12 benötigt, ein Algorithmus V entworfen werden, der bei Eingabe von F sowie einer Belegung für $x_1^1, ..., x_{i_1}^1$ in Form eines Zertifikates u_1 und einer beliebigen Belegung für $x_1^2, ..., x_{i_2}^2$ in Form eines weiteren Zertifikates u_2 nur dann 1 zurückgibt, falls das quantifizierte F wahr ist.
B_2 ist dementsprechend in Σ_2^P enthalten und es kann gezeigt werden, dass B_2 auch vollständig für Σ_2^P ist. Im Allgemeinen führt das zu folgendem Satz, der an dieser Stelle ohne Beweis angegeben werden soll:

Satz 1: Die Komplexität von B_m bzw. $\overline{B_m}$ vgl. [Wra76]
B_m bzw. $\overline{B_m}$ ist Σ_m^P - vollständig bzw. Π_m^P - vollständig.

Ähnlich wie für SAT und 3SAT gibt es ein Resultat für B_m und $\overline{B_m}$. Dafür werden die quan-

tifizierten aussagenlogischen Formeln in 3KNF mit jeweils 3 Konjunkten pro Klausel oder alternativ in 3DNF mit 3 Disjunkten pro Konjunkt betrachtet.

Satz 2: Die Komplexität von B_m bzw. $\overline{B_m}$ eingeschränkt auf Formeln in 3KNF oder 3DNF vgl. [Wra76]
$B_m \cap$ 3KNF bzw. $B_m \cap$ 3DNF ist vollständig für Σ_m^P und $\overline{B_m} \cap$ 3KNF bzw. $\overline{B_m} \cap$ 3DNF ist vollständig für Π_m^P. (Hier werden 3DNF und 3KNF als Mengen von Formeln in 3DNF bzw. 3KNF behandelt.)

Üblicherweise wird die Polynomialzeithierarchie mithilfe von Orakel Turingmaschinen definiert (siehe Anhang 8.1). Im Kontext der Presburger Arithmetik ist jedoch die folgende zu Definition 12 äquivalente Charakterisierung der Polynomialzeithierarchie mithilfe von alternierenden Turingmaschinen nützlicher.

Satz 3: Die Polynomialzeithierarchie mit alternierenden Turingmaschinen vgl. [Wra76, CS76]
Die Definition 12 ist äquivalent zu:

$$\Sigma_k^P = \Sigma_k\text{-TIME}\left(n^{O(1)}\right)$$
$$\Pi_k^P = \Pi_k\text{-TIME}\left(n^{O(1)}\right)$$

Σ_k^P bzw. Π_k^P ist demzufolge auch die Menge der Sprachen, die von einer Σ_k bzw. Π_k ATM in Polynomialzeit entschieden werden können.

3.4 Lösung eines speziellen Systems von Teilbarkeiten und der chinesische Restsatz

Im Fall des chinesischen Restsatzes wird für ein System von Kongruenzen ($x_0 \equiv c_i \pmod{\alpha_i}$ $(1 \leq i \leq n)$), eine Lösung $x_0 \leq kgV(\alpha_1, \alpha_2, ..., \alpha_n)$ gesucht, die alle Kongruenzen gleichzeitig löst. Wenn es so eine Lösung gibt, dann ergibt sich gleich ein ganzer Lösungsraum $\mathbb{L}_{x_0} = \{x \mid x \coloneqq x_0 + k * kgV(\alpha_1, \alpha_2, ..., \alpha_n) \text{ mit } k \in \mathbb{Z}\}$, sodass jedes $x \in \mathbb{L}_{x_0}$ das System von Kongruenzen löst.

Satz 4: Existenz einer Lösung durch den Chinesische Restsatz vgl. [CLRS17]
Eine Lösung x für ein System von Kongruenzen

$$x \equiv c_i \pmod{\alpha_i} \ (1 \leq i \leq n)$$

gibt es genau dann, wenn alle α_i paarweise teilerfremd sind. (Der Algorithmus zur Berechnung von x läuft in Polynomialzeit [CLRS17])

Beispiel 6: Kodierung einer Belegung

Gegeben sei eine Belegung $I(x_1, x_2, x_3, x_4) = (0, 1, 1, 0)$. Die ersten vier Primzahlen lauten $2, 3, 5, 7$ und sind notwendigerweise paarweise teilerfremd. Die Belegung lässt sich somit mittels x wie folgt kodieren:

$$x \equiv 0 \pmod 2$$
$$x \equiv 1 \pmod 3$$
$$x \equiv 1 \pmod 5$$
$$x \equiv 0 \pmod 7$$

Laut Satz 4 gibt es dann eine Lösungen $x \equiv 196 \pmod{210}$, die die Belegung kodiert. (Weiteres zur Kodierung von Belegungen in Kapitel 6.2.3)

Falls es eine Lösung x_0 gibt, kann anhand von $\mathbb{L}_{x_0}$ gesehen werden, dass es immer ein Element $x' \in \mathbb{L}_{x_0}$ gibt, das in dem Intervall $I \coloneqq [i, i + kgV(m_1, m_2, ..., m_n) - 1)](i \in \mathbb{Z})$ liegt [CLRS17]. Die Idee ist nun, ein ähnliches Resultat für ein System von Teilbarkeiten zu zeigen. In Abbildung 3 sind Lösungen des Systems auf einem Zahlenstrahl rot eingezeichnet. Wie im Fall des chinesischen Restsatzes gibt es unendlich viele Lösungen. Diese treten im Abstand von $kgV(\beta_1, \beta_2, ..., \beta_n)$ auf, sodass ein Bereich I begonnen bei i mit der Größe $kgV(\beta_1, \beta_2, ..., \beta_n) - 1$ immer eine Lösung für das System enthält.

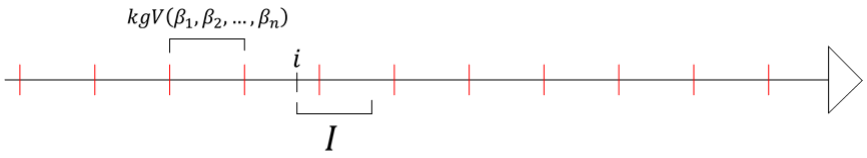

Abbildung 3: Zahlenstrahl mit Lösungen des Systems in rot und eingezeichneten Intervallen

Satz 5: Lösung eines Systems von Teilbarkeiten
Für eine System von Teilbarkeiten der Form:

$$\beta_1 \mid \alpha_1 x + t_1$$
$$\beta_2 \mid \alpha_2 x + t_2$$
$$\vdots$$
$$\beta_n \mid \alpha_n x + t_n$$

gilt, falls es eine Lösung x für das System gibt, dann gibt es auch eine Lösung x' für ein beliebiges Intervall $I \coloneqq [i, i + kgV(\beta_1, \beta_2, ..., \beta_n) - 1]$ mit $x' \in I$ $(i \in \mathbb{Z})$.

Beweis. Angenommen das System hat eine Lösung $x \notin I$, aber keine Lösung $x' \in I$. Dann ist $x < i$ oder $x > i + kgV(\beta_1, \beta_2, ..., \beta_n) - 1$.

Für den ersten Fall muss $x = i - c_1$ mit $c_1 \geq 1$ sein. Das ist aber gleichbedeutend mit $x = i + j_1 - j_2 * kgV(\beta_1, \beta_2, ..., \beta_n)$, wobei $0 \leq j_1 \leq kgV(\beta_1, \beta_2, ..., \beta_n) - 1$ und $j_2 \geq 1$ ist. Durch einsetzen ergibt sich:

$$\begin{gathered}
\beta_1 \mid \alpha_1 * (i + j_1 - j_2 * kgV(\beta_1, \beta_2, ..., \beta_n)) + t_1 \\
\beta_2 \mid \alpha_2 * (i + j_1 - j_2 * kgV(\beta_1, \beta_2, ..., \beta_n)) + t_2 \\
\vdots \\
\beta_n \mid \alpha_n * (i + j_1 - j_2 * kgV(\beta_1, \beta_2, ..., \beta_n)) + t_n \\
\Leftrightarrow \\
\beta_1 \mid \alpha_1 * i + \alpha_1 * j_1 - \alpha_1 * j_2 * kgV(\beta_1, \beta_2, ..., \beta_n) + t_1 \\
\beta_2 \mid \alpha_2 * i + \alpha_2 * j_1 - \alpha_2 * j_2 * kgV(\beta_1, \beta_2, ..., \beta_n) + t_2 \\
\vdots \\
\beta_n \mid \alpha_n * i + \alpha_n * j_1 - \alpha_n * j_2 * kgV(\beta_1, \beta_2, ..., \beta_n) + t_n \\
\Leftrightarrow \text{ (da } \beta_i \mid kgV(\beta_1, \beta_2, ..., \beta_n)) \\
\beta_1 \mid \alpha_1 * i + \alpha_1 * j_1 + t_1 \\
\beta_2 \mid \alpha_2 * i + \alpha_2 * j_1 + t_2 \\
\vdots \\
\beta_n \mid \alpha_n * i + \alpha_n * j_1 + t_n \\
\Leftrightarrow \\
\beta_1 \mid \alpha_1 * (i + j_1) + t_1 \\
\beta_2 \mid \alpha_2 * (i + j_1) + t_2 \\
\vdots \\
\beta_n \mid \alpha_n * (i + j_1) + t_n
\end{gathered}$$

Das bedeutet aber, dass es dann auch eine Lösung $x' = i + j_1 \in I$ gibt ($0 \leq j_1 \leq kgV(\beta_1, \beta_2, ..., \beta_n) - 1$). Dies ist ein Widerspruch zur Annahme, dass es in dem Fall keine Lösung in I gibt.
Für den zweiten Fall muss $x = i + kgV(\beta_1, \beta_2, ..., \beta_n) + c_2$ mit $c_2 \geq 0$ sein. Das ist wiederum gleichbedeutend mit $x = i + kgV(\beta_1, \beta_2, ..., \beta_n) + j_1 + j_2 * kgV(\beta_1, \beta_2, ..., \beta_n)$, wobei $0 \leq j_1 \leq kgV(\beta_1, \beta_2, ..., \beta_n) - 1$ und $j_2 \geq 0$ ist.
Durch einsetzen ergibt sich wieder:

$$\begin{gathered}
\beta_1 \mid \alpha_1 * (i + kgV(\beta_1, \beta_2, ..., \beta_n) + j_1 + j_2 * kgV(\beta_1, \beta_2, ..., \beta_n)) + t_1 \\
\beta_2 \mid \alpha_2 * (i + kgV(\beta_1, \beta_2, ..., \beta_n) + j_1 + j_2 * kgV(\beta_1, \beta_2, ..., \beta_n)) + t_2 \\
\vdots \\
\beta_n \mid \alpha_n * (i + kgV(\beta_1, \beta_2, ..., \beta_n) + j_1 + j_2 * kgV(\beta_1, \beta_2, ..., \beta_n)) + t_n \\
\Leftrightarrow \\
\beta_1 \mid \alpha_1 * (i + j_1) + t_1 \\
\beta_2 \mid \alpha_2 * (i + j_1) + t_2
\end{gathered}$$

$$\vdots$$
$$\beta_n \mid \alpha_n * (i + j_1) + t_n$$

Was wiederum aber bedeutet, dass es eine Lösung $x' = i + j_1 \in I$ gibt ($0 \leq j_1 \leq kgV(\beta_1, \beta_2, ..., \beta_n) - 1$). Dies ist wiederum ein Widerspruch zur Annahme, dass es in diesem Fall keine Lösung in I gibt.
Somit ist die Annahme falsch und es gilt: Falls es eine Lösung $x \notin I$ gibt, dann gibt es auch eine Lösung $x' \in I$. □

Das Ganze lässt sich auch auf Nichtteilbarkeitsrelationen erweitern: $(\beta \nmid t) \equiv \bigvee_{i=1}^{\beta-1}(\beta \mid t + i)$ [Coo72]. Für die Nichtteilbarkeit muss ein System gefunden werden, in dem einer der Teilbarkeitsrelationen auf der rechten Seite erfüllt ist. Alternativ könnte auch der linke Term durch den Rechten ersetzt werden. In beiden Fällen bleibt das β gleich, sodass der obige Satz angewandt werden kann.

Beispiel 7: Anwendung von Satz 5 mit Nichtteilbarkeiten

Es sei
$$(5 \mid 2x) \wedge (3 \mid 4x) \wedge (7 \nmid x)$$
ein System von Teilbarkeits- und Nichtteilbarkeitsrelationen. Es zeigt sich, dass $x = 15$ eine Lösung dieses Systems ist. Außerdem ist das kleinste gemeinsame Vielfache von $3, 5$ und 7 gleich 105. Dann besagt Satz 5, dass ein beliebiges $i \in \mathbb{Z}$ gewählt werden kann, sodass $I = [i, i + kgV(3, 5, 7) - 1]$ für jedes gewählte i eine Lösung für das gegebene System von Teilbarkeiten enthält.
Zum Beispiel ist für $i = 20$, $I = [20, 20+105-1] = [20, 124]$ und enthält mit $x = 120$ eine weitere Lösung des Systems. Auch für $i = -200$ enthält $I = [-200, -200 + 105 - 1] = [-200, -96]$ mit $x = -195$ eine Lösung des Systems.

3.5 Das Filtern von Belegung in quantifizierten Formeln

Bei der Polynomialzeitreduktion auf Formeln in der Presburger Arithmetik ist es häufig hilfreich, bestimmte Belegungen von Formeln ausschließen zu können. Gegeben sei die Formel $\phi := \forall x_1 \exists x_2 \; x_1 + x_2 = 1$. Dann gilt nicht $\mathcal{S}_{\text{PA}} \models \phi$, da es in dem Universum $\mathbb{N}$ keine Zahlen kleiner Null gibt. ϕ wäre doch in einer Struktur $\mathcal{S}_{\text{PA}}$ mit einem auf $\{0, 1\}$ eingeschränkten Universum wahr. Dafür wird nur eine Möglichkeit benötigt, das Universum $\mathbb{N}$ mittels einer Erweiterung der Formel einzuschränken.
Zu diesem Zweck wird zunächst das Symbol $\rightsquigarrow^k$ definiert.

Definition 13: Das Symbol $\rightsquigarrow^k$
Für eine Zahl $k \geq 1$ sei $\rightsquigarrow^k$ ein neues Symbol abhängig von der Wahl von Q_k:
$$\rightsquigarrow^k := \begin{cases} \wedge & \text{falls } Q_k = \exists \\ \rightarrow & \text{falls } Q_k = \forall \end{cases}$$

Dann zeigt sich, dass der folgende Zusammenhang gilt:

Satz 6: Filtern von Belegungen in quantifizierten Formeln
Es sei S eine Struktur mit Universum G und

$$A(x) : G \to \{0, 1\}$$

eine Filterfunktion. Außerdem sei S' die Struktur S mit eingeschränktem Universum

$$G_A = \{x \in G \mid A(x) = 1\}$$

Dann gilt für eine Funktion F ohne weitere Quantoren:

$$S \models Q_1 x_1 \ (A(x_1) \rightsquigarrow^1 (Q_2 x_2 \ (A(x_2) \rightsquigarrow^2 \cdots (Q_s x_s \ (A(x_s) \rightsquigarrow^s (F(x_1, x_2, ..., x_s)))) \cdots)))$$
$$\Leftrightarrow$$
$$S' \models Q_1 x_1 Q_2 x_2 ... Q_s x_s F(x_1, x_2, ..., x_s)$$

(Satz inspiriert durch das Vorgehen in [Sch97])

Beweis. Für den Beweis wird die gepatchte Belegung β_x^a benutzt, in der a ein Element des Universums der jeweiligen Struktur und x eine Variable ist:

$$\beta_x^a(y) = \begin{cases} a & \text{falls x = y} \\ \beta(y) & \text{sonst} \end{cases}$$

Außerdem sei im Beweis $\beta(y) \in G_A$ für alle freien Variablen y. Das kann angenommen werden, da es in der finalen Aussage keine freien Variablen gibt. Dann kann über den Formelaufbau von ϕ_1 bzw. ϕ_2 der Satz gezeigt werden:
IA) Es gilt

$$(S, \beta) \models F(x_1, x_2, ..., x_s) \Leftrightarrow (S', \beta) \models F(x_1, x_2, ..., x_s)$$

da der Unterschied zwischen den Interpretationen nur in den Universen liegt und β die freien Variablen jeweils nur mit Zahlen aus G_A belegt (Das war die obige Annahme).

IS) Es sei ϕ eine Formel. Dann gilt:

$$\begin{aligned}
(S', \beta) \models \forall x \phi \ &\Leftrightarrow \ (S', \beta_x^a) \models \phi \text{ für alle } a \in G_A \text{ und } x \text{ eine Variable} \\
&\Leftrightarrow \ (S', \beta_x^a) \models \phi \text{ für alle } a \in G, \text{ für die } A(a) = 1, \text{ und } x \text{ eine Variable} \\
&\Leftrightarrow \ (S, \beta_x^a) \models A(x) \to \phi \text{ für alle } a \in G \text{ und } x \text{ eine Variable} \\
&\Leftrightarrow \ (S, \beta) \models \forall x \ (A(x) \to \phi)
\end{aligned}$$

und

$$(S', \beta) \models \exists x \phi \ \Leftrightarrow \ (S', \beta_x^a) \models \phi \text{ für ein } a \in G_A \text{ und } x \text{ eine Variable}$$

$$
\begin{aligned}
&\Leftrightarrow && (S', \beta_x^a) \models \phi \text{ für ein } a \in G, \text{ für das} \\
&&& A(a) = 1, \text{ und } x \text{ eine Variable} \\
&\Leftrightarrow && (S, \beta_x^a) \models A(x) \wedge \phi \text{ für ein } a \in G \text{ und} \\
&&& x \text{ eine Variable} \\
&\Leftrightarrow && (S, \beta) \models \exists x \ (A(x) \wedge \phi)
\end{aligned}
$$

Das ist die Erfüllbarkeitsäquivalenz zwischen den einzigen, sich semantisch unterscheidenden, Symbolen in S und S'. Für Sätze, in denen keine freie Variablen vorkommen können, gilt folglich die obige Aussage. □

Für das obige Beispiel bedeutet das:

Beispiel 8: Anwendung von Satz 6

Wie zu Beginn des Kapitels erklärt, sei:

$$\phi_1 := \forall x_1 \exists x_2 \ x_1 + x_2 = 1$$

Gesucht ist nun eine Filterfunktion, die das Universum von $\mathcal{S}_{\text{PA}}$ auf $\{0, 1\}$ einschränkt. Dafür sei:

$$A(x) := (x = 1) \vee (x = 0)$$

Dann besagt Satz 6, dass für

$$\phi_2 := \forall x_1 \ (A(x_1) \rightarrow \exists x_2 \ (A(x_2) \wedge x_1 + x_2 = 1))$$

gilt, dass

$$\mathcal{S}_{\text{PA}} \models \phi_2 \Leftrightarrow (\{0, 1\}; +; 0, 1) \models \phi_1$$

korrekt ist.

4 Entscheidbarkeit

Eine Sprache $\mathcal{L} \subseteq \Sigma^*$ heißt entscheidbar, wenn es eine Turingmaschine gibt, die bei Eingabe von $w \in \Sigma^*$ nach endlicher Zeit zurückgibt, ob $w \in \mathcal{L}$ gilt oder nicht. Was bedeutet nun der Entscheidbarkeitsbegriff im Kontext einer Arithmetik? Dafür wird die Theorie einer Struktur $\mathcal{S}$ über einer Sprache $\mathcal{L}$ definiert.

Definition 14: Die Theorie einer Struktur $\mathcal{S}$ über einer Sprache $\mathcal{L}$ (ähnlich wie in [Grä89])

$$Th(\mathcal{S}) \coloneqq \{\phi \in \mathcal{L} \mid \mathcal{S} \models \phi \wedge \#\mathbf{frei}(\phi) = 0\}$$

Die Theorie einer Struktur $\mathcal{S}$ über einer Sprache $\mathcal{L}$ ist die Menge der Sätze aus $\mathcal{L}$, die in der Interpretation von $\mathcal{S}$ wahr sind. (Auf die Angabe von $\mathcal{L}$ kann verzichtet werden, wenn sie aus dem Kontext klar ist.) Da es sich dabei um eine Sprache oder anderes ausgedrückt um eine Menge von Aussagen in der Prädikatenlogik handelt, kann der übliche Entscheidungsbegriff (s.o.) verwendet werden.

4.1 Ein Entscheidungsalgorithmus für $Th(\mathcal{S}_{\mathbf{PA}_{\mathbb{Z}}})$

In diesem Kapitel soll mittels der Quantoreliminierung aus „Theorem proving in arithmetic without multiplication“ [Coo72] gezeigt werden, dass es einen Algorithmus gibt, der die Menge $Th(\mathcal{S}_{\mathrm{PA}_{\mathbb{Z}}})$ der „wahren“ Sätze der erweiterten Presburger Arithmetik mit Universum $\mathbb{Z}$ entscheidet. Dabei wird eine Formel $\phi_0 \in \{\phi \in \mathcal{L}_{\mathrm{PA}_{\mathbb{Z}}} \mid \#\mathbf{frei}(\phi) = 0\}$ (also ein Satz aus $\mathcal{L}_{\mathrm{PA}_{\mathbb{Z}}}$) innerhalb von s „Schritten “ in eine Formel ϕ_s gebracht, die keine Variablen mehr enthält (s sei die Anzahl der gebundenen Variablen bzw. der Quantoren in ϕ_0 (siehe Definition 21)).

Beispiel 9: Das Ziel anhand eines einfachen Beispiels

Es sei $\phi_0 \coloneqq \exists x_2 \exists x_1\ (2x_1 = 12) \wedge (3x_2 = 12)$ eine Formel mit Dimension 2. (Die Skalarmultiplikation wurde schon angewandt!) Offensichtlich ist $\phi_0 \in \mathcal{L}_{\mathrm{PA}_{\mathbb{Z}}}$ und ein Satz, da $\mathbf{frei}(\phi_0) = 0$ ist. In Worten bedeutet ϕ_0 aber nur, dass 2 und 3 Teiler von 12 sind.

$$\begin{aligned} \phi_0 &\coloneqq \exists x_2 \exists x_1\ 2x_1 = 12 \wedge 3x_2 = 12 \\ &\Leftrightarrow \\ \phi_1 &\coloneqq \exists x_2\ 2x_2 = 12 \wedge 3 \mid 12 \\ &\Leftrightarrow \\ \phi_2 &\coloneqq 2 \mid 12 \wedge 3 \mid 12 \end{aligned}$$

Es gab also eine Formel ϕ_2 ohne Variablen und Quantoren, sodass $\mathcal{S}_{\mathrm{PA}_{\mathbb{Z}}} \models \phi_0$ genau dann gilt, wenn $\mathcal{S}_{\mathrm{PA}_{\mathbb{Z}}} \models \phi_2$ gilt. Es stellt sich heraus, dass es eine ähnliche Umformung für alle Sätze der Presburger Arithmetik mit Universum $\mathbb{Z}$ gibt.

Im Beispiel wird $\mid$ als atomare Relation benötigt, weshalb die benötigten Umformungen auch nicht in der Sprache $\mathcal{L}_{\mathrm{PA}_{\leq}}$ oder $\mathcal{L}_{\mathrm{PA}}$ durchgeführt werden. Stattdessen können die Sätze aus

$Th(\mathcal{S}_{\mathrm{PA}_\mathbb{Z}})$ mit den eigenen Mitteln entschieden werden. Dabei wird genutzt, dass das Alphabet $\Sigma_{\mathrm{PA}_\mathbb{Z}}$ weitere atomare Symbole zur Verfügung hat als $\Sigma_{\mathrm{PA}_\leq}$ oder Σ_{PA}.
Die Formeln ϕ_1,...,ϕ_s, die während des Algorithmus auftreten, sind in der Interpretation von $\mathcal{S}_{\mathrm{PA}_\mathbb{Z}}$ erfüllbarkeitsäquivalent zu der ursprünglich zu überprüfende Formel $\phi_0 \in \mathcal{L}_{\mathrm{PA}_\mathbb{Z}}$. In Bezug auf $Th(\mathcal{S}_{\mathrm{PA}_\mathbb{Z}})$ bedeutet das, dass entweder für alle $i \in [0, s]$ gilt, dass $\phi_i \in Th(\mathcal{S}_{\mathrm{PA}_\mathbb{Z}})$ ist, oder aber keines der ϕ_i in $Th(\mathcal{S}_{\mathrm{PA}_\mathbb{Z}})$ enthalten ist.
Der folgende Abschnitt beschreibt die nötigen Schritte des Algorithmus (Algorithm 1), der in Pseudocode am Ende des Kapitels zu finden ist. Die verwendeten Pseudocodefunktionen werde indirekt durch Zeilenverweise hinter den beschreibenden Schritten erklärt.
Um einen Algorithmus zu entwerfen, der eine Umformung wie im Beispiel 9 durchführt, ist es zunächst sinnvoll, die Formel ϕ_0 in eine einheitliche Form zu bringen. Wie angekündigt, wird im Folgenden die Skalarmultiplikation aus Definition 4 benutzt, sodass Terme die folgende Form bekommen:

$$c_1 * x_1 + c_2 * x_2 + ... + c_s * x_s + c_{s+1} \tag{2}$$

Das ist jedoch nur eine Abkürzung, weshalb das Symbol der Skalarmultiplikation auch nicht in den Alphabeten der Strukturen der Presburger Arithmetik auftaucht. Anschließend werden die redundanten Symbole $\{\leftrightarrow, \rightarrow, <, >, \geq, =\}$ entfernt, indem die Ausdrücke auf der linken Seite der Äquivalenzen durch die Ausdrücke auf der rechten Seite ersetzt werden (Zeile 2):

$$\begin{aligned}
\phi_1 \leftrightarrow \phi_2 &\equiv (\neg\phi_1 \vee \phi_2) \wedge (\neg\phi_2 \vee \phi_1) \\
\phi_1 \rightarrow \phi_2 &\equiv \neg\phi_1 \vee \phi_2 \\
t_1 < t_2 &\equiv \neg(t_2 \leq t_1) \\
t_1 > t_2 &\equiv \neg(t_1 \leq t_2) \\
t_1 \geq t_2 &\equiv t_2 \leq t_1 \\
t_1 = t_2 &\equiv t_1 \leq t_2 \wedge t_2 \leq t_1
\end{aligned}$$

Schließlich wird ϕ_0 in Pränexnormalform gebracht (Zeile 3), wodurch sie die folgende Form bekommt:

$$Q_s x_s \cdots Q_1 x_1 \delta_1 \quad \Big| \quad \begin{array}{l} \delta_1 \text{ ist das Ergebnis der bisherigen Umformungen} \\ \text{und ohne die führenden Quantoren} \end{array}$$

Nun wird mit Laufvariable i über die Anzahl s der gebundenen Variablen von innen nach außen iteriert (Zeile 4 und 6). In jeder Iteration wird auf eine Teilformel der Form $\exists x_i \delta_i$ die Quantoreliminierung nach Cooper (siehe nächster Abschnitt) durchgeführt (Zeile 8 und 11). Zu diesem Zweck wird, falls es sich um eine Teilformel der Form $\forall x_i \delta_i$ gehandelt hat, diese zu $\neg\exists x_i \neg\delta_i$ umgeformt (Zeile 8 und 9). Schließlich kann das Ergebnis der Quantoreliminierung δ_{i+1} wieder zurückgeschrieben werden (Zeile 9 und 12):

$$\phi_i \coloneqq \phi_{i-1} \text{ wobei } Q_i x_i \delta_i \text{ durch } \begin{cases} \delta_{i+1} & \text{falls } Q_i = \exists \\ \neg\delta_{i+1} & \text{falls } Q_i = \forall \end{cases} \text{ ersetzt wurde}$$

Diese Prozedur wird nun solange durchgeführt bis ϕ_{i+1} keine Quantoren mehr enthält. Die so entstandene Formel ϕ_s enthält keine Variablen mehr, daher kann durch Auswerten festgestellt werden, ob die so erstellte erfüllbarkeitsäquivalente Formel ϕ_s wahr ist (Zeile 15). (Das Vorgehen ist eine Abwandlung des Vorgehens aus [Coo72, Grä88])

Algorithm 1 Entscheidungsalgorithmus für $Th(\mathcal{S}_{\mathrm{PA}_{\mathbb{Z}}})$

```
1:  function MAIN(Formel φ_0)
2:      φ_0 ← entferneRedundanteSymbole(φ_0)
3:      φ_0 ← erzeugePränexnormalform(φ_0)
4:      int s ← anzahlDerQuantoren(φ_0)
5:      for i = 1 bis s do
6:          Teilformel δ_i ← zugehörigeTeilformel(φ_{i−1})
7:          if Q_i = ∀ then
8:              δ_{i+1} ← quantorEliminierung(∃x_i ¬δ_i)
9:              φ_i ← einsetzen(φ_{i−1}, ¬δ_{i+1})
10:         else
11:             δ_{i+1} ← quantorEliminierung(∃x_i δ_i)
12:             φ_i ← einsetzen(φ_{i−1}, δ_{i+1})
13:         end if
14:     end for
15:     return auswerten(φ_s)
16: end function
```

Beispiel 10: Der Ablauf des Algorithmus

Gegeben sei eine Formel $\phi_0 = \exists x_3 \exists x_2 \forall x_1 F(x_1, x_2, x_3)$. Sie ist das Ergebnis des Entfernen von redundanten Symbolen (Zeile 2) und dem anschließenden Umformen in Pränexnormalform. Aus diesem Grund enthält die Funktion F auch keine weiteren Quantoren. Der Algorithmus läuft dann wie folgt ab:

1. Variablenbelegung zu Beginn der **for**-Schleife: $s = 3$, $i = 1$
2. Zugehörige Teilformel von ϕ_0: $\delta_1 = F(x_1, x_2, x_3)$
3. Es ist $Q_1 = \forall$: Wende die Quantoreliminierung auf $\exists x_1 \neg\delta_1$ an und schreibe das Ergebnis in δ_2.
4. Einsetzen: $\phi_1 = \exists x_3 \exists x_2 \neg\delta_2$
5. Zugehörige Teilformel von ϕ_1: $\delta_2 = \neg\delta_2$

   ⋮

9. Einsetzen in ϕ_3: $\phi_3 = \delta_3$
10. Auswerten von ϕ_3 liefert das Ergebnis des Algorithmus

4.2 Quantoreliminierung nach Cooper

Wie wird nun eine Teilformel $\exists x_i \delta_i$ mit $\delta_i \in \{\wedge, \vee, \neg, \leq), (, +, -, 0, 1, x, |, \nmid\}^*$, ohne weitere Quantoren in δ_i, in eine erfüllbarkeitsäquivalent Formel ohne x_i gebracht? Wie funktioniert die Methode **quantorEliminierung**?

Wie im vorherigem Abschnitt befindet sich am Ende des Kapitels der beschriebene Algorithmus (Algorithm 2) in Pseudocode und ich werde wiederum hinter den beschreibenden Schritten die zugehörigen Zeilen des Pseudocodes aufführen. (Das Vorgehen ist eine Abwandlung des Vorgehens aus [Coo72, Grä88].)
Zunächst muss δ_i mithilfe der De Morgansche Gesetzen in die Negationsnormalform gebracht werden. Das ist nötig – obwohl ϕ_0 zu Beginn in Pränexnormalform gebracht wurde – da ein $\forall$ Quantor zu einem führenden Negationszeichen $\neg$ in δ_i führt. Anschließend werden die Negationszeichen $\neg$ mithilfe folgender Äquivalenzen in die Primformeln gezogen (Zeile 2):

$$\neg(c \mid t) \equiv c \nmid t$$
$$\neg(c \nmid t) \equiv c \mid t$$
$$\neg\neg\phi \equiv \phi$$
$$\neg(t_1 \leq t_2) \equiv t_2 + 1 \leq t_1$$

An dieser Stelle sollte der zu eliminierende $\exists$ Quantor über alle möglichen $\bigvee$ aus vorausgegangenen Quantoreliminierungen gezogen werden ($\exists x \bigvee \delta \equiv \bigvee \exists x \delta$) (Zeile 3). Die Quantoreliminierung funktioniert auch ohne diesen Schritt, jedoch wird dadurch die Argumentation in Kapitel 6.2.1 vereinfacht. Darauffolgend kann durch die beidseitige Addition/Subtraktion der $c_j * x_j$ dafür gesorgt werden, dass $c_i * x_i$ alleine auf einer Seite der $\leq$ steht und $c_i > 0$ ist ($j \in [i, s]$) (Zeile 5) (Mit $c_i * x_i$ bzw. $c_j * x_j$ sind die Terme dieser Form gemeint).
δ_i' ist als das Ergebnis der bisherigen Umformungen nur noch eine Konjunktion und Disjunktion von Primformeln der folgenden Form (α's sind beliebig aus $\mathbb{N}$ gewählt, die β's einer Formel können mit $\beta_1, ..., \beta_n \in \mathbb{N}$ wie in Satz 5 aufgezählt werden, t sind Terme wie in (2) nur mit Variablen x_k mit $k > i$):

$$\alpha * x_i \leq t \tag{3}$$
$$t \leq \alpha * x_i \tag{4}$$
$$\beta \mid (\alpha * x_i + t) \tag{5}$$
$$\beta \nmid (\alpha * x_i + t) \tag{6}$$
$$\text{Primformeln ohne } x_i \text{ werden im Folgenden „mitgeschleppt“} \tag{7}$$

Für die folgenden Erläuterungen zur tatsächlichen Quantoreliminierung werden die Primformeln der jeweiligen Form in Mengen zusammengefasst:

Definition 15: Mengen von Primformeln
Jede Primformel, in einer durch den bisherigen Prozess umgeformten δ_i', kann einer der Formen (3),(4),(5),(6) oder (7) zugeordnet werden. Zu jeder dieser Formen sei (i) die zugehörige Menge ($3 \leq i \leq 7$). In einer Disjunktion $\bigvee_{(3)}(\star) \coloneqq \bigvee_{p \in (3)}(\star)$ mit $p = \alpha * x_i \leq t$, kann in $\star$ einfach auf α und t zugegriffen werden. Im Algorithmus schreibe ich daher auch $p.\alpha$ und $p.t$.

Außerdem sei die Funktion F_i wie folgt definiert:

Definition 16: Die Funktion F_i
Es sei $F_i : \mathbb{Z}^{(s-(i-1))} \to \{0,1\}$ die Funktion $F_i(x_i, ..., x_s) \coloneqq \delta_i'$ nach $i-1$ Quantoreliminierungen.

F_i hängt von $x_i, ..., x_s$ ab. Jedoch soll nur x_i eliminiert werden. Da die übrigen Variablen schon vor x_i gebunden werden, kann bei Eliminierung von x_i davon ausgegangen werden, dass die $x_{i+1}, ..., x_s$ schon belegt sind. Diese Belegungen seien im weiteren Verlauf der Arbeit in der Belegungsfunktion ϱ_i gespeichert.

Definition 17: Die Belegungsfunktion ϱ_i für $x_{i+1}, ..., x_s$ im Kontext der Quantoreliminierung
ϱ_i sei die Belegungsfunktion für die Variablen $x_{i+1}, ..., x_s$ im Kontext der Quantoreliminierung bei der i-ten Ausführung.

Beispiel 11: Ergebnis der bisherigen „Normalisierung“

Für eine ϕ_0 mit $s = 2$ ist die Funktion

$$F_2 : \mathbb{Z}^{(2-(2-1))} \to \{0,1\}$$

mit

$$F_2(x_2) = 0 \vee 5 * x_2 \leq 20 \wedge 30 \leq 10 * x_2 \vee 2 \mid 3 * x_2 + 1$$

ein mögliches Ergebnis der bisherigen „Normalisierung“. Es ergeben sich die folgenden Mengen von Primformeln:

(3) $= \{5 * x_2 \leq 20\}$

(4) $= \{30 \leq 10 * x_2\}$

(5) $= \{2 \mid 3 * x_2 + 1\}$

Die grundlegende Idee der Quantoreliminierung ist zunächst, zwischen zwei Fällen zu unterscheiden. Entweder kann ein sehr großes $x_i \in \mathbb{Z}$ gefunden werden, sodass $F_i(x_i, \varrho_i(x_{i+1}), ..., \varrho_i(x_s))$ erfüllt ist oder es gibt eine kleinste obere Schranke t für αx_i durch eine Primformel aus (3), die den ersten Fall verhindert. In Abbildung 4 sind die möglichen oberen und unteren Schranken dargestellt. Außerdem sind wie in Abbildung 3 mögliche Lösungen des Systems aus Teilbarkeiten in rot dargestellt. Der grün hinterlegte Bereich markiert die möglichen Lösungen im ersten Fall ohne obere Schranken, während der rote Bereich die möglichen Lösungen im zweiten Fall hinterlegt, in dem es eine kleinste obere Schranke t gibt.

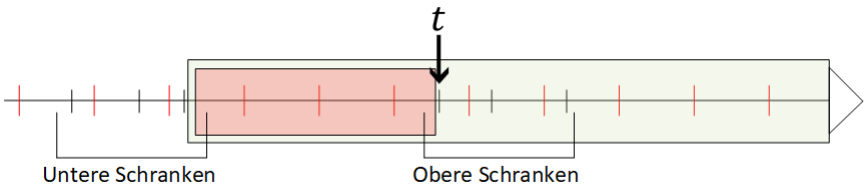

Abbildung 4: Fallunterscheidung bei der Quantoreliminierung

Der Trick ist, später mithilfe von Satz 5 beide Fälle auf endlich viele zu überprüfende Zahlen zu beschränken. Obwohl es sich bei den Formeln auch um Disjunktionen von Primformeln handeln kann, kann Satz 5 angewandt werden, da in diesem Fall mehr Zahlen als nötig getestet werden würden. Das liegt daran, dass das benötigte System der Teilbarkeiten und Nichtteilbarkeiten in diesem enthalten ist (siehe folgendes Beispiel).

Beispiel 12: Eine Disjunktion mit Systemen für Satz 5

Gegeben sei die folgende Disjunktion:

$$(4 \mid 5x) \wedge (20 \mid 2x) \vee (10 \mid 3x)$$

Dann kann Satz 5 auf die Systeme:

- $(4 \mid 5x) \wedge (20 \mid 2x)$
- $(10 \mid 3x)$

angewandt werden. Das kleinste gemeinsame Vielfache von 4 und 20 sowie von 10 ist jedoch kleiner als das kleinste gemeinsame Vielfache von 4, 20 und 10. Da das Lösungsintervall I aus Satz 5 durch einen größeres kleinstes gemeinsame Vielfache nur größer wird, kann in beiden Fällen auch der $kgV(4, 20, 10)$ verwendet werden.

Zunächst wird für den ersten Fall die Funktion F_∞ definiert. In dieser werden durch die Wahl einer sehr großen Zahl alle unteren Schranken erfüllt, wodurch es aber keine obere Schranken geben darf, die die Lösung beschränkt.

$F_\infty := F_i$ wobei Primformeln aus (3) bzw. (4) durch 0 (falsch) bzw. 1 (wahr) ersetzt werden.

Formal kann für die Fallunterscheidung die folgende Erfüllbarkeitsäquivalenz (vgl. [RL78]) formuliert werden:

$$(\mathcal{S}_{\mathrm{PA}_\mathbb{Z}}, \varrho_i) \models \exists x_i F_i(x_i, x_{i+1}, ..., x_s)$$
$$\Leftrightarrow$$
$$(\mathcal{S}_{\mathrm{PA}_\mathbb{Z}}, \varrho_i) \models \exists x F_\infty(x, x_{i+1}, ..., x_s) \vee \bigvee_{(3)} \exists x_i (F_i(x_i, x_{i+1}, ..., x_s))$$

Beweis. $\Rightarrow$: Falls $\#(3) \geq 1$ gilt, ist es trivial, da dann $\exists x_i F_i(x_i, x_{i+1}, ..., x_s)$ als Disjunkt auch rechts enthalten ist. Im Fall $\#(3) = 0$ gibt es keine Primformel der Form (3), die in F_∞ durch 0 ersetzt werden könnte. Die benötigten Primformeln in (5) und (6) werden durch das gegeben

x_i erfüllt, weshalb $F_\infty(x_i, \varrho_i(x_{i+1}), ..., \varrho_i(x_s))$ in diesem Fall auch von $(\mathcal{S}_{\mathrm{PA}_\mathbb{Z}}, \varrho_i)$ erfüllt werden muss.
$\Leftarrow$: Entweder gilt $(\mathcal{S}_{\mathrm{PA}_\mathbb{Z}}, \varrho_i) \models \exists x_i F_i(x_i, x_{i+1}, ..., x_s)$ oder $(\mathcal{S}_{\mathrm{PA}_\mathbb{Z}}, \varrho_i) \models \exists x F_\infty(x, x_{i+1}, ..., x_s)$. Der erste Fall ist genau die Aussage. Der zweite Fall bedeutet, dass es eine Belegung β für x gibt, sodass $F_\infty(\beta(x), \varrho_i(x_{i+1}), ..., \varrho_i(x_s))$ erfüllt ist. Demnach existiert eine Lösung für ein System aus Teilbarkeiten aus (5) und (6). Außerdem können Primformeln aus (3) ignoriert werden, da sie in F_∞ nicht zur Erfüllbarkeit beitragen. Wähle nun $x' = \sum_{\text{Primeformeln aus (4)}} t$, dann gibt es laut Satz 5 ein $j \in [0, kgV(\beta \text{ aus (5) und (6)}) - 1]$, sodass $F_i(x' + j, \varrho_i(x_{i+1}), ..., \varrho_i(x_s))$ erfüllt ist. (vgl. Abbildung 5). □

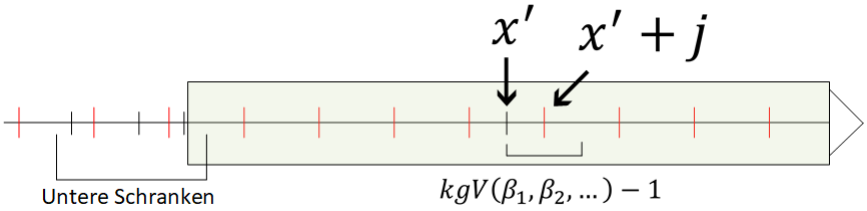

Abbildung 5: Erste Anwendung von Satz 5

In Beispiel 9 konnten die Quantoren eliminiert werden, indem stattdessen Teilbarkeiten benutzt wurden. Genau dieser Ansatz soll nun weiterverfolgt werden. Der folgende Satz ist deshalb zentral bei der Eliminierung von Quantoren.

Satz 7: Zentrale Äquivalenz der Quantoreliminierung
Es sei $F_i(\alpha x_i / x', x_{i+1}, ..., x_s)$ die Funktion, die entsteht, wenn in $F_i(x_i, x_{i+1}, ..., x_s)$ überall dort wo nicht α vor x_i steht, die Primformel mit α multipliziert und dann αx_i durch x' ersetzt wird. Die Aussage des Satzes ist die folgende Äquivalenz:

$$F_i(\alpha x_i / x', x_{i+1}, ..., x_s) \wedge (\alpha \mid x') \Leftrightarrow F_i(x_i, x_{i+1}, ..., x_s)$$

Durch das Ersetzen gilt nach der Anwendung des Satzes $x' \coloneqq \alpha x_i$ bzw. $x_i = \frac{x'}{\alpha}$.

Beweis.

$F_i(\alpha x_i / x', x_{i+1}, ..., x_s) \wedge (\alpha \mid x') \Leftrightarrow F_i(\alpha x_i / \alpha x_i, x_{i+1}, ..., x_s) \wedge (\alpha \mid \alpha x_i)$	$x' \coloneqq \alpha * x_i$
$\Leftrightarrow F_i(\alpha x_i / \alpha x_i, x_{i+1}, ..., x_s)$	$(\alpha \mid \alpha x_i)$ ist für jedes x_i richtig
$\Leftrightarrow F_i(x_i, x_{i+1}, ..., x_s)$	$\alpha x_i / \alpha x_i$ kann um α gekürzt und dann x/x durch x ersetzt werden

□

Durch Einsetzen der bewiesenen Äquivalenz ergibt sich:

$$(\mathcal{S}_{\mathrm{PA}_\mathbb{Z}}, \varrho_i) \models \exists x_i F_i(x_i, x_{i+1}, ..., x_s)$$

$$\Leftrightarrow \tag{8}$$
$$(\mathcal{S}_{\mathrm{PA}_\mathbb{Z}}, \varrho_i) \models \exists x F_\infty(x, x_{i+1}, ..., x_s) \vee \bigvee_{(3)} \exists x'(F_i(\alpha x_i / x', x_{i+1}, ..., x_s) \wedge (\alpha \mid x')$$

Zu beachten ist, dass das α nun abhängig von der jeweiligen Primformel aus (3) ist. Um das x_i zu eliminieren, muss erreicht werden, dass es nur eine endliche Anzahl an Möglichkeiten der Belegung für das x_i gibt, die dann mit $\bigvee$ aufgezählt werden können. Das ist tatsächlich der Fall:

Satz 8: Quantoreliminierung vgl. [Coo72, Grä88]

$$(\mathcal{S}_{\mathrm{PA}_\mathbb{Z}}, \varrho_i) \models \exists x_i F_i(x_i, x_{i+1}, ..., x_s)$$
$$\Leftrightarrow$$
$$(\mathcal{S}_{\mathrm{PA}_\mathbb{Z}}, \varrho_i) \models \bigvee_{j=0}^{\sigma-1} F_\infty(j, x_{i+1}, ..., x_s) \vee \bigvee_{(3)} \bigvee_{j=0}^{\alpha * \sigma - 1} (F_i(\alpha x_i / t - j, x_{i+1}, ..., x_s) \wedge (\alpha \mid t - j))$$

wobei $\sigma \coloneqq kgV(\beta_1, \beta_2, ...)$ ist. (Ideen zur Herleitung dieses Resultats aus [Coo72, YFC, RL78])

Damit aus 8 die Quantoreliminierung wie in Satz 8 folgt, sind noch die folgende Punkte zu zeigen:

1. $(\mathcal{S}_{\mathrm{PA}_\mathbb{Z}}, \varrho_i) \models \exists x F_\infty(x, x_{i+1}, ..., x_s) \Leftrightarrow (\mathcal{S}_{\mathrm{PA}_\mathbb{Z}}, \varrho_i) \models \bigvee_{j=0}^{\sigma-1} F_\infty(j, x_{i+1}, ..., x_s)$

   Beweis. $\Rightarrow$: Es gibt eine Belegung β für x, sodass $F_\infty(\beta(x), \varrho_i(x_{i+1}), ..., \varrho_i(x_s))$ wahr ist. Entweder gilt $\beta(x) \in [0, \sigma - 1]$ oder $\beta(x) \notin [0, \sigma - 1]$. Im zweiten Fall gibt es aber laut Satz 5 ein $x' \in [0, \sigma - 1]$, sodass $F_\infty(x', \varrho_i(x_{i+1}), ..., \varrho_i(x_s))$ wahr ist.
   $\Leftarrow$: Gesucht ist eine Belegung β für x. Die Voraussetzung liefert dafür $\beta(x) \coloneqq j$, sodass $F_\infty(\beta(x), \varrho_i(x_{i+1}), ..., \varrho_i(x_s))$ wahr ist. (Verdeutlichung dieses Zusammenhangs in Abbildung 6) □

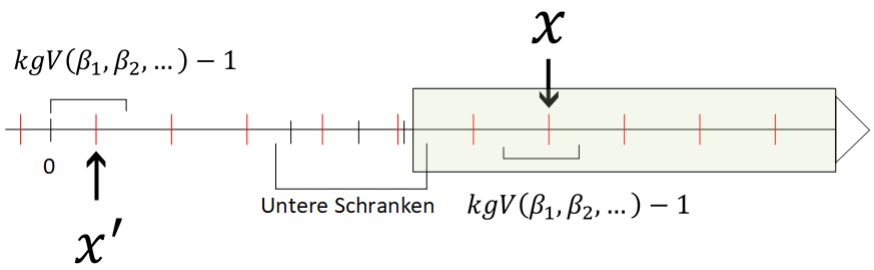

Abbildung 6: Endliche Menge an zu prüfenden Zahlen für Fall 1: Falls es ein $x' \in [0, \sigma - 1]$ gibt, liefert Satz 5 ein beliebig großes x, welches oberhalb der größten unteren Schranke liegt.

2. $$(\mathcal{S}_{\mathrm{PA}_\mathbb{Z}}, \varrho_i) \models \exists x'(F_i(\alpha x_i / x', x_{i+1}, ..., x_s) \wedge (\alpha \mid x'))$$

$$\Leftrightarrow$$

$$(\mathcal{S}_{\mathrm{PA}_{\mathbb{Z}}}, \varrho_i) \models \bigvee_{j=0}^{\alpha*\sigma-1} (F_i(\alpha x_i/t-j, x_{i+1}, ..., x_s) \wedge (\alpha \mid t-j))$$

(für eine Primformel der Form (3) (also: $\alpha * x_i \leq t$))

Beweis. $\Rightarrow$: Es gibt eine Belegung β für x', sodass $F_i(\alpha x_i/\beta(x'), \varrho_i(x_{i+1}), ..., \varrho_i(x_s)) \wedge (\alpha \mid \beta(x'))$ wahr ist. Das heißt, dass das $\beta(x')$ das aus den Primformeln aus (5) und (6) gegebene System lösen kann. Der Fall $\beta(x') \to \infty$ kann ausgeschlossen werden, da die Äquivalenz nur angewandt wird, wenn es mindestens eine obere Schranke aus Primformeln aus (3) gibt. Nun sei t die kleinste obere Schranke für $\beta(x')$. Laut Satz 5 gibt es dann aber ein $x'' \in [t-(kgV(\alpha, \beta_1, \beta_2, ...)-1), t]$, welches auch das gegebene System aus (5) und (6) löst. Da es sich bei t um die kleinste obere Schranke handelt, und es laut Annahme eine Belegung für x' gibt, die auch die Teilbarkeiten erfüllt, muss x'' auch die nötigen Primformeln $t' \leq x''$ aus (4) erfüllen (t' sind die unteren Schranken aus Primformeln aus (4)). Nun wird noch $t-j \coloneqq x''$ substituiert, wobei $j \in [0, kgV(\alpha, \beta_1, \beta_2, ...)-1]$ ist. Dann kann eine Lösung j für $F_i(\alpha x/t-j, \varrho_i(x_{i+1}), ..., \varrho_i(x_s)) \wedge (\alpha \mid t-j)$ in $[0, kgV(\alpha, \beta_1, \beta_2, ...)-1] \subset [0, \alpha * kgV(\beta_1, \beta_2, ...)-1]$ gefunden werden. (vgl. Abbildung 7)
$\Leftarrow$: Gesucht ist eine Belegung β für x'. Die Voraussetzung liefert dafür $\beta(x') \coloneqq t-j$, sodass $F_i(\alpha x_i/\beta(x'), \varrho_i(x_{i+1}), ..., \varrho_i(x_s)) \wedge (\alpha \mid \beta(x'))$ wahr ist. □

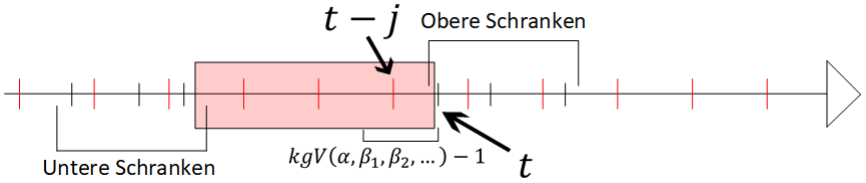

Abbildung 7: Endliche Menge an zu prüfenden Zahlen für Fall 2

Satz 8 kann zum Beispiel anhand eines negativ Beispiels demonstriert werden:

Beispiel 13: Ein negativ Beispiel für Satz 8

Es sei

$$\phi_0 = \exists x((1 \leq x) \wedge (x \leq 4) \wedge (5 \mid x) \vee (2 \mid x) \wedge (x \leq 3) \wedge (3 \leq x))$$

Offensichtlich gilt $\phi_0 \notin Th(\mathcal{S}_{\mathrm{PA}_{\leq}})$. Nun kann Satz 8 angewandt werden, der eine erfüllbarkeitsäquivalente Formel ϕ_1 ohne Variable x liefert.

$$\phi_1 = \bigvee_{j=0}^{kgV(2,5)-1} (1 \wedge 0 \wedge (5 \mid j) \vee (2 \mid j) \wedge 0 \wedge 1)$$
$$\vee \bigvee_{j=0}^{1*kgV(2,5)-1} ((1 \leq 4-j) \wedge (4-j \leq 4) \wedge (5 \mid 4-j) \vee (2 \mid 4-j) \wedge (4-j \leq 3) \wedge (3 \leq 4-j)) \wedge (1 \mid 4-j)$$

$$\vee \bigvee_{j=0}^{1*kgV(2,5)-1} ((1 \leq 3-j) \wedge (3-j \leq 4) \wedge (5 \mid 3-j) \vee (2 \mid 3-j) \wedge (3-j \leq 3) \wedge (3 \leq 3-j)) \wedge (1 \mid 3-j)$$

ϕ_1 ist wie gefordert ebenfalls nicht erfüllbar.

Mithilfe von Satz 8 ergibt sich der restliche Algorithmus in den Zeilen 7 bis 21. Die Quantoreliminierung aus Satz 8 erzeugt eine Formel, die zu $Qx_i\delta_i'$ erfüllbarkeitsäquivalent ist. Alle vorherigen Umformungen waren Äquivalenzumformungen. Daher sind alle ϕ_i $(0 \leq i \leq s)$ erfüllbarkeitsäquivalent. Wenn also $\mathcal{S}_{\mathrm{PA}_\mathbb{Z}} \models \phi_s$ gilt, dann ist auch die ursprüngliche Formel ϕ_0 in $Th(\mathcal{S}_{\mathrm{PA}_\mathbb{Z}})$ enthalten. Das ist die Entscheidbarkeit von $Th(\mathcal{S}_{\mathrm{PA}_\mathbb{Z}})$ mithilfe der Quantoreliminierung.

Beispiel 14: Die Quantoreliminierung am Beispiel der Transitivität der $\leq$-Relation

Gegeben sei die Formel $\phi_0 = \forall x_3 \forall x \forall x_1 \ (x_2 \leq x_3) \wedge (x_3 \leq x_1) \rightarrow (x_2 \leq x_1)$. Mithilfe der Quantoreliminierung (Satz 8) kann nun gezeigt werden, dass $\phi_0 \in Th(\mathcal{S}_{\mathrm{PA}_\mathbb{Z}})$ ist.
Zunächst entfernt Algorithmus 1 in Zeile 2 redundante Symbole und bringt ϕ_0 anschließend in Pränexnormalform (Zeile 3).

$$\begin{aligned}\phi_0 &\Leftrightarrow \forall x_3 \forall x_2 \forall x_1 \ (\neg((x_2 \leq x_3) \wedge (x_3 \leq x_1)) \vee (x_2 \leq x_1)) \\ &\Leftrightarrow \forall x_3 \forall x_2 \forall x_1 \ (\neg(x_2 \leq x_3) \vee \neg(x_3 \leq x_1) \vee (x_2 \leq x_1))\end{aligned}$$

Es ist

$$\delta_1 = (\neg(x_2 \leq x_3) \vee \neg(x_3 \leq x_1) \vee (x_2 \leq x_1))$$

Da $Q_1 = \forall$ ist, wird nun Algorithmus 2 auf $\exists x_1 \neg\delta_1$ angewandt.

$$\begin{aligned}\neg\delta_1 &\Leftrightarrow ((x_2 \leq x_3) \wedge (x_3 \leq x_1) \wedge \neg(x_2 \leq x_1)) \\ &\Leftrightarrow ((x_2 \leq x_3) \wedge (x_3 \leq x_1) \wedge (x_1 + 1 \leq x_2)) \\ &\Leftrightarrow ((x_2 \leq x_3) \wedge (x_3 \leq x_1) \wedge (x_1 \leq x_2 - 1))\end{aligned}$$

Dann ist

$$F_1(x_1, x_2, x_3) = ((x_2 \leq x_3) \wedge (x_3 \leq x_1) \wedge (x_1 \leq x_2 - 1))$$

Nun wird Satz 8 auf $\exists x_1 F_1(x_1, x_2, x_3)$ angewandt und es ergibt sich im Kontext von ϕ_1:

$$\phi_1 = \forall x_3 \forall x_2 \neg \left(\bigvee_{j=0}^{1-1} (x_2 \leq x_3 \wedge 1 \wedge 0) \vee \bigvee_{j=0}^{1*(1-1)} (x_2 \leq x_3 \wedge x_3 \leq x_2 - 1 - j \wedge x_2 - 1 - j \leq x_2 - 1 \wedge 1 \mid x_2 - 1 - j) \right)$$

Bei dem manuellen Durchführen des Algorithmus sollte an dieser Stelle gekürzt werden:

$$\Leftrightarrow \phi_1 = \forall x_3 \forall x_2 \neg(x_2 \leq x_3 \wedge x_3 \leq x_2 - 1)$$

Nun ist

$$\delta_2 = \neg(x_2 \leq x_3 \wedge x_3 \leq x_2 - 1)$$

Weiter ist $Q_2 = \forall$, sodass Algorithmus 2 auf $\exists x_2 \neg\delta_2$ angewandt wird. Durch das Eliminieren der Negation und dem Umformen nach Termen der Form $c_2 * x_2$ ergibt sich:

$$F_2(x_2, x_3) = x_2 \leq x_3 \wedge x_3 + 1 \leq x_2$$

Dann kann wieder Satz 8 auf $\exists x_2 F_2(x_2, x_3)$ angewendet werden:

$$\phi_2 = \forall x_3 \neg(\bigvee_{j=0}^{1-1}(0 \wedge 1) \vee \bigvee_{j=0}^{1*(1-1)}(x_3 - j \leq x_3) \wedge (x_3 + 1 \leq x_3 - j))$$

An dieser Stelle sollte gekürzt werden...

$$\begin{aligned}
&\Leftrightarrow \phi_2 = \forall x_3 \neg((x_3 \leq x_3) \wedge (x_3 + 1 \leq x_3)) \\
&\Leftrightarrow \phi_2 = \forall x_3 (\neg(x_3 \leq x_3) \vee \neg(x_3 + 1 \leq x_3)) \\
&\Leftrightarrow \phi_2 = \forall x_3 (0 \vee \neg(x_3 + 1 \leq x_3)) \\
&\Leftrightarrow \phi_2 = \forall x_3 (x_3 + 1 \leq x_3 + 1) \\
&\Leftrightarrow \phi_2 = 1
\end{aligned}$$

...sodass sich die letzte Anwendung der Quantoreliminierung gespart werden kann. Es zeigt sich, ohne weitere Auswertung, dass ϕ_2 wahr ist. Somit folgt, dass $\phi_0 \in Th(\mathcal{S}_{\mathrm{PA}_{\mathbb{Z}}})$ ist.

Algorithm 2 Entscheidungsalgorithmus für $Th(\mathcal{S}_{\mathrm{PA}_{\mathbb{Z}}})$

function QUANTORELIMINIERUNG(Teilformel $\exists x_i \delta_i$)
  Teilformel $\delta_i \leftarrow$ **eliminiereDasNegationszeichen**(δ_i)
  if δ_i hat die Form $\bigvee \delta$ **then return** ($\bigvee$ **quantorEliminierung**($\exists x_i \delta$))
  end if
  $\delta_i' \leftarrow$ **erzeugeEinheitlicheForm**(δ_i)
  ▷ ab hier kann auf Primformeln zugegriffen werden
  Funktion $F_i \leftarrow \delta_i'$
  Funktion $F_{\text{result}} \leftarrow 0$
  int[] betas $\leftarrow$ **extrahiereKonstantenVor|Oder**$\nmid(F_i)$ ▷ das sind die $\beta_1, \beta_2, ..., \beta_n$
  Funktion $F_\infty \leftarrow F_{\text{input}}$ wobei Primformeln aus (3) bzw. (4) durch 0 bzw. 1 ersetzt wird
  for int $j = 0; j <$ **kgV**(betas)$; j = j + 1$ **do**
    $F_{\text{result}} \leftarrow F_{\text{result}} \vee F_\infty(j)$
  end for
  Primformel[] pfs $\leftarrow$ Primformeln der Form (3) aus F_{input}
  for all p in pfs **do**
    for int $j = 0; j <$ **kgV**(betas) $* p.\alpha; j = j + 1$ **do**
      Funktion $F_{\text{hat } \alpha x_i} \leftarrow F_{\text{input}}$ falls vor einem x_i kein $p.\alpha$ steht multipliziere die Primformel mit $p.\alpha$
      Funktion $F_j \leftarrow F_{\text{hat } \alpha x_i}$ ersetze αx_i durch $p.t - j$
      $F_{\text{result}} \leftarrow F_{\text{result}} \vee F_j \wedge (p.\alpha \mid p.t - j)$
    end for
  end for
end function

5 Betrachtung der verschiedenen Strukturen der Presburger Arithmetik

Im vorausgehenden Kapitel konnte die Entscheidbarkeit für $Th(\mathcal{S}_{\mathrm{PA}_{\mathbb{Z}}})$ gezeigt werden. Doch wie verhält es sich mit den anderen beschränkteren Strukturen? Muss für jede Struktur der Presburger Arithmetik die Komplexität des Entscheidungsproblems zu dieser Theorie einzeln betrachtet werden?
Zur Beantwortung dieser Fragen ergibt sich der folgende Zusammenhang:

Satz 9: Der Zusammenhang zwischen den verschiedenen Theorien der Strukturen der Presburger Arithmetik
Es gilt:

$$Th(\mathcal{S}_{\mathrm{PA}}) \leq_m^P Th(\mathcal{S}_{\mathrm{PA}_{\leq}}) \leq_m^P Th(\mathcal{S}_{\mathrm{PA}_{\mathbb{Z}}})$$

Beweis. Die beiden Reduktionen sollen einzeln gezeigt werden:

1. $Th(\mathcal{S}_{\mathrm{PA}}) \leq_m^P Th(\mathcal{S}_{\mathrm{PA}_{\leq}})$
   Zunächst lässt sich feststellen, dass

   $$\Sigma_{\mathrm{PA}} \subset \Sigma_{\mathrm{PA}_{\leq}}$$

   gilt. Auch für die Sprachen, die die zugehörigen syntaktisch korrekten Ausdrücke definieren gilt, dass

   $$\mathcal{L}_{\mathrm{PA}} \subset \mathcal{L}_{\mathrm{PA}_{\leq}}$$

   ist. Da auch semantisch alle Zeichen mittels $\mathcal{S}_{\mathrm{PA}}$ und $\mathcal{S}_{\mathrm{PA}_{\leq}}$ gleich behandelt werden und es sich um das gleiche Universum $\mathbb{N}$ handelt, ist offensichtlich:

   $$Th(\mathcal{S}_{\mathrm{PA}}) \subset Th(\mathcal{S}_{\mathrm{PA}_{\leq}})$$

   Die Reduktion erfordert, dass es eine Funktion f gibt, sodass

   $$\phi \in Th(\mathcal{S}_{\mathrm{PA}}) \Leftrightarrow f(\phi) \in Th(\mathcal{S}_{\mathrm{PA}_{\leq}})$$

   gilt und f in Polynomialzeit berechnet werden kann. Aufgrund der obigen Teilmengenbeziehungen reicht dafür aber schon die Identitätsfunktion.

2. $Th(\mathcal{S}_{\mathrm{PA}_{\leq}}) \leq_m^P Th(\mathcal{S}_{\mathrm{PA}_{\mathbb{Z}}})$
   Auch hier gilt mit gleicher Argumentation:

   $$\Sigma_{\mathrm{PA}_{\leq}} \subset \Sigma_{\mathrm{PA}_{\mathbb{Z}}}$$

   und

   $$\mathcal{L}_{\mathrm{PA}_{\leq}} \subset \mathcal{L}_{\mathrm{PA}_{\mathbb{Z}}}$$

   Jedoch ist in diesem Fall nicht $Th(\mathcal{S}_{\mathrm{PA}_{\leq}}) \subset Th(\mathcal{S}_{\mathrm{PA}_{\mathbb{Z}}})$, da z.B. $\forall x\ 0 \leq x$ in beiden Sprachen existiert, aber nur von $\mathcal{S}_{\mathrm{PA}_{\leq}}$ erfüllt wird. Es sei

   $$\phi \coloneqq Q_1 x_1 Q_2 x_2 \cdots Q_s x_s\ F(x_1, x_2, ..., x_s)$$

ein Satz aus $\mathcal{L}_{\mathrm{PA}_\leq}$

Die für die Reduktion benötigte Funktion f muss also dafür sorgen, dass in $f(\phi)$ nur noch die Belegungen der Variablen relevant sind, die auch in $\mathbb{N}$ relevant wären. Für eine Variable x sind das die Belegungen mit $(0 \leq x)$. Die benötigte „Filterfunktion“ wird wie folgt definiert:

$$A(x) \coloneqq (0 \leq x)$$

Nun sei f mithilfe des Zeichens aus Definition 13 wie folgt definiert:

$$f(\phi) \coloneqq Q_1 x_1(A(x_1) \rightsquigarrow^1 Q_2 x_2(A(x_2) \rightsquigarrow^2 \cdots Q_s x_s(A(x_s) \rightsquigarrow^s F(x_1, x_2, ..., x_s)) \cdots))$$

Dann liefert Satz 6 mit $S = \mathcal{S}_{\mathrm{PA}_\mathbb{Z}}$, dem Universum $G = \mathbb{Z}$ und dementsprechend $S' = \mathcal{S}_{\mathrm{PA}_\leq}$ mit $G_A = \mathbb{N}$:

$$\mathcal{S}_{\mathrm{PA}_\leq} \models \phi \Leftrightarrow \mathcal{S}_{\mathrm{PA}_\mathbb{Z}} \models f(\phi)$$

Da es sich bei ϕ und $f(\phi)$ um Sätze handelt, ist das gleichbedeutend mit:

$$\phi \in Th(\mathcal{S}_{\mathrm{PA}_\leq}) \Leftrightarrow f(\phi) \in Th(\mathcal{S}_{\mathrm{PA}_\mathbb{Z}})$$

$f(\phi)$ kann in Polynomialzeit aus ϕ erstellt werden. Das ist die Polynomialzeitreduktion von $Th(\mathcal{S}_{\mathrm{PA}_\leq})$ auf $Th(\mathcal{S}_{\mathrm{PA}_\mathbb{Z}})$.

□

Die Polynomialzeithierarchie ist die komplexitätstheoretische Fortführung von $\mathcal{P}$ und $\mathcal{NP}$ (vgl. Kapitel 3.3). Es ist also wenig überraschend, dass aus $A \leq_m^P B$ und der Σ_m^P - schwere von A die Σ_m^P - schwere für B folgt. Auch die Inklusion überträgt sich wie im Fall von $\mathcal{P}$ und $\mathcal{NP}$. Im Fall von $A \leq_m^P B$ und $B \in \Sigma_m^P$ folgt, dass $A \in \Sigma_m^P$ gilt. [Wra76]
Aufgrund dieser Zusammenhänge reicht es bei den folgenden Betrachtungen, jeweils nur einen Fall zu analysieren, da sich das Ergebnis bei der richtigen Wahl auf die anderen Fälle überträgt.

6 Komplexität

Im vorausgegangenen Kapitel wurde erkennbar, dass die Sätze aus $Th(\mathcal{S}_{\mathrm{PA}_{\mathbb{Z}}})$ entscheidbar sind. Es stellt sich jedoch die Frage, in welcher Zeit für komplexe Sätze entschieden werden kann, ob diese wahr oder falsch sind. Dazu werde ich zunächst auf die Komplexität des allgemeinen Entscheidungsproblems $Th(\mathcal{S}_{\mathrm{PA}})$ eingehen. Die Ergebnisse lassen sich dann am Ende des Kapitels mittels Satz 9 auf die erweiterten Strukturen mit $Th(\mathcal{S}_{\mathrm{PA}_{\leq}})$ und $Th(\mathcal{S}_{\mathrm{PA}_{\mathbb{Z}}})$ verallgemeinern.

6.1 Die Komplexität des allgemeinen Entscheidungsproblems zu $Th(\mathcal{S}_{\mathbf{PA}})$

Der in Kapitel 4 erläuterte Entscheidungsalgorithmus scheint sehr aufwendig. In der Tat lässt sich zeigen, dass $Th(\mathcal{S}_{\mathrm{PA}})$ nicht schneller als in Zeit $O\left(2^{2^n}\right)$ entschieden werden kann ($n = |w|$ mit $w \in Th(\mathcal{S}_{\mathrm{PA}})$) [FR98]. Einfacher zu zeigen ist jedoch, dass $Th(\mathcal{S}_{\mathrm{PA}}) \notin \mathcal{P}$ gilt, was bedeutet, dass auch im Fall von $\mathcal{P} = \mathcal{NP}$, $Th(\mathcal{S}_{\mathrm{PA}})$ nicht effizient entscheidbar ist. Zu diesem Zweck werde ich Teile des Beweises aus [Pau78] vorstellen, die zu dem gewünschten Resultat führen.

Satz 10: $Th(\mathcal{S}_{\mathbf{PA}})$ ist nicht effizient entscheidbar
Falls es einen Polynomialzeitalgorithmus gibt, der bei Eingabe einer Turingmaschine M, die eine Sprache $L \in \mathrm{TIME}\left(2^{O(n)}\right)$ entscheidet, und einer Eingabe v, einen Satz $S_{Lv} \in \mathcal{L}_{\mathrm{PA}}$ konstruiert, sodass gilt:

$$M \text{ hält auf } v \text{ in einem akzeptierenden Zustand} \Leftrightarrow S_{Lv} \in Th(\mathcal{S}_{\mathrm{PA}})$$

Dann gilt $Th(\mathcal{S}_{\mathrm{PA}}) \notin \mathcal{P}$. ($n = |v|$)

Beweis. Zunächst gilt

$$P \overset{1.}{\subseteq} \mathrm{TIME}\left(2^n\right) \overset{2.}{\subsetneq} \mathrm{TIME}\left(2^{2n}\right) \overset{3.}{\subseteq} \mathrm{TIME}\left(2^{O(n)}\right)$$

Die Teilmengenbeziehungen können einzeln gezeigt werden:

1. $P \subseteq \mathrm{TIME}\left(2^n\right)$
   Es ist $P = \mathrm{TIME}\left(n^{O(1)}\right) = \bigcup_{k\in\mathbb{N}} \mathrm{TIME}\left(n^k\right)$. Nun kann bewiesen werden, dass $\mathrm{TIME}\left(n^k\right)$ für alle festen $k \in \mathbb{N}$ in $\mathrm{TIME}\left(2^n\right)$ ist. Zu diesem Zweck wird $n^k \in O\left(2^n\right)$ gezeigt:

$$\begin{aligned}
&\lim_{n\to\infty} \frac{n^k}{2^n} \leq \infty \\
\Leftrightarrow &\lim_{n\to\infty} \frac{k * n^{k-1}}{ln(2) * 2^n} < \infty && \text{| Anwendung des Satz von L'Hôpital} \\
&\vdots && \vdots \\
\Leftrightarrow &\lim_{n\to\infty} \frac{k!}{ln(2)^k * 2^n} < \infty && \text{| Anwendung des Satz von L'Hôpital} \\
\Leftrightarrow &\frac{k!}{ln(2)^k} * \lim_{n\to\infty} \frac{1}{2^n} < \infty
\end{aligned}$$

$$\Leftrightarrow \frac{k!}{ln(2)^k} * 0 < \infty$$
$$\Leftrightarrow 0 < \infty$$

Da für jedes festes $k \in \mathbb{N}$ gilt, dass $\text{TIME}\left(n^k\right) \subseteq \text{TIME}\left(2^n\right)$ ist, gilt auch für die Vereinigung $\bigcup_{k\in\mathbb{N}} \text{TIME}\left(n^k\right) \subseteq \text{TIME}\left(2^n\right)$.

2. $\text{TIME}\left(2^n\right) \subsetneq \text{TIME}\left(2^{2n}\right)$
   Mit dem Zeithierarchiesatz folgt aus $2^n \in o\left(\frac{2^{2n}}{ld(2^{2n})}\right)$, dass $\text{TIME}\left(2^n\right) \subsetneq \text{TIME}\left(2^{2n}\right)$ gilt. Benötigt wird demnach:

$$\lim_{n\to\infty} \frac{2^n}{\frac{2^{2n}}{ld(2^{2n})}} = 0$$
$$\Leftrightarrow \lim_{n\to\infty} \frac{2^n * ld\left(2^{2n}\right)}{2^{2n}} = 0$$
$$\Leftrightarrow \lim_{n\to\infty} \frac{2^n * 2n * ld(2)}{2^{2n}} = 0$$
$$\Leftrightarrow \lim_{n\to\infty} \frac{2^n * 2n * ld(2)}{(2^n)^2} = 0$$
$$\Leftrightarrow \lim_{n\to\infty} \frac{2n * ld(2)}{2^n} = 0$$
$$\Leftrightarrow \lim_{n\to\infty} \frac{2 * ld(2)}{ld(2) * 2^n} = 0 \qquad \text{| Anwendung des Satz von L'Hôpital}$$
$$\Leftrightarrow \lim_{n\to\infty} \frac{1}{2^{n-1}} = 0$$
$$\Leftrightarrow 0 = 0$$

3. $\text{TIME}\left(2^{2n}\right) \subseteq \text{TIME}\left(2^{O(n)}\right)$
   Das stimmt, da $\text{TIME}\left(2^{O(n)}\right) = \text{TIME}\left(2^{c*n}\right)$ für alle $c > 0$ gilt. Daher kann $c = 2$ gewählt werden, um das gewünschte Ergebnis zu erhalten: $\text{TIME}\left(2^{2*n}\right) \subseteq \text{TIME}\left(2^{O(n)}\right)$.

Aufgrund der daraus resultierenden echten Teilmengenbeziehung zwischen $\mathcal{P}$ und $\text{TIME}\left(2^{O(n)}\right)$ gibt es also Sprachen $L \in \text{TIME}\left(2^{O(n)}\right)$, die nicht in $\mathcal{P}$ liegen. Nun kann für alle $L \in \text{TIME}\left(2^{O(n)}\right)$ gezeigt werden, dass sich L in Polynomialzeit auf $Th(\mathcal{S}_{\text{PA}})$ reduzieren lässt. Dafür wird eine Instanz $v \in \Sigma^*$ in eine Instanz $S_{Lv} \in \mathcal{L}_{\text{PA}}$ $(L \subset \Sigma^*)$ übersetzt. Das geht mit dem vorausgesetzten Polynomialzeitalgorithmus und einer 2^{c*n}-zeitbeschränkten Turingmaschine M_L. Diese muss es geben, da sonst L nicht in $\text{TIME}\left(2^{O(n)}\right)$ wäre. Dann gilt:

$v \in L \Leftrightarrow$ Die Turingmaschine M_L hält auf v in einem akzeptierenden Zustand
$\Leftrightarrow$ Der vorrausgesetzte Polynomialzeitalgorithmus liefert S_{Lv} bei Eingabe $< M_L, v >$
$\Leftrightarrow S_{Lv} \in Th(\mathcal{S}_{\text{PA}})$

Das ist die Reduktion des Entscheidungsproblems „$v \in L$“ auf das Problem $S_{Lv} \in Th(\mathcal{S}_{\text{PA}})$ in Polynomialzeit.

Angenommen $Th(\mathcal{S}_{\mathrm{PA}})$ wäre in $\mathcal{P}$, dann könnten alle Sprachen $L \in \mathrm{TIME}\left(2^{O(n)}\right)$ mit obigem Verfahren in Polynomialzeit entschieden werden. Das ist aber ein Widerspruch dazu, dass es Sprachen $L \in \mathrm{TIME}\left(2^{O(n)}\right)$ gibt, die nicht in Polynomialzeit gelöst werden können.
Demzufolge gilt $Th(\mathcal{S}_{\mathrm{PA}}) \notin \mathcal{P}$. □

Damit aus Satz 10 $Th(\mathcal{S}_{\mathrm{PA}}) \notin \mathcal{P}$ folgt, wird der erwähnte Algorithmus benötigt, der in polynomieller Zeit läuft und eine Turingmaschine M und eine Eingabe v in einen Satz in der Presburger Arithmetik übersetzt, der nur dann wahr ist, wenn M auf v mit akzeptierenden Zustand hält. Dazu sei weiterhin $n = |v|$.
Es sei $M = (Z, \Sigma, \Gamma, \delta, z_0, \square, A, V)$ eine Turingmaschine, die eine Sprache $L \in \mathrm{TIME}\left(2^{O(n)}\right)$ entscheidet. Solch eine Turingmaschine benötigt maximal $T \coloneqq 2^{c*n}$ Konfigurationsübergänge um zu entscheiden, ob $v \in L$ ist und damit in einem akzeptierenden Zustand aus A hält (für ein $c > 0$). Falls also tatsächlich $v \in L$ ist, gibt es eine Konfigurationsfolge, die mit $k_0 = \square...\square z_o v \square...\square$ beginnt, und mit einer Endkonfiguration $k_T = uz_A y$ endet $(u, y \in \Gamma, z_A \in A)$. In der Konfigurationsfolge $k_0, k_1, ..., k_{T-1}, k_T$ dürfen nur Konfigurationen k_i, k_{i+1} hintereinander stehen, für die es einen entsprechenden Übergang aus δ gibt, sodass $k_i \vdash k_{i+1}$ für alle $0 \leq i \leq T-1$ gilt. Ein Satz S_{Lv} in der Presburger Arithmetik muss auf Grund dieser Zusammenhänge vereinfacht lauten:

$$\exists w((1) \wedge (2) \wedge (3)) \tag{9}$$

wobei $w \coloneqq k_0, k_1, ..., k_T$ und

(1) $\coloneqq k_0$ hat die Form $\square...\square z_o v \square...\square$

(2) $\coloneqq k_i \vdash k_{i+1}$ für alle $0 \leq i \leq T-1$

(3) $\coloneqq k_T$ hat die Form $uz_A y$ wobei $(u, y \in \Gamma, z_A \in A)$

Damit w eine Konfigurationsfolge kodieren kann, wird mindestens eine injektive Kodierung ψ zwischen der Sprache mit dem Alphabet $\mathbb{K} \coloneqq Z \cup \Gamma \cup \{,\}$ und den natürlichen Zahlen $\mathbb{N}$ benötigt. Wodurch im Satz S_{Lv} eine Zahl $\psi(w)$ gesucht werden kann, die die benötigte Konfigurationsfolge kodiert.
Ohne Beschränkung der Allgemeinheit lässt sich annehmen, dass $\#\mathbb{K} = p-1$ und p eine Primzahl ist, da es unendlich viele Primzahlen gibt, und ein $\mathbb{K}$ problemlos mit nicht verwendeten Zeichen bis $\#\mathbb{K} = p-1$, aufgefüllt werden kann.
Dann wird jedem Zeichen aus $\mathbb{K}$ eine Zahl aus $\{1, ..., p-1\}$ zugeordnet, sodass eine Zeichenreihe zu einer Zahl im Stellenwertsystem mit der Basis p wird. Eine Zahl im Stellenwertsystem zu Basis p kann problemlos zu einer Zahl im Stellenwertsystem einer anderen Basis konvertiert werden. Im Fall der Presburger Arithmetik ist das im Normalfall Basis 10. Es folgt also die Definition:

> **Definition 18: Konvertierungen im Stellenwertsystem**
> Es sei $S_{B_1 \to B_2}$ die Funktion die eine Zahl im Stellenwertsystem zur Basis B_1 in eine Zahl zur Basis B_2 konvertiert.

$S_{B_2 \to B_1}$ ist das eindeutig definierte Inverse zu $S_{B_1 \to B_2}$ und die benötigte Kodierungsfunktion $S_{p \to 10}$ ist bijektiv. Für die gesamte Kodierung lässt definieren:

Definition 19: Die Kodierungsfunktion für Wörter
Es sei $\psi(w)$ mit $w \in \mathbb{K}^+$ die Funktion, die zunächst jedem Zeichen $a \in \mathbb{K}$ eine Zahl aus $\{1, ..., p-1\}$ zuordnet und diese in w einzeln ersetzt, sodass sich $w' \in \{1, ..., p-1\}^+$ ergibt. Die Funktion gibt dann $S_{p\to 10}(w') \in \{1, 2, ..., 9\}^+$ zurück. Außerdem sei $\psi(\epsilon) = 0$.

Die Kodierung ψ ist injektiv jedoch nicht surjektiv, da es zu Zahlen mit Basis p, die eine 0 und andere Zahlen enthalten, kein Wort $w \in \mathbb{K}^+$ gibt. Ansonsten ergibt sich eine eindeutige Zuordnung zwischen Worten $w \in \mathbb{K}$ und den natürlichen Zahlen $\mathbb{N}$.

Beispiel 15: Kodierung eines Beispielalphabets mit natürlichen Zahlen

Es sei $\Sigma_{\text{Beispiel}} = \{a, b, c, d, e\}$. Dann wird Σ_{Beispiel} um ein Symbol o^1 erweitert, damit $\#\Sigma_{\text{Beispiel}} = p - 1$ für die Primzahl $p = 7$ gilt. Die Zuordnung kann wie folgt gewählt werden:

$$\begin{aligned} a &\to 1 & b &\to 2 \\ c &\to 3 & d &\to 4 \\ e &\to 5 & o^1 &\to 6 \end{aligned}$$

Mithilfe der Zuordnung ergibt sich zu einem $w_1 = acdc$ eine Zahl $w_1' = 1343$ zur Basis 7, sodass folgendermaßen kodiert werden kann:

$$S_{7\to 10}(w_1') = 1 * 7^3 + 3 * 7^2 + 4 * 7^1 + 3 * 7^0 = 343 + 147 + 28 + 3 = 521$$

Im umgekehrten Fall kann dekodiert werden. Zu einem Wort w_2 mit $S_{7\to 10}(w_2') = 1337$ ergibt sich

$$S_{10\to 7}(1337) = 3 * 7^3 + 6 * 7^2 + 2 * 7 + 1 * 7^0 = 3621$$

sodass dekodiert das Wort $w_2 = co^1ba$ entsteht.

Letztendlich ist das Problem an dieser Stelle nur noch, dass $S_{B_1\to B_2}$ mit der Potenzierung und der Multiplikation arbeitet, die es im Allgemeinen in der Presburger Arithmetik nicht gibt. Es kann jedoch die Skalarmultiplikation aus Definition 4 noch etwas erweitert werden. Die Multiplikation $a * b = c$ ist gleichbedeutend mit $\bigvee_{i=1}^{\max(a)}(a = i \wedge i * b = c)$, wenn eine obere Grenze $\max(a)$ für a bekannt ist. $i * b$ kann mit Presburger Arithmetik ausgedrückt werden, da i fest gewählt ist, und somit die Skalarmultiplikation aus Definition 4 angewandt wird. Diese Idee und das Auftrennen einer Multiplikation in mehrere Teilrechnungen sorgt dafür, dass die Multiplikation $a * b = c$ für ein $p_n \geq 2^{2^n}$ mit einem Prädikat $m_n(a, b, c)$ mit der Bedeutung $a \leq p_n \wedge a * b = c$ und in Platz $O(|n|)$ ausgedrückt werden kann (weitere Erläuterung diesbezüglich in [Pau78] und im Kapitel 8.2 zu finden im Anhang).
Es zeigt sich, dass das $\psi(w) \leq p^{|w|}$ und $|w| = (T+1) * (\max_{i=0}^{T}(|k_i|) + 1)$ ist, da die Turingmaschine M bei Eingabe v nach maximal $T = 2^{c*n}$ Schritten für ein $c > 0$ zum Ergebnis $v \in L$ oder $v \notin L$ kommt. Dabei durchläuft sie $T + 1$ Konfigurationen der Länge $\max_{i=0}^{T}(|k_i|)$ und nach jeder Konfiguration folgt in w ein Komma. Dann lässt sich abschätzen:

$$\begin{aligned} \psi(w) &\leq p^{|w|} \\ &= 2^{ld(p)*|w|} \end{aligned}$$

$$
\begin{aligned}
&\leq 2^{ld(p)*(T+1)*(\max_{i=0}^{T}(|k_i|)+1)} \\
&\leq 2^{ld(p)*(2^{c*n}+1)*(2^{c*n}+(n+1))} \\
&\leq 2^{2^{c*n}*(2^{c*n+1})*(2^{c*n+1})} \\
&\leq 2^{(2^{c*n+1})^3} \\
&= 2^{2^{3*c*n+3}}
\end{aligned}
$$

Die Kodierung der in S_{Lv} gesuchten Zahl $\psi(w)$ ist also kleiner als $2^{2^{3cn+3}}$. Es ist $3cn+3 \in O(n)$ und daher gibt es ein $p_{3cn+3} \geq 2^{2^{3cn+3}} \geq \psi(w)$ und ein dazu passendes Prädikat m_{3cn+3} für die Multiplikation, welches nur Platz $O(n)$ benötigt und ebenfalls in Polynomialzeit gefunden werden kann. Das führt dazu, dass auf der Suche nach dem $\psi(w)$ die Multiplikation verwendet werden kann und dementsprechend Formeln für (1), (2) und (3) in Polynomialzeit aufgestellt werden können. Insgesamt kann also (9) in Polynomialzeit erstellt werden (explizite Formeln in [Pau78]).
Der in Satz 10 erwähnte Polynomialzeitalgorithmus existiert demnach, sodass folgt:

Satz 11: Sätze der Presburger Arithmetik sind nicht effizient entscheidbar
Es gilt:

1. $Th(\mathcal{S}_{\mathrm{PA}}) \notin \mathcal{P}$
2. $Th(\mathcal{S}_{\mathrm{PA}_{\leq}}) \notin \mathcal{P}$
3. $Th(\mathcal{S}_{\mathrm{PA}_{\mathbb{Z}}}) \notin \mathcal{P}$

Beweis. Jedes Element kann einzeln aus den bisherigen Ergebnissen gefolgert werden:

1. $Th(\mathcal{S}_{\mathrm{PA}}) \notin \mathcal{P}$
   Folgt aus dem in [Pau78] vollständig gegebenen Polynomialzeitalgorithmus (bzw. aus den oben gegebenen Ideen aus [Pau78]) und Satz 10.

2. $Th(\mathcal{S}_{\mathrm{PA}_{\leq}}) \notin \mathcal{P}$
   Angenommen es ist $Th(\mathcal{S}_{\mathrm{PA}_{\leq}}) \in \mathcal{P}$. Außerdem besagt Satz 9, dass $Th(\mathcal{S}_{\mathrm{PA}}) \leq_m^P Th(\mathcal{S}_{\mathrm{PA}_{\leq}})$ gilt. In Kombination heißt das aber, dass es einen Polynomialzeitalgorithmus f gibt, sodass
   $$\phi \in Th(\mathcal{S}_{\mathrm{PA}}) \Leftrightarrow f(\phi) \in Th(\mathcal{S}_{\mathrm{PA}_{\leq}})$$
   gilt. Damit kann dann aber einen Polynomialzeitalgorithmus erstellt werden, der $Th(\mathcal{S}_{\mathrm{PA}})$ entscheidet, indem er auf seine Eingabe w f anwendet und dann $f(w) \in Th(\mathcal{S}_{\mathrm{PA}_{\leq}})$ in Polynomialzeit entscheidet (Das geht laut Annahme!). Das ist aber ein Widerspruch dazu, dass $Th(\mathcal{S}_{\mathrm{PA}}) \notin \mathcal{P}$ aus 1. gilt.

3. $Th(\mathcal{S}_{\mathrm{PA}_{\mathbb{Z}}}) \notin \mathcal{P}$
   Analoge Argumentation zu 2. nur wird $Th(\mathcal{S}_{\mathrm{PA}_{\leq}}) \leq_m^P Th(\mathcal{S}_{\mathrm{PA}_{\mathbb{Z}}})$ verwendet (ebenfalls aus Satz 9). Falls $Th(\mathcal{S}_{\mathrm{PA}_{\mathbb{Z}}}) \in \mathcal{P}$ ist, lässt sich wieder ein Polynomialzeitalgorithmus finden, der $Th(\mathcal{S}_{\mathrm{PA}_{\leq}})$ entscheidet. Das ist ein Widerspruch zu 2.

□

6.2 Die Komplexität bestimmter Teilklassen der Presburger Arithmetik

In der Mathematik und deren Anwendung kommt es jedoch selten vor, dass Sätze beliebig viele Quantoren besitzen. Daher liegt die Idee nah, das Entscheidungsproblem auf bestimmte Teilklassen der Presburger Arithmetik einzuschränken. In den folgenden Definitionen sei

$$X \in \{\mathrm{PA}, \mathrm{PA}_{\leq}, \mathrm{PA}_{\mathbb{Z}}\}$$

Jeder Satz aus $\mathcal{L}_{\mathrm{X}}$ lässt sich in Polynomialzeit zu einem äquivalenten Satz in Pränexnormalform $Q_1x_1Q_2x_2...Q_sx_sF(x_1, x_2, ..., x_s)$ wiederum in $\mathcal{L}_{\mathrm{X}}$ umformen. $Q_1, Q_2, ..., Q_s$ wird das Quantorenprefix des Satzes genannt und soll im Folgenden für die Definition von Teilklassen von $Th(\mathcal{S}_{\mathrm{X}})$ genutzt werden.

Definition 20: Feste Teilklassen von $Th(\mathcal{S}_{\mathbf{X}})$
Für ein Quantorenprefix $Q_1, Q_2, ..., Q_s$ gibt es Teilklassen $[Q_1, Q_2, ..., Q_s] \cap Th(\mathcal{S}_{\mathrm{X}})$. Das sind die Sätze aus $Th(\mathcal{S}_{\mathrm{X}})$ die in Pränexnormalform die Form $Q_1x_1Q_2x_2...Q_sx_sF(x_1, x_2, ..., x_s)$ haben und keine weiteren Quantoren in F besitzen.

Diese Teilklassen besitzen nun bestimmte Eigenschaften, die sich als nützlich erweisen, um die Komplexität solch einer Teilklasse zu bestimmen.

Definition 21: Dimension
Die Dimension von $[Q_1, Q_2, ..., Q_s] \cap Th(\mathcal{S}_{\mathrm{X}})$ ist die Länge s des Quantorenprefix $Q_1, Q_2, ..., Q_s$ $(s \in \mathbb{N})$.

Definition 22: Quantorenwechsel
Das Quantorenprefix $Q_1, Q_2, ..., Q_s$ der Teilklasse $[Q_1, Q_2, ..., Q_s] \cap Th(\mathcal{S}_{\mathrm{X}})$ besitzt m Quantorenwechsel, wenn es genau m Mal den Fall gibt, dass Q_i sich von Q_{i+1} unterscheidet. $(1 \leq i \leq s-1)$

Definition 23: Quantorenblöcke
Das Quantorenprefix $Q_1, Q_2, ..., Q_s$ der Teilklasse $[Q_1, Q_2, ..., Q_s] \cap Th(\mathcal{S}_{\mathrm{X}})$ besitzt m Quantorenblöcke, wenn es genau $m-1$ Quantorenwechsel gibt.

Dann kann eine Gruppe von Teilklassen zusammengefasst werden:

Definition 24: Variable Teilklassen von $Th(\mathcal{S}_{\mathbf{PA}})$
Es sei $\Upsilon^s_m(X)$ die Menge der Teilklassen $[Q_1, Q_2, ..., Q_s] \cap Th(\mathcal{S}_{\mathrm{X}})$, die ein Quantorenprefix mit m Quantorenblöcke bzw. $m-1$ Quantorenwechsel (und Dimension s) haben.

Satz 9 lässt sich mit analoger Argumentation wie im Beweis zu Satz 9 auf diese Teilklassen erweitern:

Satz 12: Der Zusammenhang zwischen den festen und variablen Teilklassen
Es gilt:

$$[Q_1, Q_2, ..., Q_s] \cap Th(\mathcal{S}_{\mathrm{PA}}) \leq_m^P [Q_1, Q_2, ..., Q_s] \cap Th(\mathcal{S}_{\mathrm{PA}_\leq}) \leq_m^P [Q_1, Q_2, ..., Q_s] \cap Th(\mathcal{S}_{\mathrm{PA}_\mathbb{Z}})$$

Und auch:

$$\Upsilon_m^s(\mathrm{PA}) \leq_m^P \Upsilon_m^s(\mathrm{PA}_\leq) \leq_m^P \Upsilon_m^s(\mathrm{PA}_\mathbb{Z})$$

6.2.1 Obere Schranken für die zu testenden Belegungen der gebundenen Variablen und Analyse des Entscheidungsalgorithmus

Um eine Einordnung der oben genannten Teilklassen in die Polynomialzeithierarchie zu bekommen, sollen im Folgenden die beiden Algorithmen 1 und 2 analysiert werden. Dabei wird die Eingabe für Algorithmus 1 auf Formeln $\phi \in \mathcal{L}_{\mathrm{PA}_\mathbb{Z}}$ beschränkt, in denen kein $\mid$ oder $\nmid$ vorkommt. Im Satz 8 gibt es die $\bigvee$ bzw. im Algorithmus 2 in Zeile 11 und 16 die **for**-Schleifen über endlich viele Zahlen, die getestet werden mussten, um einen Quantor zu eliminieren.

Beispiel 16: Einfaches Beispiel für eine endliche Menge von zu prüfenden Zahlen

Es sei $\phi_1 = \exists x\ 5x \leq 50 \wedge (4 \mid x)$. Dann führt Satz 8 zu einer erfüllbarkeitsäquivalenten Formel:

$$\phi_2 = 0 \wedge (4 \mid x) \vee \bigvee_{j=0}^{5*kgV(4)-1} (50 - j \leq 50) \wedge (5 * 4 \mid 50 - j) \wedge (5 \mid 50 - j)$$

Offensichtlich war ϕ_1 erfüllbar, deshalb ist auch ϕ_2 erfüllbar. Um das festzustellen musste nur die Menge $[0, 1, ..., 5 * kgV(4) - 1]$ für mögliche j überprüft werden. Das waren gerade mal 20 Zahlen.

Die Idee ist es, nun zu zeigen, dass

$$\begin{gathered} \mathcal{S}_{\mathrm{PA}_\mathbb{Z}} \models Q_1 x_1 \cdots Q_s x_s F_1(x_1, ..., x_s) \\ \Leftrightarrow \\ ([-w_s, w_s]; +; \leq, \mid, \nmid; 0, 1) \models Q_1 x_1 \cdots Q_s x_s F_1(x_1, ..., x_s) \end{gathered} \tag{10}$$

für ein $w_s \in \mathbb{N}$ gilt. Die Beschränkung des Universum $\mathbb{Z}$ auf $[-w_s, w_s]$ sorgt dafür, dass für gebundene Variable nur noch endlich viele Belegungen getestet werden müssen. Mit Satz 6 lässt sich dieser Zusammenhang auch in einer äquivalenten Formel in $\mathcal{S}_{\mathrm{PA}_\mathbb{Z}}$ ausdrücken. Dafür wird die folgende Filterfunktion gewählt:

$$A_{w_s}(x) = (x \leq w_s) \wedge (-w_s \leq x)$$

Es ergibt sich eine zu (10) äquivalente Aussage:

$$\begin{gathered} \mathcal{S}_{\mathrm{PA}_\mathbb{Z}} \models Q_1 x_1 \cdots Q_s x_s F_1(x_1, ..., x_s) \\ \Leftrightarrow \end{gathered}$$

$$\mathcal{S}_{\mathrm{PA}_\mathbb{Z}} \models Q_1 x_1 \ (A_{w_s}(x_1) \rightsquigarrow^1 \cdots (Q_s x_s \ (A_{w_s}(x_s) \rightsquigarrow^s F_1(x_1, ..., x_s)$$

Mithilfe der Filterfunktion lässt sich jede gebundene Variable einzeln beschränken. Zu diesem Zweck lässt sich die folgende Abkürzung definieren:

Definition 25: Quantifizierung betragsmäßig beschränkter Variablen
Für eine Formel ϕ sei

$$(Q_k|x_k| \leq w) \ \phi \coloneqq Q_k x_k \ (A_w(x_k) \rightsquigarrow^k \phi)$$

eine Abkürzung.

Dann lässt sich

$$\mathcal{S}_{\mathrm{PA}_\mathbb{Z}} \models Q_1 x_1 \cdots Q_s x_s F_1(x_1, ..., x_s) \Leftrightarrow \mathcal{S}_{\mathrm{PA}_\mathbb{Z}} \models (Q_1|x_1| \leq w_s) \cdots (Q_s|x_s| \leq w_s) F_1(x_1, ..., x_s)$$

mit Schranke w_s als Ziel der folgenden Analyse formulieren.
Dafür wird die Teilformel δ_{r+1} betrachtet, die mit der Quantoreliminierung aus einer erfüllbarkeitsäquivalenten Formel $\exists x_r F_r(x_r, \varrho_r(x_{r+1}), ..., \varrho_r(x_s))$ entstanden ist. Um eine obere Schranke für den Betrag der Belegung des x_r zu bestimmen, wird zunächst davon ausgegangen, dass auch die Belegungen der Variablen x_i $(r+1 \leq i \leq s)$, die noch nicht eliminiert wurden, beschränkt sind:

$$|\varrho_r(x_i)| \leq w \ (w \in \mathbb{N})$$

Im Fall der Erfüllbarkeit von δ_{r+1} muss entweder $F_r(\alpha x_r/t - j, \varrho_r(x_{r+1}), ..., \varrho_r(x_s)) \wedge (\alpha \mid t - j)$ für ein $t - j$ oder $F_\infty(j, \varrho_r(x_{r+1}), ..., \varrho_r(x_s))$ für ein j wahr sein. Im ersten Fall kann Satz 7 angewandt werden, der besagt, dass

$$F_r(\alpha x_r/t - j, x_{r+1}, ..., x_s) \wedge (\alpha \mid t - j) \Leftrightarrow F_r(x_r, x_{r+1}, ..., x_s)$$

gilt, wenn $x_r \coloneqq \frac{t-j}{\alpha}$ gesetzt wird.

Beispiel 17: Fortsetzung von Beispiel 16

Um zu sehen, dass ϕ_2 wahr ist, wird das Disjunkt mit $j = 10$ betrachtet. In diesem Fall ist $(40 \leq 50) \wedge (20 \mid 40) \wedge (5 \mid 40)$ wahr. In ϕ_1 gab es nur eine Primformel aus (3) mit $\alpha = 5$ und $t = 50$, sodass nur eine endliche Menge getestet werden musste. Aufgrund von Satz 7 ist nun $\frac{50-10}{5} = 8$ eine Lösung für $5x \leq 50 \wedge (4 \mid x)$.

Im zweiten Fall ist es hingegen so, dass es keinen für die Lösung relevante Primformel aus (3) gibt, die x_r nach oben beschränkt. Jedoch kann es trotzdem eine kleinste untere Schranke t' aus einer Primformel aus (4) geben. Da es eine Lösung j für $F_\infty(j, \varrho_r(x_{r+1}), ..., \varrho_r(x_s))$ gibt, welche das nötige System aus Teilbarkeiten löst, kann nun laut Satz 5 eine Lösung $t' + j'$ mit $j' \in [0, kgV(\alpha', \beta_1, \beta_2, ...) - 1] \subset [0, \alpha' * kgV(\beta_1, \beta_2, ...) - 1]$ gefunden werden (vgl. Abbildung 8).

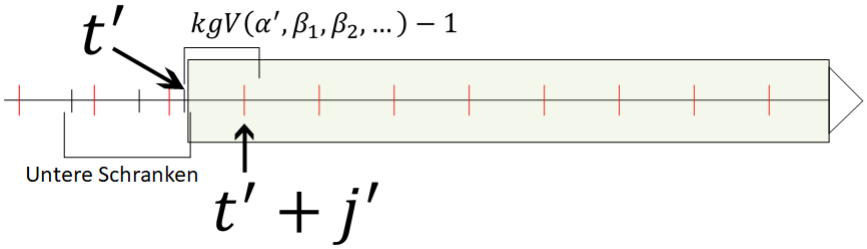

Abbildung 8: Der Fall aus Abbildung 6: Es wird eine Belegung für x_r gesucht, sodass auch die Primformeln aus (4) erfüllt sind. Das wird sonst nicht benötigt, da F_∞ im Fall der Erfüllbarkeit schon die Erfüllbarkeitsäquivalenz liefert (siehe Herleitung zu Satz 8)

Nun kann wieder Satz 7 angewandt werden, der liefert, dass $x_r := \frac{t'+j'}{\alpha'}$ eine Lösung für $F_r(x_r, \varrho_r(x_{r+1}), ..., \varrho_r(x_s))$ ist.

Beispiel 18: Erweiterung von Beispiel 16

Wenn es in Formel ϕ_1 $5x \geq 50$ gewesen wäre, dann wäre

$$\phi_2 = 1 \wedge (4 \mid x)$$

offensichtlich ebenfalls wahr gewesen. Auch in diesem Fall lässt sich mit Satz 7 eine Lösung für das geänderte ϕ_1 konstruieren. In diesem Fall ist $t' = 50$ die kleinste untere Schranke für $5x$ (also $\alpha' = 5$). Es ist $j' \in [0, kgV(5, 4)]$. Beispielsweise liefert $j' = 10$ mit $x = \frac{50+10}{5} = 12$ eine Lösung für die geänderte Formel ϕ_1.

Insgesamt lässt sich aus einer Lösung für δ_{r+1}, bestehend aus einer Primformel $p \in (3)$ bzw. $p \in (4)$ und einem j auf eine Lösung für F_r schließen. (Herleitung zu diesem Resultat aus [RL78]) Das Finden einer obere Schranke für ein $t + j$

$$|x| = \left|\frac{t-j}{\alpha}\right| \leq t + j \tag{11}$$

bzw. für

$$|x| = \left|\frac{t'+j'}{\alpha'}\right| \leq t' + j' \tag{12}$$

genügt demnach, um auf eine obere Schranke für $|x|$ zu schließen. Ein Term $t = \sum_{j=1}^{k} c_i * x_i + c_{k+1}$ und $j \in [0, \alpha * kgV(\beta_1, \beta_2, ...) - 1]$ besteht aus

1. den Koeffizienten $\alpha_1, ..., \alpha_n$ bzw. $c_1, ..., c_n$
2. dem $\sigma := kgV(\beta_1, ..., \beta_n)$
3. der Summe der additiven Konstanten c_{s+1} und j

Für diese Zahlen sollen im Folgenden obere Schranken bestimmt werden, um dann auf eine obere Schranke für die zu testenden Belegungen für x_i $(1 \leq i \leq s)$ schließen zu können. In der ganzen Analyse sei $n := |\phi_0|$ und die Belegung der Variablen im Binärsystem notiert.

Definition 26: Die obere Schranke K_r für Koeffizienten
K_r ist eine obere Schranke für die Größe der Koeffizienten α, die nach der Eliminierung von r Quantoren vor einer Variable stehen. (Nach r-maligem Ausführen von Algorithmus 2 ist bei dem $(r+1)$. Mal $\alpha \leq K_r$)

Satz 13: Obere Schranke für K_r
Es ist $K_r \leq 2^{2^r(n+2r)}$ für alle $r \geq 0$. (Satz und Beweis folgen der Argumentation aus [Grä88] jedoch mit größer gewählter oberer Schranke)

Beweis. Im Algorithmus 2 in Zeile 5 wird δ_i in eine einheitliche Form gebracht. Zu diesem Zweck werden diverse $c_j * x_j$ addiert und subtrahiert, um das $c_i * x_i$ allein auf eine Seite der $\leq$ zu bekommen ($i \leq j$ und $j \in [1, s]$). Dabei werden maximal zwei Koeffizienten vor demselben x_i addiert. Des Weiteren wird darauffolgend in Zeile 17, falls vor einem x_i kein α steht, diese Primformel mit α multipliziert. α war jedoch auch nur ein Koeffizient aus einer Primformel der Form (3).
Sodass insgesamt $K_r \leq (K_{r-1} + K_{r-1})^2 = 4 * K_{r-1}^2$ gilt. Nun kann die Behauptung per vollständiger Induktion bewiesen werden:
IA) Für $r = 0$ ist $K_0 \leq 2^n \left(= 2^{2^0(n+2*0)}\right)$, da eine Formel ϕ_0 aus mehr als einem Koeffizient besteht, und ein Koeffizient der Länge $n-1$ im Dezimalsystem kleiner als 2^n ist.
IS) Zu zeigen: Angenommen $K_r \leq 2^{2^r(n+2r)}$ stimmt, dann gilt $K_{r+1} \leq 2^{2^{r+1}(n+2(r+1))}$.

$$
\begin{aligned}
K_{r+1} &= 4 * K_r^2 \\
&\leq 4 * \left(2^{2^r(n+2r)}\right)^2 && \mid \text{Einsetzen der Induktionsvorraussetzung} \\
&= 2^2 * 2^{2*(2^r(n+2r))} && \mid (2^a)^b = 2^{a*b} \text{ und } 4 = 2^2 \\
&= 2^2 * 2^{2^{r+1}(n+2r)} && \mid 2 * 2^r = 2^{r+1} \\
&= 2^{2^{r+1}(n+2r)+2} && \mid 2^2 * 2^a = 2^{a+2} \\
&\leq 2^{2^{r+1}(n+2r)+2*2^{r+1}} && \mid 2 \leq 2 * 2^{r+1} \\
&= 2^{2^{r+1}(n+2r+2)} \\
&= 2^{2^{r+1}(n+2(r+1))}
\end{aligned}
$$

Also gilt $K_r \leq 2^{2^r(n+2r)}$ für alle $r \geq 0$. □

Beispiel 19: Die Koeffizienten K_r

Es sei $\phi_0 = \forall x_3 \forall x_2 \exists x_1 (4x_1 \leq x_2 - 5x_1) \wedge (5x_1 - 5x_3 \leq 6x_3 + 7)$, dann handelt es sich bei $1, 4, 5, 6$ um die Koeffizienten. In Algorithmus 2 wird zunächst nach x_1 umgeformt:

$$\phi_0 \Leftrightarrow \forall x_3 \forall x_2 \exists x_1 ((4+5)x_1 \leq x_2) \wedge (5x_1 \leq (6+5)x_3 + 7)$$

und dann mit verschiedenen Koeffizienten multipliziert, um mit $t - j$ substituieren zu

können. Also zum Beispiel:

$$\phi_0 \Leftrightarrow \forall x_3 \forall x_2 \exists x_1((4+5)x_1 \leq x_2) \wedge ((4+5)*5x_1 \leq (4+5)*(6+5)x_3 + (4+5)*7)$$

Zu beachten ist die Form des so entstandenen Koeffizient $(5+4)*(6+5)$ von x_3. Das Ersetzen der Koeffizienten durch K_r liefert die Formel aus dem Beweis: $(K_r + K_r)^2 = 4 * K_r^2$

Für die folgenden oberen Schranken werden zusätzlich zu der Anzahl der durchgeführten Quantoreliminierungen r noch die Anzahl m der dabei aufgetretenen Quantorenwechsel benötigt (siehe Definition 22). Die Anzahl der Durchgeführten Quantoreliminierungen r kann weiter aufgeteilt werden. Es sei $r = \sum_{i=1}^{m+1} r_i$ (ein r_i bezeichnet die Anzahl der Quantoren im i-ten Quantorenblock, von denen es bei m Quantorenwechsel $m+1$ Stück gibt). Für einen Satz mit m Quantorenwechseln und Dimension r lässt sich beispielsweise

$$Q_{m+1}x_{r_{m+1}}^{m+1}Q_{m+1}x_{r_{m+1}-1}^{m+1}...Q_{m+1}x_1^{m+1}\cdots Q_1x_{r_1}^1Q_1x_{r_1-1}^1...Q_1x_1^1F_1(x_1^1,...,x_{r_{m+1}}^{m+1}) \tag{13}$$

schreiben.

Definition 27: Die obere Schranke S_r^m für $kgV(\beta_1, \beta_2, ...)$
S_r^m ist eine obere Schranke für die Größe der $kgV(\beta_1, \beta_2, ...)$ nach der Eliminierung von r Quantoren und m Quantorenwechsel. (Nach r-maligem Ausführen von Algorithmus 2 bei denen m Quantorenwechsel aufgetreten sind, ist bei dem $(r+1)$. Mal $kgV(\beta_1, \beta_2, ...) \leq S_r^m$)

In einer Formel ϕ_0 aus $\mathcal{L}_{\mathrm{PA}_\mathbb{Z}}$ ist $S_0^m = 1$, da die Eingabe nach der Einschränkung vom Anfang des Kapitels kein $|$ oder $\nmid$ enthält. S_r^m kann immer dann größer werden, wenn es ein neues β aus Teilbarkeiten aus Primformeln der Form (5) und (6) gibt. Diese Primeformeln entstehen ausschließlich im Algorithmus 2 in Zeile 19. Für jede Primformel $p \in pfs$ der Form (3) aus Zeile 14 entsteht ein neues β aus einem Koeffizient $p.\alpha$. Nun werden mit η_r^m nur die verschiedenen $p.\alpha$ nach r Quantoreliminierungen und m Quantorenwechsel gezählt, da doppelte Zahlen im $kgV(...)$ irrelevant sind.
Dann ergibt sich bei der $(r+1)$-ten Ausführung

$kgV(p.\alpha_1, p.\alpha_2, ..., p.\alpha_{\eta_{r-1}^m}, \beta_1, \beta_2, ...)$	neue $p.\alpha_i$ aus r-ten Ausführung und evtl. $\eta_{r-1}^{m-1} \leq \eta_{r-1}^m$
$\leq p.\alpha_1 * p.\alpha_2 * ... * p.\alpha_{\eta_{r-1}^m} * kgV(\beta_1, \beta_2, ...)$	$kgV(a, b, ...) \leq a * kgV(b, ...)$
$\leq K_{r-1}^{\eta_{r-1}^m} * S_{r-1}^m$	$p.\alpha_i \leq K_{r-1}$ und $kgV(\beta_1, \beta_2, ...) \leq S_{r-1}$ (r-te Ausführung $\Rightarrow r-1$ vorausgegangene Quantoreliminierungen)

sodass die folgende Rekursion aufgestellt werden kann.

$$S_r^m \coloneqq K_{r-1}^{\eta_{r-1}^m} * S_{r-1}^m$$

Diese kann aber sofort aufgelöst werden:

$$S_r^m = \prod_{i=0}^{r-1} K_i^{\eta_i^m} \leq \left(K_{r-1}^{\eta_{r-1}^m}\right)^r \quad | \; K_i \leq K_{r-1} \text{ und } \eta_i^m \leq \eta_{r-1}^m \text{ für alle } 0 \leq i \leq r-1 \tag{14}$$

Um S_r^m zu bestimmen wird noch eine obere Schranke für η_{r-1}^m benötigt. Dafür lässt sich definieren:

Definition 28: Die Anzahl q_r^m der durch Koeffizienten unterscheidbaren Primformeln
q_r^m ist die Anzahl der durch Koeffizienten unterscheidbaren Primformeln der Form (3) nach r Quantoreliminierungen und m Quantorenwechsel. (Bem.: Es ist $\eta_r^m \leq q_r^m$ für alle r und m, da q_r^m feiner unterscheidet.(siehe nächstes Beispiel))

Beispiel 20: Durch Koeffizienten unterscheidbare Primformeln

Es seien die folgenden Primformeln der Form (3) gegeben:

1. $5x_1 \leq 4x_2 + 7$
2. $5x_1 \leq 5x_2 + 7$
3. $5x_1 \leq 4x_2 + 100$

Dann sind 1 und 2, als auch 2 und 3, durch Koeffizienten unterscheidbare Primformeln, während 1 und 3 nicht unterscheidbar ist. Demnach wäre $q = 2$. Jedoch ist $\eta = 1$, da alle „$p.\alpha$“ vor x_1 gleich sind.

Satz 14: Obere Schranke für q_r^m
Es ist $q_r^m \leq n^{(r+1)^{(m+1)}}$ (Satz und Beweis folgen der Argumentation aus [RL78])

Beweis. Zunächst gibt es zu Beginn maximal n durch Koeffizienten unterscheidbare Primformeln. Daher gilt:

$$q_0^0 \leq n \tag{15}$$

Wenn ein Block von r_{i+1} gleichen $\exists$ Quantoren eliminiert wird, wird die Anzahl der durch Koeffizienten unterscheidbaren Primformeln maximal r_{i+1} Mal ver-$q^i_{r_1+\ldots+r_i}$-facht. In der folgenden Rechnung steht an einer Formel ein $*$, wenn Teile der Formel vor die Formel gezogen wurden. Für diesen Fall gilt:

1. F_∞ besitzt keine Primformeln der Form (3), diese wurden durch 0 ersetzt. (Algorithmus 2 in Zeile 10)

2. Beim Einsetzen von Zahlen für j entstehen keine neuen durch Koeffizienten unterscheidbare Primformeln. Demzufolge besitzt $\bigvee_{j=0}^{\alpha*\sigma-1}(F_i(\alpha x/t-j) \wedge (\alpha \mid t-j))$ genauso viele durch Koeffizienten unterscheidbare Primformeln, wie $(F_i(\alpha x/t-j) \wedge (\alpha \mid t-j))$.

3. $\exists$ Quantoren lassen sich über $\bigvee$ ziehen. (Algorithmus 2 in Zeile 3)

4. Die Primformel $(\alpha \mid t-j)$ hat nicht die Form (3)

5. Durch die Quantoreliminierung wird F_i zu F^*_{i+1} (bzw. F^*_i zu F^*_{i+1}), dabei wird jedoch nur umgeformt und eingesetzt. Die Anzahl der durch Koeffizienten unterscheidbaren Primformeln der Form (3) bleibt dabei gleich. (Beispielsweise kann aus der Primformel $\alpha x_1 + 5x_2 \leq 3x_3$ die Primformel $t - j + 5x_2 \leq 3x_3$ werden. Die Primformel hat sich dadurch nicht vermehrt.)

Daher wird in folgender Rechnung auf die Darstellung von F_∞ (aufgrund von 1.), $\bigvee_{j=0}^{\alpha*\sigma-1}$ (aufgrund von 2.), $(\alpha \mid t - j)$ (aufgrund von 4.) und $\alpha x_i / t - j$ (aufgrund von 5.) verzichtet. Vereinfacht lässt sich für die mehrfache Anwendung der Quantoreliminierung auf (13) nach $r_1 + ... + r_i$ Quantoreliminierungen für einen Block von $\exists$ Quantoren schreiben:

$$\mathcal{S}_{\mathrm{PA}_\mathbb{Z}} \models Q_m x^m_{r_m} Q_m x^m_{r_m-1} ... Q_m x^m_1 \cdots \exists x^{i+1}_{r_{i+1}} \exists x^{i+1}_{r_{i+1}-1} ... \exists x^{i+1}_1 \underbrace{F_{r_1+...+r_i}(x^{i+1}_1, ..., x^m_{r_m})}_{q^i_{r_1+...+r_i} \text{ Stück}}$$

$$\Leftrightarrow \mathcal{S}_{\mathrm{PA}_\mathbb{Z}} \models Q_m x^m_{r_m} Q_m x^m_{r_m-1} ... Q_m x^m_1 \cdots \exists x^{i+1}_{r_{i+1}} \exists x^{i+1}_{r_{i+1}-1} ... \exists x^{i+1}_2 \underbrace{\bigvee_{(3)}}_{\#(3) \leq q^i_{r_1+...+r_i}} \underbrace{F^*_{r_1+...+r_i+1}(x^{i+1}_2, ..., x^m_{r_m})}_{q^i_{r_1+...+r_i} \text{ Stück}}$$

$$\Leftrightarrow \mathcal{S}_{\mathrm{PA}_\mathbb{Z}} \models Q_m x^m_{r_m} Q_m x^m_{r_m-1} ... Q_m x^m_1 \cdots \underbrace{\bigvee_{(3)}}_{\#(3) \leq q^i_{r_1+...+r_i}} \exists x^{i+1}_{r_{i+1}} \exists x^{i+1}_{r_{i+1}-1} ... \exists x^{i+1}_2 \underbrace{F^*_{r_1+...+r_i+1}(x^{i+1}_2, ..., x^m_{r_m})}_{q^i_{r_1+...+r_i} \text{ Stück}}$$

$$\Leftrightarrow \mathcal{S}_{\mathrm{PA}_\mathbb{Z}} \models Q_m x^m_{r_m} Q_m x^m_{r_m-1} ... Q_m x^m_1 \cdots \underbrace{\bigvee_{(3)}}_{\#(3) \leq q^i_{r_1+...+r_i}} \exists x^{i+1}_{r_{i+1}} \exists x^{i+1}_{r_{i+1}-1} ... \exists x^{i+1}_3 \underbrace{\bigvee_{(3)}}_{\#(3) \leq q^i_{r_1+...+r_i}} \underbrace{F^*_{r_1+...+r_i+2}(x^{i+1}_3, ..., x^m_{r_m})}_{q^i_{r_1+...+r_i} \text{ Stück}}$$

$$\vdots$$

$$\Leftrightarrow \mathcal{S}_{\mathrm{PA}_\mathbb{Z}} \models Q_m x^m_{r_m} Q_m x^m_{r_m-1} ... Q_m x^m_1 \cdots \underbrace{\bigvee_{(3)}}_{\#(3) \leq q^i_{r_1+...+r_i}} \cdots \underbrace{\bigvee_{(3)}}_{\#(3) \leq q^i_{r_1+...+r_i}} \underbrace{F^*_{r_1+...+r_i+r_{i+1}}(x^{i+2}_1, ..., x^m_{r_m})}_{q^i_{r_1+...+r_i} \text{ Stück}}$$

$\bigvee_{(3)}$ bedeutet, dass für jedes $p \in pfs$, $p.\alpha * x$ durch $p.t - j$ ersetzt wird. Dadurch kann jede durch Koeffizienten unterscheidbare Primformel $(p.\alpha x \leq t)$ $q^i_{r_1+...+r_i}$-Mal neue Koeffizienten bekommen: $p.t - j \leq t \equiv -j \leq t - p.t$. Das Ganze passiert r_{i+1}-Mal. Außerdem wurde ein ganzer Block von existenziellen Quantoren eliminiert, wodurch die Anzahl der Quantorenwechsel um eins sinkt.

$$q^{i-1}_{r_1+r_2+...+r_{i+1}} \leq (q^i_{r_1+r_2+...+r_i})^{(r_{i+1}+1)} \tag{16}$$

Für den Fall, dass es sich um einen Block von $\forall ... \forall$ Quantoren gehandelt hat, werden diese im Algorithmus zu $\neg\exists ... \exists\neg$ umgeformt. Anders ausgedrückt: Immer dann, wenn es einen Quantorenwechsel gibt, steht zwischen den $\exists$ Quantoren ein $\neg$ (Algorithmus 1 in Zeile 9 und 12). Diese $\neg$ sorgen im Algorithmus 2 in Zeile 2 dafür, dass durch das Gesetz von De Morgen die durch die vorherige Eliminierungen erzeugten $\bigvee_{(3)} ... \bigvee_{(3)}$ zu $\bigwedge_{(3)} ... \bigwedge_{(3)}$ werden. Dann lassen sich die nächsten $\exists$ nicht wie in der obigen Umformung in die Formel ziehen, sodass neu potenziert werden muss:

$$q^m_r = q^m_{r_1+...+r_{m+1}} \qquad \Big| \quad r = \sum_{i=1}^{m+1} r_i$$

$$
\begin{aligned}
&\leq \left(q^{m-1}_{r_1+\ldots+r_m}\right)^{(r_{m+1}+1)} && \text{| Anwenden von (16)}\\
&\leq \left(\left(q^{m-1}_{r_1+\ldots+r_{m-1}}\right)^{(r_m+1)}\right)^{(r_{m+1}+1)} && \text{| Anwenden von (16)}\\
&\leq \left(\left(\left(q^{0}_{r_1}\right)^{(r_2+1)}\right)^{\cdots}\right)^{(r_{m+1}+1)} && \text{| Wiederholtes Anwenden von (16)}\\
&\leq \left(\left(\left(n^{(r_1+1)}\right)^{(r_2+1)}\right)^{\cdots}\right)^{(r_{m+1}+1)} && \text{| Betrachte die obige Umformung für den Fall ohne vorrausgehende Quantorenblöcke, außerdem ist } q^0_0 \leq n \text{ laut (15) (Für erneute Rechnung siehe Anhang)}\\
&\leq \left(\left(n^{(r+1)}\right)^{\cdots}\right)^{(r+1)} && \text{| } r_i \leq r \text{ für alle } 1 \leq i \leq m+1\\
&= n^{(r+1)^{(m+1)}} && \text{| Mehrfaches Anwenden von } (a^b)^b = a^{b*b}
\end{aligned}
$$

□

Satz 15: Obere Schranke für S^m_r
Es ist $S^m_r \leq 2^{n^{(r+2)^{(m+2)}}}$ für $n \geq r$ und $n \geq 2^2$ (Satz und Beweis folgen der Argumentation aus [Grä88], jedoch mit größer gewählter oberer Schranke)

Beweis. Durch Einsetzen in (14) und Abschätzen ergibt sich:

$$
\begin{aligned}
S^m_r &\leq \left(K^{q^m_{r-1}}_{r-1}\right)^r && \text{| (14) und } \eta^m_{r-1} \leq q^m_{r-1}\\
&\leq \left(K^{n^{r^{(m+1)}}}_{r-1}\right)^r && \text{| Anwenden von Satz 14}\\
&\leq \left(\left(2^{2^{(r-1)}(n+2(r-1))}\right)^{n^{r^{(m+1)}}}\right)^r && \text{| Anwenden von Satz 13}\\
&= 2^{2^{2(r-1)}*(n+2(r-1))*n^{r^{(m+1)}}*r} && \text{| } ((2^x)^a)^b = 2^{x*a*b}\\
&\leq 2^{n^{(r-1)}*(n+2n)*n^{r^{(m+1)}}*n} && \text{| } n \geq r\\
&\leq 2^{n^{(r-1)}*(2^2*n)*n^{r^{(m+1)}}*n} && \text{| } 2^2 \geq 3\\
&\leq 2^{n^{(r-1)}*(n*n)*n^{r^{(m+1)}}*n} && \text{| } n \geq 2^2\\
&= 2^{n^{(r-1)}*n^3*n^{r^{(m+1)}}} && \text{| } n*n*n = n^3\\
&= 2^{n^{(r+2)}*n^{r^{(m+1)}}} && \text{| } x^a * x^b = x^{a+b}\\
&\leq 2^{n^{(r+2)^{(m+1)}}*n^{(r+2)^{(m+1)}}} && \text{| } n^{(r+2)} \leq n^{(r+2)^{(m+1)}}
\end{aligned}
$$

$$
\begin{aligned}
&= 2^{\left(n^{(r+2)^{(m+1)}}\right)^2} && \mid x * x = x^2 \\
&= 2^{n^{2*(r+2)^{(m+1)}}} && \mid 2^{(x^a)^b} = 2^{(x^{b*a})} \\
&\leq 2^{n^{(r+2)*(r+2)^{(m+1)}}} && \mid (r+2) \geq 2 \\
&= 2^{n^{(r+2)^{(m+2)}}} && \mid a * a^b = a^{b+1}
\end{aligned}
$$

□

Definition 29: Die obere Schranke A_r^m für additive Konstanten
A_r^m ist eine obere Schranke für die Größe der additiven Konstanten nach der Eliminierung von r Quantoren und m Quantorenwechsel. (Im Algorithmus 2 also z.B. $j \leq A_r^m$ und $c_{s+1} \leq A_r^m$)

Satz 16: Obere Schranke für A_r^m
Es ist $A_r^m \leq 2^{n^{(r+3)^{(m+2)}}}$ für $n \geq r$ und $n \geq 2^2$.(Satz und Beweis folgen der Argumentation aus [Grä88], jedoch mit größer gewählter oberer Schranke)

Beweis. Beweis per vollständiger Induktion über r.
IA) Zu Beginn muss $A_0^m \leq 2^n \left(\leq 2^{n^{(0+3)^{(m+2)}}}\right)$ sein, da ϕ_0 nicht nur aus einer Konstanten besteht, und eine Konstante der Länge $(n-1)$ im Dezimalsystem kleiner als 2^n ist.
IS) Zu zeigen ist: Angenommen $A_{r-1}^m \leq 2^{n^{(r+2)^{(m+2)}}} := A_{r-1}'^m$ gilt, dann folgt $A_r^m \leq 2^{n^{(r+3)^{(m+2)}}}$.
Zunächst kann eine additive Konstante in Algorithmus 2 in Zeile 5 mit einer anderem Konstante zusammengefasst werden. Danach wird so eine additive Konstante in Zeile 17 maximal mit einem Koeffizient $\alpha \leq K_{r-1}$ multipliziert und mit $j \leq K_{r-1} * S_{r-1}^m$ (in Zeile 18) summiert. Insgesamt ergibt sich:

$$A_r^m \leq \underbrace{2 * A_{r-1}^m}_{Zusammenfassen} * \underbrace{K_{r-1}}_{\geq \alpha} + \underbrace{K_{r-1} * S_{r-1}^m}_{\geq j} \tag{17}$$

Nun lässt sich S_{r-1}^m weiter nach oben abschätzen:

$$S_{r-1}^m \leq 2^{n^{(r+1)^{(m+2)}}} \leq \underbrace{2^{n^{(r+2)^{(m+2)}}}}_{A_{r-1}'^m}$$

Daraus resultiert $S_{r-1}^m \leq A_{r-1}'^m$. Mit der Induktionsannahme $A_{r-1}^m \leq A_{r-1}'^m$ kann (17) weiter abgeschätzt werden:

$$A_r^m \leq 2 * A_{r-1}'^m * K_{r-1} + K_{r-1} * A_{r-1}'^m = 3 * A_{r-1}'^m * K_{r-1} \tag{18}$$

Außerdem kann auch $3 * K_{r-1}$ weiter abgeschätzt werden:

$$3 * K_{r-1} \leq 3 * 2^{2^{(r-1)}(n+2(r-1))} \qquad \mid \text{Anwenden von Satz 13}$$

$$
\begin{aligned}
&\leq 2^2 * 2^{2^{(r-1)}(n+2(r-1))} && \mid 3 \leq 2^2 \\
&= 2^{2^{(r-1)}(n+2(r-1))+2} && \mid 2^2 * 2^x = 2^{x+1} \\
&\leq 2^{2^{(r-1)}(n+2(r-1))+2*2^{(r-1)}} && \mid 2 \leq 2 * 2^{(r-1)} \\
&= 2^{2^{(r-1)}(n+2r)} && \mid \text{Einklammern} \\
&\leq 2^{n^{(r-1)}*3n} && \mid n \geq r \\
&\leq 2^{n^{(r-1)}*n^2} && \mid n \geq 3 \\
&= 2^{n^{(r+1)}} && \mid x^a * x^2 = x^{a+2} \\
&\leq \underbrace{2^{n^{(r+2)^{(m+2)}}}}_{A'^m_{r-1}} && \mid x^{a+2} \leq x^{(a+2)^{b+2}} \text{ für beliebige } b \geq 0
\end{aligned}
$$

Schließlich wird (18) weiter abgeschätzt:

$$
\begin{aligned}
A^m_r &\leq A'^m_{r-1} * 3 * K_{r-1} && \mid (18) \\
&\leq (A'^m_{r-1})^2 && \mid 3 * K_{r-1} \leq A'^m_{r-1} \\
&\leq \left(2^{n^{(r+2)^{(m+2)}}}\right)^2 && \mid \text{Definition von } A'^m_{r-1} \\
&= 2^{2*n^{(r+2)^{(m+2)}}} && \mid (2^a)^2 = 2^{2*a} \\
&\leq 2^{n*n^{(r+2)^{(m+2)}}} && \mid n \geq 2 \\
&\leq 2^{n^{(r+3)^{(m+2)}}} && \mid n * n^{a^{b+2}} \leq n^{(a+1)^{b+2}}
\end{aligned}
$$

□

An dieser Stelle kann eine obere Schranke für $t + j$ bestimmt werden, da alle nötigen oberen Schranken für die Komponenten von $t + j$ bestimmt wurden. Zur Erinnerung an die Situation: Gesucht ist eine obere Schranke für den Betrag der zu testenden Belegungen von x_r aus F_r. F_r ist die Funktion, die durch $r - 1$ vorher angewandte Quantoreliminierungen entstanden ist. Zu diesem Zweck wird δ_{r+1} betrachtet, in der x_r mittels der Quantoreliminierung angewandt auf $Q_r x_r$ F_r eliminiert wurde. Bei der Quantoreliminierung wird unter anderem x_r so substituiert, dass $t + j$ eine obere Schranke für die zu testenden Belegungen von x_r ist. Dieses $t + j$ taucht jedoch erst in δ_{r+1} auf, weshalb die oberen Schranken aus Satz 13 - 16 nach r Quantoreliminierungen benötigt werden. (vgl. Abbildung 9)

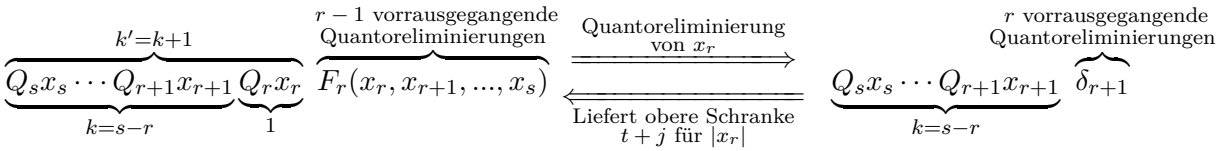

Abbildung 9: Eine obere Schranke für $|x_r|$

Satz 17: Eine obere Schranke für die zu testende Belegung der letzten von $k+1$ zu beschränkende Belegungen
Es sei F_r die Funktion aus Definition 16 mit der Belegungsfunktion ϱ_r. Jedoch sei

$$|\varrho_r(x_i)| \leq w$$

für alle $(r+1 \leq i \leq s)$ und x_r die noch zu beschränkende Variable. Dann gilt:

$$\mathcal{S}_{\mathrm{PA}_\mathbb{Z}} \models Qx_r \ F_r(x_r, \varrho_r(x_{r+1}), ..., \varrho_r(x_s)) \Leftrightarrow \mathcal{S}_{\mathrm{PA}_\mathbb{Z}} \models (Q|x_r| \leq p) \ F_r(x_r, \varrho_r(x_{r+1}), ..., \varrho_r(x_s))$$

für $p = (k{*}w{+}1){*}\left(2^{n^{(r+3)^{(m+2)}}}\right)^2$ mit $k = s{-}r$ per Voraussetzung beschränkten Variablen.
Die ursprüngliche Formel ϕ_0 hatte m Quantorenwechsel. (Satz und Beweisstruktur aus [RL78])

Beweis. Zunächst für den Fall $Q = \exists$:
$\Rightarrow$: $t + j$ (vgl. (11) und (12)) benötigt eine obere Schranke, die nach der r-ten Eliminierung auftritt (vgl. Argumentation vom Anfang des Kapitels).

$$t + j = \left(\sum_{j=1}^{k} c_i * x_i\right) + c_{k+1} + j \quad | \text{ Definition von } t$$

$$\leq k * K_r * w + A_r^m + K_r * S_r^m \quad | \text{ Abschätzen der Anzahl der Quantorenwechsel nach der r-ten Eliminierung auf m, Anwenden der Definition von } K_r, A_r^m \text{ und } j \leq K_r * S_r^m$$

$$\leq k * 2^{2^r(n+2r)} * w + 2^{n^{(r+3)^{(m+2)}}} + 2^{2^r(n+2r)} * 2^{n^{(r+2)^{(m+2)}}} \quad | \text{ Anwenden von Sätzen 13,15 und 16}$$

$$\leq k * 2^{n^r(n+2n)} * w + 2^{n^{(r+3)^{(m+2)}}} + 2^{n^r(n+2n)} * 2^{n^{(r+2)^{(m+2)}}} \quad | \ r \leq n$$

$$\leq k * 2^{n^r(n^2)} * w + 2^{n^{(r+3)^{(m+2)}}} + 2^{n^r(n^2)} * 2^{n^{(r+2)^{(m+2)}}} \quad | \ 3 \leq n$$

$$\leq k * 2^{n^{(r+2)}} * w + 2^{n^{(r+3)^{(m+2)}}} + 2^{n^{(r+2)}} * 2^{n^{(r+2)^{(m+2)}}} \quad | \ n^a * n^2 = n^{a+2}$$

$$\leq k * 2^{n^{(r+2)^{(m+2)}}} * w + 2^{n^{(r+3)^{(m+2)}}} + 2^{n^{(r+2)^{(m+2)}}} * 2^{n^{(r+2)^{(m+2)}}} \quad | \ 2^a \leq 2^{a^b}$$

$$\leq k * 2^{n^{(r+2)^{(m+2)}}} * w + 2^{n^{(r+3)^{(m+2)}}} + 2^{2*n^{(r+2)^{(m+2)}}} \quad | \ (2^{a^b})^2 = 2^{2*a^b}$$

$$\leq k * 2^{n^{(r+2)^{(m+2)}}} * w + 2^{n^{(r+3)^{(m+2)}}} + 2^{n^{(r+3)^{(m+2)}}} \quad | \ 2 \leq n$$

$$= k * 2^{n^{(r+2)^{(m+2)}}} * w + 2 * 2^{n^{(r+3)^{(m+2)}}} \quad | \text{ Zusammenfassen}$$

$$\leq k * 2 * 2^{n^{(r+3)^{(m+2)}}} * w + 2 * 2^{n^{(r+3)^{(m+2)}}} \quad | \ a \leq 2a$$

$$\leq (k * w + 1) * 2 * 2^{n^{(r+3)^{(m+2)}}} \quad | \text{ Einklammern}$$

$$\leq (k * w + 1) * \left(2^{n^{(r+3)^{(m+2)}}}\right)^2$$

$\Leftarrow$: Wenn es eine Belegung für x_r kleiner gleich p gibt, welches F_r erfüllt, dann gibt es ebenfalls eine Belegung für ein unbeschränktes x_r.

Der Fall $Q = \forall$ kann auf den obigen Fall zurückgeführt werden. Auf die Frage, ob es eine Belegung für x_r gibt welches $\neg F_r$ erfüllt, kann zunächst wie im Algorithmus 2 in Zeile 2 das Negationszeichen eliminiert werden. Anschließend kann wie im obigen Fall ein p gefunden werden, welches die zu prüfenden Möglichkeiten für die Belegungen von x_r beschränkt. □

Satz 17 liefert den Schritt von $r-1$ zu r durchgeführten Quantoreliminierungen. Dabei wird eine obere Schranke für den Betrag der zu testenden Belegungen von x_r gefunden. Diese ist jedoch abhängig von der Belegung der $k = s - r$ noch zu eliminierende gebundenen Variablen (vgl. Abbildung 9). Es bietet sich demnach eine Induktion an, die über die Anzahl $k' := k + 1$ der zu beschränkenden gebundenen Variablen geht und den folgenden Satz als Ziel hat:

Satz 18: Obere Schranken für die zu testenden Belegungen der gebundenen Variablen während der Quantoreliminierung eines Satzes aus $Th(\mathcal{S}_{\mathbf{PA}_\mathbb{Z}})$
Für Formeln F_r aus Definition 16 und $k' := k+1 = s-r+1$ zu beschränkenden gebundene Variablen gilt für alle $k' \in [1, s]$:

$$\mathcal{S}_{\mathrm{PA}_\mathbb{Z}} \models Q_s x_s ... Q_r x_r F_r(x_r, ..., x_s)$$
$$\Leftrightarrow$$
$$\mathcal{S}_{\mathrm{PA}_\mathbb{Z}} \models (Q_s |x_s| \leq w_{k'}) ... (Q_r |x_r| \leq w_{k'}) F_r(x_r, ..., x_s)$$

wobei $w_{k'} = \left(2^{n^{(s+3)^{(m+2)}}}\right)^{(8^{k'})}$ (Satz ähnlich wie in [RL78], jedoch mit anderen Schranken und weiter ausformuliertem Beweis)

Beweis. Beweis mithilfe von Induktion über $k' = k + 1$ der Anzahl der noch zu eliminierenden Variablen ($k' \in [1, s]$):
IA): Für $k' = 1$ bzw. $k = 0$ handelt es sich um den Fall:

$$\mathcal{S}_{\mathrm{PA}_\mathbb{Z}} \models Q_s x_s F_s(x_s) \Leftrightarrow \mathcal{S}_{\mathrm{PA}_\mathbb{Z}} \models (Q_s |x_s| \leq w_1) F_s(x_s)$$

Die Äquivalenz liefert Satz 17. Für $k = 0$ besteht t nur noch aus einer Konstante c_{s+1}, sodass keine schon beschränkte Variable aus der Voraussetzung des Satzes benötigt wird.

$$|x_1| \leq (0 * w + 1) * \left(2^{n^{(s+3)^{(m+2)}}}\right)^2 \leq \left(2^{n^{(s+3)^{(m+2)}}}\right)^{(8^1)}$$

IS): Zu zeigen ist: Falls für die Formeln F_{r+1} mit r schon eliminierten Variablen und mit $k'-1$ noch nicht eliminierten Variablen gilt, dass

$$\mathcal{S}_{\mathrm{PA}_\mathbb{Z}} \models Q_s x_s ... Q_{r+1} x_{r+1} F_{r+1}(x_{r+1}, ..., x_s)$$
$$\Leftrightarrow \tag{19}$$
$$\mathcal{S}_{\mathrm{PA}_\mathbb{Z}} \models (Q_s |x_s| \leq w_{k'-1}) ... (Q_{r+1} |x_{r+1}| \leq w_{k'-1}) F_{r+1}(x_{r+1}, ..., x_s)$$

ist, dann folgt die Aussage

$$\begin{gathered}\mathcal{S}_{\mathrm{PA}_\mathbb{Z}} \models Q_s x_s ... Q_r x_r F_r(x_s, ..., x_r) \\ \Leftrightarrow \\ \mathcal{S}_{\mathrm{PA}_\mathbb{Z}} \models (Q_s|x_s| \leq w_{k'}) ... (Q_r|x_r| \leq w_{k'}) F_r(x_r, ..., x_s)\end{gathered} \tag{20}$$

für k' (Das ist der Fall mit $k+1$ zu beschränkenden Variablen).

Zunächst lassen sich die oberen Schranken $w_{k'-1}$ der Belegungen von x_i aus (19) mit $r+1 \leq i \leq s$ auf F_r übertragen. Das liegt daran, dass der Schritt von F_r nach F_{r+1} die Eliminierung von $Q_r x_r$ ist, die aber nur Äquivalenzoperationen auf Primformeln mit x_i ausführt (vgl. Satz 7). Dies liefert:

$$\begin{gathered}\mathcal{S}_{\mathrm{PA}_\mathbb{Z}} \models Q_s x_s ... Q_r x_r F_r(x_s, ..., x_r) \\ \Leftrightarrow \\ \mathcal{S}_{\mathrm{PA}_\mathbb{Z}} \models (Q_s|x_s| \leq w_{k'-1}) ... (Q_{r+1}|x_{r+1}| \leq w_{k'-1}) Q_r x_r F_r(x_r, ..., x_s)\end{gathered} \tag{21}$$

Das ist genau der Fall aus Satz 17. Es gibt k Variablen, für die eine obere Schranke $w = w_{k'-1}$ gegeben ist und $F = F_r$ ist das Ergebnis von $r-1$ Quantoreliminierungen. Nun wird eine obere Schranke für die zu testenden Belegungen von $|x_r|$ gesucht:

$$\begin{aligned}
|x_r| &\leq (k * w_{k'-1} + 1) * \left(2^{n^{(r+3)^{(m+2)}}}\right)^2 && \text{Satz 17 liefert eine Schranke für die letzte von } k+1 \text{ zu beschränkende Variablen} \\
&\leq \left(k * \left(2^{n^{(s+3)^{(m+2)}}}\right)^{\left(8^{k'-1}\right)} + 1\right) * \left(2^{n^{(r+3)^{(m+2)}}}\right)^2 && \text{Einsetzen der Schranke } w_k \text{ aus der Induktionsannahme} \\
&\leq \left(2^{n^{(s+3)^{(m+2)}}} * \left(2^{n^{(s+3)^{(m+2)}}}\right)^{\left(8^{k'-1}\right)} + 1\right) * \left(2^{n^{(s+3)^{(m+2)}}}\right)^2 && k \leq 2^{n^{(s+3)^{(m+2)}}} \text{ und } r \leq s \\
&= \left(\left(2^{n^{(s+3)^{(m+2)}}}\right)^{\left(8^{k'-1}+1\right)} + 1\right) * \left(2^{n^{(s+3)^{(m+2)}}}\right)^2 && a * a^b = a^{b+1} \\
&\leq \left(2^{n^{(s+3)^{(m+2)}}}\right)^{\left(8^{k'-1}+2\right)} * \left(2^{n^{(s+3)^{(m+2)}}}\right)^2 \\
&= \left(2^{n^{(s+3)^{(m+2)}}}\right)^{\left(8^{k'-1}+4\right)} && a^b * a^2 = a^{b+2} \\
&\leq \left(2^{n^{(s+3)^{(m+2)}}}\right)^{\left(8^{(k'-1)+1}\right)} \\
&= w_{k'}
\end{aligned}$$

Also beschränkt $w_{k'}$ die Anzahl der zu testenden Belegungen für $|x_r|$. Dann können in (21) die $w_{k'-1}$ auf $w_{k'}$ abgeschätzt werden. Dadurch werden nur weitere Belegungen getestet, die

eigentlich nicht getestet werden müssten, da die Erfüllbarkeitsäquivalenz schon vorher galt. Dann folgt insgesamt aber (20).

□

Satz 19: Erfüllbarkeitsäquivalenz im Fall von beschränkter Belgungswahl für gebundene Variablen
Wenn $Q_1x_1..Q_sx_s$ mit Länge n, m Quantorenblöcke enthält, dann gilt für $w = 2^{n^{3*s+(s+3)^{(m+1)}}}$:

$$\mathcal{S}_{\mathrm{PA}_\mathbb{Z}} \models Q_1x_1, ...Q_sx_sF(x_1, ..., x_s) \Leftrightarrow \mathcal{S}_{\mathrm{PA}_\mathbb{Z}} \models (Q_1|x_1| \leq w), ...(Q_s|x_s| \leq w)F(x_1, ..., x_s)$$

Beweis. Für den Beweis soll Satz 18 genutzt werden. Dort ging es jedoch um ein Quantorprefix mit m Quantorenwechseln. Ein Quantorenprefix mit m Quantorenblöcken enthält jedoch nur $m-1$ Quantorenwechsel. Also wird in Satz 18 für $m, m-1$ eingesetzt und w_s weiter abgeschätzt:

$$\begin{aligned}
w_s &= \left(2^{n^{(s+3)^{(m+1)}}}\right)^{(8^s)} && \mid \text{Anwenden von Satz 18} \\
&= 2^{(8^s)*n^{(s+3)^{(m+1)}}} && \mid (2^a)^b = 2^{b*a} \\
&= 2^{\left(2^3\right)^s*n^{(s+3)^{(m+1)}}} && \mid 8 = 2^3 \\
&= 2^{2^{3*s}*n^{(s+3)^{(m+1)}}} && \mid (2^a)^b = 2^{b*a} \\
&\leq 2^{n^{3*s}*n^{(s+3)^{(m+1)}}} && \mid n \geq 2 \\
&= 2^{n^{3*s+(s+3)^{(m+1)}}} && \mid n^a * n^b = n^{a+b}
\end{aligned}$$

□

6.2.2 Einordnung in die Polynomialzeithierarchie

Mit den bisherigen Erkenntnissen ist es nun möglich, die verschiedenen Teilklassen Υ^s_m in die Polynomialzeithierarchie einzuordnen, d.h. die Inklusion zu zeigen. Dabei wird die Vorgehensweise aus [Grä88] auf Υ^s_m angewandt.
Satz 19 liefert zunächst einen alternativen Entscheidungsalgorithmus für $\Upsilon^s_m(\mathrm{PA}_\mathbb{Z})$. Zunächst wird mithilfe der festgelegten s und m sowie der bestimmbaren Länge n, das w aus Satz 19 berechnet. w beschränkt die Anzahl der zu testenden Belegungen, sodass danach im schlechtesten Fall nur noch alle $(2*w)^s$ Kombinationen für Belegungen der x_i getestet werden müssen ($2*w$, da die x_i sowohl positiv, als auch negative Belegungen bekommen können).
In Satz 3 konnten Sprachen, die mit einer ATM mit $k-1$ „Zustandsmengenwechsel“ in Polynomialzeit gelöst werden können, in die Polynomialzeithierarchie eingeordnet werden. Das soll nun genutzt werden, indem der alternative Entscheidungsalgorithmus mithilfe einer ATM, die in Polynomialzeit läuft (Begründung siehe unten), ausgeführt wird.
Die Idee ist es, den Nichtdeterminismus von ATM's zum Belegen der Variablen zu nutzen und dabei zwischen existenziellen und universellen Zuständen, abhängig von dem Quantor vor der jeweiligen Variable zu wechseln. Eine Dezimalzahl $\varrho(x_i) \leq 2^{n^{3*s+(s+3)^{(m+1)}}}$ ist im Binärsystem

maximal $n^{3*s+(s+3)^{(m+1)}}$ Stellen lang. Hinzu kommt noch eine Stelle, wenn negative Zahlen mithilfe der Vorzeichenbetragsdarstellung codiert werden. Das führt jedoch dazu, dass das nicht-deterministische Belegen im Folgendem in Polynomialzeit läuft.

Algorithm 3 Alternativer Entscheidungsalgorithmus für eine ATM Σ_m bzw. Π_m

```
1:  function MAIN(Formel φ_0, int n := |φ_0|, int s_m := 3 * s + (s + 3)^(m+1))
2:                  ▷ Startzustand aus ⋁ bzw. ⋀ falls sich um eine ATM Σ_m bzw. Π_m handelt
3:      Formel φ_0 ← erzeugePränexnormalform(φ_0)
4:      Quantor[] Q ← extrahiereQuantorprefix(φ_0)          ▷ [Q_1, ..., Q_s] ist ein Array
5:      Funktion F ← ohneQuantorprefix(φ_0)
6:      Belegung ϱ ← ∅
7:      for int i ← 0 to s − 1 do
8:          ϱ ← ϱ ∪ Setze Belegung für x_{i+1} nicht deterministisch mit Länge n^{s_m} + 1
9:          if Q[i + 1] = ∀ then
10:             Zustand ← beliebig ∈ ⋀
11:         else
12:             Zustand ← beliebig ∈ ⋁
13:         end if
14:     end for
15:     if (S_{PA_ℤ}, ϱ) ⊨ F then Zustand ← beliebig ∈ A
16:     else Zustand ← beliebig ∈ V
17:     end if
18: end function
```

Nachfolgend soll Algorithmus 3 erklärt werden. Dabei möchte ich zunächst auf den Prozess des nicht-deterministischen Belegens eingehen.
Um beispielsweise eine Variable x nichtdeterministisch mit Binärzahlen einer bestimmten Länge n zu belegen, wird in jedem Schritt der Maschine die Belegung $\varrho(x)$ um eine Stelle erweitert, bis sie n Stellen besitzt. Wenn x existenziell quantifiziert ist, muss sich die ATM währenddessen in Zuständen aus $\bigvee$ befinden, andernfalls in Zuständen aus $\bigwedge$. Dann können, wie in Abbildung 10, die Konfigurationsübergänge zum Belegen einer Variable zusammengefasst werden. Wenn nun mehrere Belegungen in den jeweils passenden Zustandsmengen gewählt werden, ergibt sich ein Baum wie in Abbildung 11. Die Blätter des Baumes akzeptieren nur dann, wenn die quantifizierte Funktion mit der vorher gewählten Belegung wahr ist. Dann kann mit der Funktion f aus (1) ausgewertet werden.
Zur weiteren Erklärung des Algorithmus sei

$$\phi_0 = Q_1 x_1 Q_2 x_2 ... Q_s x_s F(x_1, x_2, ..., x_s)$$

ein Satz in Pränexnormalform aus $\Upsilon_m^s(\mathrm{PA}_\mathbb{Z})$. Davon kann ausgegangen werden, da der Algorithmus zu Beginn die Formel in Pränexnormalform bringt (Zeile 3). Nun kann induktiv über die Anzahl der noch nicht belegten Variablen die Funktionalität des Algorithmus nachvollzogen werden. Zu diesem Zweck wird Algorithmus 3 von hinten nach vorne betrachtet. In den letzten Schritten (Zeile 15 ff.) wird deterministisch entschieden, ob das vorausgegangene nicht-deterministisch Belegen zu einer „akzeptierenden Konfiguration“ führt. Das ist der Fall falls

$$F(\varrho(x_1), \varrho(x_2), ..., \varrho(x_s)) = 1$$

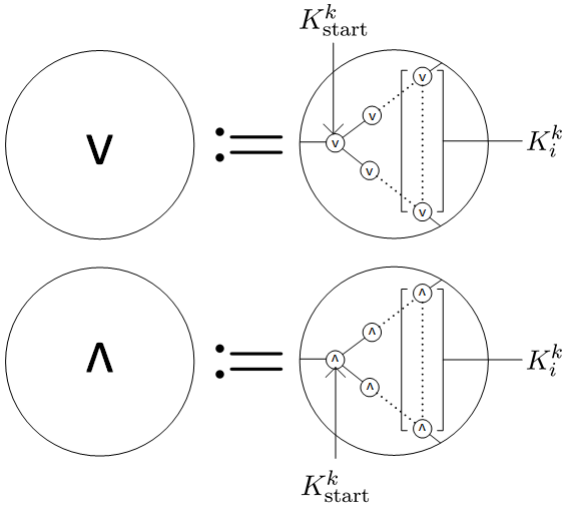

Abbildung 10: Konfigurationsübergänge bei dem nichtdeterministischen Belegen von x_k

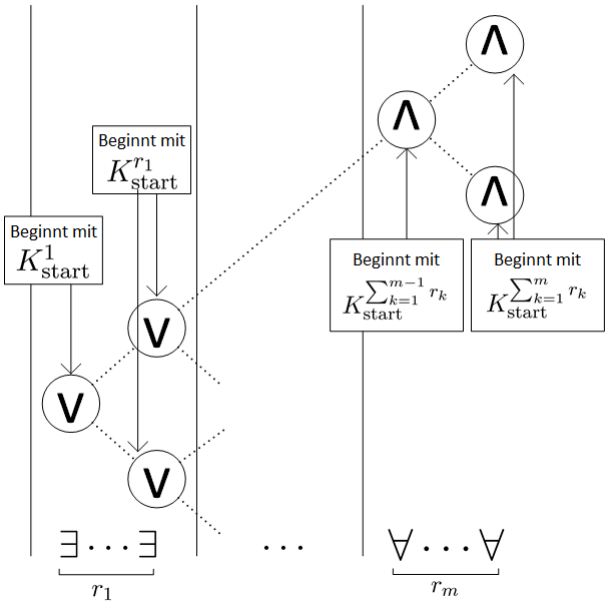

Abbildung 11: Idee des Algorithmus 3 mit nichtdeterministischen Belegen aus Abbildung 10 für einen Satz mit m Quantorenblöcken

ist, also $(\mathcal{S}_{\mathrm{PA}_\mathbb{Z}}, \varrho) \models F$ („Induktionsanfang“). Dabei ist egal, ob die ATM sich gerade in einem existenziellen oder in einem universellen Zustand befindet, da bei dem deterministischen Auswerten jede Konfiguration nur eine Folgekonfiguration besitzt, die dementsprechend wahr sein muss, damit die Konfiguration zu Beginn der Auswertung wahr wird.
Die vorausgegangenen Schritte dienen dem nicht-deterministischen Belegen der x_i (Zeile 7 - 14) unter Ausnutzung der Zustandsmengen der ATM.
Der „Induktionsschritt“ besteht darin zu zeigen, dass

$$\mathcal{S}_{\mathrm{PA}_\mathbb{Z}} \models Q_k x_k \; Q_{k+1} x_{k+1} ... Q_s x_s F(\varrho(x_1), ..., \varrho(x_{k-1}), x_k, x_{k+1}, ..., x_s)$$

nur dann gilt, wenn die Konfiguration K^k_{start}, in dem sich die ATM zu Beginn des k-ten Durchlaufs der **for**-Schleife befindet, eine Akzeptierenden wird $f(K^k_{\mathrm{start}}) = 1$. Dafür darf benutzt werden, dass

$$\mathcal{S}_{\mathrm{PA}_\mathbb{Z}} \models Q_{k+1} x_{k+1} ... Q_s x_s F(\varrho(x_1), ..., \varrho(x_{k-1}), \varrho(x_k), x_{k+1}, ..., x_s)$$

nur dann gilt, wenn die Konfiguration K^{k+1}_{start}, die die ATM zu Beginn des nächsten **for**-Schleifendurchlaufs hat, eine Akzeptierende wird $f(K^{k+1}_{\mathrm{start}}) = 1$ („Induktionsvoraussetzung“).
Dann kann zwischen dem Fall $Q_k = \exists$ und $Q_k = \forall$ unterschieden werden. Im ersten Fall ist

$$\exists x_k \; Q_{k+1} x_{k+1} ... Q_s x_s F(\varrho(x_1), ..., \varrho(x_{k-1}), x_k, x_{k+1}, ..., x_s)$$

gegeben.
Nun wird mithilfe von nicht-deterministischen Belegens eine Belegung $\varrho(x_k)$ gesucht, sodass

$$\mathcal{S}_{\mathrm{PA}_\mathbb{Z}} \models Q_{k+1} x_{k+1} ... Q_s x_s F(\varrho(x_1), ..., \varrho(x_{k-1}), \varrho(x_k), x_{k+1}, ..., x_s)$$

gilt und damit per Voraussetzung $f(K^{k+1}_{\mathrm{start}}) = 1$ ist. In Zeile 12 des vorausgegangenen Durchlaufs ist die ATM in einen Zustand aus $\bigvee$ gewechselt, da $Q_k = \exists$ ist. Bei dem nicht-deterministischen Belegen (Zeile 8) bleibt die Zustandsmenge erhalten. Es bildet sich eine Verzweigung, wie im oberen Bild von Abbildung 10 angedeutet. Es seien nun die Konfigurationen am Ende diese Baumes K^k_i mit $i \in \mathbb{N}$ bezeichnet, dann lässt sich mit f auswerten:

$$f(K^k_{\mathrm{start}}) = \bigvee_{i>1} f(K^k_i)$$

Auf K^k_i folgt deterministisch die Konfiguration K^{k+1}_{start} (vgl. Abbildung 11), sodass

$$f(K^k_{\mathrm{start}}) = 1$$

ist, falls

$$\mathcal{S}_{\mathrm{PA}_\mathbb{Z}} \models \exists x_k \; Q_{k+1} x_{k+1} ... Q_s x_s F(\varrho(x_1), ..., \varrho(x_{k-1}), x_k, x_{k+1}, ..., x_s)$$

gilt. Im zweiten Fall ist

$$\forall x_k \; Q_{k+1} x_{k+1} ... Q_s x_s F(\varrho(x_1), ..., \varrho(x_{k-1}), x_k, x_{k+1}, ..., x_s)$$

gegeben.
Nun wird wie im ersten Fall vorgegangen, jedoch mit dem Unterschied, dass im vorausgegangenen Durchgang in einen Zustand aus $\bigwedge$ gewechselt wurde (Zeile 10). Das bedeutet, dass für alle Belegungen $\varrho(x_k)$

$$\mathcal{S}_{\mathrm{PA}_\mathbb{Z}} \models Q_{k+1} x_{k+1} ... Q_s x_s F(\varrho(x_1), ..., \varrho(x_{k-1}), \varrho(x_k), x_{k+1}, ..., x_s)$$

gelten muss, sodass sich

$$f(K^k_{\text{start}}) = \bigwedge_{i>1} f(K^k_i)$$

ergibt und

$$f(K^k_{\text{start}}) = 1$$

nur dann gilt, wenn

$$\mathcal{S}_{\text{PA}_\mathbb{Z}} \models \forall x_k \; Q_{k+1}x_{k+1}...Q_s x_s F(\varrho(x_1), ..., \varrho(x_{k-1}), x_k, x_{k+1}, ..., x_s)$$

gültig ist. Insgesamt bildet sich ein Baum, wie in Abbildung 11, und die ATM akzeptiert nur dann, wenn die benötigten Konfigurationen akzeptieren, sodass die Wurzel (Das ist K^1_{start} in Abbildung 11) des Baumes akzeptiert. An dieser Stelle stellt sich die Frage nach der Laufzeit dieses Vorgehens.
Zu Beginn wird die Formel in Pränexnormalform gebracht und es werden Informationen aus der Eingabe extrahiert. Das geht beides in Polynomialzeit. Darauf folgt das nichtdeterministische Belegen. Bei diesem werden s Mal Variablen mit Binärzahlen der Länge $n^{3*s+(s+3)^{(m+1)}}$ belegt. Da s und m aber von $\Upsilon^s_m(\text{PA}_\mathbb{Z})$ festgelegt sind, funktioniert das ebenfalls in Polynomialzeit. (Die Länge der resultierenden Berechnungspfade hängt von einem Polynom in n ab.) Danach wird nur noch ausgewertet.
Insgesamt kann Algorithmus 3 die Teilklassen $\Upsilon^s_m(\text{PA}_\mathbb{Z})$ mit einer alternierenden Turingmaschine Σ_m bzw. Π_m in Polynomialzeit entscheiden. Für den Fall, dass das Quantorenprefix mit $\exists$ beginnt, wird Σ_m genutzt, andernfalls wird Π_m verwendet, sodass sich die ATM zu Beginn des Algorithmus 3 in einer Konfiguration mit Zustand aus der richtigen Zustandsmenge befindet. Es ergibt sich:

Satz 20: Inklusion der variablen Teilklassen von $Th(\mathcal{S}_{\text{PA}_\mathbb{Z}})$ in die Polynomialzeithierarchie
Es sei $X \in \{\text{PA}, \text{PA}_\leq, \text{PA}_\mathbb{Z}\}$, dann liegt $\Upsilon^s_m(\text{X})$ auf der m-ten Stufe der Polynomialzeithierarchie.

Beweis. $\Upsilon^s_m(\text{PA}_\mathbb{Z})$ ist die Menge der festen Teilklassen $[Q_1, Q_2, ..., Q_s] \cap Th(\mathcal{S}_{\text{PA}_\mathbb{Z}})$, die mit Algorithmus 3 in Polynomialzeit entschieden werden können. Dann gilt mit Anwendung von Satz 3:

$$[Q_1, Q_2, ..., Q_s] \cap Th(\mathcal{S}_{\text{PA}_\mathbb{Z}}) \in \begin{cases} \Sigma_m\text{-TIME}\left(n^{O(1)}\right) = \Sigma^P_m & \text{falls } Q_1 = \exists \\ \Pi_m\text{-TIME}\left(n^{O(1)}\right) = \Pi^P_m & \text{falls } Q_1 = \forall \end{cases}$$

Mittels Satz 12 lässt sich dieses Resultat auf $\Upsilon^s_m(\text{PA}_\leq)$ und $\Upsilon^s_m(\text{PA})$ erweitern. □

Für PA und $\text{PA}_\leq$ kann das Resultat um eine Stufe der Hierarchie verbessert werden, wenn das Quantorprefix genau festgelegt wird. In [Sca84, LJ83] wird gezeigt, dass der folgende Satz gilt:

Satz 21: Komplexität der Teilklassen $[Q^t] \cap Th(\mathcal{S}_{\text{PA}_\leq})$
Es sei $X \in \{\text{PA}, \text{PA}_\leq\}$ und $t \in \mathbb{N}$ fest gewählt, dann gilt:

$$[\exists^t] \cap Th(\mathcal{S}_{\text{X}}) \in \mathcal{P}$$

$$[\forall^t] \cap Th(\mathcal{S}_{\text{X}}) \in \mathcal{P}$$

Tatsächlich zeigt sich, dass die Menge der wahren Formeln der Presburger Arithmetik mit nur einem Quantorenblock mit festgelegter Länge, die einzige Teilklasse ist, für die ein Entscheidungsalgorithmus gefunden werden konnte, der in Polynomialzeit läuft.
Dann kann der obige Algorithmus erweitert werden und es ergibt sich:

Satz 22: Inklusion der festen Teilklassen von $Th(\mathcal{S}_{\mathbf{PA}})$ und $Th(\mathcal{S}_{\mathbf{PA}_\leq})$ in die Polynomialzeithierarchie vgl. [Grä88]
Es sei $X \in \{\mathrm{PA}, \mathrm{PA}_\leq\}$, dann gilt für $[Q_1, Q_2, ..., Q_s] \cap Th(\mathcal{S}_\mathrm{X})$ mit m Quantorenblöcken und dem Quantorenprefix fest gewählt:

$$[Q_1, Q_2, ..., Q_s] \cap Th(\mathcal{S}_\mathrm{X}) \in \begin{cases} \Sigma_{m-1}\text{-TIME}\left(n^{O(1)}\right) = \Sigma_{m-1}^P & \text{falls } Q_1 = \exists \\ \Pi_{m-1}\text{-TIME}\left(n^{O(1)}\right) = \Pi_{m-1}^P & \text{falls } Q_1 = \forall \end{cases}$$

Beweis. Es sei $s = \sum_{i=1}^m t_i$ ($t_i \in \mathbb{N}$), dann lassen sich zunächst die Quantoren umbenennen.

$$[Q_1^{t_1}, Q_2^{t_2}, ..., Q_m^{t_m}] \cap Th(\mathcal{S}_{\mathrm{PA}_\leq}) \coloneqq [Q_1, Q_2, ..., Q_s] \cap Th(\mathcal{S}_{\mathrm{PA}_\leq})$$

Anschließend kann das Quantorenprefix nach dem $m-1$-ten Quantorenblock, also nach den vorderen $\sum_{i=1}^{m-1} t_i$ Quantoren, geteilt werden. Dies liefert den ersten Teil $Q_1 x_1^1 ... Q_1 x_{t_1}^1 Q_2 x_1^2 ... Q_2 x_{t_2}^2 \cdots Q_{m-1} x_1^{m-1} ... Q_{m-1} x_{t_{m-1}}^{m-1}$ und den zweiten Teil $Q_m x_1^m ... Q_m x_{t_m}^m$.
Die Belegung des ersten Teils wird wie in Algorithmus 3 nichtdeterministisch mit einer ATM ermittelt. Da es sich in diesem Fall jedoch um $\mathcal{S}_{\mathrm{PA}_\leq}$ handelt, müssen nur positive Belegungen getestet werden. Dies wird erreicht, indem die, im Text erwähnte, +1 für die Vorzeichenbetragsdarstellung weggelassen wird. Jedoch wird nur eine Σ_{m-1} bzw. Π_{m-1} ATM benötigt, da das Quantorenprefix einen Quantorenblock weniger besitzt. Diese liefert die zu testende Belegung $\varrho(x_1^1), ..., \varrho(x_{t_{m-1}}^{m-1})$ für

$$Q_m x_1^m ... Q_m x_{t_m}^m F(\varrho(x_1^1), ..., \varrho(x_{t_{m-1}}^{m-1}), x_1^m, ..., x_{t_m}^m) \tag{22}$$

Laut Satz 21 kann dann aber deterministisch in Polynomialzeit entschieden werden, ob (22) in $[Q_m^{t_m}] \cap Th(\mathcal{S}_{\mathrm{PA}_\leq})$ enthalten ist. (Zu beachten ist, dass die Anzahl der letzten t_m Quantoren dafür festgelegt sein muss, was in Satz 20 nicht der Fall war.)
Insgesamt ergibt sich wieder ein Polynomialzeitalgorithmus für eine ATM, die wie in Algorithmus 3 funktioniert, jedoch vor Zeile 15 den Algorithmus aus Satz 21 benutzt, und nur positive Belegungen testet. Dann wird wieder Satz 3 angewendet, der das gesuchte Ergebnis für $\mathcal{S}_{\mathrm{PA}_\leq}$ liefert.
Aufgrund der Polynomialzeitreduktion in Satz 12 lässt sich das Ergebnis auf $\mathcal{S}_\mathrm{PA}$ übertragen (siehe auch Kapitel 5). □

6.2.3 Die Vollständigkeit bestimmter fester Teilklassen von $Th(\mathcal{S}_{\mathbf{PA}})$

In dem vorausgegangenen Kapitel wurde nur die Inklusion in die Polynomialzeithierarchie gezeigt. Jedoch ist es für bestimmte Teilklassen auch möglich zu zeigen, dass die Probleme mindestens so schwer sind, wie alle anderen Probleme der Komplexitätsklasse. Diese Eigenschaft wird Schwere genannt. Zusammen mit der Inklusion heißt die Eigenschaft Vollständigkeit für eine Komplexitätsklasse. Dementsprechend wird ein Problem, das für eine Stufe der Polynomialzeithierarchie vollständig ist, Σ_m^P - vollständig bzw. Π_m^P - vollständig genannt (Für ein bestimmtes $m \geq 0$).

Wie üblich wird auch die Σ_m^P - schwere bzw. die Π_m^P-schwere eines Problems über die Reduktion auf ein anderes Problem gezeigt, welches für Σ_m^P bzw. Π_m^P vollständig ist. Im Fall von festen Teilklassen von $Th(\mathcal{S}_{\mathrm{PA}})$ bietet sich das Problem B_m bzw. $\overline{B_m}$ (vgl. Definition 10) an. Für die Reduktion muss also in Polynomialzeit aus einer quantifizierten aussagenlogischen Formel eine passende Formel in der Presburger Arithmetik gefunden werden. Dafür werden die folgenden aus [Sch97] erschlossenen Zusammenhänge benötigt.
In einer quantifizierten aussagenlogischen Formel gibt es Quantorenblöcke (vgl. Definition 10). Die Belegungen der Variablen $x_1^k, ..., x_{i_k}^k$ des $k-$ten Blockes sollen mit einer Zahl $z_k \in \mathbb{N}$ kodiert werden. Dafür wird eine Funktion ψ definiert, die Belegungen $\{0,1\}^{i_k}$ $(1 \leq k \leq m)$ mit Zahlen $\mathbb{N}$ kodiert.

Definition 30: Kodierung von Belegungen mit ψ
Es sei I_k die Belegungsfunktion von $x_1^k, ..., x_{i_k}^k$ und $p_1, ..., p_{i_k}$ die ersten i_k Primzahlen. Dann sei

$$\psi : Belegung \to \mathbb{N}$$

mit $\psi(I_k) = z_k$ und

$$\begin{aligned} z_k &\equiv I_k(x_1^k) \pmod{p_1} \\ z_k &\equiv I_k(x_2^k) \pmod{p_2} \\ &\vdots \\ z_k &\equiv I_k(x_{i_k}^k) \pmod{p_{i_k}} \end{aligned}$$

die Kodierungsfunktion für Belegungen.

ψ ist korrekt definiert, da Satz 4 in diesem Fall besagt, dass es für das System von Kongruenzen eine Lösung gibt, da $p_1, ..., p_{i_k}$ paarweise teilerfremd sind. Außerdem kann jede Zahl $z_k \in \mathbb{N}$ von ψ nur einmal getroffen werden:

Satz 23: Die Kodierungfunktion ψ ist injektiv
ψ ist injektiv.

Beweis. Angenommen es gäbe zwei verschiedene Belegungen I_1 und I_2, sodass $\psi(I_1) = \psi(I_2)$ ist. Das bedeutet aber, dass $z_1 \equiv z_2 \pmod{p_i}$ für alle $1 \leq i \leq i_k$ gleich ist. Das ist jedoch ein Widerspruch zur Annahme, dass I_1 und I_2 verschieden waren. □

Bei der benötigten Reduktion gibt es aber gar keine freien Variablen, sodass einem das manuelle Übersetzen mittels ψ erspart bleibt. Vielmehr soll die Inverse von ψ verwendet werden, um beliebige natürliche Zahlen in Wahrheitswerte zu übersetzen. Diese Dekodierung soll dann später in Formeln der Presburger Arithmetik verwendet werden.
Für valide Belegungen, also Belegungen, in denen Variablen nur die Werte 0 und 1 zugewiesen bekommen, kann die folgende Dekodierungsfunktion definiert werden:

Satz 24: Die Dekodierungsfunktion ψ^{-1}
Wenn umgekehrt eine Kodierung $z_k \in \mathbb{N}$ einer validen Belegung vorliegt, dann darf fol-

gendermaßen dekodiert werden:

$$(I_k(x_i) \equiv z_k \pmod{p_i})$$

Beweis. Es sei I_k eine Belegung. Anwenden von ψ liefert eine Zahl z_k, sodass

$$z_k \equiv I_k(x_i^k) \pmod{p_i}$$

gilt. Nun wird die i-te Gleichung umgeformt:

$$\Leftrightarrow I_k(x_i^k) \equiv z_k \pmod{p_i}$$

□

Da es sich bei ψ jedoch um eine injektive Funktion und keine bijektive Funktion handelt (vgl. Satz 23 und Beispiel 21), gibt es $z_k \in \mathbb{N}$, die keine valide Belegung repräsentieren.

Beispiel 21: Die Dekodierung durch die Inverse der nicht surjektiven Funktion ψ

Es sei $z_1 = 196$ und $z_2 = 197$. Dann ergibt sich (vgl. umgekehrter Fall Beispiel 6 eine Belegung für z_1, während z_2 keine valide Belegung repräsentiert.

$$196 \pmod{2} \equiv 0$$
$$196 \pmod{3} \equiv 1$$
$$196 \pmod{5} \equiv 1$$
$$196 \pmod{7} \equiv 0$$

$(0, 1, 1, 0)$ ist eine korrekte Belegung für $(x_1, ..., x_4)$. Während z_2 keine valide Belegung repräsentiert, da unter anderem

$$197 \bmod 5 \equiv 2$$

ist.

Die Idee ist nun Satz 6 zu benutzen, um in Formeln das Universum $\mathbb{N}$ auf die Zahlen einzuschränken, zu denen ψ^{-1} eine valide Belegung liefert. Der längste Quantorenblock benötigt die größte Einschränkung auf $\mathbb{N}$, da die meisten Primzahl Kongruenzen genutzt werden. Die Zahl $i_{\max} \geq i_k$ für alle $k \in [1, ..., m]$ ist mindestens so groß, wie der längste Quantorenblock. Dann wird eine Funktion A definiert, die das Universum $\mathbb{N}$ wie benötigt einschränkt:

Definition 31: Die Filterfunktion für Kodierungen, die keine Belegung kodieren
Es sei A mit $A : \mathbb{N} \to \{0, 1\}$ und

$$A(z_k) = \bigwedge_{i=1}^{i_{\max}} (z_k \mod p_i) \in \{0, 1\}$$

die „Filterfunktion“.

Mit A kann jetzt $\mathbb{N}_A \coloneqq \{x \in \mathbb{N} \mid A(x) = 1\}$ aufgestellt werden. $\mathbb{N}_A$ ist das benötigte Universum, sodass die Kodierungsfunktion

$$\psi : Belegung \mapsto \mathbb{N}_A$$

mit geänderter Zielmenge laut Konstruktion von $\mathbb{N}_A$ bijektiv ist. Die Dekodierungsfunktion liefert dementsprechend für alle $z_k \in \mathbb{N}_A$ eine valide Belegung für $x_1^k, ..., x_{i_k}^k$.
Weiter stellt sich heraus, dass die benötigten Funktionen auch in der Presburger Arithmetik ausgedrückt werden können (vgl. Beispiel 4):

Name	Bez.	Formel	Formel in PA
Filterfunktion	$A(z_k)$	$\bigwedge_{i=1}^{i_{\max}}(z_k \mod p_i) \in \{0,1\}$	$\forall u \bigwedge_{i=1}^{i_{\max}} \bigwedge_{j=2}^{p_i-1} \neg(p_i * u + j = z_k)$
Dekodierung	$D_i(z_k)$	$I_k(x_i) \equiv z_k \pmod{p_i}$	$\exists u(p_i * u + I_k(x_i) = z_k)$
¬ Dekodierung	$\overline{D_i(z_k)}$	$I_k(\neg x_i) \equiv z_k \pmod{p_i}$	$\exists u(p_i * u + I_k(\neg x_i) = z_k)$

Abbildung 12: Filtern und Dekodieren in der Presburger Arithmetik

Die Formeln sind äquivalent, jedoch ist die Argumentation mit den normalen Formeln intuitiver, weshalb ich im Folgenden weiterhin mit diesen argumentiere. In den resultierenden Formeln werden jedoch die Ausdrücke in der Presburger Arithmetik verwendet. In späteren Umformungen kann auch $\forall u A^*(z_k, u)$ bzw. $\exists u D^*(z_k, u)$ für $A(z_k)$ bzw. $D(z_k)$ geschrieben werden.
Nun können Sätze aus B_m wie in [Sch97] in Sätze in der Presburger Arithmetik übersetzt werden.

Satz 25: Die Übersetzung von Formeln aus B_m bzw. $\overline{B_m}$
Es sei

$$\phi_0 \coloneqq Q_1 x_1^1 ... Q_1 x_{i_1}^1 \; Q_2 x_1^2 ... Q_2 x_{i_2}^2 \; \cdots \; Q_m x_1^m ... Q_m x_{i_m}^m F(x_1^1, ..., x_{i_m}^m)$$

eine quantifizierte aussagenlogische Formel mit m Quantorenblöcken, $i_k \in \mathbb{N}_{>0}$ für alle $1 \leq k \leq m$ und F ohne weitere Quantoren. Weiter sei

$$\phi_2 = Q_1 z_1 (A(z_1) \rightsquigarrow^1 Q_2 z_2 (A(z_2) \rightsquigarrow^2 \cdots Q_m z_m (A(z_m) \rightsquigarrow^m F'(z_1, z_2, ..., z_m)) \cdots))$$

und F' das Ergebnis der Ersetzung von $\neg x_i^k$ bzw. x_i^k in F' durch $(0 \equiv z_k \pmod{p_i})$ bzw. $(1 \equiv z_k \pmod{p_i})$. Dann gilt:

$$\phi_2 \in Th(\mathcal{S}_{\text{PA}}) \Leftrightarrow \phi_0 \in \begin{cases} B_m & \text{falls } Q_1 = \exists \\ \overline{B_m} & \text{sonst} \end{cases}$$

Beweis. Zunächst wird ϕ_0 in ϕ_1 übersetzt, sodass

$$\phi_0 \in \begin{cases} B_m & \text{falls } Q_1 = \exists \\ \overline{B_m} & \text{sonst} \end{cases} \Leftrightarrow \phi_1 \text{ ist wahr in der Struktur } \mathcal{S}_{\text{PA}} \text{ mit Universum } \mathbb{N}_A \tag{23}$$

gilt.

In ϕ_0 muss für alle $k \in [1, m]$ $Q_k x_1^k ... Q_k x_{i_k}^k$ durch $Q_k z_k$ ersetzt werden, da die Kodierungsfunktion ψ die Belegung eines ganzen Quantorenblocks kodiert. Bei der Übersetzung von F wird die Dekodierungsfunktion $I_k(x_i^k) \equiv z_k \pmod{p_i}$ benutzt, die in der Interpretation mit Universum $\mathbb{N}_A$ uneingeschränkt angewandt werden darf (Satz 24).
Dann kann in F $\neg x_i^k$ und x_i^k übersetzt werden:

$$\begin{aligned}\neg x_i^k &\Leftrightarrow I(x_i^k) = 0 \\ &\Leftrightarrow 0 \equiv z_k \pmod{p_i} \\ &\Leftrightarrow \exists x \; p_i * u + 0 = z_k\end{aligned}$$

$$\begin{aligned}x_i^k &\Leftrightarrow I(x_i^k) = 1 \\ &\Leftrightarrow 1 \equiv z_k \pmod{p_i} \\ &\Leftrightarrow \exists x \; p_i * u + 1 = z_k\end{aligned}$$

Die obige Übersetzung, sowie die Ersetzung der x_i^k liefert:

$$\phi_1 \coloneqq Q_1 z_1 Q_2 z_2 \cdots Q_m z_m \; F'(z_1, z_2, ..., z_m)$$

Da die Zeichen in den quantifizierten aussagenlogischen Formeln mit der gleichen Semantik auch in der Presburger Arithmetik auftreten, kann die Formel ansonsten übernommen werden. Aufgrund der obigen Äquivalenzen, die eine aussagenlogische Variable (atomare Formel in der Aussagenlogik) in eine äquivalente Formel der Presburger Arithmetik übersetzt, gilt dann (23). Für die finale Aussage muss noch

$$\phi_1 \text{ ist wahr in der Struktur } \mathcal{S}_{\text{PA}} \text{ mit Universum } \mathbb{N}_A \Leftrightarrow \mathcal{S}_{\text{PA}} \models \phi_2$$

gelten. Genau diese Äquivalenz liefert Satz 6 für $S = \mathcal{S}_{\text{PA}}$ mit Universum $G = \mathbb{N}$ und $S' = \mathcal{S}_{\text{PA}}$ mit durch die Filterfunktion A eingeschränktem Universum $\mathbb{N}_A$.
Da ϕ_2 ein Satz ist, ergibt sich insgesamt die benötigte Äquivalenz:

$$\phi_2 \in Th(\mathcal{S}_{\text{PA}}) \Leftrightarrow \phi_0 \in \begin{cases} B_m & \text{falls } Q_1 = \exists \\ \overline{B_m} & \text{sonst} \end{cases}$$

□

Beispiel 22: Beispielübersetzung

Es sei $F(x_1^1, x_2^1) \coloneqq (\neg x_1^1 \wedge x_2^1)$ die zu übersetzende Funktion ($F : \{0,1\}^2 \mapsto \{0,1\}$). Es ist $i_{max} = 2$, deshalb werden die ersten beiden Primzahlen 2 und 3 benötigt. Nun wird F zu $F'(z_1) \coloneqq (0 \equiv z_1 \bmod 2) \wedge (1 \equiv z_1 \bmod 3)$ ($F' : \mathbb{N}_A \mapsto \{0,1\}$) mithilfe der Dekodierungsfunktion übersetzt.
Eine Formel $\phi_0 = \exists x_1^1 \; \exists x_2^1 \; F(x_1^1, x_2^1)$ wird dann unter Verwendung von Satz 25 zu: $\exists z_1 (A(z_1) \wedge F'(z_1))$

Der Prozess der Übersetzung ist nicht sonderlich komplex, sodass sie in Polynomialzeit läuft.

Satz 26: Die Übersetzung läuft in Polynomialzeit
Die Übersetzung aus Satz 25 läuft in Polynomialzeit abhängig von $n := |\phi_0|$. (Resultat wird in [Sch97] vorausgesetzt.)

Beweis. Bei der Übersetzung werden grundsätzlich nur Terme konstanter Länge durch andere Terme konstanter Länge ersetzt. Dafür muss ein zu ersetzender Term gefunden und dann ersetzt werden. Das geht in linearer Zeit. Des Weiteren müssen für die Dekodierungsfunktion die ersten $i_{\max} \geq i_j$ Primzahlen generiert werden ($1 \leq j \leq m$). Offensichtlich ist die Länge der Formel n größer gleich der Anzahl der Quantoren in einem Quantorenblock: $n \geq i_{\max}$.
Mit dem AKS-Primzahltest kann in Polynomialzeit geprüft werden, ob es sich bei einer Zahl um eine Primzahl handelt [SS05]. Die Primzahlfunktion $\Pi(x) = \frac{x}{ln(x)}$ liefert die approximierte Anzahl der Primzahlen bis zur Zahl x [CLRS17]. Gesucht ist nun ein x, sodass $\Pi(x) \geq n \geq i_{\max}$ ist. Es sei $x_0 = n^2 * ln(n)$, dann ist:

$$\begin{aligned}
\Pi(x_0) &= \frac{x_0}{ln(x_0)} \\
&= \frac{n^2 * ln(n)}{ln(n^2 * ln(n))} \\
&= \frac{n^2 * ln(n)}{2 * ln(n) + ln(ln(n))} \\
&\geq \frac{n^2 * ln(n)}{2 * ln(n) + ln(n)} && \mid ln(ln(x)) \leq ln(x) \\
&= \frac{n^2 * ln(n)}{3 * ln(n)} \\
&= \frac{n^2}{3} \\
&\geq n
\end{aligned}$$

Da die Anzahl der zu prüfenden Primzahlen x_0 nur ein Polynom von n ist, können die ersten $i_{\max}$ Primzahlen in Polynomialzeit abhängig von n gefunden werden. □

Mithilfe von Satz 25 kann mit der benötigten Reduktion begonnen werden:
Zunächst wird definiert:

$$B = \begin{cases} B_{m-1} & \text{falls } Q_1 = \exists \\ \overline{B_{m-1}} & \text{falls } Q_1 = \forall \end{cases} \qquad G = \begin{cases} \text{3DNF} & \text{falls } m \text{ gerade und } B = \overline{B_{m-1}} \\ \text{3KNF} & \text{falls } m \text{ gerade und } B = B_{m-1} \\ \text{3DNF} & \text{falls } m \text{ ungerade und } B = B_{m-1} \\ \text{3KNF} & \text{falls } m \text{ ungerade und } B = \overline{B_{m-1}} \end{cases}$$

$B \cap G$ ist vollständig für $\begin{cases} \Sigma^P_{m-1} & \text{falls } B = B_{m-1} \\ \Pi^P_{m-1} & \text{falls } B = \overline{B_{m-1}} \end{cases}$ (vgl. Satz 2). Dann lässt sich $B \cap G \leq^P_m [Q_1, ..., Q_s] \cap Th(\mathcal{S}_{\text{PA}})$ reduzieren. Dafür wird Satz 25 benutzt, der in Polynomialzeit zu einer

Formel

$$\phi_B \coloneqq Q_1 x_1^1 ... Q_1 x_{i_1}^1 \; Q_2 x_1^2 ... Q_2 x_{i_2}^2 \; \cdots \; Q_{m-1} x_1^{m-1} ... Q_{m-1} x_{i_{m-1}}^{m-1} F(x_1^1, ..., x_{i_{m-1}}^{m-1})$$

eine Formel

$$\phi_{PA} \coloneqq Q_1 z_1 (A(z_1) \rightsquigarrow^1 Q_2 z_2 (A(z_2) \rightsquigarrow^2 \cdots \rightsquigarrow^{m-3} \\ Q_{m-1} z_{m-1} (A(z_{m-1}) \rightsquigarrow^{m-1} F'(z_1, z_2, ..., z_{m-1})) \cdots))$$

findet, sodass

$$\phi_B \in B \cap G \Leftrightarrow \phi_{PA} \in Th(\mathcal{S}_{\mathrm{PA}})$$

gilt. Übrig bleibt zu zeigen, dass es eine zu ϕ_{PA} äquivalente Formel gibt, die das Quantorenprefix $Q_1, ..., Q_s$ mit m Quantorenblöcken besitzt. Zu diesem Zweck kann ϕ_{PA} auch geschrieben werden als:

$$\phi_{PA} \coloneqq Q_1 z_1 ((\forall u \; A^*(z_1, u)) \rightsquigarrow^1 Q_2 z_2 ((\forall u \; A^*(z_2, u)) \rightsquigarrow^2 \cdots \rightsquigarrow^{m-3} \\ Q_{m-1} z_{m-1} ((\forall u \; A^*(z_{m-1}, u)) \rightsquigarrow^{m-1} F'(z_1, z_2, ..., z_{m-1})$$

Dann können die Quantoren zusammengefasst werden. Dazu wird zunächst der Fall mit $Q_k = \exists$ und dementsprechend $Q_{k+1} = \forall$ und $\rightsquigarrow^k = \wedge$ betrachtet.

$$\begin{aligned} &\exists z_k ((\forall u \; A^*(z_k, u)) \wedge \forall z_{k+1}(...)) \\ \Leftrightarrow &\exists z_k (\forall z_{k+1} \; (A^*(z_k, z_{k+1}) \wedge (...))) \end{aligned}$$

Auch im Fall $Q_{k+1} = \forall$, in dem $Q_{k+1} = \exists$ und $\rightsquigarrow^k = \rightarrow$ ist, kann zusammengefasst werden:

$$\begin{aligned} &\forall z_k ((\forall u \; A^*(z_k, u)) \rightarrow \exists z_{k+1}(...)) \\ \Leftrightarrow &\forall z_k (\exists u \; (A^*(z_k, u) \rightarrow \exists z_{k+1}(...))) \\ \Leftrightarrow &\forall z_k (\exists z_{k+1} \; (A^*(z_k, z_{k+1}) \rightarrow (...))) \end{aligned}$$

Unabhängig von Q_k ergibt sich:

$$\begin{aligned} &Q_k z_k ((\forall u \; A^*(z_k, u)) \rightsquigarrow^k Q_{k+1} z_{k+1}(...)) \\ \Leftrightarrow \; &Q_k z_k (Q_{k+1} z_{k+1} (A^*(z_k, z_{k+1}) \rightsquigarrow^k (...))) \end{aligned}$$

Durch wiederholte Anwendung obiger Zusammenfassung von Quantoren ergibt sich:

$$\phi_{PA} \Leftrightarrow \; Q_1 z_1 (Q_2 z_2 \; (A^*(z_1, z_2) \rightsquigarrow^1 (Q_3 z_3 (A^*(z_2, z_3) \rightsquigarrow^2 \cdots \rightsquigarrow^{m-3} \\ (Q_{m-1} z_{m-1} (A^*(z_{m-2}, z_{m-1}) \rightsquigarrow^{m-2} ((\forall u \; A^*(z_{m-1}, u)) \rightsquigarrow^{m-1} F'(z_1, z_2, ..., z_{m-1})))) \cdots))))$$

Schließlich können die Quantoren Q_i für $1 \leq i \leq m-1$ vor die Formel gezogen werden:

$$\phi_{PA} \Leftrightarrow \; Q_1 z_1 \; Q_2 z_2 \; Q_3 z_3 \; \cdots \; Q_{m-1} z_{m-1} (A^*(z_1, z_2) \rightsquigarrow^1 (A^*(z_2, z_3) \rightsquigarrow^2 \cdots \rightsquigarrow^{m-3} \\ (A^*(z_{m-2}, z_{m-1}) \rightsquigarrow^{m-2} ((\forall u \; A^*(z_{m-1}, u)) \rightsquigarrow^{m-1} F'(z_1, z_2, ..., z_{m-1})))) \cdots)))$$

Da es sich bei F um eine Funktion in G handelte, ist auch F' dieser Form mit Atomen $D_i(z_k)$.

Im Fall $G = 3\text{KNF}$ können die einzelne Konjunkte $v_1 \vee v_2 \vee v_3$ umgeformt werden:

$$\begin{aligned} v_1 \vee v_2 \vee v_3 &\Leftrightarrow \neg\neg(v_1 \vee v_2 \vee v_3) \\ &\Leftrightarrow \neg(\neg v_1 \wedge \neg v_2 \wedge \neg v_3) \end{aligned}$$

Im Rahmen der Dekodierung bedeutet die Negierung einer Variable v, dass sich die Belegung von $I(v)$ in der Dekodierungsfunktion von 0 zu 1 oder umgekehrt ändert, sodass $\overline{D(z_k)}$ benutzt wird. Insgesamt ergibt sich nach diesem Schritt ein F', welches wie folgt aufgebaut ist:

$$F' = \begin{cases} \bigvee \left(\bigwedge_{j=1}^{3} (\exists u \ D_i^*(u, z_k)) \right) & \text{falls } G = 3\text{DNF} \\ \bigwedge \neg \left(\bigwedge_{j=1}^{3} (\exists u \ \overline{D_i^*(u, z_k)}) \right) & \text{falls } G = 3\text{KNF} \end{cases}$$

In $D_i^*(u, z_k)$ steht i und k an dieser Stelle und in den folgenden Vorkommen für die Werte des ersetzten x_i^k.
Die Reduktion läuft bis zu diesem Punkt in Polynomialzeit, da Satz 25 in Polynomialzeit funktioniert, und alle folgenden Umformungen maximal lineare Zeit benötigen.
Aufbauend auf dieser Reduktion können nun die zwei folgenden Sätze bewiesen werden. Zunächst kann gezeigt werden, dass schon das Problem zu entscheiden, ob ein Satz der Presburger Arithmetik mit nur zwei verschiedenen Quantoren wahr ist, vollständig für $\mathcal{NP}$ bzw. $co\mathcal{NP}$ ist.

Satz 27: $\mathcal{NP}$ bzw $co\mathcal{NP}$ -vollständigkeit für Feste Teilklassen von $Th(\mathcal{S}_{\mathbf{PA}})$

1. $[\forall, \exists] \cap Th(\mathcal{S}_{\text{PA}})$ ist vollständig für die Klasse $co\mathcal{NP}$ bzw. Π_1^P.
2. $[\exists, \forall] \cap Th(\mathcal{S}_{\text{PA}})$ ist vollständig für die Klasse $\mathcal{NP}$ bzw. Σ_1^P

(Das Resultat und das Vorgehen im Beweis aus [Sch97])

Beweis. Laut Satz 22 ist sowohl $[\exists, \forall] \cap Th(\mathcal{S}_{\text{PA}})$ als auch $[\forall, \exists] \cap Th(\mathcal{S}_{\text{PA}})$ in Σ_1^P bzw. Π_1^P, da es sich um ein festes Quantorenprefix mit zwei Quantorenblöcken handelt.
Nun wird obige Reduktion für $m = 2$ fortgeführt. Dann ist:

$$\phi_{PA} \Leftrightarrow Q_1 z_1((\forall u \ A^*(z_1, u)) \rightsquigarrow^1 F'(z_1))$$

Nun kann jedes $\bigwedge_{j=1}^{3} \exists u \ D_i^*(u, z_1)$ bzw. $\bigwedge_{j=1}^{3} (\exists u \ \overline{D_i^*(u, z_1)})$ von F' mit dem chinesischen Restsatz zusammengefasst werden ($j_o = i$ der jeweiligen Iteration für $1 \leq o \leq 3$). Für das System

$$\begin{aligned} a &\equiv I(x_{j_1}^1) \pmod{p_{j_1}} \\ a &\equiv I(x_{j_2}^1) \pmod{p_{j_2}} \\ a &\equiv I(x_{j_3}^1) \pmod{p_{j_3}} \end{aligned}$$

simultaner Kongruenzen gibt es laut Satz 4 eine Lösung $a \mod p_{j_1} * p_{j_2} * p_{j_3}$, da p_{j_1}, p_{j_2} und p_{j_3} als Primzahlen paarweise teilerfremd sind. Dann muss für jedes Disjunkt nur noch getestet werden, ob $z_1 \equiv a \mod p_{j_1} * p_{j_2} * p_{j_3}$ wahr ist. Das kann wiederum mithilfe der Presburger Arithmetik ausdrückt werden:

$$\exists u \ p_{j_1} * p_{j_2} * p_{j_3} * u + a = z_1$$

Durch Einsetzen und Umformen ergibt sich eine zu ϕ_{PA} äquivalente Formel, die das jeweils benötigte Quantorenprefix besitzt.
Im Fall $[\forall, \exists] \cap Th(\mathcal{S}_{\text{PA}})$ ergibt sich:

$$\begin{aligned}
\phi_{PA} &\Leftrightarrow \forall z_1((\forall u\ A^*(z_1,u)) \to F'(z_1)) \\
&\Leftrightarrow \forall z_1 \left((\forall u\ A^*(z_1,u)) \to \bigvee \left(\bigwedge_{j=1}^{3} (\exists u\ D_i^*(u,z_k)) \right) \right) \\
&\Leftrightarrow \forall z_1((\forall u\ A^*(z_1,u)) \to \bigvee \exists u\ p_{j_1} * p_{j_2} * p_{j_3} * u + a = z_1) \\
&\Leftrightarrow \forall z_1(\exists u\ (A^*(z_1,u) \to \exists u \bigvee p_{j_1} * p_{j_2} * p_{j_3} * u + a = z_1)) \\
&\Leftrightarrow \forall z_1 \exists u\ ((A^*(z_1,u) \to \bigvee p_{j_1} * p_{j_2} * p_{j_3} * u + a = z_1))
\end{aligned}$$

Die Reduktion läuft in Polynomialzeit, da der erste Teil der Reduktion in Polynomialzeit lief, und die Anwendung des chinesischen Restsatzes ebenfalls in Polynomialzeit läuft. Danach musste nur noch ersetzt und umgeformt werden. Das geht in linearer Zeit.
Somit folgt aus der Reduktion von $B_1 \cap$ 3KNF auf $[\forall, \exists] \cap Th(\mathcal{S}_{\text{PA}})$, dass $[\forall, \exists] \cap Th(\mathcal{S}_{\text{PA}})$ Π_1^P - schwer und mit obiger Inklusion auch Π_1^P-vollständig ist.
Im umgekehrten Fall $[\exists, \forall] \cap Th(\mathcal{S}_{\text{PA}})$ ergibt sich mit gleicher Argumentation die Vollständigkeit für Σ_1^P:

$$\begin{aligned}
\phi_{PA} &\Leftrightarrow \exists z_1((\forall u\ A^*(z_1,u)) \wedge F'(z_1)) \\
&\Leftrightarrow \exists z_1 \left(\forall u\ A^*(z_1,u) \wedge \bigwedge \neg \left(\bigwedge_{j=1}^{3} (\exists u\ \overline{D_i^*(u,z_k)}) \right) \right) \\
&\Leftrightarrow \exists z_1 \left(\forall u\ A^*(z_1,u) \wedge \bigwedge \neg (\exists u\ p_{j_1} * p_{j_2} * p_{j_3} * u + a = z_1) \right) \\
&\Leftrightarrow \exists z_1 \left(\forall u\ A^*(z_1,u) \wedge \bigwedge \forall u\ \neg (p_{j_1} * p_{j_2} * p_{j_3} * u + a = z_1) \right) \\
&\Leftrightarrow \exists z_1 \forall u\ \left(A^*(z_1,u) \wedge \bigwedge \neg (p_{j_1} * p_{j_2} * p_{j_3} * u + a = z_1) \right)
\end{aligned}$$

□

Auch für eine beliebige Anzahl m an Quantorenblöcken kann für bestimmte feste Teilklassen aus $Th(\mathcal{S}_{\text{PA}})$ die Vollständigkeit für Stufen der Polynomialzeithierarchie gefolgert werden:

Satz 28: Die Vollständigkeit bestimmter fester Teilklassen aus $Th(\mathcal{S}_{\text{PA}})$ für Stufen der Polynomialzeithierarchie
$[Q_1, Q_2, ..., Q_{m-1}, Q_m, Q_m, Q_m] \cap Th(\mathcal{S}_{\text{PA}})$ mit m Quantorenblöcken ist
$\begin{cases} \sum_{m-1}^P\text{-vollständig} & \text{falls } Q_1 = \exists \\ \Pi_{m-1}^P\text{-vollständig} & \text{falls } Q_1 = \forall \end{cases}$
(Resultat und Vorgehen im Beweis aus [Sch97])

Beweis. Zunächst liefert Satz 22, dass $[Q_1, ..., Q_s] \cap Th(\mathcal{S}_{\text{PA}})$ mit $m \geq 2$ Quantorenblöcken in Σ_{m-1}^P falls $Q_1 = \exists$ oder in Π_{m-1}^P falls $Q_1 = \forall$ ist, da das Quantorenprefix $Q_1, ..., Q_s$, m Quantorenblöcke enthält und vollständig festgelegt ist.

Nun wird die obige Reduktion für ein variables m weiter fortgeführt. Aufbauend kann das $\exists u$ aus F' über das $\bigwedge$ gezogen werden, indem u zu u_j umbenannt wird.

$$F' = \begin{cases} \bigvee \exists u_1\ \exists u_2\ \exists u_3\ \left(\bigwedge_{j=1}^{3}(D_i^*(u_j, z_k))\right) & \text{falls } G = \text{3DNF} \\ \bigwedge \neg\exists u_1\ \exists u_2\ \exists u_3\ \left(\bigwedge_{j=1}^{3}(\overline{D_i^*(u_j, z_k)})\right) & \text{falls } G = \text{3KNF} \end{cases}$$

Dann muss aufgrund der Form von F' unterschieden werden:

1. In dem Fall, dass F' in 3KNF ist und $Q_{m-1} = \exists$ gilt:

$$\begin{aligned} (\forall u\ A^*(z_{m-1}, u)) &\rightsquigarrow^{m-1} F'(z_1, z_2, ..., z_{m-1}) \\ &\Leftrightarrow \forall u\ A^*(z_{m-1}, u) \wedge F'(z_1, z_2, ..., z_{m-1}) \\ &\Leftrightarrow \forall u\ A^*(z_{m-1}, u) \wedge \bigwedge \neg\exists u_1\ \exists u_2\ \exists u_3\ \left(\bigwedge_{j=1}^{3}(\overline{D_i^*(u_j, z_k)})\right) \\ &\Leftrightarrow \forall u\ A^*(z_{m-1}, u) \wedge \bigwedge \forall u_1\ \forall u_2\ \forall u_3\ \left(\bigvee_{j=1}^{3}(\neg\overline{D_i^*(u_j, z_k)})\right) \\ &\Leftrightarrow \forall u_1\ \forall u_2\ \forall u_3\ A^*(z_{m-1}, u_1) \wedge \bigwedge \left(\bigvee_{j=1}^{3}(\neg\overline{D_i^*(u_j, z_k)})\right) \end{aligned}$$

2. In dem Fall, dass F' in 3DNF ist und $Q_{m-1} = \forall$ gilt:

$$\begin{aligned} (\forall u\ A^*(z_{m-1}, u)) &\rightsquigarrow^{m-1} F'(z_1, z_2, ..., z_{m-1}) \\ &\Leftrightarrow (\forall u\ A^*(z_{m-1}, u)) \rightarrow F'(z_1, z_2, ..., z_{m-1}) \\ &\Leftrightarrow (\forall u\ A^*(z_{m-1}, u)) \rightarrow \bigvee \exists u_1\ \exists u_2\ \exists u_3\ \left(\bigwedge_{j=1}^{3}(D_i^*(u_j, z_k))\right) \\ &\Leftrightarrow \exists u\ \left(A^*(z_{m-1}, u) \rightarrow \bigvee \exists u_1\ \exists u_2\ \exists u_3\ \left(\bigwedge_{j=1}^{3}(D_i^*(u_j, z_k))\right)\right) \\ &\Leftrightarrow \exists u_1\ \exists u_2\ \exists u_3\ \left(A^*(z_{m-1}, u_1) \rightarrow \bigvee \left(\bigwedge_{j=1}^{3}(D_i^*(u_j, z_k))\right)\right) \end{aligned}$$

Die Ergebnisse der vorangehenden Umformungen können wie folgt eingesetzt werden. Außerdem können die übrigen Quantoren nach vorne gezogen werden. Dabei wird u und u_1 zu einer gebundenen Variable.

1. m gerade und $Q_1 = \exists$

$$\begin{aligned} \phi_{PA} \Leftrightarrow \exists z_1\ \cdots\ \exists z_{m-1}(A^*(z_1, z_2) \wedge (A^*(z_2, z_3) \rightarrow \cdots \rightarrow \\ (\forall u\ A^*(z_{m-1}, u) \wedge F'(z_1, z_2, ..., z_{m-1})) \cdots)) \\ \Leftrightarrow \exists z_1\ \cdots\ \exists z_{m-1} \forall u_1\ \forall u_2\ \forall u_3\ \Bigg(A^*(z_1, z_2) \wedge \cdots \rightarrow \\ \left(A^*(z_{m-1}, u_1) \wedge \bigwedge\left(\bigvee_{j=1}^{3}(\neg\overline{D_i^*(u_j, z_k)})\right)\right) \cdots\Bigg) \end{aligned}$$

2. m ungerade und $Q_1 = \exists$

$$\begin{aligned}\phi_{PA} \Leftrightarrow\ & \exists z_1\ \cdots\ \forall z_{m-1}(A^*(z_1,z_2) \wedge (A^*(z_2,z_3) \rightarrow \cdots \wedge \\ & \qquad ((\forall u\ A^*(z_{m-1},u)) \rightarrow F'(z_1,z_2,...,z_{m-1}))\cdots)) \\ \Leftrightarrow\ & \exists z_1\ \cdots\ \forall z_{m-1} \exists u_1\ \exists u_2\ \exists u_3 \left(A^*(z_1,z_2) \wedge \cdots \wedge \right. \\ & \qquad \left. \left(A^*(z_{m-1},u_1) \rightarrow \bigvee \left(\bigwedge_{j=1}^{3} (D_i^*(u_j,z_k)) \right) \right) \cdots \right)\end{aligned}$$

3. m gerade und $Q_1 = \forall$

$$\begin{aligned}\phi_{PA} \Leftrightarrow\ & \forall z_1\ \cdots\ \forall z_{m-1}(A^*(z_1,z_2) \wedge (A^*(z_2,z_3) \rightarrow \cdots \wedge \\ & \qquad ((\forall u\ A^*(z_{m-1},u)) \rightarrow F'(z_1,z_2,...,z_{m-1}))\cdots)) \\ \Leftrightarrow\ & \forall z_1\ \cdots\ \forall z_{m-1} \exists u_1\ \exists u_2\ \exists u_3 \left(A^*(z_1,z_2) \wedge \cdots \wedge \right. \\ & \qquad \left. \left(A^*(z_{m-1},u_1) \rightarrow \bigvee \left(\bigwedge_{j=1}^{3} (D_i^*(u_j,z_k)) \right) \right) \cdots \right)\end{aligned}$$

4. m ungerade und $Q_1 = \forall$

$$\begin{aligned}\phi_{PA} \Leftrightarrow\ & \forall z_1\ \cdots\ \exists z_{m-1}(A^*(z_1,z_2) \wedge (A^*(z_2,z_3) \rightarrow \cdots \rightarrow \\ & \qquad (\forall u\ A^*(z_{m-1},u) \wedge F'(z_1,z_2,...,z_{m-1}))\cdots)) \\ \Leftrightarrow\ & \forall z_1\ \cdots\ \exists z_{m-1} \forall u_1\ \forall u_2\ \forall u_3 \left(A^*(z_1,z_2) \wedge \cdots \rightarrow \right. \\ & \qquad \left. \left(A^*(z_{m-1},u_1) \wedge \bigwedge \left(\bigvee_{j=1}^{3} (\neg\overline{D_i^*(u_j,z_k)}) \right) \right) \cdots \right)\end{aligned}$$

In jedem der möglichen Fälle besitzt die Formel das benötigte Quantorenprefix und ist äquivalent zu ϕ_{PA}. Die Reduktion läuft in Polynomialzeit, da der erste Teil der Reduktion in Polynomialzeit lief, und das weitere Ersetzen und Umformen in linearer Zeit funktioniert.
Somit folgt aus der Reduktion von $B \cap G$ auf $[Q_1,Q_2,...,Q_{m-1},Q_m,Q_m,Q_m] \cap Th(\mathcal{S}_{\mathrm{PA}})$ mit m Quantorenblöcken, dass $[Q_1,Q_2,...,Q_{m-1},Q_m,Q_m,Q_m] \cap Th(\mathcal{S}_{\mathrm{PA}})$ $\begin{cases}\sum_{m-1}^{P}\text{-schwer} & \text{falls } Q_1 = \exists \\ \Pi_{m-1}^{P}\text{-schwer} & \text{falls } Q_1 = \forall\end{cases}$ ist und mit obiger Inklusion auch vollständig für die jeweilige Stufe der Polynomialzeithierarchie ist. □

6.2.4 Weitere Vollständigkeitsresultate für feste Teileklassen von $Th(\mathcal{S}_{\mathbf{PA}})$ sowie für $Th(\mathcal{S}_{\mathbf{PA}_{\leq}})$

In Satz 27 und 28 wird die Vollständigkeit für feste Teilklassen aus $Th(\mathcal{S}_{\mathrm{PA}})$ gezeigt und in Satz 22 kam es nicht auf die Dimension des Quantorprefixes an, sondern nur auf die Anzahl der Quantorenblöcke. Daher liegt die Idee nah, die bisherigen Resultate auf weitere feste Teilklassen mit längerem Quantorprefix auszudehnen.
Dafür wird die folgende Reduktion benutzt:

Satz 29: Reduktion für weitere Vollständigkeitsresultate von festen Teilklassen
$[Q_1, Q_2, ..., Q_{s-1}, Q_s] \cap Th(\mathcal{S}_{\text{PA}})$ mit m Quantorenblöcken, lässt sich in Polynomialzeit auf $[Q_1^{t_1}, Q_2^{t_2}, ..., Q_{s-1}^{t_{s-1}}, Q_s^{t_s}] \cap Th(\mathcal{S}_{\text{PA}})$ mit m Quantorenblöcken reduzieren ($t_i \geq 1$ für alle $i \in [1, s]$)

Beweis. Gesucht ist ein Polynomialzeitalgorithmus f, der Formeln aus $\mathcal{L}_{\text{PA}}$ übersetzt, sodass gilt:

$$\phi \in [Q_1, Q_2, ..., Q_{s-1}, Q_s] \cap Th(\mathcal{S}_{\text{PA}}) \coloneqq A \Leftrightarrow f(\phi) \in [Q_1^{t_1}, Q_2^{t_2}, ..., Q_{s-1}^{t_{s-1}}, Q_s^{t_s}] \cap Th(\mathcal{S}_{\text{PA}}) \coloneqq B$$

Dazu nimmt f in $\phi \in \mathcal{L}_{\text{PA}}$ für jedes $i \in [1, s]$ die folgenden Ersetzungen vor:

1. $Q_i x_i$ wird durch $Q_i x_i^1 \cdots Q_i x_i^{t_i}$ ersetzt
2. x_i wird durch $(\sum_{j=1}^{t_i} x_i^j)$ ersetzt

Dann wird aus einer Formel ϕ der Form

$$Q_1 x_1 \cdots Q_s x_s F(x_1, ..., x_s)$$

eine Formel $f(\phi)$ der Form

$$Q_1 x_1^1 ... Q_1 x_1^{t_1} \cdots Q_s x_s^1 ... Q_s x_s^{t_s} F(\sum_{j=1}^{t_1} x_1^j, ..., \sum_{j=1}^{t_s} x_s^j)$$

sodass jetzt gilt

$$\mathcal{S}_{\text{PA}} \models \phi \Leftrightarrow \mathcal{S}_{\text{PA}} \models f(\phi)$$

Dies lässt sich mithilfe einer Fallunterscheidung erkennen. Im Fall $Q_i = \exists$ gibt es für jede Belegung des x_i passende Belegungen für $x_i^1, ..., x_i^{t_i}$ und umgekehrt, sodass $x_i = \sum_{j=1}^{t_i} x_i^j$ gilt. Im Fall $Q_i = \forall$ gibt es für jede Kombination von $x_i^1, ..., x_i^{t_i}$ ein x_i und umgekehrt, sodass ebenfalls $x_i = \sum_{j=1}^{t_i} x_i^j$ gilt. Dabei wird jedes x_i getroffen.
Das ist die Polynomialzeitreduktion von A auf B mittels f. □

Die Schwere der Teilklassen überträgt sich, mittels der Transitivität der Reduktion noch auf weitere feste Teilklassen:

Satz 30: Weitere Vollständigkeitsresultate für feste Teilklassen von $Th(\mathcal{S}_{\mathbf{PA}})$ und $Th(\mathcal{S}_{\mathbf{PA}_{\leq}})$
Es seien $t_i \geq 1$ Zahlen aus $\mathbb{N}$ für jedes $i \in [1, ..., m]$ und $X \in \{\text{PA}, \text{PA}_{\leq}\}$. Dann ist $[Q_1^{t_1}, Q_2^{t_2}, ..., Q_{m-1}^{t_{m-1}}, Q_m^{t_m}, Q_m, Q_m] \cap Th(\mathcal{S}_{\text{X}})$ mit m Quantorenblöcken $\begin{cases} \sum_{m-1}^P\text{-vollständig} & \text{falls } Q_1 = \exists \\ \Pi_{m-1}^P\text{-vollständig} & \text{falls } Q_1 = \forall \end{cases}$.
(Satzidee aus [Grä88]; als Erweiterung des Resultats aus [Sch97])

Beweis. Zunächst besteht das Quantorenprefix $Q_1^{t_1}, Q_2^{t_2}, ..., Q_{m-1}^{t_{m-1}}, Q_m^{t_m}, Q_m, Q_m$ aus m Quantorenblöcken, sodass für $[Q_1^{t_1}, Q_2^{t_2}, ..., Q_{m-1}^{t_{m-1}}, Q_m^{t_m}, Q_m, Q_m] \cap Th(\mathcal{S}_{\mathrm{X}})$ mit Satz 22 weiterhin gilt:

$$[Q_1^{t_1}, Q_2^{t_2}, ..., Q_{m-1}^{t_{m-1}}, Q_m^{t_m}, Q_m, Q_m] \cap Th(\mathcal{S}_{\mathrm{X}}) \in \begin{cases} \Sigma_{m-1}^P & \text{falls } Q_1 = \exists \\ \Pi_{m-1}^P & \text{falls } Q_1 = \forall \end{cases}$$

Die Schwere überträgt sich mittels der Transitivität der $\leq_m^P$ Relationen, sodass zunächst aus Satz 28 und der Reduktion in Satz 29 folgt

$$[Q_1, Q_2, ..., Q_{m-1}, Q_m, Q_m, Q_m] \cap Th(\mathcal{S}_{\mathrm{PA}}) \leq_m^P [Q_1^{t_1}, Q_2^{t_2}, ..., Q_{m-1}^{t_{m-1}}, Q_m^{t_m}, Q_m, Q_m] \cap Th(\mathcal{S}_{\mathrm{PA}})$$

sodass

$$[Q_1^{t_1}, Q_2^{t_2}, ..., Q_{m-1}^{t_{m-1}}, Q_m^{t_m}, Q_m, Q_m] \cap Th(\mathcal{S}_{\mathrm{PA}}) \begin{cases} \Sigma_{m-1}^P\text{-schwer} & \text{falls } Q_1 = \exists \\ \Pi_{m-1}^P\text{-schwer} & \text{falls } Q_1 = \forall \end{cases}$$

ist.
Auch für die festen Teilkassen mit dem „$\leq$“ Zeichen erweiterten Struktur der Presburger Arithmetik $\mathrm{PA}_{\leq}$ folgt die Schwere mit Satz 12:

$$[Q_1^{t_1}, Q_2^{t_2}, ..., Q_{m-1}^{t_{m-1}}, Q_m^{t_m}, Q_m, Q_m] \cap Th(\mathcal{S}_{\mathrm{PA}}) \leq_m^P [Q_1^{t_1}, Q_2^{t_2}, ..., Q_{m-1}^{t_{m-1}}, Q_m^{t_m}, Q_m, Q_m] \cap Th(\mathcal{S}_{\mathrm{PA}_{\leq}})$$

□

Mit analoger Argumentation folgt auch der folgende Satz:

Satz 31: Weitere $\mathcal{NP}$ bzw. $co\mathcal{NP}$ -vollständigkeitsresultate für feste Teilklassen von $Th(\mathcal{S}_{\mathbf{PA}})$ und $Th(\mathcal{S}_{\mathbf{PA}_{\leq}})$
Es seien $t_1, t_2 \geq 1$ Zahlen aus $\mathbb{N}$ und $X \in \{\mathrm{PA}, \mathrm{PA}_{\leq}\}$.

1. $[\forall^{t_1}, \exists^{t_2}] \cap Th(\mathcal{S}_{\mathrm{X}})$ ist vollständig für die Klasse $co\mathcal{NP}$ bzw. Π_1^P.
2. $[\exists^{t_1}, \forall^{t_2}] \cap Th(\mathcal{S}_{\mathrm{X}})$ ist vollständig für die Klasse $\mathcal{NP}$ bzw. Σ_1^P

(Satzidee aus [Grä88]; als Erweiterung des Resultats aus [Sch97])

7 Fazit und Ausblick

Die Sätze der Presburger Arithmetik sind entscheidbar. Jedoch zeigt sich, dass wenn keine Einschränkungen am Aufbau eines Satzes vorgenommen werden, das Problem als Ganzes betrachtet, nicht effizient entschieden werden kann. Durch das Aufteilen des Problems in Teilklassen abhängig von dem Quantorenprefix, ist es jedoch möglich, diese dann eingeschränkte Probleme in die Polynomialzeithierarchie einzuordnen. So konnten sowohl für die festen als auch die variablen Teilklassen obere Schranken für die Laufzeit gefunden werden. Während die variablen Teilklassen mit m Quantorenblöcken mit einer ATM mit m Zustandsmengenwechsel entschieden werden konnten und damit auf der m-ten Stufe der Polynomialzeithierarchie einzuordnen sind, können die festen Teilklassen aus Definition 20 mit m Quantorenblöcken eine Stufe niedriger eingeordnet werden. Für viele Teilkassen gilt auch, dass sie für diese Stufe der Polynomialzeithierachie schwer und damit vollständig für die Stufe der Polynomialzeithierarchie sind. Ein Beispiel dafür ist Satz 27 und deren Erweiterung auf alle Sätze mit zwei Quantorenblöcken in Satz 31. Die Schwere für $m > 2$ Quantorenblöcke konnte nur dann gezeigt werden, wenn der letzte Quantor mindestens drei Mal vorkommt (vgl. Satz 30). Es lässt sich jedoch vermuten, dass für jede feste Teilklasse mit m Quantorenblöcke auch gilt, dass das zugehörige Entscheidungsproblem für die $m-1$ Stufe der Polynomialzeithierarchie vollständig ist [Haa18].

Während die Frage, ob $\mathcal{P} = \mathcal{NP}$ gilt, ungelöst bleibt, steht nicht fest, ob die festen Teilklassen der Presburger Arithmetik nicht doch effizient gelöst werden können. Im Fall $\mathcal{P} = \mathcal{NP}$ kollabiert die Polynomialzeithierarchie und jede feste Teilklasse der Presburger Arithmetik könnte in Polynomialzeit entschieden werden. Jedoch wird vermutet, dass die Polynomialzeithierarchie eine echte Hierarchie ist, sodass $\Sigma_m^P \subsetneq \Sigma_{m+1}^P$ für alle $m \geq 0$ gilt [AB09]. In diesem Fall wären alle festen Teilklassen aus Definition 20 mit mehr als einem Quantorenblock nicht effizient lösbar, also nicht in $\mathcal{P}$.

Schließlich lassen sich die zusammengefassten komplexitätstheoretischen Resultate dieser Arbeit in einer Tabelle (Abbildung 13) zusammenfassen:

Struktur	TK	QB	Inklusion	Schwere	Satz	$\in \mathcal{P}$?
PA,PA$_{\leq}$,PA$_{\mathbb{Z}}$	-	-	-	TIME $\left(2^{O(n)}\right)$	10,11	nein
PA,PA$_{\leq}$,PA$_{\mathbb{Z}}$	variable	m	Σ_m^P bzw. Π_m^P	-	20	falls $\mathcal{P} = \mathcal{NP}$
PA,PA$_{\leq}$	feste	2	$\mathcal{NP}$ bzw. $co\mathcal{NP}$	$\mathcal{NP}$ bzw. $co\mathcal{NP}$	22,31	falls $\mathcal{P} = \mathcal{NP}$
PA,PA$_{\leq}$	feste $\star$	m	Σ_{m-1}^P bzw. Π_{m-1}^P	Σ_{m-1}^P bzw. Π_{m-1}^P	22,30	falls $\mathcal{P} = \mathcal{NP}$

Abbildung 13: Komplexitätsresultate: TK := Teilklassen; QB := Quantorenblöcke; $\star$:= Der letzte Quantorenblock besteht aus mindestens 3 Quantoren

Alle aufgeführten Resultate gelten für Sätze beliebiger Länge, d.h. es wird nur das Quantorenprefix eingeschränkt. Jedoch zeigt ein Resultat von Nguyen und Pak, dass durch Einschränkung der Anzahl der Ungleichungen und der Anzahl der Variablen die Komplexität um eine weitere Stufe der Polynomialzeithierarchie reduziert werden kann [NP17]. Dieses Resultat bezieht sich aber nur auf ein Quantorenprefix mit mindestens 3 Quantorenblöcken, sodass sich aus der zugehörigen Polynomialzeitreduktion auch kein weiterer Algorithmus ergibt, der kurze Formeln in Polynomialzeit entscheiden könnte. Auch im diesen Fall ist das nur im Fall von $\mathcal{P} = \mathcal{NP}$ möglich.

Für den Fall von $\mathcal{P} \neq \mathcal{NP}$ kann demzufolge weiterhin nur die Klasse von festen Teilklassen mit nur einem Quantorenblock in Polynomialzeit entschieden werden (Satz 21).
Im Bezug auf die Einleitung könnten also Datenbankanfragen oder Integritätsbedingungen, die im Tupelkalkül mit nur einem Quantorenblock ausgedrückt werden können und nur Symbole der Presburger Arithmetik benutzen, in Datenbanksystemen in Polynomialzeit auf mögliche Erfüllbarkeit getestet werden (wie beispielsweise die Anfrage in Beispiel 1). In komplexeren Fällen (wie beispielsweise die Integritätsbedingung in Beispiel 1) ist ein effizientes Testen jedoch nur im Fall von $\mathcal{P} = \mathcal{NP}$ möglich.

8 Anhang

8.1 Übliche Definition der Polynomialzeithierarchie

Wie erwähnt, wird die Polynomialzeithierarchie üblicherweise mithilfe von Orakel Turingmaschinen definiert. Da diese Definition jedoch keinen Mehrwert im Zusammenhang mit der Komplexität der Presburger Arithmetik bietet, gebe ich die Definition nun an dieser Stelle an:

Definition 32: Orakel Turingmaschinen vgl. [Sto76]
Eine Orakel Turingmaschine $M(B)$ ist eine (nicht) deterministische Turingmaschine mit einem extra Band für Orakel Anfragen. Das Orakel einer solchen Maschine ist eine Sprache $B \subseteq \Sigma_B$ für die in einem Schritt bestimmt werden kann, ob $w \in B$ ist ($w \in \Sigma_B$). Dazu gibt es drei erweiternde Zustände $z_{Anfrage}, z_{ja}, z_{nein}$. Wenn M eine Orakel Anfrage stellt, muss auf dem extra Band ein Wort $w \in \Sigma_B$ stehen und die Turingmaschine in den Zustand $z_{Anfrage}$ wechseln. Im nächsten Schritt wechselt M in den Zustand z_{ja}, falls $w \in B$ ist und sonst in z_{nein}. Das extra Band wird geleert.

Dann wird $\mathcal{P}(B)$ bzw. $\mathcal{NP}(B)$ für die Menge der deterministischen bzw. nicht deterministischen Turingmaschinen mit Orakel B geschrieben, die in Polynomialzeit arbeiten. Darauf aufbauend kann $\mathcal{P}(\zeta) = \bigcup_{B \in \zeta} \mathcal{P}(B)$ für eine Menge von Sprachen ζ geschrieben werden. Nun kann äquivalent zu Definition 12 formuliert werden:

Satz 32: Polynomialzeithierarchie mit Orakel Turingmaschinen vgl. [Sto76]
Die Polynomialzeithierarchie aus Definition 12 ist äquivalent zu $\{\Sigma_k^P, \Pi_k^P, \Delta_k^P \mid k \geq 0\}$, wobei

$$\Sigma_0^P = \Pi_0^P = \Delta_0^P = \mathcal{P}$$

und für $k > 0$

$$\begin{aligned} \Sigma_{k+1}^P &= \mathcal{NP}(\Pi_k^P) \\ \Pi_{k+1}^P &= co\mathcal{NP}(\Sigma_k^P) \\ \Delta_{k+1}^P &= \mathcal{P}(\Sigma_k^P) \end{aligned}$$

gilt.

8.2 Ein Prädikat der Presburger Arithmetik für die Multiplikation von beschränkten Faktoren in linearem Platz

In „Komplexitätstheorie“ von Wolfgang Paul [Pau78] wird ein Prädikat $m_n(a, b, c)$ vorgestellt, mit dem es möglich ist in Platz $O(n)$ die Multiplikation $a*b = c$ für Zahlen $a \leq p_n$ auszudrücken ($p_n \geq 2^{2^n}$). Dafür wird zunächst bewiesen, dass es für jede natürliche Zahl a natürliche Zahlen $a_1, a_2, a_3, a_4 \leq \lfloor \sqrt{a} \rfloor$ gibt, sodass $a = a_1 * a_2 + a_3 + a_4$ gilt. Außerdem wird m_0 mittels $\bigvee_{i=1}^{4}(a = i \wedge i * b = c)$, also der im Abschnitt 6.1 erklärten Skalarmultiplikation mit Schranke $max(a) = 4 = p_0$, definiert. Schließlich werden die m_n mit $n > 0$ rekursiv aufgestellt. Dazu wird $a * b = c$

in ein System von Gleichungen aufgeteilt:

$$
\begin{aligned}
a*b=c \Leftrightarrow & \begin{pmatrix} (a_1*a_2+a_3+a_4)*b=c \\ \wedge\ a_1*a_2+a_3+a_4=a \end{pmatrix} \\
\Leftrightarrow & \begin{pmatrix} a_1*a_2*b+a_3*b+a_4*b=c \\ \wedge\ a_1*a_2+a_3+a_4=a \end{pmatrix} \\
\Leftrightarrow & \begin{pmatrix} a_1*c_1+c_2+c_3=c \\ \wedge\ a_2*b=c_1 \\ \wedge\ a_3*b=c_2 \\ \wedge\ a_4*b=c_3 \\ \wedge\ a_1*a_2+a_3+a_4=a \end{pmatrix} \\
\Leftrightarrow & \begin{pmatrix} c_4+c_2+c_3=c \\ \wedge\ a_1*c_1=c_4 \\ \wedge\ a_2*b=c_1 \\ \wedge\ a_3*b=c_2 \\ \wedge\ a_4*b=c_3 \\ \wedge\ a_1*a_2=c_5 \\ \wedge\ c_5+a_3+a_4=a \end{pmatrix}
\end{aligned}
$$

Nun kann m_{n+1} mittels m_n aufgestellt werden, dabei wird nur ein Term konstanter Länge zu m_n hinzugefügt.

$$
\begin{aligned}
m_{n+1} = \exists a_1 ... \exists a_4 \exists c_1 ... \exists c_5 \forall d \forall e \forall f ((& (d=a_1 \wedge e=c_1 \wedge f=c_4) \\
& \vee (d=a_2 \wedge e=b \wedge f=c_1) \\
& \vee (d=a_3 \wedge e=b \wedge f=c_2) \\
& \vee (d=a_4 \wedge e=b \wedge f=c_3) \\
& \vee (d=a_1 \wedge e=a_2 \wedge f=c_5) \\
& \rightarrow m_n(d,e,f)) \wedge c_4+c_2+c_3=c \wedge c_5+a_3+a_4=a
\end{aligned}
$$

m_n besagt, dass $a = b * c$ gilt, falls $a \leq p_n$ für ein $p_n \geq 2^{2^n}$ ist. Die Beschränkung auf p_n überträgt sich durch $a = a_1 * a_2 + a_3 + a_4$, sodass $a \leq {p_n}^2 + 2 * p_n$ gilt. $p_n \geq 2^{2^n}$ galt laut Annahme (gilt für $p_0 = 4 \geq 2^{2^0} = 2$ danach induktiv), sodass nun ${p_n}^2 \geq 2^{2^{n+1}}$ ist, und damit $p_{n+1} > 2^{2^{n+1}}$ gilt.
m_n kann für alle aufgezählten d, e, f verwendet werden, da es für das a auch $a_i \leq \lfloor \sqrt{a} \rfloor$ gibt und aus $a \leq p_{n+1} \leq 2^{2^{(n+1)}}$ folgt, dass

$$
a_i \leq \left\lfloor \sqrt{2^{2^{(n+1)}}} \right\rfloor = \left\lfloor 2^{2^n} \right\rfloor
$$

für alle $(1 \leq i \leq 4)$ ist.
m_n benötigt nur Platz $O(n)$, da m_0 konstanter Größe ist, und bei jedem Schritt von m_n zu m_{n+1} nur ein Term konstanter Größe hinzu kommt. (Herleitung aus [Pau78] mit weiteren Erläuterungen und korrigierter Formel für m_{n+1})

8.3 Rechnung für $q^0_{r_1}$

In Kapitel 6.2.1 wurde eine obere Schranke für die Anzahl der durch Koeffizienten unterscheidbaren Primformeln q^m_r gefunden. Die Rechnung für den Übergang von $r_1 + ... + r_i$ zu

$r_1+...+r_i+1$ durchgeführten Quantoreliminierungen wurde dort schon durchgeführt, wodurch sich die Ungleichung (16) ergeben hat. Bei der darauffolgenden Rechnung für eine geschlossene Formel für q_r^m wird jedoch auch benutzt, dass $q_{r_1}^0 \leq n^{(r_1+1)}$ ist. Diese Formel folgt genau genommen nicht aus der Ungleichung (16), sodass erneut gerechnet werden müsste:

$$
\begin{aligned}
&\mathcal{S}_{\mathrm{PA}_\mathbb{Z}} \models \exists x_{r_1}, \exists x_{r_1-1}, ..., \exists x_1 \underbrace{F_1(x_1, x_2, ..., x_{r_1})}_{q_0^0 \leq n} \\
\Leftrightarrow\ &\mathcal{S}_{\mathrm{PA}_\mathbb{Z}} \models \exists x_{r_1}, \exists x_{r_1-1}, ..., \exists x_2 \underbrace{\bigvee_{(3)}}_{|(3)|\leq n} \underbrace{F_2^*(x_2, x_3, ..., x_{r_1})}_{q_0^0 \leq n} \\
\Leftrightarrow\ &\mathcal{S}_{\mathrm{PA}_\mathbb{Z}} \models \underbrace{\bigvee_{(3)}}_{|(3)|\leq n} \exists x_{r_1}, \exists x_{r_1-1}, ..., \exists x_2 \underbrace{F_2^*(x_2, x_3, ..., x_{r_1})}_{q_0^0 \leq n} \\
\Leftrightarrow\ &\mathcal{S}_{\mathrm{PA}_\mathbb{Z}} \models \underbrace{\bigvee_{(3)}}_{|(3)|\leq n} \exists x_{r_1}, \exists x_{r_1-1}, ..., \exists x_3 \underbrace{\bigvee_{(3)}}_{|(3)|\leq n} \underbrace{F_3^*(x_3, x_4, ..., x_{r_1})}_{q_0^0 \leq n} \\
&\quad\vdots \\
\Leftrightarrow\ &\mathcal{S}_{\mathrm{PA}_\mathbb{Z}} \models \underbrace{\underbrace{\bigvee_{(3)}}_{|(3)|\leq n} ... \underbrace{\bigvee_{(3)}}_{|(3)|\leq n}}_{r_1 \text{ Mal}} \underbrace{F_{r_1+1}^*()}_{q_0^0 \leq n}
\end{aligned}
$$

Auch für den Fall, dass es keine Quantorenwechsel mehr gibt, gilt dass sich die Anzahl der durch Koeffizienten unterscheidbaren Primformeln nur maximal r_1 Mal ver-q_0^0-facht. Da $q_0^0 \leq n$ galt, gilt $q_{r_1}^0 \leq n^{(r_1+1)}$.

8.4 Selbstständigkeitserklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig angefertigt, nicht anderweitig zu Prüfungszwecken vorgelegt und keine anderen als die angegebenen Hilfsmittel verwendet habe. Sämtliche wissentlich verwendete Textausschnitte, Zitate oder Inhalte anderer Verfasser wurden ausdrücklich als solche gekennzeichnet. Mit der Übermittlung meiner Arbeit auch an externe Dienste zur Plagiatsprüfung durch Plagiatssoftware erkläre ich mich einverstanden.

Hannover, den 08.08.2019

Julian Gercke

9 Literatur

[AB09] Arora, Sanjeev ; Barak, Boaz: *Computational complexity: a modern approach*. Cambridge University Press, 2009

[CLRS17] Cormen, Thomas H. ; Leiserson, Charles E. ; Rivest, Ronald ; Stein, Clifford: *Algorithmen-Eine Einführung*. Walter de Gruyter GmbH & Co KG, 2017

[Coo72] Cooper, David C.: Theorem proving in arithmetic without multiplication. In: *Machine intelligence* 7 (1972), Nr. 91-99, S. 300

[CS76] Chandra, Ashok K. ; Stockmeyer, Larry J.: Alternation. In: *17th Annual Symposium on Foundations of Computer Science (sfcs 1976)* IEEE, 1976, S. 98–108

[FR98] Fischer, Michael J. ; Rabin, Michael O.: Super-exponential complexity of Presburger arithmetic. In: *Quantifier Elimination and Cylindrical Algebraic Decomposition*. Springer, 1998, S. 122–135

[Göd31] Gödel, Kurt: Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme I. In: *Monatshefte für mathematik und physik* 38 (1931), Nr. 1, S. 173–198

[Grä88] Grädel, Erich: Subclasses of Presburger arithmetic and the polynomial-time hierarchy. In: *Theoretical Computer Science* 56 (1988), Nr. 3, S. 289–301

[Grä89] Grädel, Erich: Dominoes and the complexity of subclasses of logical theories. In: *Annals of Pure and Applied Logic* 43 (1989), Nr. 1, S. 1–30

[Haa18] Haase, Christoph: A survival guide to presburger arithmetic. In: *ACM SIGLOG News* 5 (2018), Nr. 3, S. 67–82

[Hof17] Hoffmann, Dirk W.: *Die Gödel'schen Unvollständigkeitssätze: Eine geführte Reise durch Kurt Gödels historischen Beweis*. Springer-Verlag, 2017

[LJ83] Lenstra Jr, Hendrik W.: Integer programming with a fixed number of variables. In: *Mathematics of operations research* 8 (1983), Nr. 4, S. 538–548

[NP17] Nguyen, Danny ; Pak, Igor: Short Presburger arithmetic is hard. In: *2017 IEEE 58th Annual Symposium on Foundations of Computer Science (FOCS)* IEEE, 2017, S. 37–48

[Pau78] Paul, Wolfgang J.: *Komplexitätstheorie*. Bd. 39. Teubner, 1978

[RL78] Reddy, Cattamanchi R. ; Loveland, Donald W.: Presburger arithmetic with bounded quantifier alternation. In: *Proceedings of the tenth annual ACM symposium on Theory of computing* ACM, 1978, S. 320–325

[Sca84] Scarpellini, Bruno: Complexity of subcases of Presburger arithmetic. In: *Transactions of the American Mathematical Society* 284 (1984), Nr. 1, S. 203–218

[Sch97] Schöning, Uwe: Complexity of Presburger arithmetic with fixed quantifier dimension. In: *Theory of Computing Systems* 30 (1997), Nr. 4, S. 423–428

[SS05] Salembier, Robert G. ; Southerington, Paul: An implementation of the aks primality test. In: *Computer Engineering* (2005)

[Sto76] Stockmeyer, Larry J.: The polynomial-time hierarchy. In: *Theoretical Computer Science* 3 (1976), Nr. 1, S. 1–22

[Wra76] Wrathall, Celia: Complete sets and the polynomial-time hierarchy. In: *Theoretical Computer Science* 3 (1976), Nr. 1, S. 23–33

[YFC] Yu-Fang Chen, Isil D.: *Quantifier Elimination for Presburger Arithmetic*. `http://flolac.iis.sinica.edu.tw/flolac15/_media/download:5-quantifier-elimination.pdf`

[Zyg91] Zygmunt, Jan: Mojżesz presburger: life and work. In: *History and Philosophy of Logic* 12 (1991), Nr. 2, S. 211–223

10 Abbildungsverzeichnis