30. Theorietag

Automaten und Formale Sprachen

25. September 2020

Technischer Bericht

Vorwort

Die Theorietage der GI Fachgruppe “Automaten und Formalen Sprachen” finden seit 1991 jährlich statt. Seit 2006 war bereits geplant, dass der 30. Theorietag in Wien stattfinden sollte. Auf Grund der Corona-Pandemie war es aber schließlich nicht möglich, den 30. Theorietag wie geplant vor Ort an der TU Wien zu organisieren.

In der langen Tradition der Theorietage war dies daher das erste Mal, dass der Theorietag nicht vor Ort, also wie geplant an der TU Wien, sondern nur im virtuellen Raum stattfinden konnte. Leider gab es damit auch nur wenige Beiträge, die gerade mal einen Tag füllen konnten. Umso mehr danke ich allen Kolleginnen und Kollegen aus Deutschland, die mit ihren Beiträgen und Präsentationen schließlich doch noch die Durchführung des 30. Theorietages im „virtuellen Wiener Raum“ ermöglicht haben.

Insgesamt waren schließlich 21 Personen aus Deutschland (20) und Österreich (1) angemeldet. Überdies konnten am 25. September 2020 noch kurzfristig weitere Personen im virtuellen Raum teilnehmen. Schließlich wurden beim Theorietag 9 Beiträge präsentiert. Außerdem wurde der für 2021 geplante Theorietag vorgestellt und ein Bericht aus der Fachgruppe gegeben.

Ich hoffe dass alle, die am 25. September 2020 beim 30. Theorietag, wenn auch nur virtuell, dabei waren, sich gefreut haben, ihre Kolleginnen und Kollegen zumindest im virtuellen Raum sehen und hören zu können.

Rudi Freund

Stockerau, am 29. September 2020

Liste der Beiträge und Vorträge:

FERNAU Henning
Was Einfüge-Lösch-Systeme mit verbotenen Kontextbedingungen so alles können 1

FREUND Rudi
New Results for P Systems with only One Catalyst
– P Systems with Limiting the Number of Objects in Membranes 7
– Catalytic P Systems with Weak Priority of Catalytic Rules 17

HOFFMANN Stefan
News on Constrained Synchronization 25

KOß Tore
Efficiently Testing Simon's Congruence 29

MANEA Florin
On the Structure of Solution Sets to Regular Word Equations /
Efficiently Testing Simon's Congruence 33

SIEMER Stefan
The Edit Distance to k-Subsequence Universality 37

VU Martin
Insertion-deletion Systems with Dissolution 41

WOLF Petra
Synchronizing Deterministic Push-Down Automata Can Be Really Hard 45

PROGRAMM für Freitag, 25. September 2020

10:15 – 10:30 Eröffnung des Theorietags 2020 (Rudi FREUND)

10:30 – 11:00 MANEA Florin, Universität Göttingen

11:00 – 11:30 KOß Tore, Universität Göttingen

11:30 – 12:00 SIEMER Stefan, Universität Göttingen

12:00 – 13:00 „Mittagspause"

13:00 – 13:30 HOFFMANN Stefan, Universität Trier

13:30 – 14:00 VU Martin, Universität Trier

14:00 – 14:30 WOLF Petra, Universität Trier

14:30 - 15:00 „Kaffeepause"

15:00 – 15:30 FERNAU Henning, Universität Trier

15:30 – 16:15 FREUND Rudolf, TU Wien

16:15 – 16:30 Vorstellung des Theorietags 2021, Universität Leipzig (Andreas MALETTI)

16:30 – 16:45 Bericht aus der Fachgruppe (Henning FERNAU)

16:45 Abschluss des Theorietags 2020 (Rudi FREUND)

THEORIE-TAG 2020

Rudi Freund (Herausgeber): Theorietag 2020, Wien, 24. – 25.9.2020
TU Wien, Technischer Bericht, S. 1–6

Was Einfüge-Lösch-Systeme mit verbotenen Kontextbedingungen so alles können

Henning Fernau[(A)] Lakshmanan Kuppusamy[(B)]
Indhumathi Raman[(C)]

[(A)]Fachbereich 4 – Abteilung Informatikwissenschaften, CIRT
Universität Trier, 54286 Trier, Deutschland. fernau@uni-trier.de

[(B)]School of Computer Science and Engineering
VIT, Vellore-632 014, Indien. klakshma@vit.ac.in

[(C)]Department of Applied Mathematics and Computational Sciences
PSG College of Technology, Coimbatore-641 004, Indien. ind.amcs@psgtech.ac.in

Zusammenfassung

Einfüge-Lösch-Systeme sind bekanntermaßen Turing-mächtig. Das kann sich naturgemäß ändern, wenn natürliche Ressourcen dieser Systeme (und ihrer Varianten) beschränkt werden. Wir untersuchen diese Grenzen der Beschreibungsmächtigkeit für Einfüge-Lösch-Systeme mit verbotenen Kontextbedingungen. Eine ausführlichere Fassung dieser Arbeit ist im Tagungsband der DCFS 2020 enthalten, der in naher Zukunft erscheinen wird.

1. Einleitung

Einfüge-Lösch-Systeme wurden ursprünglich als ein formales Modell im Bereich *DNA Computing* eingeführt [8]. Es handelt sich hierbei im Grunde genommen um Phrasenstrukturgrammatiken mit zwei (eingeschränkten) Grundoperationen, dem Einfügen bzw. Löschen von Zeichenketten in der augenblicklichen Satzform, jeweils unter möglicher Betrachtung des Kontexts der Einfüge- bzw. Löschstelle. Genauer entspricht eine $(\alpha,\beta,\gamma)_{ins}$ notierte Einfügeregel der Ersetzungsregel $\alpha\gamma \to \alpha\beta\gamma$, während eine $(\alpha,\beta,\gamma)_{del}$ notierte Löschregel der Ersetzungsregel $\alpha\beta\gamma \to \alpha\gamma$ entspricht. Die Beschreibungskomplexität von solchen System lässt sich daher beschreiben durch ein Tupel der Form $(n,i',i'';m,j',j'')$ mit folgender Bedeutung für die Regeln besagten Systems:

- n ist die maximale Länge einer eingefügten Zeichenkette,
- i' ist die maximale Länge irgendeines linken Kontexts einer Einfügeregel,
- i'' ist die maximale Länge irgendeines rechten Kontexts einer Einfügeregel,
- m,j',j'' haben analoge Bedeutungen für Löschregeln.

Sodann betrachtet man Sprachfamilien wie $\mathsf{ID}(n,i',i'';m,j',j'')$, die alle Sprachen enthalten, die durch Einfüge-Lösch-Systeme beschrieben werden können, deren Beschreibungskomplexität (komponentenweise) durch $(n,i',i'';m,j',j'')$ beschränkt ist. Beispielsweise ist bekannt, dasss $\mathsf{ID}(1,1,1;1,1,1) = \mathsf{RE}$ gilt; viele derartige Ergebnisse findet man in der Übersicht [15].

Gleichzeitig ist bekannt, dass mancherlei Parameterbeschränkungen zu schwach sind, um ganz RE zu beschreiben. Das motiviert (aus mathematischer Sicht), ähnlich wie bei der klassischen Theorie der Phrasenstrukturgrammatiken, Regulationsmechanismen zu betrachten, siehe [1]. So haben S. Ivanov und S. Verlan in [7] den ursprünglich von Gh. Păun [13] (im Wesentlichen für die Regulierung kontextfreier Grammatiken) eingeführten Mechanismus *semi-conditional* für Einfüge-Lösch-Systeme betrachtet. Als Spezialfall hiervon wiederum hat A. Meduna in [11] verbotene Kontextbedingungen eingeführt *((generalized) forbidding context)*; diese wurden auch in [2, 10, 9, 12] weiter betrachtet. Das heißt, dass man Regeln endliche Mengen von Wörtern zuordnet, die alle keine Teilwörter sein dürfen in der aktuellen Satzform, will man solch eine Regel anwenden. In der vorgelegten Arbeit wird dieses Konzept übertragen auf Einfüge-Lösch-Systeme. Dies hat auch durchaus eine gewisse biologische Motivation, da es auf der molekularen Ebene durchaus Substanzen gibt, deren Vorhandensein gewisse Reaktionen blockiert.

Etwas formaler führt das auf Sprachfamilien, die wir mit $\mathsf{GFID}(k;n,i',i'';m,j',j'')$ notieren, wobei der erste Parameter den Grad des Systems angibt, d.h., die maximal zugelassene Länge eines verbotenen Teilwortes. Aus [7] folgt, dass $\mathsf{GFID}(1;1,1,0;x,y,z) \neq \mathsf{RE}$ gilt für $(x,y,z) \in \{(2,0,0),(1,1,0),(1,0,1)\}$. Weitere Fälle ergeben sich durch Symmetrie. Ebenso ist bekannt, dass $\mathsf{ID}(2,0,0;2,0,0) \neq \mathsf{RE}$ gilt. Es stellt sich die Frage, ob kurze verbotene Kontextbedingungen genügen, insbesondere für diese Parameterwerte Turingmächtigkeit zu erzielen.

2. Vorstellung der Ergebnisse

Wir konnten Turingmächtigkeit für folgende Sprachfamilien nachweisen:

- GFID(2;2,0,0;2,0,0), GFID(2;1,1,0;2,0,0), GFID(2;1,1,0;1,1,0), GFID(2;1,1,0;1,0,1), sowie GFID(2;2,0,0;1,1,0);
- GFID(2;1,0,1;2,0,0), GFID(2;1,0,1;1,0,1), GFID(2;1,0,1;1,1,0), GFID(2;2,0,0;1,0,1).

Die zweite Gruppe von Ergebnissen folgt aus der ersten durch Symmetriebetrachtungen, genauer aus dem Abschluss von RE gegenüber der Spiegeloperation.

Zum Beweis der ersten Gruppe von Ergebnissen wird eine Phrasenstrukturgrammatik G in einer Sonderform der Speziellen Geffert Normalform (SGNF) betrachtet, kurz ssSGNF. Zunächst sei daran erinnert, dass I. Petre und S. Verlan die SGNF eingeführt hatten in [14]. Wesentlich für ein erstes Verständnis für die ssSGNF ist, dass das Nichtterminalalphabet N von G zerfällt in N' und $N'' = \{A,B,C,D,E,F\}$. N' wiederum zerfällt in N_S und $N_{S'}$ mit $S \in N_S$ und $S' \in N_{S'}$. G besitzt löschende Regeln der Form $AB \to \lambda$, $CD \to \lambda$ und $EF \to \lambda$ sowie $S' \to \lambda$ und evtl. $S \to \lambda$. Darüber hinaus gibt es nichtlöschende kontextfreie Regeln folgender zwei Bauarten:

- $X \to Yb$ oder $X \to b'Y$ ($X \in N_S$, $Y \in N'$, $X \neq Y$, $b \in T$, $b' \in \{A,C,E\}$);
- $X \to Yb$ oder $X \to b'Y$ ($X,Y \in N_{S'}$, $X \neq Y$, $b \in \{B,D,F\}$, $b' \in \{A,C,E\}$).

In Phase I der Ableitung der ssSGNF-Grammatik G begegnen wir Satzformen der Bauart

$$\{EA,EC\}^* N_S T^* \cup \{EA,EC\}^* N_{S'} \{BF,DF\}^* T^* \cup \{EA,EC\}^* \{BF,DF\}^* T^* .$$

Die hier angewendeten Regeln haben aufgrund der Aufspaltung des Nichtterminalalphabets einen rechts- bzw. linkslinearen Charakter. Dieser ist typisch für SGNF-Grammatiken. In Phase

II werden die nicht-kontextfreien löschenden Regeln auf Satzformen der Bauart

$$\{EA, EC\}^*\{\lambda, EF\}\{BF, DF\}^*T^*$$

angewendet. Die Nichtterminale E, F dienen zum Aufspreizen der normalen SGNF-Satzformen; ss steht nämlich für *space-separating*. Der Vorteil besteht darin, dass wir leichter verbotene Kontexte definieren können. Denn man kann zeigen, dass ableitbare Satzformen nie Teilwörter aus $\{AA, BB, CC, DD, EE, FF\}$ enthalten. Das kann man in den Simulationen ausnutzen.

Wir erklären im Folgenden nur die allererste der Simulationen, GFID(2;2,0,0;2,0,0) betreffend. Da Zeichenketten der Länge 2 gelöscht werden dürfen, bereitet die Simulation der löschenden Regeln insgesamt keinerlei Schwierigkeiten. Die Simulation einer „rechtslinearen" Regel $p : X \to bY$ ist deutlich komplexer. Wir haben dafür folgende Regeln vorgeschlagen:

$$\begin{array}{ll}
p1 = [(\lambda, pp', \lambda)_{ins}, & \mathcal{M}^v \cup (N' \setminus \{X\}) \cup \Phi] \\
p2 = [(\lambda, p'X, \lambda)_{del}, & (\mathcal{M}^v \setminus \{p, p'\}) \cup (N' \setminus \{X\})] \\
p3 = [(\lambda, bp'', \lambda)_{ins}, & (\mathcal{M}^v \setminus \{p\}) \cup N' \cup \Phi] \\
p4 = [(\lambda, Yp''', \lambda)_{ins}, & (\mathcal{M}^v \setminus \{p, p''\}) \cup N' \cup \{p''Z \mid Z \in V \setminus \{p\}\} \cup \\
 & \qquad \{Zp \mid Z \in V \setminus \{p''\}\} \cup \Phi] \\
p5 = [(\lambda, p^v p^{iv}, \lambda)_{ins}, & (\mathcal{M}^v \setminus \{p, p'', p'''\}) \cup (N' \setminus \{Y\}) \cup \{bp''\} \cup \\
 & \qquad \{Zp \mid Z \in V \setminus \{p''\}\} \cup \Phi] \\
p6 = [(\lambda, p^{iv} p''', \lambda)_{del}, & (\mathcal{M}^v \setminus \{p, p'', p''', p^{iv}, p^v\}) \cup (N' \setminus \{Y\}) \cup \{Yp''', bp''\} \cup \\
 & \{p'''Z \mid Z \in V \setminus \{p''\}\} \cup \{Zp \mid Z \in V \setminus \{p''\}\}] \\
p7 = [(\lambda, p'', \lambda)_{del}, & (\mathcal{M}^v \setminus \{p, p'', p^v\}) \cup (N' \setminus \{Y\}) \cup \{Yp'''\} \cup \{Zp'' \mid Z \in V \setminus \{p^v\}\}] \\
p8 = [(\lambda, p^v p, \lambda)_{del}, & (\mathcal{M}^v \setminus \{p^v, p\}) \cup (N' \setminus \{Y\})]
\end{array}$$

Hierbei haben wir für die $|P|$ vielen Regeln der zu simulierenden Grammatik G (Mengen von) Markierungen eingeführt. Diese Markierungen sind in den Satzformen des simulierenden GFID-Systems als einzelne Zeichen enthalten, um den Fortschritt der Simulation anzuzeigen.

$$\begin{array}{llll}
M & = \{m \mid m \in [1 \ldots |P|]\}, & M' & = \{m' \mid m \in [1 \ldots |P|]\}, \\
M'' & = \{m'' \mid m \in [1 \ldots |P|]\}, & M''' & = \{m''' \mid m \in [1 \ldots |P|]\}, \\
M^{iv} & = \{m^{iv} \mid m \in [1 \ldots |P|]\}, & M^v & = \{m^v \mid m \in [1 \ldots |P|]\}, \\
\mathcal{M}'' & = M \cup M' \cup M'', & \mathcal{M}''' & = M \cup M' \cup M'' \cup M''', \\
\mathcal{M}^{iv} & = \mathcal{M}''' \cup M^{iv}, & \mathcal{M}^v & = \mathcal{M}''' \cup M^{iv} \cup M^v,
\end{array}$$

Schließlich umfässt Φ eine Standardmenge verbotener Teilwörter:

$$\Phi = \{A, C, E, \sigma\}(\{B, D, F, \sigma\} \cup T) \cup T\{A, C, E\} \cup \{ZZ \mid Z \in \{A, B, C, D, E, F\}\}.$$

Eine Ersetzungsregel $p : X \to bY$ wird wie folgt simuliert.

$$\begin{array}{c}
\alpha X \beta \Rightarrow_{p1} \alpha pp' X \beta \Rightarrow_{p2} \alpha p \beta \Rightarrow_{p3} \alpha bp'' p \beta \Rightarrow_{p4} \alpha bY p''' p'' p \beta \Rightarrow_{p5} \\
\alpha bY p^v p^{iv} p''' p'' p \beta \Rightarrow_{p6} \alpha bY p^v p'' p \beta \Rightarrow_{p7} \alpha bY p^v p \beta \Rightarrow_{p8} \alpha bY \beta.
\end{array}$$

Entsprechend wird be „linkslinearen" Regeln verfahren. Die beweistechnische Schwierigkeit besteht nun darin nachzuweisen, dass keine unerwünschten Ableitungen möglich sind. So verwehrt $\mathcal{M}^v$ als (u.a.) verbotener Kontext bei $p1$ die Anwendung der entsprechenden Einfügeregel dann, wenn irgendeine Regelmarkierung in der Satzform vorhanden ist, sei es in der Grundform

oder in irgendeiner gestrichenen Variante. Insbesondere kann also $p1$ nicht zweimal unmittelbar hintereinander ausgeführt werden. So muss man Schritt für Schritt (induktiv) argumentieren, dass nur die intendierte Ableitung zu einem Terminalwort führen kann.

Die Notwendigkeit der Verwendung von ssSGNF (im Gegensatz zu SGNF) erschließt sich noch nicht bei dieser Simulation, sie wird aber benötigt in anderen Fällen. Deshalb geben wir im Folgenden noch an, wie eine löschende Regel $f : AB \to \lambda$ simuliert werden kann mit den Ressourcen $\mathsf{GFID}(2;1,0,0;1,1,0)$; diese Simulation kann also verwendet werden für $\mathsf{GFID}(2;1,1,0;1,1,0)$ und für $\mathsf{GFID}(2;2,0,0;1,1,0)$, sowie für $\mathsf{GFID}(2;1,1,0;1,0,1)$ (in symmetrischer Weise).

$$
\begin{array}{|ll|}
\hline
f1 = [(\lambda, f', \lambda)_{ins}, & N' \cup \mathcal{M}''' \cup (N''_\ell N''_r \setminus \{AB\})] \\
f2 = [(\lambda, f, \lambda)_{ins}, & N' \cup (\mathcal{M}''' \setminus \{f'\}) \cup \{f'Z \mid Z \in V \setminus \{B\}\} \cup N''_\ell N''_r] \\
f3 = [(\lambda, f'', \lambda)_{ins}, & N' \cup (\mathcal{M}''' \setminus \{f, f'\}) \cup \{f'Z \mid Z \in V \setminus \{B\}\} \cup \\
 & \quad \{fZ \mid Z \in V \setminus \{A\}\} \cup N''_\ell N''_r] \\
f4 = [(f, A, \lambda)_{del}, & N' \cup (\mathcal{M}''' \setminus \{f, f', f''\}) \cup \{f'Z \mid Z \in V \setminus \{B\}\} \cup \\
 & \quad \{Zf'' \mid Z \in V \setminus \{B\}\} \cup N''_\ell N''_r] \\
f5 = [(f', B, \lambda)_{del}, & N' \cup (\mathcal{M}''' \setminus \{f, f', f''\}) \cup \{fZ \mid Z \in V \setminus \{f'\}\} \cup \\
 & \quad \{Zf'' \mid Z \in V \setminus \{B\}\} \cup N''_\ell N''_r] \\
f6 = [(f, f', \lambda)_{del}, & N' \cup (\mathcal{M}''' \setminus \{f, f', f''\}) \cup \{f'Z \mid Z \in V \setminus \{f''\}\} \cup \\
 & \quad \{Zf'' \mid Z \in V \setminus \{f'\}\} \cup N''_\ell N''_r] \\
f7 = [(f, f'', \lambda)_{del}, & N' \cup (\mathcal{M}''' \setminus \{f, f''\}) \cup N''_\ell N''_r] \\
f8 = [(\lambda, f, \lambda)_{del}, & \{f', f''\}] \\
\hline
\end{array}
$$

Beabsichtigt ist hiermit eine Simulation der folgenden Bauart:

$$\alpha AB\beta t \Rightarrow_{f1} \alpha Af'B\beta t \Rightarrow_{f2} \alpha fAf'B\beta t \Rightarrow_{f3} \alpha fAf'Bf''\beta t \Rightarrow_{f4} \alpha ff'Bf''\beta t \Rightarrow_{f5} \alpha ff'f''\beta t \Rightarrow_{f6} \alpha ff''\beta t \Rightarrow_{f7} \alpha f\beta t \Rightarrow_{f8} \alpha f\beta t$$

Wichtig zu beobachten ist nun, dass die Löschregeln $f4$ bzw. $f5$ nicht mehrfach hintereinander angewendet werden können, weil hier insbesondere die Teilwörter AA und BB nicht auftreten können. In der üblichen SGNF ist das aber sehr wohl möglich.

Als Kuriosität merken wir abschließend noch an, dass die Simulation rechts- (und auch links-)linearer Regeln des „asymmetrischen Falles“ (AS) $\mathsf{GFID}(2;1,1,0;1,0,1)$ sehr viel einfacher gelingt als im „symmetrischen Falle“ (S) von $\mathsf{GFID}(2;1,1,0;1,1,0)$. Grund dafür ist, dass die Informationsübertragung sich einfacher im asymmetrischen Fall gestaltet., d.h., wenn Einfügen und Löschen aus unterschiedlichen Richtungen seine Kontextinformationen erhält.

AS:
$$
\begin{array}{|l|}
\hline
p1 = [(X, p, \lambda)_{ins}, \mathcal{M}'' \cup (N' \setminus \{X\}) \cup \Phi] \\
p2 = [(\lambda, X, p)_{del}, (\mathcal{M}'' \setminus \{p\}) \cup (N' \setminus \{X\})] \\
p3 = [(p, Y, \lambda)_{ins}, (\mathcal{M}'' \setminus \{p\}) \cup N'] \\
p4 = [(p, p', \lambda)_{ins}, (\mathcal{M}'' \setminus \{p\}) \cup (N' \setminus \{Y\}) \cup (\{pZ \mid Z \in V \setminus \{Y\}\})] \\
p5 = [(p', b, \lambda)_{ins}, (\mathcal{M}'' \setminus \{p, p'\}) \cup (N' \setminus \{Y\}) \cup (\{p'Z \mid Z \in V \setminus \{Y\}\})] \\
p6 = [(\lambda, p, p')_{del}, \{p'Y\}] \\
p7 = [(\lambda, p', \lambda)_{del}, \{p\}] \\
\hline
\end{array}
$$

Die Simulation von $p : X \to bY$ läuft wie folgt ab:

$$\alpha X\beta \Rightarrow_{p1} \alpha Xp\beta \Rightarrow_{p2} \alpha p\beta \Rightarrow_{p3} \alpha pY\beta \Rightarrow_{p4} \alpha pp'Y\beta \Rightarrow_{p5} \alpha pp'bY\beta \Rightarrow_{p6} \alpha p'bY\beta \Rightarrow_{p7} \alpha bY\beta.$$

Hingegen benötigen wir mehr Simulationsregeln für S:

$$
\begin{array}{ll}
p1 = [(\lambda, p, \lambda)_{ins}, & \mathcal{M}''' \cup (N' \setminus \{X\}) \cup \Phi] \\
p2 = [(X, p', \lambda)_{ins}, & (\mathcal{M}''' \setminus \{p\}) \cup (N' \setminus \{X\})] \\
p3 = [(p, X, \lambda)_{del}, & (\mathcal{M}''' \setminus \{p, p'\}) \cup (N' \setminus \{X\}) \cup (\{X\gamma \mid \gamma \in V \setminus \{p'\}\}] \\
p4 = [(p', p'', \lambda)_{ins}, & (\mathcal{M}''' \setminus \{p, p'\}) \cup N' \cup (\{pZ \mid Z \in V \setminus \{p'\}\})] \\
p5 = [(p'', p''', \lambda)_{ins}, & (\mathcal{M}''' \setminus \{p, p', p''\}) \cup N' \cup (\{p'Z \mid Z \in V \setminus \{p''\}\})] \\
p6 = [(\lambda, p, \lambda)_{del}, & (\mathcal{M}''' \setminus \{p, p', p'', p'''\}) \cup N' \cup \{p'Z \mid Z \in V \setminus \{p''\}\}] \\
p7 = [(p', b, \lambda)_{ins}, & \{p'Z \mid Z \in V \setminus \{p''\}\} \cup \{p''Z \mid Z \in V \setminus \{p'''\}\} \cup N' \cup \{p\}] \\
p8 = [(p'', Y, \lambda)_{ins}, & \{p'Z \mid Z \in V \setminus \{b\}\} \cup \{p''Z \mid Z \in V \setminus \{p'''\}\} \cup N' \cup \{p\}] \\
p9 = [(Y, p''', \lambda)_{del}, & \{p, p'p'', p''p'''\} \cup (N' \setminus \{Y\})] \\
p10 = [(b, p'', \lambda)_{del}, & \{p, p''', p'p''\} \cup (N' \setminus \{Y\})] \\
p11 = [(\lambda, p', \lambda)_{del}, & \{p, p'', p'''\}]
\end{array}
$$

Die Simulation von $p : X \to bY$ sollte hierbei folgendermaßen ablaufen:

$$
\begin{aligned}
\alpha X\beta \Rightarrow_{p1} \alpha pX\beta \Rightarrow_{p2} \alpha pXp'\beta \Rightarrow_{p3} \alpha pp'\beta \Rightarrow_{p4} \alpha pp'p''\beta \Rightarrow_{p5} \alpha pp'p''p'''\beta \\
\Rightarrow_{p6} \alpha p'p''p'''\beta \Rightarrow_{p7} \alpha p'bp''p'''\beta \Rightarrow_{p8} \alpha p'bp''Yp'''\beta \Rightarrow_{p9} \alpha p'bp''Y\beta \\
\Rightarrow_{p10} \alpha p'bY\beta \Rightarrow_{p11} \alpha bY\beta.
\end{aligned}
$$

3. Was noch zu tun (und offen) bleibt

- Unsere bisherigen Simulationen benötigen alle Regeln vom Grad 2. Gibt es RE-Ergebnisse für Grammatiken mit Regeln vom Grad 1? Oder helfen umgekehrt Regeln vom Grad 3?
- Insbesondere ist unbekannt, ob GFID(1;2,0,0;2,0,0) turingmächtig ist.
- Unsere Simulationen benötigen große Mengen von verbotenen Teilwörtern. Kann man die Mächtigkeit dieser Mengen beschränken, ohne die RE-Mächtigkeit zu verlieren?
- Tests auf Teilwörter der Länge 2 scheinen besonders notwendig zu sein. Man könnte auch deren Anzahl als Beschreibungskomplexitätsmaß studieren.
- Die ssSGNF scheint hilfreich zu sein bei Systemen mit einseitigen Löschungen, da sie Mehrfachlöschungen vermeiden hilft. Daher schlagen wir vor, diese Normalform auch in anderen derartigen Simulationen zu verwenden. Bisherige Ergebnisse in Kombination mit anderen Regulationsmechanismen (die natürliche Kandidaten für eine solche Anwendung darstellen) finden sich in [3, 4, 5, 6] und der dort zitierten Literatur.
- Alle Konstruktionen, die (ss)SGNF verwenden, zeichnen sich dadurch aus, dass sie sehr viele (beliebig viele) Nichtterminale verwenden. In dieser Arbeit kommt noch hinzu, dass die Regeln (insbesondere die kontextfreien Regeln) über Markierungssymbole in die Anzahl der Nichtterminale der Systeme einfließen. Daher wäre die Nichtterminalkomplexität (zusätzlich zu den anderen Parametern) ein weiteres interessantes Beschreibungskomplexitätsmaß, bislang ohne irgendwelche Ergebnisse.

Literatur

[1] J. DASSOW, GH. PĂUN, *Regulated Rewriting in Formal Language Theory*. EATCS Monographs in Theoretical Computer Science 18, Springer-Verlag Berlin, 1989.

[2] H. FERNAU, L. KUPPUSAMY, R. O. OLADELE, I. RAMAN, Improved Descriptional Complexity Results on Generalized Forbidding Grammars. In: S. P. PAL, A. VIJAYAKUMAR (eds.), *5th Annual International Conference on Algorithms and Discrete Applied Mathematics CALDAM*. LNCS 11394, Springer, 2019, 174–188. Long version accepted with Discrete Applied Mathematics.

[3] H. FERNAU, L. KUPPUSAMY, I. RAMAN, On the computational completeness of graph-controlled insertion-deletion systems with binary sizes. *Theoretical Computer Science* **682** (2017), 100–121. Special Issue on Languages and Combinatorics in Theory and Nature.

[4] H. FERNAU, L. KUPPUSAMY, I. RAMAN, Investigations on the power of matrix insertion-deletion systems with small sizes. *Natural Computing* **17** (2018) 2, 249–269.

[5] H. FERNAU, L. KUPPUSAMY, I. RAMAN, Computational completeness of simple semi-conditional insertion-deletion systems of degree (2, 1). *Natural Computing* **18** (2019) 3, 563–577.

[6] H. FERNAU, L. KUPPUSAMY, I. RAMAN, On path-controlled insertion-deletion systems. *Acta Informatica* **56** (2019) 1, 35–59.

[7] S. IVANOV, S. VERLAN, Random Context and Semi-conditional Insertion-deletion Systems. *Fundamenta Informaticae* **138** (2015), 127–144.

[8] L. KARI, *On insertions and deletions in formal languages*. Ph.D. thesis, University of Turku, Finland, 1991.

[9] T. MASOPUST, A. MEDUNA, Descriptional Complexity of Grammars Regulated by Context Conditions. In: R. LOOS, S. Z. FAZEKAS, C. MARTÍN-VIDE (eds.), *LATA 2007. Proceedings of the 1st International Conference on Language and Automata Theory and Applications*. Report 35/07, Research Group on Mathematical Linguistics, Universitat Rovira i Virgili, Tarragona, Spain, 2007, 403–412.

[10] T. MASOPUST, A. MEDUNA, Descriptional complexity of semi-conditional grammars. *Information Processing Letters* **104** (2007) 1, 29–31.

[11] A. MEDUNA, Generalized forbidding grammars. *International Journal of Computer Mathematics* **36** (1990), 31–39.

[12] A. MEDUNA, M. SVEC, Descriptional Complexity of Generalized Forbidding Grammars. *International Journal of Computer Mathematics* **80** (2003) 1, 11–17.

[13] GH. PĂUN, A variant of random context grammars: semi-conditional grammars. *Theoretical Computer Science* **41** (1985), 1–17.

[14] I. PETRE, S. VERLAN, Matrix insertion-deletion systems. *Theoretical Computer Science* **456** (2012), 80–88.

[15] S. VERLAN, Recent Developments on Insertion-Deletion Systems. *The Computer Science Journal of Moldova* **18** (2010) 2, 210–245.

THEORIE-TAG 2020

Rudi Freund (Herausgeber): Theorietag 2020, Wien, 24. – 25.9.2020
TU Wien, Technischer Bericht, S. 7–16

Computationally Complete Catalytic P Systems with One Catalyst

Artiom Alhazov[(A)] Rudolf Freund[(B)] Sergiu Ivanov[(C)]

[(A)] Vladimir Andrunachievici Institute of Mathematics and Computer Science
Academiei 5, Chișinău, MD-2028, Moldova
artiom@math.md

[(B)] Faculty of Informatics, TU Wien
Favoritenstraße 9–11, 1040 Wien, Austria
rudi@emcc.at

[(C)] IBISC, Univ Évry, Paris-Saclay University
23, boulevard de France 91034 Évry, France
sergiu.ivanov@ibisc.univ-evry.fr

Abstract

Catalytic P systems are among the first variants of membrane systems ever considered in this area. This variant of systems also features some prominent computational complexity questions, and in particular the problem of using only one catalyst: is one catalyst enough to allow for generating all recursively enumerable sets of multisets? Several additional ingredients have been shown to be sufficient for obtaining even computational completeness with only one catalyst. In this paper we show that one catalyst is sufficient for obtaining computational completeness if either catalytic rules have weak priority over non-catalytic rules or else instead of the standard maximal parallel derivation mode we use the derivation mode *maxobjects*, i.e., we only take those multisets of rules which affect the maximal number of objects in the underlying configuration.

1. Introduction

Membrane systems were introduced in [8] as a multiset-rewriting model of computing inspired by the structure and the functioning of the living cell. During two decades now membrane computing has attracted the interest of many researchers, and its development is documented in two textbooks, see [9] and [10]. For actual information see the P systems webpage [12] and the issues of the Bulletin of the International Membrane Computing Society and of the Journal of Membrane Computing.

One basic feature of P systems already presented in [8] is the maximally parallel derivation mode, i.e., using non-extendable multisets of rules in every derivation step. The result of a computation can be extracted when the system halts, i.e., when no rule is applicable any more. Catalysts are special symbols which allow only one object to evolve in its context (in contrast to

promoters) and in their basic variant never evolve themselves, i.e., a catalytic rule is of the form $ca \to cv$, where c is a catalyst, a is a single object and v is a multiset of objects. In contrast, non-catalytic rules in catalytic P systems are non-cooperative rules of the form $a \to v$.

From the beginning, the question how many catalysts are needed for obtaining computational completeness has been one of the most intriguing challenges regarding (catalytic) P systems. In [3] it has already been shown that two catalysts are enough for generating any recursively enumerable set of multisets, without any additional ingredients like a priority relation on the rules as used in the original definition. As already known from the beginning, without catalysts only regular (semi-linear) sets can be generated when using the standard halting mode, i.e., a result is extracted when the system halts with no rule being applicable any more. As shown, for example, in [5], using various additional ingredients, i.e., additional control mechanisms, one catalyst can be sufficient: in P systems with label selection, only rules from one set of a finite number of sets of rules in each computation step are used; in time-varying P systems, the available sets of rules change periodically with time. On the other hand, for catalytic P systems with only one catalyst a lower bound has been established in [6]: P systems with one catalyst can simulate partially blind register machines, i.e., they can generate more than just semi-linear sets.

In [1], we returned to the idea of using a priority relation on the rules, but take only a very weak form of such a priority relation: we only required that overall in the system catalytic rules have weak priority over non-catalytic rules. This means that the catalyst c must not stay idle if the current configuration contains an object a with which it may cooperate in a rule $ca \to cv$; all remaining objects evolve in the maximally parallel way with non-cooperative rules. On the other hand, if the current configuration does not contain an object a with which the catalyst c may cooperate in a rule $ca \to cv$, c may stay idle and *all* objects evolve in the maximally parallel way with non-cooperative rules. Even without using more than this weak priority of catalytic rules over the non-catalytic (non-cooperative) rules, we could establish computational completeness for catalytic P systems with only one catalyst. In this paper, we recall the result from [1], but show a somehow much stronger result using a similar construction as in [1]: we show computational completeness for catalytic P systems with only one catalyst using the derivation mode *maxobjects*, i.e., we only take those multisets of rules which affect the maximal number of objects in the underlying configuration.

2. Definitions

For an alphabet V, by V^* we denote the free monoid generated by V under the operation of concatenation, i.e., containing all possible strings over V. The *empty string* is denoted by λ. For any $a \in V$ and any string w over A, w_a denotes the number of symbols a in w.

For further notions and results in formal language theory we refer to textbooks like [2] and [11].

2.1. Register Machines

Register machines are well-known universal devices for computing (or generating or accepting) sets of vectors of natural numbers.

Definition 2.1 *A* register machine *is a construct* $M = (m, B, l_0, l_h, P)$ *where*
– m *is the number of registers,*
– P *is the set of instructions bijectively labeled by elements of* B*,*
– $l_0 \in B$ *is the initial label, and*
– $l_h \in B$ *is the final label.*

The instructions of M *can be of the following forms:*

– $p : (ADD(r), q, s)$*, with* $p \in B \setminus \{l_h\}$*,* $q, s \in B$*,* $1 \leq r \leq m$*.*
Increase the value of register r *by one, and non-deterministically jump to instruction* q *or* s*.*
– $p : (SUB(r), q, s)$*, with* $p \in B \setminus \{l_h\}$*,* $q, s \in B$*,* $1 \leq r \leq m$*.*
If the value of register r *is not zero then decrease the value of register* r *by one (*decrement *case) and jump to instruction* q*, otherwise jump to instruction* s *(*zero-test *case).*
– $l_h : HALT$*.*
Stop the execution of the register machine.

A configuration *of a register machine is described by the contents of each register and by the value of the current label, which indicates the next instruction to be executed.* M *is called* deterministic *if the ADD-instructions all are of the form* $p : (ADD(r), q)$*.*

In the *computing* case, a computation starts with the input of a l-vector of natural numbers in its first l registers and by executing the first instruction of P (labeled with l_0); it terminates with reaching the $HALT$-instruction and the output of a k-vector of natural numbers in its last k registers. Without loss of generality, we may assume all registers except the last k output registers to be empty at the end of the computation.

In the *accepting* case, no output registers are needed.

In the *generating* case, no input registers are needed.

For useful results on the computational power of register machines, we refer to [7]; for proving computational completeness results for specific variants of P systems, the most important observation is that – in addition to the input and output registers – only two working registers are needed.

Proposition 2.2 *Register machines can compute any partial recursive relation on vectors of natural numbers with* l *components as input and vectors of natural numbers with* k *components as output using precisely* $l + 2 + k$ *registers, where without loss of generality, we may assume that at the end of a successful computation the first* $l + 2$ *registers are empty, and, moreover, on the output registers, i.e., the last* k *registers, no SUB-instruction is ever used.*

2.2. Partially Blind Register Machines

A *partially blind register machine* is defined as a register machine where the *SUB*-instructions are of the following form:
$p : (SUB(r), q)$, with $p \in B \setminus \{l_h\}$, $q \in B$, $1 \leq r \leq m$.If the value of register r is not zero then decrease the value of register r by one and jump to instruction q, otherwise abort the computation, i.e., *partially blind* register machines cannot check a register for zero. A computation works as for a register machine, yet with the restriction that a computation is aborted if one tries to decrement a register which is zero. Moreover, computing, accepting or generating now also requires all registers (except output registers) to be empty at the end of the computation.

The standard model of (hierarchical) P systems can be defined as follows, for example, see [10] for several variants:

Definition 2.3 *A* catalytic P system *of degree* $m \geq 1$ *is a construct*

$$\Pi = (O, C, \mu, w_1, \ldots, w_m, R_1, \ldots, R_m, i_0) \text{ where}$$

- O *is the alphabet of objects;*
- $C \subset O$ *is the set of* catalysts*,*
- μ *is a membrane structure of degree* m *with membranes labeled in a one-to-one manner with the natural numbers* $1, \ldots, m$*;*
- $w_1, \ldots, w_m \in O^*$ *are the multisets of objects initially present in the* m *regions of* μ*;*
- R_i*,* $1 \leq i \leq m$*, are finite sets of* evolution rules *over* O *associated with the regions* $1, 2, \ldots, m$ *of* μ*; the evolution rules are of the forms* $ca \to cv$ *(*catalytic rules*) or* $a \to v$ *(*non-cooperative rules*), where* c *is a catalyst,* a *is an object from* $O \setminus C$*, and* v *is a string from* $((O \setminus C) \times \{here, out, in\})^*$*.;*
- $i_0 \in \{0, 1, \ldots, m\}$ *indicates the output region of* Π*.*

The membrane structure and the multisets in Π constitute a *configuration* of the P system; the *initial configuration* is given by the initial multisets $w_1, \ldots, w_m$. A transition between configurations is governed by the application of the evolution rules, which is done in a given derivation mode. In this paper we consider the maximally parallel derivation mode (*max* for short), i.e., only applicable multisets of rules which cannot be extended by further rules are to be applied to the objects in all membrane regions, as well as the derivation mode *maxobjects* where in every membrane region only applicable multisets of rules which affect the maximal number of objects are to be applied. We remark that all the multisets which can be used in the derivation mode *maxobjects* can also be used in the derivation mode *max*.

The application of a rule $u \to v$ in a region containing a multiset M results in subtracting from M the multiset identified by u, and then in adding the multiset identified by v. The objects can eventually be transported through membranes due to the targets *in* and *out*. We refer to [10] for further details and examples.

The P system continues with applying multisets of rules according to the derivation mode until there remain no applicable rules in any region of Π. Then the system *halts*. We consider the number of objects from $O \setminus C$ contained in the output region i_0 at the moment when the system halts as the *result* of the underlying computation of Π. The system is called *extended* since the catalytic objects in C are not counted to the result of a computation. Yet as often done in the literature, in the following we will omit the term *extended* and just speak of *catalytic P systems*, especially as we will restrict ourselves to P systems with only one catalyst.

The set of results of all computations possible in Π using the derivation mode δ is called the set of natural numbers *generated by* Π using δ and it is denoted by $N(\Pi, \delta)$ if we only count the total number of objects in the output membrane; if wedistinguish between the multiplicities of different objects, we obtain aset of vectors of natural numbers denoted by $Ps(\Pi, \delta)$.

Remark 2.4 *As in this paper we only consider catalytic P systems with only one catalyst, without loss of generality, we can restrict ourselves to one-membrane catalytic P systems with the single catalyst in the skin membrane, by taking into account the well-known flattening process, e.g., see [4].*

Remark 2.5 *Finally, we make the convention that a one-membrane catalytic P system with the single catalyst in the skin membrane and with internal output in the skin membrane, not taking into account the single catalyst c for the results, throughout the rest of the paper will be described without specifying the trivial membrane structure or the output region (assumed to be the skin membrane), i.e., we will just write $\Pi = (O, \{c\}, w, R)$ where O is the set of objects, c is the single catalyst, w is the initial input specifying the initial configuration, and R is the set of rules.*

As already mentioned earlier, the following result was shown in [6], establishing a lower bound for the computational power of catalytic P systems with only one catalyst:

Proposition 2.6 *Catalytic P systems with only one catalyst working in the derivation mode* max *have at least the computational power of partially blind register machines.*

Example 2.7 *In [6] it was shown that the vector set $S = \{(n, m) \mid 0 \leq n, n \leq m \leq 2^n\}$ (which is not semi-linear) can be generated by some (even extended version of a) PBRM and therefore by a P system with only one catalyst and* 19 *rules.*

As already shown in [3], register machines with $n \geq 2$ decrementable registers can be simulated by catalytic P systems with n catalysts and by purely catalytic P systems with $n + 1$ catalysts. Hence, both catalytic P systems and purely catalytic P systems are computationally complete.

3. Weak Priority of Catalytic Rules

In this section we now study catalytic P systems with only one catalyst in which the catalytic rules have weak priority over the non-catalytic rules.

Example 3.1 *To illustrate this weak priority of catalytic rules over the non-catalytic rules, consider the rules $ca \to cb$ and $a \to d$. If the current configuration contains $k > 0$ copies of a, then the catalytic rule $ca \to cb$* must *be applied to one of the copies, while the rest of objects a may be taken up by the non-catalytic rule $a \to d$. In particular, if $k = 1$, only $ca \to cb$ may be applied.*

We would like to highlight the fact that weak priority of catalytic rules is much weaker than the general weak priority, as the priority relation is only constrained by the types of rules.

Remark 3.2 *The reverse weak priority, i.e., non-catalytic rules having priority over catalytic rules, is useless, since it is equivalent to removing all catalytic rules for which there are non-catalytic rules with the same symbol on the left-hand side of the rule. In that way we just end up with an even restricted variant of P systems with only one catalyst.*

3.1. Computational Completeness with Weak Priority

In this section, we show that catalytic P systems with one catalyst only and with weak priority of catalytic rules are computationally complete.

Theorem 3.3 *Catalytic P systems with only one catalyst and with weak priority of catalytic rules over the non-cooperative rules when working in the derivation mode* max *are computationally complete.*

Proof. Given an arbitrary register machine $M = (m, B, l_0, l_h, P)$ we will construct a corresponding catalytic P system with one membrane and one catalyst $\Pi = (O, \{c\}, w, R)$ simulating M. Without loss of generality, we may assume that, depending on its use as an accepting or generating or computing device, the register machine M fulfills the condition that on the output registers we never apply any *SUB*-instruction. The following proof is given for the most general case of a register machine computing any partial recursive relation on vectors of natural numbers with l components as input and vectors of natural numbers with k components as output using precisely $l+2+k$ registers, where without loss of generality, we may assume that at the end of a successful computation the first $l+2$ registers are empty, and, moreover, on the output registers, i.e., the last k registers, no *SUB*-instruction is ever used. In fact, the proof works for any number n of decrementable registers, no matter how many of them are the l input registers and the working registers, respectively.

The main idea behind our construction is that all the symbols except the catalyst c and the output symbols (representing the contents of the output registers) go through a cycle of length $n+2$ where n is the number of decrementable registers of the simulated register machine. When the symbols are traversing the r-th section of the n sections, they "know" that they are to probably simulate a *SUB*-instruction on register r of the register machine M.

Observing that $n = m - k$, in total we get the following set of objects:

$$\begin{aligned} O = &\{a_r \mid n+1 \le r \le m\} \\ &\cup \{(a_r, i) \mid 1 \le r \le n, 1 \le i \le n+2\} \\ &\cup \{(p, i) \mid p \in B_{ADD}, 1 \le i \le n+2\} \\ &\cup \{(p, i) \mid p \in B_{SUB(r)}, 1 \le i \le r+1\} \\ &\cup \{(p, i)^-, (p, i)^0 \mid p \in B_{SUB(r)}, r+2 \le i \le n+2\} \\ &\cup \{c, e, \#\}. \end{aligned}$$

B_{ADD} denotes the set of labels of *ADD*-instructions. $B_{SUB(r)}$ denotes the set of labels of all *SUB*-instruction $p : (SUB(r), q, s)$ of decrementable registers r

The starting configuration of Π is $w = c(l_0, 1)\alpha_0$, where l_0 is the starting label of the machine and α_0 is the multiset encoding the initial values of the registers.

All register symbols a_r, $1 \le r \le n$, representing the contents of decrementable registers, are equipped with the rules evolving them throughout the whole cycle:

$$(a_r, i) \to (a_r, i+1), 1 \le r \le n+1; \qquad (a_r, n+2) \to (a_r, 1). \tag{1}$$

The construction also includes the trap rule $\# \to \#$: once the trap symbol # is introduced, it will always keep the system busy and prevent it from halting and thus from producing a result.

For simulating *ADD*-instructions we also need the following rules:

Increment $p : (ADD(r), q, s)$*:*

The (variants of the) symbol p cycles together with all the other symbols, always involving the catalyst:

$$c(p,i) \to c(p,i+1), \;\; 1 \leq i \leq n+1. \tag{2}$$

At the end of the cycle, the register is incremented and the non-deterministic jump to q or s occurs: for r being a decrementable register, we take

$$c(p,n+2) \to c(q,1)(a_r,1), \qquad c(p,n+2) \to c(s,1)(a_r,1), \tag{3}$$

whereas for r being a register never to be decremented, we take

$$c(p,n+2) \to c(q,1)a_r, \qquad c(p,n+2) \to c(s,1)a_r \tag{4}$$

The output symbols need not undergo the cycle, in fact, they must not do that because otherwise the computation would never stop. When the computation of the register machine halts, only output symbols will be present, as we have assumed that at the end of a computation all decrementable registers will be empty, i.e., no cycling symbols will be present any more in the P system. Finally, we have to mention that if q or s is the final label l_h, then we take λ instead, which means that also the P system will halt, because, as already explained above, the only symbols left in the configuration will be output symbols, for which no rules exist.

The state symbol is not allowed to evolve without the catalyst:

$$(p,i) \to \#, \;\; 1 \leq i \leq n+2. \tag{5}$$

Hence, in that way it is guaranteed that the catalyst cannot be used in another way, i.e., affecting a symbol (a_r,i) as explained below during the simulation of a *SUB*-instruction on register r.

Decrement and zero-test $p : (SUB(r), q, s)$*:*

The simulation of a *SUB* instruction is carried out in three steps of the cycle, i.e., in steps r, $r+1$, and $r+2$.

Before reaching simulation phase r, i.e., step r, the state symbol goes through the cycle, necessarily involving the catalyst:

$$c(p,i) \to c(p,i+1) \;\; > \;\; (p,i) \to \#, \qquad 1 \leq i < r. \tag{6}$$

Although by the definition of the P systems with priority of catalytic rules, the catalytic rule has priority over the non-catalytic rule for (p,i), we indicate this general priority relation by the sign $<$ (or $>$ for the reverse relation) in order to make the situation even clearer.

In the first step of the simulation phase r, i.e., in step r, the state symbol releases the catalyst to try to perform the decrement and to produce a witness symbol e if register r is not empty:

$$(p,r) \to (p,r+1), \qquad c(a_r,r) \to ce. \tag{7}$$

Note that due to the counters identifying the position of the register symbols in the cycle, it is guaranteed that the catalytic rule transforming (a_r,r) picks the correct register symbol. Furthermore, due to the priority of the catalytic rules, one of the the register symbols (a_r,r) *must* be transformed by the catalytic rule if present, instead of continuing along its cycle.

In the second step of simulation phase r, i.e., in step $r+1$, the detection of the possible decrement happens. The outcome is stored in the state symbol:

$$\begin{array}{lcl} ce \to c & > & e \to \#, \\ (p,r+1) \to (p,r+2)^- & < & c(p,r+1) \to c(p,r+2)^0. \end{array} \tag{8}$$

Observe that in this case of register r being empty, both rules $(p,r+1) \to (p,r+2)^-$ and $c(p,r+1) \to c(p,r+2)^0$ would be applicable, but due to the priority of the catalytic rules, the second rule must be preferred, thus producing $(p,r+2)^0$. Therefore, the superscript of the state symbol correctly reflects the outcome of the decrement: it is $-$ if the decrement succeeded, and 0 if it did not.

After the simulation of the decrement, the state symbols evolve to the end of the cycle and produce the corresponding next state symbols:

$$\begin{array}{ll} c(p,i)^- \to c(p,i+1)^-, \ r+2 \le i \le n+1, & c(p,n+2)^- \to c(q,1), \\ c(p,i)^0 \to c(p,i+1)^0, \ r+2 \le i \le n+1, & c(p,n+2)^0 \to c(s,1), \\ (p,i)^- \to \#, \ (p,i)^0 \to \#, \ r+2 \le i \le n+2. & \end{array} \tag{9}$$

The additional two steps $n+1$ and $n+2$ are needed to correctly finish the decrement and zero test cases even for $r = n$.

Finally, we again mention that if q or s is the final label l_h, then we take λ instead, which means that not only the register machine but also the P system halts, because, as already explained above, the only symbols left in the configuration will be output symbols, for which no rules exist. □

In case we have enough decrementable registers, the additional steps can be reduced:

Theorem 3.4 *For any register machine with at least three decrementable registers we can construct a catalytic P system with only one catalyst and with weak priority of catalytic rules over the non-cooperative rules and working in the derivation mode* max *which can simulate every step of the register machine in n steps where n is the number of decrementable registers.*

Theorem 3.5 *For any register machine with at least two decrementable registers we can construct a catalytic P system with only one catalyst and with weak priority of catalytic rules over the non-cooperative rules and working in the derivation mode* max *which can simulate every step of the register machine in $n+1$ steps where n is the number of decrementable registers.*

As the number of decrementable registers in generating register machines needed for generating any recursively enumerable set of (vectors of) natural numbers is only two, from Theorem 3.4 we obtain the following result:

Corollary 3.6 *For any generating register machine with two decrementable registers we can construct a catalytic P system with only one catalyst and with weak priority of catalytic rules over the non-cooperative rules which can simulate every step of the register machine in* 3 *steps, and therefore such catalytic P systems with only one catalyst and with weak priority of catalytic rules over the non-cooperative rules can generate any recursively enumerable set of (vectors of) natural numbers.*

4. Catalytic P Systems With Only One Catalyst Working in the Derivation Mode *maxobjects*

In this section we now study catalytic P systems with only one catalyst which work in the derivation mode *maxobjects* and show that computational completeness can be obtained with only one catalyst and no further ingredients.

Theorem 4.1 *For any register machine with at least three decrementable registers we can construct a catalytic P system with only one catalyst and working in the derivation mode* maxobjects *which can simulate every step of the register machine in n steps where n is the number of decrementable registers.*

Proof. We can take over the proof of Theorem 3.4. The priority of catalytic rules then is somehow regained by the fact that a catalytic rule $ca \to cv$ affects two objects, whereas the corresponding non-catalytic rule $a \to v$ only affects one object. Going carefully through the proof we even realize that as the only rule introducing the trap symbol now the single rule $e \to \#$ remains. □

5. Conclusion

The results obtained in Section 4 are optimal with respect to the number of catalysts for catalytic P systems working in the derivation mode *maxobjects*, as the results for P systems only using non-cooperative rules are the same for both the derivation mode *maxobjects* and the derivation mode *max*; for example, in the generating case, P systems only using non-cooperative rules can only generate semi-linear sets.

Although in this paper we have come tantalizingly close to showing computational completeness of catalytic P system with only one catalyst, we still believe that the answer to the classic problem whether catalytic P system with only one catalyst working in the derivation mode *max* are computationally complete still remains one of the biggest challenges in the theory of P systems.

References

[1] Artiom Alhazov, Rudolf Freund, and Sergiu Ivanov. Catalytic P systems with weak priority of catalytic rules. In Rudolf Freund, editor, *Proceedings ICMC 2020, September 14–18, 2020*. TU Wien, 2020. *To appear*.

[2] Jürgen Dassow and Gheorghe Păun. *Regulated Rewriting in Formal Language Theory*. Springer, 1989.

[3] Rudolf Freund, Lila Kari, Marion Oswald, and Petr Sosík. Computationally universal P systems without priorities: two catalysts are sufficient. *Theoretical Computer Science*, 330(2):251–266, 2005.

[4] Rudolf Freund, Alberto Leporati, Giancarlo Mauri, Antonio E. Porreca, Sergey Verlan, and Claudio Zandron. Flattening in (tissue) P systems. In Artiom Alhazov, Svetlana Cojocaru, Marian Gheorghe, Yurii Rogozhin, Grzegorz Rozenberg, and Arto Salomaa, editors, *Membrane Computing*, volume 8340 of *Lecture Notes in Computer Science*, pages 173–188. Springer, 2014.

[5] Rudolf Freund, Marion Oswald, and Gheorghe Păun. Catalytic and purely catalytic P systems and P automata: Control mechanisms for obtaining computational completeness. *Fundam. Inform.*, 136(1–2):59–84, 2015.

[6] Rudolf Freund and Petr Sosík. On the power of catalytic P systems with one catalyst. In Grzegorz Rozenberg, Arto Salomaa, José M. Sempere, and Claudio Zandron, editors, *Membrane Computing – 16th International Conference, CMC 2015, Valencia, Spain, August 17–21, 2015, Revised Selected Papers*, volume 9504 of *Lecture Notes in Computer Science*, pages 137–152. Springer, 2015.

[7] Marvin L. Minsky. *Computation. Finite and Infinite Machines*. Prentice Hall, Englewood Cliffs, NJ, 1967.

[8] Gheorghe Păun. Computing with membranes. *Journal of Computer and System Sciences*, 61(1):108–143, 2000.

[9] Gheorghe Păun. *Membrane Computing: An Introduction*. Springer, 2002.

[10] Gheorghe Păun, Grzegorz Rozenberg, and Arto Salomaa, editors. *The Oxford Handbook of Membrane Computing*. Oxford University Press, 2010.

[11] Grzegorz Rozenberg and Arto Salomaa, editors. *Handbook of Formal Languages*. Springer, 1997.

[12] The P Systems Website. `http://ppage.psystems.eu/`.

THEORIE-TAG 2020

Rudi Freund (Herausgeber): Theorietag 2020, Wien, 24. – 25.9.2020
TU Wien, Technischer Bericht, S. 17–24

P Systems with Limiting the Number of Objects

Artiom Alhazov[(A)] Rudolf Freund[(B)] Sergiu Ivanov[(C)]

[(A)] Vladimir Andrunachievici Institute of Mathematics and Computer Science
Academiei 5, Chișinău, MD-2028, Moldova
artiom@math.md

[(B)] Faculty of Informatics, TU Wien
Favoritenstraße 9–11, 1040 Wien, Austria
rudi@emcc.at

[(C)] IBISC, Univ Évry, Paris-Saclay University
23, boulevard de France 91034 Évry, France
sergiu.ivanov@ibisc.univ-evry.fr

Abstract

P systems are a model of compartmentalized multiset rewriting inspired by the structure and functioning of the living cell. In this paper, we focus on a variant in P systems in which membranes have limited capacity, i.e., the number of objects they may hold is statically bounded. This feature corresponds to an important physical property of cellular compartments. We propose several possible semantics of limited capacity and show that one of them allows real-time simulations of partially blind register machines, while the other one allows for obtaining computational completeness.

1. Introduction

Membrane systems were introduced in [9] as a multiset-rewriting model of computing inspired by the structure and the functioning of the living cell. Among the basic features of the original model are the hierarchical arrangement of the membranes and the parallel evolution of the objects contained in the membrane compartments. Usually a result is obtained if the computation halts, i.e., if no rule is applicable any more.

In this paper we consider an additional feature also inspired by biology, namely the limited capacity of cells to include objects – in total or of a specific kind. When the number of cells is not bounded as in P systems with active membranes, this biological feature of limited capacity can be kept for all cells below a given fixed bound. On the other hand, in the standard hierarchical model with a static number of cells, or, even if we allowed membrane dissolution, with a fixed upper bound for the number of cells, we can only limit the number of specific objects and have to allow an unbounded number of other objects when aiming at non-trivial theoretical results.

When the number of cells is not bounded because of using membrane creation and/or membrane division (together with membrane dissolution), the number of objects in one cell/membrane

can even be restricted to one, still allowing for obtaining computational completeness, thereby counting the number of membranes/cells instead of the number of objects in an output membrane/cell; for example, see [1, 4, 2].

In this paper we investigate P systems limiting the number of (specific) objects to be contained in a membrane region and two different semantics of how to treat the situation when the application of a (multiset of) rule(s) would violate this limiting condition, both of them blocking or aborting computations which try to apply a multiset of rules leading to a violation of the limited capacity conditions. For the first variant we show that it allows for real-time simulations of partially blind register machines, while the other variant allows for obtaining computational completeness.

The development of the fascinating area of membrane computing during the last two decades is documented in two textbooks, see [10] and [11]. For actual information see the P systems webpage [13] and the issues of the Bulletin of the International Membrane Computing Society and of the Journal of Membrane Computing.

2. Definitions

For an alphabet V, by V^* we denote the free monoid generated by V under the operation of concatenation, i.e., containing all possible strings over V. The *empty string* is denoted by λ. For any $a \in V$ and any string w over A, w_a denotes the number of symbols a in w.

For further notions and results in formal language theory we refer to textbooks like [5] and [12].

2.1. Register Machines

Register machines are well-known universal devices for computing (or generating or accepting) sets of vectors of natural numbers.

Definition 2.1 *A* register machine *is a construct* $M = (m, B, l_0, l_h, P)$ *where*
- m *is the number of registers,*
- P *is the set of instructions bijectively labeled by elements of* B*,*
- $l_0 \in B$ *is the initial label, and*
- $l_h \in B$ *is the final label.*

The instructions of M *can be of the following forms:*
- $p : (ADD(r), q, s)$*, with* $p \in B \setminus \{l_h\}$*,* $q, s \in B$*,* $1 \leq r \leq m$*.*
  Increase the value of register r *by one, and non-deterministically jump to instruction* q *or* s*.*
- $p : (SUB(r), q, s)$*, with* $p \in B \setminus \{l_h\}$*,* $q, s \in B$*,* $1 \leq r \leq m$*.*
  If the value of register r *is not zero then decrease the value of register* r *by one (*decrement *case) and jump to instruction* q*, otherwise jump to instruction* s *(*zero-test *case).*
- $l_h : HALT$*.*
  Stop the execution of the register machine.

A configuration *of a register machine is described by the contents of each register and by the value of the current label, which indicates the next instruction to be executed.* M *is called* deterministic *if the ADD-instructions all are of the form* $p : (ADD(r), q)$.

In the *computing* case, a computation starts with the input of a l-vector of natural numbers in its first l registers and by executing the first instruction of P (labeled with l_0); it terminates with reaching the $HALT$-instruction and the output of a k-vector of natural numbers in its last k registers. Without loss of generality, we may assume all registers except the last k output registers to be empty at the end of the computation.

In the *accepting* case, no output registers are needed.

In the *generating* case, no input registers are needed.

For useful results on the computational power of register machines, we refer to [8]; for proving computational completeness results for specific variants of P systems, the most important observation is that – in addition to the input and output registers – only two working registers are needed.

Proposition 2.2 *Register machines can compute any partial recursive relation on vectors of natural numbers with* l *components as input and vectors of natural numbers with* k *components as output using precisely* $l + 2 + k$ *registers, where without loss of generality, we may assume that at the end of a successful computation the first* $l + 2$ *registers are empty, and, moreover, on the output registers, i.e., the last* k *registers, no SUB-instruction is ever used.*

2.2. Partially Blind Register Machines

A *partially blind register machine* is defined as a register machine where the *SUB*-instructions are of the following form:
$p : (SUB(r), q)$, with $p \in B \setminus \{l_h\}$, $q \in B$, $1 \leq r \leq m$.If the value of register r is not zero then decrease the value of register r by one and jump to instruction q, otherwise abort the computation, i.e., *partially blind* register machines cannot check a register for zero. A computation works as for a register machine, yet with the restriction that a computation is aborted if one tries to decrement a register which is zero. Moreover, computing, accepting or generating now also requires all registers (except output registers) to be empty at the end of the computation.

2.3. P Systems

The standard model of (hierarchical) P systems can be defined as follows, for example, see [11] for several variants:

Definition 2.3 *A* catalytic P system *of degree* $m \geq 1$ *is a construct*

$$\Pi = (O, C, \mu, w_1, \ldots, w_m, R_1, \ldots, R_m, i_0) \text{ where}$$

- O *is the alphabet of objects;*
- $C \subset O$ *is the set of* catalysts,
- μ *is a membrane structure of degree* m *with membranes labeled in a one-to-one manner with the natural numbers* $1, \ldots, m$*;*

- $w_1, \ldots, w_m \in O^*$ *are the multisets of objects initially present in the* m *regions of* μ*;*
- R_i, $1 \leq i \leq m$, *are finite sets of* evolution rules *over* O *associated with the regions* $1, 2, \ldots, m$ *of* μ*; the evolution rules are of the forms* $ca \to cv$ *(*catalytic rules*) or* $a \to v$ *(*non-cooperative rules*), where* c *is a catalyst,* a *is an object from* $O \setminus C$*, and* v *is a string from* $((O \setminus C) \times \{here, out, in\})^*$*.;*
- $i_0 \in \{0, 1, \ldots, m\}$ *indicates the output region of* Π.

The membrane structure and the multisets in Π constitute a *configuration* of the P system; the *initial configuration* is given by the initial multisets $w_1, \ldots, w_m$. A transition between configurations is governed by the application of the evolution rules, which is done in the maximally parallel way, i.e., only applicable multisets of rules which cannot be extended by further rules are to be applied to the objects in all membrane regions.

The application of a rule $u \to v$ in a region containing a multiset M results in subtracting from M the multiset identified by u, and then in adding the multiset identified by v. The objects can eventually be transported through membranes due to the targets *in* and *out*.

The P system continues with applying multisets of rules in the maximally parallel way until there remain no applicable rules in any region of Π. Then the system *halts*. We consider the number of objects from O contained in the output region i_0 at the moment when the system halts as the *result* of the underlying computation of Π. The set of results of all computations possible in Π is called the set of natural numbers *generated by* Π and it is denoted by $N(\Pi)$ if we only count the total number of objects in the output membrane; if we distinguish between the multiplicities of different objects, we obtain a set of vectors of natural numbers denoted by $Ps(\Pi)$. We refer to [11] for further details and examples.

Since the beginning, the question how many catalysts are needed in catalytic and purely catalytic P systems for obtaining computational completeness has been a challenging theoretical question. The following result was shown in [7], establishing a lower bound for the computational power of catalytic P systems with only one catalyst:

Proposition 2.4 *Catalytic P systems with only one catalyst have at least the computational power of partially blind register machines.*

Example 2.5 *In [7] it was shown that the vector set* $S = \{(n, m) \mid 0 \leq n, n \leq m \leq 2^n\}$ *(which is not semi-linear) can be generated by some (even extended version of a) PBRM and therefore by a P system with only one catalyst and* 19 *rules.*

As already shown in [6], register machines with $n \geq 2$ decrementable registers can be simulated by catalytic P systems with n catalysts and by purely catalytic P systems with $n + 1$ catalysts. Hence, both catalytic P systems and purely catalytic P systems are computationally complete.

3. Limited Capacity

In most of the variants of P systems considered in the literature the number of objects in a membrane region is not limited. In [3], we proposed variants in which the number of objects a membrane may contain is bounded, with the bound already being given in the definition of

the system, either limiting the total number of objects in a cell or only limiting the number of specific objects in a cell, respectively. The following definitions are given as in [3].

Definition 3.1 *A P system with* per-membrane limited capacity *is the following construct:*

$$\Pi = (O, \mu, w_1, \ldots, w_n, k_1, \ldots, k_n, R_1, \ldots R_n, i_0),$$

where $k_i \in \mathbb{N} \cup \{\infty\}$ is the total capacity *of membrane i, $1 \leq i \leq n$, meaning that, for $|v_i|$ denoting the contents of membrane i in the current configuration, the condition $|v_i| \leq k_i$ must always be enforced, unless $k_i = \infty$. The other components of the tuple are as in Subsection 2.3.*

Definition 3.2 *A P system with* per-symbol limited capacity *is the following construct:*

$$\Pi = (O, \mu, w_1, \ldots, w_n, K_1, \ldots, K_n, R_1, \ldots R_n, i_0),$$

where $K_i : O \to \mathbb{N} \cup \{\infty\}$ are functions defining the per-symbol *capacity of membrane i. For w denoting the union of the contents of all memranes in the system, the condition $w_a \leq K(a)$ therefore must be enforced at all times, for any $a \in O$, unless $K(a) = \infty$.*

Remark 3.3 *We immediately remark that the flattening technique which is folklore in the membrane computing community can be applied in the case of P systems with per-symbol limited capacity. Without loss of generality, we therefore in Section 4 will only consider 1-membrane systems, which can be written in a simplified version as follows with omitting the trivial membrane structure and taking the skin membrane 1 as the output membrane:*

$$\Pi = (O, w, K, R) \text{ and } \Pi = (O, C, w, K, R) \text{ for catalytic P systems.}$$

3.1. Semantics of Limited Capacity

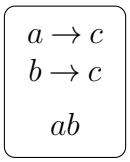
What should happen if a membrane is about to exceed its capacity (total or per-symbol)? Multiple kinds of behaviors may be considered, see [3] for several variants. We here only consider the *blocking behavior*: prohibit the application of (multisets of) rules which would produce more objects. Attempting to apply such rules blocks the system, and yields no result.

Although in this paper we want to focus on the blocking behavior, there are still at least two possible semantics for the blocking behavior itself under the maximally parallel derivation mode:

Semantics 1: Take all the applicable multisets of rules in the maximally parallel derivation mode, but discard all those multisets which would violate the constraints.

Semantics 2: Take all the applicable multisets of rules in the asynchronous derivation mode, discard the multisets which would violate the constraints, and then pick the non-extendable, i.e., maximal multisets out of these applicable multisets of rules.

To illustrate the difference between these two semantics, consider the following 1-membrane system with limited capacity:



It can formally be written as $\Pi_{ab} = (\{a,b\}, ab, K_{ab}, \{a \to c, b \to c\})$ where $K_{ab}(c) = 1$ and $K_{ab}(a) = K_{ab}(b) = \infty$. In the case of Semantics 1, no multisets of rules not violating the constraint of limiting the capacity of symbols c in the resulting configuration would be applicable, and the P system will block /abort this computation. On the other hand, under Semantics 2, Π_{ab} would be allowed to apply *either* $a \to c$ *or* $b \to c$, but not both.

4. Computational Power

In this section we investigate the computational power of P systems with limited per-symbol capacity: when operating with Semantics 1, they at least can simulate partially blind register machines in real time; when operating with Semantics 2, they can simulate purely catalytic P systems (and thus register machines) and therefore are computationally complete.

4.1. Semantics 1 Allows for Simulating a PBRM in Real Time

In comparison with the result stated in [7] showing that P systems with one catalyst can simulate partially blind register machines (without any further ingredients), we here obtain a real-time simulation, see [3], whereas the result there needs a cycle of $n+3$ for each step of the register machine, with n being the number of decrementable registers.

Theorem 4.1 *Catalytic P systems with one catalyst and per-symbol limited capacity operating with Semantics 1 can simulate partially blind register machines (PBRM) in real time, plus three additional cleanup steps at the end of the computation.*

4.2. Semantics 2 Allows for Computational Completeness

Theorem 4.2 *P systems with per-symbol limited capacity operating with Semantics 2 without catalysts can simulate purely catalytic P systems.*

Proof. Let $\Pi = (O, C, w, K, R)$ be an arbitrary purely catalytic P system. We now construct a P system $\Pi' = (O, w, K, R')$ with per-symbol limited capacity, operating with Semantics 2, without catalysts:

$$\begin{aligned}
\Pi' &= (O, w, K, R'),\\
w_0 &= l_0\, \alpha_0,\\
K &= \{(c,1) \mid c \in C\} \cup \{(x,\infty) \mid x \in O \setminus C\},\\
R' &= \{a \to cu \mid ca \to cu \in R, c \in C\}\\
&\quad \cup \{c \to \lambda \mid c \in C\}.
\end{aligned}$$

α_0 is the multiset encoding the initial values of the registers.

In the construction of Π' the symbols $c \in C$ now are not catalysts any more as in Π, but guarantee the same effect by their number being limited by 1.

□

Corollary 4.3 *P systems with per-symbol limited capacity operating with Semantics 2 without catalysts can simulate catalytic P systems.*

Proof. Let $\Pi = (O, C, w, K, R)$ be an arbitrary catalytic P system. As in the Proof of Theorem 4.2 we construct a P system $\Pi' = (O, w, K, R')$ with per-symbol limited capacity, operating with Semantics 2, without catalysts:

$$\begin{aligned}
\Pi' &= (O, w, K, R'), \\
w_0 &= l_0\, \alpha_0, \\
K &= \{(c, 1) \mid c \in C\} \cup \{(x, \infty) \mid x \in O \setminus C\}, \\
R' &= \{a \to cu \mid ca \to cu \in R, c \in C\} \\
&\quad \cup \{a \to u \mid a \to u \in R\} \\
&\quad \cup \{c \to \lambda \mid c \in C\}.
\end{aligned}$$

In fact, we have just added the non-cooperative rules $a \to u$ from R to R'.

□

Since catalytic as well as purely catalytic P systems are computationally complete, for example see [6], we immediately derive the following corollary.

Corollary 4.4 *P systems with per-symbol limited capacity operating with Semantics 2 are computationally complete, even without using catalysts.*

5. Conclusion

In this paper, we have introduced the idea of bounding the number of symbols that may appear in the membranes of a P system. This is a quite natural restriction to consider, given that actual biological membranes are of limited capacity, too. We defined limited total and per-symbol capacities, and defined two possible semantics for handling the overflow. We then showed that Semantics 1 allows non-cooperative P systems to simulate partially blind register machines in real time, with 3 additional cleanup steps at the end of the computation. We also showed that non-cooperative P systems operating under Semantics 2 of limited capacity directly simulate purely catalytic and catalytic P systems (in real time), yet without needing catalysts, and therefore are computationally complete.

This paper only scratches the surface of the study of P systems with limited capacity. One immediate open problem is that of computational completeness of (catalytic, purely catalytic) P systems with limited capacity operating with Semantics 1 or else characterizing the computational power of these systems.

Acknowledgements

Sergiu Ivanov is partially supported by the Paris region via the project DIM RFSI n°2018-03 “Modèles informatiques pour la reprogrammation cellulaire”.

References

[1] A. ALHAZOV, P systems without multiplicities of symbol-objects. *Inf. Process. Lett.* **100** (2006) 3, 124–129.

[2] A. ALHAZOV, R. FREUND, S. IVANOV, Length P Systems. *Fundam. Inform.* **134** (2014) 1-2, 17–37.

[3] A. ALHAZOV, R. FREUND, S. IVANOV, P Systems with Limited Capacity. In: *Proceedings 18th Brainstorming Week on Membrane Computing, Sevilla, February 4–8, 2020*. Fénix Editora, Sevilla, 2020. *To appear*.

[4] A. ALHAZOV, R. FREUND, A. RISCOS-NÚÑEZ, Membrane division, restricted membrane creation and object complexity in P systems. *Int. J. Comput. Math.* **83** (2006) 7, 529–547.

[5] J. DASSOW, GH. PĂUN, *Regulated Rewriting in Formal Language Theory*. Springer, 1989. `https://www.springer.com/de/book/9783642749346`

[6] R. FREUND, L. KARI, M. OSWALD, P. SOSÍK, Computationally universal P systems without priorities: two catalysts are sufficient. *Theoretical Computer Science* **330** (2005) 2, 251–266.

[7] R. FREUND, P. SOSÍK, On the Power of Catalytic P Systems with One Catalyst. In: G. ROZENBERG, A. SALOMAA, J. M. SEMPERE, C. ZANDRON (eds.), *Membrane Computing – 16th International Conference, CMC 2015, Valencia, Spain, August 17–21, 2015, Revised Selected Papers*. Lecture Notes in Computer Science 9504, Springer, 2015, 137–152.

[8] M. L. MINSKY, *Computation. Finite and Infinite Machines*. Prentice Hall, Englewood Cliffs, NJ, 1967.

[9] GH. PĂUN, Computing with Membranes. *Journal of Computer and System Sciences* **61** (2000) 1, 108–143.

[10] GH. PĂUN, *Membrane Computing: An Introduction*. Springer, 2002.

[11] GH. PĂUN, G. ROZENBERG, A. SALOMAA (eds.), *The Oxford Handbook of Membrane Computing*. Oxford University Press, 2010.

[12] G. ROZENBERG, A. SALOMAA (eds.), *Handbook of Formal Languages*. Springer, 1997.

[13] The P Systems Website. `http://ppage.psystems.eu/`.

THEORIE-TAG 2020

Rudi Freund (Herausgeber): Theorietag 2020, Wien, 24. – 25.9.2020
TU Wien, Technischer Bericht, S. 25–27

Synchronisierbarkeit unter kommutativen und polyzyklischen regulären Einschränkungen

Stefan Hoffmann[(A)]

[(A)]Informatikwissenschaften, FB IV, Universität Trier, Universitätsring 15, 54296 Trier, Deutschland, hoffmanns@informatik.uni-trier.de

Zusammenfassung

Wir geben eine vollständige Klassifikation der Berechnungskomplexität des Synchronisierbarkeitsproblems für endliche Halbautomaten unter kommutativen regulären Einschränkungen. Weiter zeigen wir, dass unter polyzyklischen regulären Einschränkungen das Problem der Synchronisierbarkeit stets in NP liegt.

1. Ergebnisse

Wir setzen Kenntnisse und Notationen der Automatentheorie, formaler Sprachen und der Komplexitätstheorie, wie in [7] aufgeführt, voraus. Eine Sprache $L \subseteq \Sigma^*$ heißt *kommutativ*, wenn sie unter dem Vertauschen von Buchstaben abgeschlossen ist; genauer, aus $uawbv \in L$, für $u, w, v \in \Sigma^*$ und $a, b \in \Sigma$, folgt $ubwav \in L$. Ein *Halbautomat* ist ein Tupel $A = (\Sigma, S, \delta)$, wobei Σ ein endliches Alphabet, S eine endliche Zustandsmenge und $\delta : S \times \Sigma \to S$ die Überführungsfunktion bezeichnet. Letztere kann, genau wie bei endlichen Automaten [7], auf Wörter und Teilmengen erweitert werden. Ein Halbautomat heißt *synchronisierend*, falls ein Wort $w \in \Sigma^*$ existiert, so dass $\delta(s, w) = \delta(\hat{s}, w)$ für alle $s, \hat{s} \in S$ gilt. Das Problem der *Synchronisierbarkeit unter regulären Einschränkungen* wurde in [1] eingeführt. Gegeben sei eine (fixe) reguläre Sprache L.

Definition 1.1 ([1]) L-CONSTR-SYNC
Input*: Halbautomat* $A = (\Sigma, Q, \delta)$.
Question*: Gibt es ein synchronisierendes Wort* w *for* A *mit* $w \in L$*?*

In [4] wurde folgendes gezeigt.

Theorem 1.2 ([4]) *Für kommutative reguläre Sprachen* L *ist* L-CONSTR-SYNC *entweder in P, NP-vollständig oder PSPACE-vollständig, und jeder dieser Komplexitätsklassen wird realisiert. Weiter ist, zu gegebenem* L*, die Komplexität von* L-CONSTR-SYNC *effektiv entscheidbar.*

Eine genauere Formulierung und zahlreiche Beispiele finden sich in [4][1]. Ein nicht-deterministischer Automat $A = (\Sigma, S, \delta, s_0, F)$ heißt *polyzyklisch*, wenn es zu jedem Zustand $s \in S$ ein Wort

[1]Eine Langfassung wird in einem Sonderband zur COCOON 2020, in der Zeitschrift *Theoretical Computer Science (TCS)*, erscheinen.

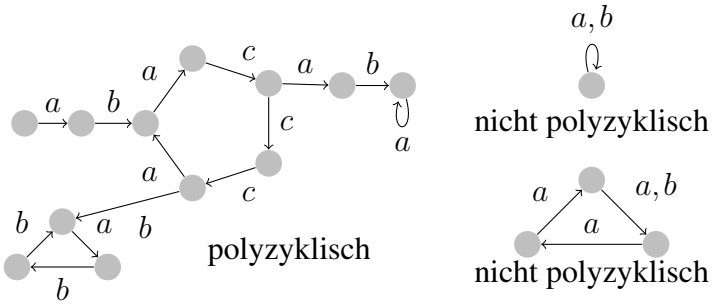


Figure 1: Polyzyklische und nicht polyzyklische Automaten.

$w \in \Sigma^*$ gibt, so dass $\{u \in \Sigma \mid s \in \delta(s,u)\} \subseteq w^*$ gilt[2]. Eine Sprache $L \subseteq \Sigma^*$ heißt *polyzyklisch*, wenn sie durch einen polyzyklischen Automaten akzeptiert wird.

Proposition 1.3 ([6]) *Die polyzyklischen Sprachen können ebenso durch partielle deterministische Automaten charakterisiert werden. Weiter sind sie abgeschlossen unter Konkatenation und Vereinigung.*

In [6] konnte gezeigt werden.

Theorem 1.4 ([6]) *Für polyzyklische Sprachen L ist L-CONSTR-SYNC stets in NP.*

Wir verweisen auf [6] für Beispiele zu NP-vollständigen Problemen und Problemen in P. Eine Sprache $L \subseteq \Sigma^*$ heißt *bounded* [2], sofern es Wörter $w_1, \ldots, w_n \in \Sigma^*$ gibt, so dass $L \subseteq w_1^* \cdots w_n^*$. Jede reguläre Sprache, welche bounded ist, ist polyzyklisch, wie mit einem einfachen Satz von Ginsburg [3] folgt.

Corollary 1.5 *Ist $L \subseteq \Sigma^*$ bounded und regulär, so ist L-CONSTR-SYNC in NP.*

2. Offene Probleme

(i) Bisher sind alle bekannten Probleme, für die L-CONSTR-SYNC NP-vollständig ist, durch polyzyklische Sprachen gegeben[3]. Ob dies im Allgemeinen so ist, ist ein offenes Problem.
(ii) Gibt es für polyzyklische Einschränkungssprachen nur Probleme, die entweder in P oder NP-vollständig sind, d.h. gilt eine Dichotomie?
(iii) Allgemein: Gilt für reguläres L, dass L-CONSTR-SYNC entweder in P, NP-vollständig oder PSPACE-vollständig ist?

Remark 2.1 *In der arXiv-Version von [4], siehe [5], wurde ein kleiner Fehler korrigiert. In zwei Propositionen und dem Hauptsatz fehlte eine Annahme. Siehe auch den Kommentar zur letzten arXiv-Version [5] in der arXiv-Ansicht.*

[2] In [2] werden solche Sprachen als kommutativ bezeichnet, wir verstehen hier aber unter kommutativen Sprachen etwas anderes.

[3] Es gibt allerdings eine Vielzahl von nicht-polyzyklischen Sprachen L, für die das Problem in P ist.

Literatur

[1] H. FERNAU, V. V. GUSEV, S. HOFFMANN, M. HOLZER, M. V. VOLKOV, P. WOLF, Computational Complexity of Synchronization under Regular Constraints. In: P. ROSSMANITH, P. HEGGERNES, J. KATOEN (eds.), *44th International Symposium on Mathematical Foundations of Computer Science, MFCS 2019, August 26-30, 2019, Aachen, Germany*. LIPIcs 138, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019, 63:1–63:14.
https://doi.org/10.4230/LIPIcs.MFCS.2019.63

[2] S. GINSBURG, E. H. SPANIER, Bounded Algol-Like Languages. *Transactions of the American Mathematical Society* **113** (1964) 2, 333–368.
http://www.jstor.org/stable/1994067

[3] S. GINSBURG, E. H. SPANIER, Bounded Regular Sets. *Proceedings of the American Mathematical Society* **17** (1966) 5, 1043–1049.
http://www.jstor.org/stable/2036087

[4] S. HOFFMANN, Computational Complexity of Synchronization Under Regular Commutative Constraints. In: D. KIM, R. N. UMA, Z. CAI, D. H. LEE (eds.), *Computing and Combinatorics - 26th International Conference, COCOON 2020, Atlanta, GA, USA, August 29-31, 2020, Proceedings*. Lecture Notes in Computer Science 12273, Springer, 2020, 460–471.
https://doi.org/10.1007/978-3-030-58150-3_37

[5] S. HOFFMANN, Computational Complexity of Synchronization under Regular Commutative Constraints. *CoRR* **abs/2005.04042** (2020).
https://arxiv.org/abs/2005.04042

[6] S. HOFFMANN, On A Class of Constrained Synchronization Problems in NP. In: G. CORDASCO, L. GARGANO, A. RESCIGNO (eds.), *Proceedings of the 21th Italian Conference on Theoretical Computer Science, ICTCS 2020, Ischia, Italy, September 9-11, 2019*. CEUR Workshop Proceedings, CEUR-WS.org, 2020.

[7] J. E. HOPCROFT, R. MOTWANI, J. D. ULLMAN, *Introduction to Automata Theory, Languages, and Computation*. 2nd edition, Addison-Wesley, 2001.

THEORIE-TAG 2020

Rudi Freund (Herausgeber): Theorietag 2020, Wien, 24. – 25.9.2020
TU Wien, Technischer Bericht, S. 29–31

Efficiently Testing Simon's Congruence

Paweł Gawrychowski[(A)] Maria Kosche[(B)] Tore Koß[(B)]
Florin Manea [(B)] Stefan Siemer[(B)]

[(A)]University of Wrocław, Faculty of Mathematics and Computer Science, Poland
gawry@cs.uni.wroc.pl

[(B)]Göttingen University, Computer Science Department, Germany
{maria.kosche,tore.koss,florin.manea,stefan.siemer}@cs.uni-goettingen.de

Abstract

A subsequence (also called scattered factor or subword, especially in automata and language theory) of a word w is a word u such that there exist (possibly empty) words $v_0, \ldots, v_n, u_1, \ldots, u_n$ with $u = u_1 \ldots u_n$ and $w = v_0 u_1 v_1 u_2 \ldots u_n v_n$. Intuitively, the subsequences of a word w are exactly those words obtained by deleting some of the letters of w, so, in a sense, they can be seen as lossy-representations of the word w. Accordingly, subsequences may be a natural mathematical model for situations where one has to deal with input strings with missing or erroneous symbols sequencing, such as processing DNA data or digital signals. Due to this very simple and intuitive definition as well as the apparently large potential for applications, there is a high interest in understanding the fundamental properties that can be derived in connection to the sets of subsequences of words. This is reflected in the consistent literature developed around this topic. J. Sakarovitch and I. Simon in [9, Chapter 6] overview some of the most important combinatorial and language theoretic properties of sets of subsequences. The theory of subsequences was further developed in various directions: combinatorics on words, automata theory, formal verification, or string algorithms.

One particularly interesting notion related to subsequences was introduced by Simon in [11]. He defined the relation $\sim_k$ (called now Simon's congruence) as follows. Two words are $\sim_k$-equivalent if they have the same set of subsequences of length at most k. In [11], as well as in [9, Chapter 6], many fundamental properties (mainly of combinatorial nature) of $\sim_k$ are discussed; this line of research was continued in, e.g., [1, 2, 6–8] where the focus was on the properties of some special classes of this equivalence. From an algorithmic point of view, a natural decision problem and its optimisation variant stand out:

Problem 1 SIMK: Given two words s and t over an alphabet Σ, with $|s| = n$ and $|t| = n'$, with $n \geq n'$, and a natural number k, decide whether $s \sim_k t$.

Problem 2 MAXSIMK: Given two words s and t over an alphabet Σ, with $|s| = n$ and $|t| = n'$, with $n \geq n'$, find the maximum k for which $s \sim_k t$.

The problems above were usually considered assuming (although not always explicitly) the Word RAM model with words of logarithmic size. This is a standard computational model in algorithm design: for an input of size n, the memory consists of words consisting of $\Theta(\log n)$ bits. Basic operations (including arithmetic and bitwise boolean operations) on words take constant time, and any word can be accessed in constant time. In this model, the two input words are just sequences of integers, each integer stored in a single memory word. To be able to compare our results to the existing literature, we assume too the standard Word RAM model.

Both problems SIMK and MAXSIMK were considered thoroughly in the literature. In particular, Hebrard [5] presents MAXSIMK as computing a similarity measure between strings and mentions a solution of Simon [10] for MAXSIMK which runs in $O(|\Sigma|nn')$ (the same solution is mentioned in [4]). He goes on and improves this (see [5]) in the case when Σ is a binary alphabet: given two bitstrings s and t, one can find the maximum k for which $s \sim_k t$ in linear time. The problem of finding optimal algorithms for MAXSIMK, or even SIMK, for general alphabets was left open in [5, 10] as the methods used in the latter paper for binary strings did not seem to scale up. In [4], Garel considers MAXSIMK and presents an algorithm based on finite automata, running in $O(|\Sigma|n)$, which computes all *distinguishing words* u of minimum length, i.e., words which are factors of only one of the words s and t from the problem's statement. Finally, in an extended abstract from 2003 [12], Simon presents another algorithm based on finite automata solving MAXSIMK which runs in $O(|\Sigma|n)$, and he conjectures that it can be implemented in $O(|\Sigma| + n)$. Unfortunately, the last claim is only vaguely and insufficiently substantiated, and obtaining an algorithm with the claimed complexity seems to be non-trivial. Although Simon announced that a detailed description of this algorithm will follow shortly, we were not able to find it in the literature.

In [3], a new approach to efficiently solving SIMK was introduced. This idea was to compute, for the two given words s and t and the given number k, their shortlex forms: the words which have the same set of subsequences of length at most k as s and t, respectively, and are also lexicographically smallest among all words with the respective property. Clearly, $s \sim_k t$ if and only if the shortlex forms of s and t for k coincide. The shortlex form of a word s of length n over Σ was computed in $O(|\Sigma|n)$ time, so SIMK was also solved in $O(|\Sigma|n)$. A more efficient implementation of the ideas introduced in [3] was presented in [1]: the shortlex form of a word of length n over Σ can be computed in linear time, so SIMK can be solved in optimal linear time. By binary searching for the smallest k, this gives an $O(n \log n)$ time solution for MAXSIMK. This brings up the challenge of designing an optimal linear-time algorithm for non-binary alphabets.

We confirm Simon's claim from 2003 [12]. We present a complete algorithm solving MAXSIMK in linear time on Word RAM with words of size $\Theta(\log n)$. This closes the problem of finding an optimal algorithm for MAXSIMK. Our approach is not based on finite automata (as the one suggested by Simon), nor on the ideas from [1, 3]. Instead, it works as follows. Firstly, looking at a single word, we partition the respective word into k-blocks: contiguous intervals of positions inside the word, such that all suffixes of the word inside the same block have exactly the same subsequences of length at most k (i.e., they are $\sim_k$-equivalent). Since the partition in $(k+1)$-blocks refines the partition in k-blocks, one can introduce the *Simon-tree* associated to a word: its nodes are the k-blocks (for k from 1 to at most n), and each node on level k has as children exactly the $(k+1)$-blocks in which it is partitioned. We first show how to compute efficiently the Simon-tree of a word. Then, to solve MAXSIMK, we

show that one can maintain in linear time a connection between the nodes on the same levels of the Simon-trees associated to the two input words. More precisely, for all ℓ, we connect two nodes on level ℓ of the two trees if the suffixes starting in those blocks, in their respective words, have exactly the same subsequences of length at most ℓ. It follows that the value k required in MAXSIMK is the lowest level of the trees on which the blocks containing the first position of the respective input words are connected. Using the Simon-trees of the two words and the connection between their nodes, we can also compute in linear time a distinguishing word of minimal length for s and t. Achieving the desired complexities is based on a series of combinatorial properties of the Simon-trees, as well as on a rather involved data structures toolbox.

References

[1] L. BARKER, P. FLEISCHMANN, K. HARWARDT, F. MANEA, D. NOWOTKA, Scattered Factor-Universality of Words. *To appear in Proc. DLT 2020, CoRR* **abs/2003.04629** (2020).

[2] J. D. DAY, P. FLEISCHMANN, F. MANEA, D. NOWOTKA, k-Spectra of Weakly-c-Balanced Words. In: *Proc. DLT 2019*. Lecture Notes in Computer Science 11647, Springer, 2019, 265–277.

[3] L. FLEISCHER, M. KUFLEITNER, Testing Simon's congruence. In: *Proc. MFCS 2018*. LIPIcs 117, Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2018, 62:1–62:13.

[4] E. GAREL, Minimal Separators of Two Words. In: *Proc. CPM 1993*. Lecture Notes in Computer Science 684, Springer, 1993, 35–53.

[5] J.-J. HEBRARD, An algorithm for distinguishing efficiently bit-strings by their subsequences. *Theoretical Computer Science* **82** (1991) 1, 35–49.

[6] P. KARANDIKAR, M. KUFLEITNER, P. SCHNOEBELEN, On the index of Simon's congruence for piecewise testability. *Inf. Process. Lett.* **115** (2015) 4, 515–519.

[7] P. KARANDIKAR, P. SCHNOEBELEN, The Height of Piecewise-Testable Languages with Applications in Logical Complexity. In: *Proc. CSL 2016*. LIPIcs 62, 2016, 37:1–37:22.

[8] P. KARANDIKAR, P. SCHNOEBELEN, The height of piecewise-testable languages and the complexity of the logic of subwords. *Logical Methods in Computer Science* **15** (2019) 2.

[9] M. LOTHAIRE, *Combinatorics on Words*. Cambridge University Press, 1997.

[10] I. SIMON, An Algorithm to Distinguish Words efficiently by their Subwords. *unpublished* .

[11] I. SIMON, Piecewise testable events. In: *Autom. Theor. Form. Lang., 2nd GI Conf.*. LNCS 33, Springer, 1975, 214–222.

[12] I. SIMON, Words distinguished by their subwords (extended Abstract). In: *Proc. WORDS 2003*. TUCS General Publication 27, 2003, 6–13.

THEORIE-TAG 2020
Rudi Freund (Herausgeber): Theorietag 2020, Wien, 24. – 25.9.2020
TU Wien, Technischer Bericht, S. 33–36

Efficiently Testing Simon's Congruence

Joel D. Day[(A)] Florin Manea [(B)]

[(A)]Loughborough University, UK
J.Day@lboro.ac.uk

[(B)]Göttingen University, Computer Science Department, Germany
florin.manea@cs.uni-goettingen.de

A *word equation* is a tuple (α, β), usually written $\alpha \doteq \beta$, such that α and β are words comprised of letters from a *terminal alphabet* $\Sigma = \{\mathtt{a}, \mathtt{b}, \ldots\}$ and *variables* from a set $X = \{x, y, z, \ldots\}$. Solutions are substitutions of the variables for words in Σ^* making both sides identical. For example, one solution to the word equation $x\mathtt{ab}y \doteq y\mathtt{ba}x$ is given by $x \to \mathtt{b}$ and $y \to \mathtt{bab}$. A system of equations is a set of equations, and a solution to the system is a substitution for the variables which is a solution to all the equations in the system.

One of the most fundamental questions concerning word equations is the satisfiability problem: determining whether or not a word equation has a solution. Makanin [19] famously showed in 1977 that the satisfiability problem for word equations is decidable by giving a general algorithm. Since then, several further algorithms have been presented. Most notable among these are the algorithm given by Plandowski [22] which demonstrated that the satisfiability problem is in PSPACE, the algorithm based on Lempel-Ziv encodings by Plandowksi and Rytter [23], and the method of recompression by Jeż, which has since been shown to require only non-deterministic linear space [14, 15]. On the other hand, it is easily seen that solving word equations is NP-hard due to fact that the subcase when one side of the equation consists only of terminals is exactly the pattern matching problem which is NP-complete [3, 11]. It remains a long-standing open problem whether or not the satisfiability problem for word equations is contained in NP.

Recently, there has been elevated interest in solving more general versions of the satisfiability problem, originating from practical applications in e.g. software verification where several *string solving* tools capable of solving word equations are being developed [1, 2, 4, 5, 16] and database theory [12, 13], where one asks whether a given (system of) word equation(s) has a solution which satisfies some additional constraints. Prominent examples include requiring that the substitution for a variable x belongs to some regular language $\mathcal{L}_x$ (regular constraints), or that the lengths of the substitutions of the variables satisfy a set of given linear diophantine equations. Adding regular constraints makes the problem PSPACE complete (see [9,22,24], while it is another long standing open problem whether the satisfiability problem with length constraints is decidable. There are also many other kinds of constraints, however many lead to undecidable variants of the satisfiability problem [6, 17]. The main difficulty in dealing with additional constraints is that the solution-sets to word equations are often infinite sets with complex structures. For example, they are not parametrizable [21], and the set of lengths of solutions is generally not definable in Presburger arithmetic [18]. Thus, a better understanding of the solution-sets

and their structures is a key aspect of improving our ability to solve problems relating to word equations both in theory and practice.

Quadratic word equations (QWEs) are equations in which each variable occurs at most twice. For QWEs, a conceptually simple and easily implemented algorithm exists which produces a representation of the set of all solutions as a graph. Despite this, however, the satisfiability problem for quadratic equations remains NP-hard, even for severely restricted subclasses [7, 10], while inclusion in NP, and whether the satisfiability problem with length constraints is decidable, have remained open for a long time, just as for the general case.

The algorithm solving QWEs is based on iteratively rewriting the equation(s) according to some simple rules called *Nielsen transformations*. If there exists a sequence of transformations from the original equation to the trivial equation $\varepsilon \doteq \varepsilon$, then the equation has a solution. Otherwise, there is no solution. Hence the satisfiability problem becomes a reachability problem for the underlying rewriting transformation relation, which we denote $\Rightarrow_{NT}$. It is natural to represent this relation as a directed graph $\mathcal{G}^{\Rightarrow_{NT}}$ in which the vertices are word equations and the edges are the rewriting transformations. This has the advantage that the set of all solutions to an equation E corresponds exactly to the set of walks in the graph starting at E and finishing at the trivial equation $\varepsilon \doteq \varepsilon$. Consequently, the properties of the subgraph of $\mathcal{G}^{\Rightarrow_{NT}}$ containing all vertices reachable from E (denoted $\mathcal{G}^{\Rightarrow_{NT}}_{[E]}$) are also informative about the set of solutions to the equation. For example, in [21] a connection is made between the non-parameterisability of the solution set of E and the occurrence of combinations of cycles in the graph. Since equations with a parametrisable solution set are much easier to work with when dealing with additional constraints, this also establishes a connection between the structure of $\mathcal{G}^{\Rightarrow_{NT}}_{[E]}$ and the potential (un)decidability of variants of the satisfiability problem. Moreover, new insights into the structure and symmetries of these graphs are necessary for better understanding and optimising the practical performance of the algorithm.

We consider a subclass of QWEs called regular equations (RWEs) introduced in [20]. A word equation is *regular* if each variable occurs at most once on each side of the equation. Thus, for example, $x\mathtt{ab}y \doteq y\mathtt{ba}x$ is regular while $x\mathtt{ab}x \doteq y\mathtt{ba}y$ is not. Understanding RWEs is a vital step towards understanding the quadratic case, not only because they constitute a significant and general subclass, but also because many non-regular quadratic equations can exhibit the same behaviour as regular ones (consider, e.g. $zz \doteq x\mathtt{ab}yy\mathtt{ba}x$ for which all solutions must satisfy $z = x\mathtt{ab}y = y\mathtt{ba}x$). The satisfiability problem was shown in [7] to be NP-hard for RWEs, and shown to be NP-complete in [8] for some restricted subclasses of RWEs including the classes of regular-reversed and regular-ordered equations.

For RWEs E, we investigate the structure of the graphs $\mathcal{G}^{\Rightarrow_{NT}}_{[E]}$, and as a consequence, are able to describe some of their most important properties. In particular, we show that the diameter of the full graph $\mathcal{G}^{\Rightarrow_{NT}}_{[E]}$ is polynomial, and consequently, that the satisfiability problem for RWEs is NP-complete. This can be generalised to systems of equations satisfying a natural extension of the regularity property.

References

[1] P. A. ABDULLA, M. F. ATIG, Y. CHEN, L. HOLÍK, A. REZINE, P. RÜMMER, J. STENMAN, Norn: An SMT Solver for String Constraints. In: *Proc. Computer Aided Verification (CAV)*. Lecture Notes

in Computer Science (LNCS) 9206, 2015, 462–469.

[2] M. ALKHALAF, T. BULTAN, F. YU, STRANGER: An Automata-based String Analysis Tool for PHP. In: *Proc. Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*. Lecture Notes in Computer Science (LNCS) 6015, 2010.

[3] D. ANGLUIN, Finding patterns common to a set of strings. *Journal of Computer and System Sciences* **21** (1980), 46–62.

[4] C. BARRETT, C. L. CONWAY, M. DETERS, L. HADAREAN, D. JOVANOVIĆ, T. KING, A. REYNOLDS, C. TINELLI, CVC4. In: *Proc. Computer Aided Verification (CAV)*. Lecture Notes in Computer Science (LNCS) 6806, 2011, 171–177.

[5] M. BERZISH, V. GANESH, Y. ZHENG, Z3str3: A string solver with theory-aware heuristics. In: *Proc. Formal Methods in Computer-Aided Design (FMCAD)*. IEEE, 2017, 55–59.

[6] J. D. DAY, V. GANESH, P. HE, F. MANEA, D. NOWOTKA, The Satisfiability of Word Equations: Decidable and Undecidable Theories. In: I. POTAPOV, P. REYNIER (eds.), *In Proc. 12th International Conference on Reachability Problems, RP 2018*. Lecture Notes in Computer Science (LNCS) 11123, 2018, 15–29.

[7] J. D. DAY, F. MANEA, D. NOWOTKA, The Hardness of Solving Simple Word Equations. In: *Proc. Mathematical Foundations of Computer Science (MFCS)*. LIPIcs 83, 2017, 18:1–18:14.

[8] J. D. DAY, F. MANEA, D. NOWOTKA, Upper Bounds on the Length of Minimal Solutions to Certain Quadratic Word Equations. In: *Proc. Mathematical Foundations of Computer Science (MFCS)*. LIPIcs 138, 2019, 44:1–44:15.

[9] V. DIEKERT, A. JEŻ, W. PLANDOWSKI, Finding all solutions of equations in free groups and monoids with involution. *Information and Computation* **251** (2016), 263–286.

[10] V. DIEKERT, J. M. ROBSON, On Quadratic Word Equations. In: *Proc. 16th Annual Symposium on Theoretical Aspects of Computer Science, STACS*. Lecture Notes in Computer Science (LNCS) 1563, 1999, 217–226.

[11] A. EHRENFEUCHT, G. ROZENBERG, Finding a Homomorphism Between Two Words is NP-Complete. *Information Processing Letters* **9** (1979), 86–88.

[12] D. D. FREYDENBERGER, A Logic for Document Spanners. *Theory of Computing Systems* **63** (2019) 7, 1679–1754.

[13] D. D. FREYDENBERGER, M. HOLLDACK, Document Spanners: From Expressive Power to Decision Problems. *Theory of Computing Systems* **62** (2018) 4, 854–898.

[14] A. JEŻ, Recompression: A Simple and Powerful Technique for Word Equations. *Journal of the ACM* **63** (2016).

[15] A. JEŻ, Word Equations in Nondeterministic Linear Space. In: *Proc. International Colloquium on Automata, Languages and Programming (ICALP)*. LIPIcs 80, 2017, 95:1–95:13.

[16] A. KIEZUN, V. GANESH, P. J. GUO, P. HOOIMEIJER, M. D. ERNST, HAMPI: a solver for string constraints. In: *Proc. ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA)*. ACM, 2009, 105–116.

[17] A. W. LIN, P. BARCELÓ, String solving with word equations and transducers: towards a logic for analysing mutation XSS. In: *ACM SIGPLAN Notices*. 51, ACM, 2016, 123–136.

[18] A. W. LIN, R. MAJUMDAR, Quadratic Word Equations with Length Constraints, Counter Systems, and Presburger Arithmetic with Divisibility. In: S. K. LAHIRI, C. WANG (eds.), *In Proc. 16th International Symposium on Automated Technology for Verification and Analysis (ATVA)*. Lecture Notes in Computer Science (LNCS) 11138, Springer, 2018, 352–369.

[19] G. S. MAKANIN, The problem of solvability of equations in a free semigroup. *Sbornik: Mathematics* **32** (1977) 2, 129–198.

[20] F. MANEA, D. NOWOTKA, M. L. SCHMID, On the Complexity of Solving Restricted Word Equations. *International Journal of Foundations of Computer Science* **29** (2018) 5, 893–909.

[21] E. PETRE, An Elementary Proof for the Non-parametrizability of the Equation $xyz = zvx$. In: *Proc. 29th International Symposium on Mathematical Foundations of Computer Science (MFCS)*. Lecture Notes in Computer Science (LNCS) 3153, 2004, 807–817.

[22] W. PLANDOWSKI, Satisfiability of word equations with constants is in PSPACE. In: *Proc. Foundations of Computer Science (FOCS)*. IEEE, 1999, 495–500.

[23] W. PLANDOWSKI, W. RYTTER, Application of Lempel-Ziv Encodings to the Solution of Words Equations. In: *Proc. International Colloquium on Automata, Languages and Programming (ICALP)*. Lecture Notes in Computer Science (LNCS) 1443, 1998, 731–742.

[24] K. U. SCHULZ, Makanin's algorithm for word equations-Two improvements and a generalization. In: *International Workshop on Word Equations and Related Topics*. Springer, 1990, 85–150.

THEORIE-TAG 2020

Rudi Freund (Herausgeber): Theorietag 2020, Wien, 24. – 25.9.2020
TU Wien, Technischer Bericht, S. 37–40

The Edit Distance to k-Subsequence Universality

Pamela Fleischmann[(A)] Maria Kosche[(B)] Tore Koß[(B)]
Florin Manea [(B)] Stefan Siemer[(B)]

[(A)]Kiel University, Computer Science Department, Germany
fpa@informatik.uni-kiel.de

[(B)]Göttingen University, Computer Science Department, Germany
{maria.kosche,tore.koss,florin.manea,stefan.siemer}@cs.uni-goettingen.de

Abstract

A word v is a subsequence (also called scattered factor) of a word w if there exist (possibly empty) words $x_1, \ldots, x_{\ell+1}$ and $v_1, \ldots, v_\ell$ such that $v = v_1 \ldots v_\ell$ and $w = x_1 v_1 \ldots x_\ell v_\ell x_{\ell+1}$. That is, v is obtained from w by removing some of its letters.

The study of the relationship between words and their subsequences is a central topic in combinatorics on words and string algorithms, as well as in language and automata theory (see, e.g., the chapter *Subwords* by J. Sakarovitch and I. Simon in [23, Chapter 6] for an overview of the fundamental aspects of this topic, and [24] for a more practical perspective).

A major area of research related to subsequences is the study of the set of all subsequences of bounded length of a word, initiated by Simon in his PhD thesis [25]. In particular, Simon defined and studied (see [23, 26]) the relation $\sim_k$ (called now Simon's congruence) between words having exactly the same set of subsequences of length at most k. The surveys [20, 21] overview some of the extensions of Simon's seminal work from 1972 in various areas related to automata theory. Moreover, $\sim_k$ is a well-studied relation in the area of string algorithms too. The problems of deciding whether two given words are $\sim_k$-equivalent, for a given k, and to find the largest k such that two given words are $\sim_k$-equivalent (and their applications) were heavily investigated in the literature, see [9] and the references therein. This year, optimal solutions were given for both these problems [2, 9]. In [2] it was shown how to compute the shortlex normal form of a given word in linear time, i.e., the minimum representative of a $\sim_k$-equivalence class w.r.t. shortlex ordering. This can be directly applied to test whether two words are $\sim_k$-equivalent: they need to have the same shortlex normal form. In [9], a tree-like structure, called Simon-tree, was used to represent the equivalence classes induced by $\sim_k$ on the set of suffixes of a word, for all possible values of k, and then, given two words, a correspondence between their Simon-trees was constructed to compute in linear time the largest k for which they are $\sim_k$-equivalent.

Extending the algorithmic work on $\sim_k$, the following problem seems interesting: given two words w and u and an integer k, which is the minimal number of edit operations we need to perform on w to obtain a word v such that $v \sim_k u$? As edit operations we consider, as usual, insertion, deletion, and substitution of letters. Rephrasing, we ask how far (w.r.t. the *edit*

distance) are two words from being $\sim_k$-equivalent, or, how far is the word w from the set of all words which are $\sim_k$-equivalent to u. To this end, we can replace the target-word u by a set U of words of length k and ask for the minimal number of edit operations we need to perform on w to obtain a word v whose set of subsequences of length k is (or includes) U.

This direction of research is not new, and has always been a source of interesting problems. One of the most classical and well-understood string-problems is computing the edit distance between words [18], for which an optimal (up to poly-logarithmic factors) solution exists [1,19]. The problem of computing the edit distance between a word and a language, or between two languages, is also a well-studied problem, in various settings (see, e.g., [3, 5, 6, 13, 15]).

We make some initial steps in the study of the problems introduced above. While we do not solve their general form, we investigate one of their particular cases which seems interesting, meaningful, and well motivated. We follow the line of research of [2, 7, 16] and focus on a special $\sim_k$-class of words. A word w is *k-subsequence universal* (for short k-universal) with respect to an alphabet Σ if its set of subsequences of length k equals Σ^k. In the problems we consider, Σ will be the set $alph(w)$ of letters occurring in the input w. The maximum k for which a word w is k-universal is *the universality index* of w. In this context, we consider the problem of computing for a given word w and an integer k the minimal number of edit operations we need to perform on w in order to obtain a k-universal word. That is, we are interested in the edit distance from the input word w to the set of k-universal words w.r.t. $alph(w)$.

Before presenting our results, we briefly discuss the motivation of considering k-subsequence universal words. Firstly, using the name *universal* in this context is not unusual. The classical universality problem (see, e.g., [14]) is whether a given language L (over an alphabet Σ, and specified by an automaton or grammar) is equal to Σ^*. The works [10, 22] and the references therein discuss many variants of and results on the universality problem for various language generating and accepting formalisms. The universality problem was considered for words [8] and partial words [11] w.r.t. their factors. More precisely, one is interested in finding, for a given ℓ, a word w over an alphabet Σ, such that each word of length ℓ over Σ occurs exactly once as a contiguous factor of w. De Bruijn sequences [8] fulfil this property and have many applications in computer science or combinatorics, see [4, 11] and the references therein. In [16, 17] the authors define the notion of k-rich words in relation to the height of piecewise testable languages, a class of simple regular languages with applications in learning theory, databases theory, or linguistics (see [17] and the references therein). The class of k-rich words coincides with that of k-subsequence universal words. The study of k-subsequence universal words was continued, from a combinatorial point of view, in [2, 7]. So, it seems that investigating this class also from an algorithmic perspective is motivated by, fits in, and even enriches this well-developed and classical line of research.

Our results. Firstly, we note that when we want to increase the universality of a word by edit operations, it is enough to use only insertions. Similarly, when we want to decrease the universality of a word, it is enough to consider deletions. So, to measure the edit distance to the class of k-subsequence universal words, for a given k, it is enough to consider either insertions or deletions. However, changing the universality of a word by substitutions (both increasing and decreasing it) is interesting in itself as one can see the minimal number of substitutions needed to transform a word w into a k-universal word as the *Hamming distance* [12] between w and the set of k-universal words. Thus, we consider all these operations independently and propose efficient algorithms computing the minimal number of insertions, deletions, and substitutions,

respectively, needed to apply to a given word w in order to reach the class of k-universal words (w.r.t. the alphabet of w), for a given k. The time needed to compute these numbers is $O(nk)$ in the case of deletions and substitutions, as well as in the case of insertions if $k \in O(c^n)$ for some constant c (for even larger values of k we need to add roughly the time complexity of multiplying n and k).

Our algorithms are based, like most edit distance algorithms, on a dynamic programming approach. However, implementing such an approach within the time complexities stated above does not seem to follow directly from the known results on the word-to-word or word-to-language edit distance. In particular, we do not explicitly construct any k-universal word nor any representation (e.g., automaton or grammar) of the set of k-universal words, when computing the distance from the input word w to this set. Rather, we can compute the k-universal word which is closest w.r.t. edit distance to w as a byproduct of our algorithms. In our approach, we first develop several efficient data structures. Then, for each of the considered operations, we make several combinatorial observations, allowing us to restrict the search space of our algorithms, and creating a framework where our data structures can be used efficiently.

References

[1] A. BACKURS, P. INDYK, Edit Distance Cannot Be Computed in Strongly Subquadratic Time (Unless SETH is False). *SIAM J. Comput.* **47** (2018) 3, 1087–1097.

[2] L. BARKER, P. FLEISCHMANN, K. HARWARDT, F. MANEA, D. NOWOTKA, Scattered Factor-Universality of Words. In: N. JONOSKA, D. SAVCHUK (eds.), *Proc. DLT 2020*. Lecture Notes in Computer Science 12086, 2020, 14–28.

[3] K. BRINGMANN, F. GRANDONI, B. SAHA, V. V. WILLIAMS, Truly Sub-cubic Algorithms for Language Edit Distance and RNA-Folding via Fast Bounded-Difference Min-Plus Product. In: *Proc. FOCS 2016*. 2016, 375–384.

[4] H. Z. Q. CHEN, S. KITAEV, T. MÜTZE, B. Y. SUN, On universal partial words. *Electronic Notes in Discrete Mathematics* **61** (2017), 231–237.

[5] H. CHEON, Y. HAN, Computing the Shortest String and the Edit-Distance for Parsing Expression Languages. In: *Proc. DLT 2020*. Lecture Notes in Computer Science 12086, 2020, 43–54.

[6] H. CHEON, Y. HAN, S. KO, K. SALOMAA, The Relative Edit-Distance Between Two Input-Driven Languages. In: *Proc. DLT 2019*. Lecture Notes in Computer Science 11647, 2019, 127–139.

[7] J. D. DAY, P. FLEISCHMANN, F. MANEA, D. NOWOTKA, k-Spectra of Weakly-c-Balanced Words. In: *Proc. DLT 2019*. Lecture Notes in Computer Science 11647, 2019, 265–277.

[8] N. G. DE BRUIJN, A combinatorial problem. *Koninklijke Nederlandse Akademie v. Wetenschappen* **49** (1946), 758–764.

[9] P. GAWRYCHOWSKI, M. KOSCHE, T. KOSS, F. MANEA, S. SIEMER, Efficiently Testing Simon's Congruence. *CoRR* **abs/2005.01112** (2020).
`https://arxiv.org/abs/2005.01112`

[10] P. GAWRYCHOWSKI, M. LANGE, N. RAMPERSAD, J. O. SHALLIT, M. SZYKULA, Existential Length Universality. In: *Proc. STACS 2020*. LIPIcs 154, 2020, 16:1–16:14.

[11] B. GOECKNER, C. GROOTHUIS, C. HETTLE, B. KELL, P. KIRKPATRICK, R. KIRSCH, R. W. SOLAVA, Universal partial words over non-binary alphabets. *Theor. Comput. Sci* **713** (2018), 56–65.

[12] R. W. HAMMING, Error detecting and error correcting codes. *The Bell System Technical Journal* **29** (1950) 2, 147–160.

[13] Y. HAN, S. KO, K. SALOMAA, The Edit-Distance between a Regular Language and a Context-Free Language. *Int. J. Found. Comput. Sci.* **24** (2013) 7, 1067–1082.

[14] M. HOLZER, M. KUTRIB, Descriptional and computational complexity of finite automata - A survey. *Inf. Comput.* **209** (2011) 3, 456–470.
https://doi.org/10.1016/j.ic.2010.11.013

[15] R. JAYARAM, B. SAHA, Approximating Language Edit Distance Beyond Fast Matrix Multiplication: Ultralinear Grammars Are Where Parsing Becomes Hard! In: *Proc. ICALP 2017*. LIPIcs 80, 2017, 19:1–19:15.

[16] P. KARANDIKAR, M. KUFLEITNER, P. SCHNOEBELEN, On the index of Simon's congruence for piecewise testability. *Inf. Process. Lett.* **115** (2015) 4, 515–519.
https://doi.org/10.1016/j.ipl.2014.11.008

[17] P. KARANDIKAR, P. SCHNOEBELEN, The height of piecewise-testable languages and the complexity of the logic of subwords. *Logical Methods in Computer Science* **15** (2019) 2.

[18] V. I. LEVENSHTEIN, Binary codes capable of correcting deletions, insertions, and reversals. *Soviet Physics Doklady* **10** (1966) 8, 707–710.

[19] W. J. MASEK, M. PATERSON, A Faster Algorithm Computing String Edit Distances. *J. Comput. Syst. Sci.* **20** (1980) 1, 18–31.

[20] J. PIN, The Consequences of Imre Simon's Work in the Theory of Automata, Languages, and Semigroups. In: *Proc. LATIN 2004*. Lecture Notes in Computer Science 2976, 2004, 5.

[21] J. PIN, The influence of Imre Simon's work in the theory of automata, languages and semigroups. *Semigroup Forum* **98** (2019), 1–8.

[22] N. RAMPERSAD, J. SHALLIT, Z. XU, The Computational Complexity of Universality Problems for Prefixes, Suffixes, Factors, and Subwords of Regular Languages. *Fundam. Inf.* **116** (2012) 1-4, 223–236.
http://dl.acm.org/citation.cfm?id=2385073.2385090

[23] J. SAKAROVITCH, I. SIMON, Subwords. In: M. LOTHAIRE (ed.), *Combinatorics on Words*. chapter 6, Cambridge University Press, 1997, 105–142.

[24] D. SANKOFF, J. KRUSKAL, *Time Warps, String Edits, and Macromolecules The Theory and Practice of Sequence Comparison*. Cambridge University Press, 2000 (reprinted). Originally published in 1983.

[25] I. SIMON, *Hierarchies of events with dot-depth one - Ph.D. thesis*. University of Waterloo, 1972.

[26] I. SIMON, Piecewise testable events. In: *Autom. Theor. Form. Lang., 2nd GI Conf.*. LNCS 33, 1975, 214–222.

THEORIE-TAG 2020

Rudi Freund (Herausgeber): Theorietag 2020, Wien, 24. – 25.9.2020
TU Wien, Technischer Bericht, S. 41–44

An Overview over Insertion-Deletion Systems with Substitution

Martin Vu[(A)] Henning Fernau[(B)]

[(A)]FB3 - Informatik, Universität Bremen, Germany
{martin.vu@uni-bremen.de

[(B)]Universität Trier, Fachber. 4 – Abteilung Informatikwissenschaften
54286 Trier, Germany
fernau@uni-trier.de

Abstract

We give a short introduction to ins-del-sub systems, that is, insertion-deletion systems which are extended with substitutions as a further type of operations. Additionally we give an overview over the results presented in [6] and [7], in which the computational power of a number of ins-del-sub systems is investigated. For more details on the results we refer to [6] and [7].

1. Introduction

Insertion-deletion systems, or *ins-del systems* for short, are well-established as computational devices and as a research topic within Formal Languages throughout the past nearly 30 years, starting off with the PhD thesis of Lila Kari [2].

Corresponding to the mismatched annealing of DNA sequences, both the insertion and deletion operation have a strong biological background which led to their study in the molecular computing framework (cf. [4]). Insertion rules add a substring to a string, given a specified left and right context, while deletion rules remove a substring from a string given a specified left and right context. Studies about the context dependency of ins-del systems revolve around the question how much context information is (in-)sufficient for a ins-del system to reach computational completeness, given the ability to add a substring of length n and to delete a substring of length p.

The replacement of single letters (possibly within some context) by other letters by an operation called *substitution* is discussed in [1, 3], again from a biocomputing background. Interestingly, all theoretical studies on grammatical mechanisms involving insertions and deletions omitted including the substitution operation in their studies. We started stepping into this gap in [6, 7].

2. Basic Definitions

An *ins-del-sub system* is a -tuple $ID = (V, T, A, I, D)$, consisting of two alphabets V and T with $T \subseteq V$, a finite language A over V, a set of *insertion* rules I and a set of *deletion* rules D. Both sets of rules are formally defined as sets of triples of the form (u, a, v) with $a, u, v \in V^*$ and $a \neq \lambda$. We call elements occurring in T *terminal* symbols, while referring to elements of $V \setminus T$ as *nonterminals*. Elements of A are called *axioms*.

Let w_1uvw_2, with $w_1, u, v, w_2 \in V^*$, be a string. Applying the insertion rule $(u, a, v) \in I$ inserts the string $a \in V^*$ between u and v, which results in the string w_1uavw_2.

The application of a deletion rule $(u, a, v) \in D$ results in the removal of an substring a from the context (u, v). More formally let $w_1uavw_2 \in V^*$ be a string. Then, applying $(u, a, v) \in D$ results in the string w_1uvw_2.

We define the relation $\Longrightarrow$ as follows: Let $x, y \in V^*$. Then we write $x \Longrightarrow y$ if y can be obtained by applying an insertion or deletion rule to x. The *language generated by ID* is defined by $L(ID) = \{w \in T^* \mid \alpha \Longrightarrow^* w, \alpha \in A\}$.
The *size* of *ID* describes its complexity and is defined by a vector $(n, m, m'; p, q, q')$, where

$$n = \max\{|a| \mid (u, a, v) \in I\},\ p = \max\{|a| \mid (u, a, v) \in D\},$$
$$m = \max\{|u| \mid (u, a, v) \in I\},\ q = \max\{|u| \mid (u, a, v) \in D\}, \text{ and}$$
$$m' = \max\{|v| \mid (u, a, v) \in I\},\ q' = \max\{|v| \mid (u, a, v) \in D\}.$$

By $\mathrm{INS}_n^{m,m'}\mathrm{DEL}_p^{q,q'}$ we denote the family of languages generated by ins-del systems of size $(n, m, m'; p, q, q')$.[5]

We define *substitution rules* to be of the form $(u, a \to b, v);\ u, v \in V^*;\ a, b \in V$. Let $w_1uavw_2;\ w_1, w_2 \in V^*$ be a string over V. Then applying the substitution rule $(u, a \to b, v)$ allows us to substitute a single letter a with another letter b in the context of u and v, resulting in the string w_1ubvw_2. Formally, we define an *ins-del-sub system* to be a 6-tuple $ID_\varsigma = (V, T, A, I, D, S)$, where V, T, A, I and D are defined as in the case of usual ins-del systems and S is a set of substitution rules. Let $x = w_1uavw_2$ and $y = w_1ubvw_2$ be strings over V. We write $x \Longrightarrow_{\text{sub}} y$ iff there is a substitution rule $(u, a \to b, v)$. In the context of ins-del-sub systems, we write $\hat{\Longrightarrow}$ to denote any of the relations $\Longrightarrow$ or $\Longrightarrow_{\text{sub}}$. The *language generated by an ins-del-sub system* ID_ς is defined as $L(ID_\varsigma) = \{w \in T^* \mid \alpha \hat{\Longrightarrow}^* w,\ \alpha \in A\}$.

As with ins-del systems, the complexity of an ins-del-sub system $ID_\varsigma = (V, T, A, I, D, S)$ is measured via its *size*, which is defined as a tuple $(n, m, m'; p, q, q'; r, r')$, where n, m, m', p, q, and q' are defined as in ins-del systems while $r = \max\{|u| \mid (u, a \to b, v) \in S\}$ and $r' = \max\{|v| \mid (u, a \to b, v) \in S\}$. We refer to the family of languages generated by ins-del-sub systems of size $(n, m, m'; p, q, q'; r, r')$ by $\mathrm{INS}_n^{m,m'}\mathrm{DEL}_p^{q,q'}\mathrm{SUB}^{r,r'}$.

3. Results

The main focus of our investigation is the question when is the addition of substitution rules enough increase the computational of an ins-del system to computational completeness and when is it insufficient. The main results of [6] and [7] are presented in Table 1 and show which kind of substitution rules are sufficient for a non-complete systems to reach computational completeness completeness and which systems still cannot reach computational completeness.

family of insertion-deletion system with substitution rules	Language family relation	source
$\mathrm{INS}_1^{1,0}\mathrm{DEL}_1^{1,0}\mathrm{SUB}^{0,1}$	$=$ RE	[7]
$\mathrm{INS}_1^{1,0}\mathrm{DEL}_0^{0,0}\mathrm{SUB}^{0,1}$	$\subseteq$ CS	[7]
$\mathrm{INS}_1^{1,1}\mathrm{DEL}_1^{1,0}\mathrm{SUB}^{1,0}$	$=$ RE	[7]
$\mathrm{INS}_1^{1,0}\mathrm{DEL}_2^{0,0}\mathrm{SUB}^{0,1}$	$=$ RE	[7]
$\mathrm{INS}_2^{0,0}\mathrm{DEL}_2^{0,0}\mathrm{SUB}^{0,1}$	$=$ RE	[6]
$\mathrm{INS}_1^{0,0}\mathrm{DEL}_1^{0,0}\mathrm{SUB}^{1,1}$	$=$ RE	[6]
$\mathrm{INS}_1^{0,0}\mathrm{DEL}_0^{0,0}\mathrm{SUB}^{1,1}$	$=$ CS	[6]
$\mathrm{INS}_1^{1,0}\mathrm{DEL}_1^{0,0}\mathrm{SUB}^{0,0}$	$\subset$ CF	[6]
$\mathrm{INS}_1^{1,0}\mathrm{DEL}_1^{1,1}\mathrm{SUB}^{0,0}$	$\subset$ RE	[6]

Table 1: main results of [6] and [7]

For instance consider the family of ins-del systems of size $(1,1,0;1,1,0)$. These systems are not computationally complete. (In fact there are regular languages which even ins-del systems of size $(1,1,0;1,1,1)$ cannot generate [5, Theorem 5.3]). It is shown that extending such systems with substitution rules which have single letter right context is sufficient to reach computational completeness. More precisely: It is known that the family of ins-del systems of size $(1,1,1;1,1,1)$ is computationally complete. Ins-del-sub systems of size $(1,1,0;1,1,0;0,1)$ can simulate such systems and hence the family of ins-del-sub systems of size $(1,1,0;1,1,0;0,1)$ is computationally complete as well.

On the other hand extending ins-del systems of size $(1,1,0;0,0,0)$ with substitution rules which have single letter right context is still not enough to become equivalent to RE. In fact, it is shown that ins-del-sub systems which omit deletion rules cannot become more powerful CS.

We note that ins-del-sub systems are closed under *reversal* operator $^{\mathcal{R}}$, that is, it holds that $L \in \mathrm{INS}_n^{m,m'}\mathrm{DEL}_p^{q,q'}\mathrm{SUB}^{r,r'}$ if and only if $L^{\mathcal{R}} \in \mathrm{INS}_n^{m',m}\mathrm{DEL}_p^{q',q}\mathrm{SUB}^{r',r}$. Therefore, as RE is closed under reversal as well, it holds that $\mathrm{INS}_n^{m,m'}\mathrm{DEL}_p^{q,q'}\mathrm{SUB}^{r,r'} = \mathrm{RE}$ if and only if $\mathrm{INS}_n^{m',m}\mathrm{DEL}_p^{q',q}\mathrm{SUB}^{r',r} = \mathrm{Re}$. Hence, $\mathrm{INS}_1^{0,1}\mathrm{DEL}_1^{0,1}\mathrm{SUB}^{1,0} = \mathrm{RE}$ holds for instance as well.

References

[1] D. BEAVER, Computing with DNA. *Journal of Computational Biology* **2** (1995) 1, 1–7.

[2] L. KARI, *On insertions and deletions in formal languages*. Ph.D. thesis, University of Turku, Finland, 1991.

[3] L. KARI, DNA computing: Arrival of biological mathematics. *The Mathematical Intelligencer* **19** (1997) 2, 9–22.

[4] L. KARI, GH. PĂUN, G. THIERRIN, S. YU, At the crossroads of DNA computing and formal languages: Characterizing recursively enumerable languages using insertion-deletion systems. In: H. RUBIN, D. H. WOOD (eds.), *DNA Based Computers III*. DIMACS Series in Discrete Mathematics and Theretical Computer Science 48, 1999, 329–338.

[5] S. VERLAN, Recent Developments on Insertion-Deletion Systems. *The Computer Science Journal of Moldova* **18** (2010) 2, 210–245.
http://www.math.md/publications/csjm/issues/v18-n2/10288/

[6] M. VU, H. FERNAU, Insertion-Deletion Systems with Substitutions I. In: M. ANSELMO, G. D. VEDOVA, F. MANEA, A. PAULY (eds.), *Beyond the Horizon of Computability - 16th Conference on Computability in Europe, CiE 2020, Fisciano, Italy, June 29 - July 3, 2020, Proceedings*. Lecture Notes in Computer Science 12098, Springer, 2020, 366–378.
https://doi.org/10.1007/978-3-030-51466-2_33

[7] M. VU, H. FERNAU, Insertion-Deletion Systems with Substitutions II. In: *to appear in DCFS 2020*. 2020.

THEORIE-TAG 2020

Rudi Freund (Herausgeber): Theorietag 2020, Wien, 24. – 25.9.2020
TU Wien, Technischer Bericht, S. 45–48

Synchronizing Deterministic Push-Down Automata Can Be Really Hard

Henning Fernau[(A)] Petra Wolf[(A)] Tomoyuki Yamakami[(B)]

[(A)]Fachbereich 4 – Abteilung Informatikwissenschaften, Universität Trier, 54286 Trier, Germany
{fernau, wolfp}@eine-einrichtung.de

[(B)]University of Fukui, Faculty of Engineering, 3-9-1 Bunkyo, Fukui 910-8507, Japan
tomoyukiyamakami@gmail.com

Abstract

This is a summary of the results presented in [2], where the synchronization problem for finite state automata is generalized to deterministic real-time push-down automata and subclasses thereof, such as deterministic real-time one counter automata (DCAs) and partially blind DCAs. In [2] the complexity of these generalizations is considered.

1. Definitions

We refer to the empty word as ϵ. For a finite alphabet Σ we denote with Σ^* the set of all words over Σ and with $\Sigma^+ = \Sigma\Sigma^*$ the set of all non-empty words. For $i \in \mathbb{N}$ we set $[i] = \{1, 2, \ldots, i\}$. For $w \in \Sigma^*$ we denote with $|w|$ the length of w, with $w[i]$ for $i \in [|w|]$ the i'th symbol of w, and with $w[i..j]$ for $i, j \in [|w|]$ the factor $w[i]w[i+1]\ldots w[j]$ of w. We call $w[1..i]$ a prefix and $w[i..|w|]$ a suffix of w. The reversal of w is denoted by w^R, i.e., for $|w| = n$, $w^R = w[n]w[n-1]\ldots w[1]$.

We call $A = (Q, \Sigma, \delta, q_0, F)$ a *deterministic finite automaton* (DFA for short) if Q is a finite set of states, Σ is a finite input alphabet, δ is a transition function $Q \times \Sigma \to Q$, q_0 is the initial state and $F \subseteq Q$ is the set of final states. The transition function δ is generalized to words by $\delta(q, w) = \delta(\delta(q, w[1]), w[2..|w|])$ for $w \in \Sigma^*$. A word $w \in \Sigma^*$ is accepted by A if $\delta(q_0, w) \in F$ and the language accepted by A is defined by $\mathcal{L}(A) = \{w \in \Sigma^* \mid \delta(q_0, w) \in F\}$. We extend δ to sets of states $Q' \subseteq Q$ or to sets of letters $\Sigma' \subseteq \Sigma$, letting $\delta(Q', \Sigma') = \{\delta(q', \sigma') \mid (q', \sigma') \in Q' \times \Sigma'\}$. Similarly, we may write $\delta(Q', \Sigma') = p$ to define $\delta(q', \sigma') = p$ for each $(q', \sigma') \in Q' \times \Sigma'$.

The synchronization problem for DFAs (called DFA-SYNC) asks for a given DFA A whether there exists a synchronizing word for A. A word w is called a *synchronizing word* for a DFA A if it brings all states of the automaton to one single state, i.e., $|\delta(Q, w)| = 1$.

We call $M = (Q, \Sigma, \Gamma, \delta, q_0, \bot, F)$ a *deterministic push-down automaton* (DPDA for short) if Q is a finite set of states; the finite sets Σ and Γ are the input and stack alphabet, respectively; δ is a transition function $Q \times \Sigma \times \Gamma \to Q \times \Gamma^*$; q_0 is the initial state; $\bot \in \Gamma$ is the stack bottom

[(A)]The second author was supported by *DFG project FE 560/9-1*.

symbol which is only allowed as the first (lowest) symbol in the stack, i.e., if $\delta(q,a,\gamma)=(q',\gamma')$ and γ' contains $\perp$, then $\perp$ only occurs in γ' as its prefix and moreover, $\gamma=\perp$; and F is the set of final states. We will only consider *real-time* push-down automata and forbid ϵ-transitions, as can be seen in the definition.[1] Notice that the bottom symbol can be removed, but then the computation gets stuck.

Following [1], a *configuration* of M is a tuple $(q,\upsilon)\in Q\times\Gamma^*$. For a letter $\sigma\in\Sigma$ and a stack content υ with $|\upsilon|=n$ we write $(q,\upsilon)\xrightarrow{\sigma}(q',\upsilon[1..(n-1)]\gamma)$ if $\delta(q,\sigma,\upsilon[n])=(q',\gamma)$. This means that the top of the stack υ is the right end of υ. We also denote with $\longrightarrow$ the reflexive transitive closure of the union of $\xrightarrow{\sigma}$ over all letters in Σ. The input words on top of $\longrightarrow$ are concatenated accordingly, so that $\longrightarrow=\bigcup_{w\in\Sigma^*}\xrightarrow{w}$. The language $\mathcal{L}(M)$ accepted by a DPDA M is $\mathcal{L}(M)=\{w\in\Sigma^*\mid(q_0,\perp)\xrightarrow{w}(q_f,\gamma),q_f\in F\}$. We call the sequence of configurations $(q,\perp)\xrightarrow{w}(q',\gamma)$ the *run* induced by w, starting in q, and ending in q'. We might also call q' the *final state* of the run.

We will discuss three different concepts of synchronizing DPDAs. For all concepts we demand that a synchronizing word $w\in\Sigma^*$ maps all states, starting with an empty stack, to the same synchronizing state, i.e., for all $q,q'\in Q$: $(q,\perp)\xrightarrow{w}(\overline{q},\upsilon),(q',\perp)\xrightarrow{w}(\overline{q},\upsilon')$. In other words, for a synchronizing word all runs started on some states in Q end up in the same final state.

In addition to synchronizing the states of a DPDA we will consider the following two conditions for the stack content: (1) $\upsilon=\upsilon'=\perp$, (2) $\upsilon=\upsilon'$.

We will call (1) the *empty stack model* and (2) the *same stack model*. In the third case, we do not put any restrictions on the stack content and call this the *arbitrary stack model*.
As we are only interested in synchronizing a DPDA, we can neglect the start and final states.

It turns out that synchronizability of DPDAs is undecidable, which is in stark contrast to the situation with DFAs, where this problem is solvable in polynomial time. Hence, it is interesting to discuss deterministic variants of classical sub-classes of push-down automata. Here, we focus on one-counter languages and on linear languages and related classes.

A *deterministic (one) counter automaton* (DCA) is a DPDA where $|\Gamma\backslash\{\perp\}|=1$. Note that our DCAs can perform zero-tests by checking if the bottom-of-stack symbol is on top of the stack. As we will see that also with this restriction, synchronizability is still undecidable, we further restrict them to the *partially blind* setting [4]. This means in our formalization that a transition $\delta(q,\sigma,x)=(q',\gamma)$ either satisfies $x\neq\perp$, or x is a prefix of γ, i.e., $\gamma=x\gamma'$, and then both $\delta(q,\sigma,1)=(q',1\gamma')$ (for $\Gamma=\{1,\perp\}$) and $\delta(q,\sigma,\perp)=(q',\perp\gamma')$. The situation is even more delicate with one-turn or, more general, finite-turn DPDAs, whose further discussion and formal definition we defer to the specific section below.

We are now ready to define a family of synchronization problems, the complexity of which will be our main subject.

Definition 1.1 (SYNC-DPDA-EMPTY)
Given: DPDA $M=(Q,\Sigma,\Gamma,\delta,\perp)$.
Question: Does there exist a word $w\in\Sigma^*$ *that synchronizes* M *in the empty stack model?*

[1] Allowing ϵ-transitions, the considered automaton model is closer related to NFAs than to DFAs, and gives rise to adapt the concept of D1, D2, and D3 directing words (introduced in [5]).

For the same stack model, we refer to the synchronization problem above as SYNC-DPDA-SAME and as SYNC-DPDA-ARB in the arbitrary stack model. Variants of these problems are defined by replacing the DPDA in the definition above by a DCA, a deterministic partially blind counter automaton (DPBCA), or by adding turn restrictions, in particular, whether the automaton is allowed to make zero or one turns of its stack movement.

Finite-turn PDAs are introduced in [3]. From the formal language side, it is known that one-turn PDAs characterize the rather familiar family of linear context-free languages, usually defined via grammars. In our setting, the automata view is more interesting. We adopt the definition in [6]. For a DPDA M, an *upstroke* of M is a sequence of configurations induced by an input word w such that no transition decreases the stack-height. Accordingly, a *downstroke* of M is a sequence of configurations in which no transition increases the stack-height. A stroke is either an upstroke or a downstroke. Note that exchanging the top symbol of the stack is allowed in both an up- and a downstroke. A DPDA M is an n-turn DPDA if for all $w \in \mathcal{L}(M)$ the sequence of configurations induced by w can be split into at most $n+1$ strokes. Especially for 1-turn DPDAs, each sequence of configurations induced by an accepting word consists of one upstroke followed by a most one downstroke. There are two subtleties when translating this concept to synchronization: (a) there is no initial state so that there is no way to associate a stroke counter to a state, and (b) there is no language of accepted words that restricts the set of words on which the number of strokes should be limited. We therefore generalize the concept of finite-turn DPDAs to finite-turn synchronization for DPDAs in the following way. This opens up quite an interesting complexity landscape.

Definition 1.2 n-TURN-SYNC-DPDA-EMPTY
Given: DPDA $M = (Q, \Sigma, \Gamma, \delta, q_0, \bot, F)$.
Question: Is there a synchronizing word $w \in \Sigma^*$ *in the empty stack model, such that for all states* $q \in Q$*, the sequence of configurations* $(q, \bot) \xrightarrow{w} (\overline{q}, \bot)$ *consists of at most* $n+1$ *strokes?*

We call such a synchronizing word w an *n-turn synchronizing word* for M. We define n-TURN-SYNC-DPDA-SAME and n-TURN-SYNC-DPDA-ARB accordingly for the same stack and arbitrary stack models. Further, we extend the problem definition to real-time DCAs.

Results of the paper

class of automata/problem	empty stack model	same stack model	arbitrary stack model
DPDA	undecidable	undecidable	undecidable
1-Turn-Sync-DPDA	undecidable	undecidable	undecidable
0-Turn-Sync-DPDA	PSPACE-complete	undecidable	PSPACE-complete
DCA	undecidable	undecidable	undecidable
1-Turn-Sync-DCA	PSPACE-complete	PSPACE-complete	PSPACE-complete
0-Turn-Sync-DCA	PSPACE-complete	PSPACE-complete	PSPACE-complete
DPBCA	decidable	decidable	decidable

Table 1: Complexity status of the synchronization problem for different classes of deterministic real-time push-down automata in different stack synchronization modes as well as finite-turn variants of the respective synchronization problem.

We summarize our results in Table 1. In short, while already seemingly innocuous extensions of finite automata (with counters or with 1-turn push-downs) result in an undecidable synchronizability problem, some extensions do offer some algorithmic synchronizability checks, although nothing efficient.

References

[1] D. CHISTIKOV, P. MARTYUGIN, M. SHIRMOHAMMADI, Synchronizing Automata over Nested Words. *Journal of Automata, Languages and Combinatorics* **24** (2019) 2-4, 219–251.

[2] H. FERNAU, P. WOLF, T. YAMAKAMI, Synchronizing Deterministic Push-Down Automata Can Be Really Hard. In: J. ESPARZA, D. KRÁL' (eds.), *45th International Symposium on Mathematical Foundations of Computer Science, MFCS 2020, August 24-28, 2020, Prague, Czech Republic*. LIPIcs 170, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020, 33:1–33:15. https://doi.org/10.4230/LIPIcs.MFCS.2020.33

[3] S. GINSBURG, E. H. SPANIER, Finite-Turn Pushdown Automata. *SIAM Journal on Control* **4** (1966) 3, 429–453.

[4] S. A. GREIBACH, Remarks on Blind and Partially Blind One-Way Multicounter Machines. *Theoretical Computer Science* **7** (1978), 311–324.

[5] B. IMREH, M. STEINBY, Directable Nondeterministic Automata. *Acta Cybernetica* **14** (1999) 1, 105–115.

[6] L. G. VALIANT, *Decision Procedures for Families of Deterministic Pushdown Automata*. Ph.D. thesis, University of Warwick, Coventry, UK, 1973. http://wrap.warwick.ac.uk/34701/