数理解析研究所講究録
1533 巻 2007 年 24-38

Characterizations and Problems on LOGCFL

黒田覚 (Satoru Kuroda)
群馬県立女子大学
Gunma Prefectural Women's University

1 Introduction

The class LOGCFL is defined as those sets LOGSPACE-reducible to context-free languages and it turned out that the class has alternative characterizations based on Boolean circuits, word problems on groupoids and so on. The class is also known to enjoy approaches from logic such as descriptive complexity, bounded arithmetic and satisfiability problem. In this note we will review some of these results together with open problems.

2 What is LOGCFL?

First we will define the class LOGCFL. Throughout this note, we will consentrate on the string language. Let $\Sigma = \{0, 1\}$. We denote the set of finite strings over Σ by Σ^*. We denote the length of a string x by $|x|$.

We assume the reader the familiarity with basic definitions of formal language theory.

Definition 1 *A context-free grammar (CFG) is a quadraple $G = \langle N, \Sigma, P, S\rangle$ where,*

- *N is a finite set of nonterminal symbols,*
- *P is a finite set of rules of the form $A \overset{G}{\Rightarrow} \alpha$ where $A \in N$ and $\alpha \in (N \cup \Sigma)^*$,*
- *$S \in N$ is called the start symbol.*

A string $w \in \Sigma^*$ is generated by G if there exists a finite sequence such that

$$S \overset{G}{\Rightarrow} \alpha_1 \overset{G}{\Rightarrow} \alpha_2 \overset{G}{\Rightarrow} \cdots \overset{G}{\Rightarrow} w.$$

A set $A \subseteq \Sigma^*$ is a context-free language (CFL) if there exists a CFG G such that

$$\forall w \in \Sigma^*(w \in A \Leftrightarrow w \text{ is generated by } G).$$

We will also assume basic notions of Turing machines and computational complexity.

Definition 2 *Let $A, B \subseteq \Sigma^*$. A is logspace many-one reducible to B (denoted by $A \leq_m^{log} B$) if there exists a logspace bounded computable function f such that*

$$\forall w \in \Sigma^*(w \in A \Leftrightarrow f(w) \in B).$$

Now we can define the class LOGCFL as

Definition 3 $LOGCFL = \{A \subseteq \Sigma^* : \exists B : CFL(A \leq_m^{log} B)\}$.

Remark. The class LOGCFL is a "uniform" class, that is all sets in LOGCFL are recursive.

3 Circuit characterization

The class LOGCFL has several alternative characterizations. First we will give a circuit based characterzation established by H Venkateswaran [11].

A Boolean circuit with n inputs is a directed acyclic graph C such that ehac vertices are labeled as follows:

- a vertice of indegree 0 is called an input and labeled by either one of $x_1, \ldots, x_n, \bar{x_1}, \ldots, \bar{x_n}$,
- a vertice of indegree ≥ 1 is callled a (inner) gate and labeled by either one of $\wedge$ and $\vee$.

A gate with outdegree 0 is called an output of the circuit. In this note we consider only single-output circuits.

The computation of a circuit C on input x with $|x| = n$ is done by assigning the ith bit of x to each input x_i, the complement of ith bit of x to each input $\bar{x_i}$, and evaluating each inner gate using the truth table of its label. Thus a circuit with n input computes a predicate on Σ^n.

On the other hand, any infinite set $A \subseteq \Sigma^*$ contains strings of arbitrary lengths. So we need infinitely many circuits to decide it. We formalize this as follows:

Definition 4 *A set $A \subseteq \Sigma^*$ is computed by a circuit family $C = \{C_n\}_{n\in\omega}$ if for all $n \in \omega$, C_n is a circuit with n inputs and*

$$\forall x \in \Sigma^n (x \in A \Leftrightarrow C_n \text{ accepts } x.$$

We are interested in deciding the complexity of the circuit family which computs a given $A \subseteq \Sigma^*$. To measure the complexity, we introduce the following notations:

Let C be a circuit. The size of C (denoted by $size(C)$) is the number of gates in C. The depth of C (denoted by $depth(C)$) is the maximum of lengths of paths from an input to the output. Note that the size and the depth of a circuit family are functions of the length n of inputs.

A circuit family $\{C_n\}_{n\in\omega}$ is called bounded fan-in if indegrees of gates in C_n are bounded by some constant. Otherwise it is called unbounded fan-in. Further more it is called semi-unbounded fan-in if indegrees of $\wedge$-gates are bounded by a constant while those of $\vee$-gates are not bounded.

Now we define circuit complexity classes.

Definition 5 *Let $i \in \omega$. We define the class AC^i to be the class of sets $A \subseteq \Sigma^*$ which can be computed by an unbounded fan-in family of circuits with depth $O((\log n)^i)$ and size $n^{O(1)}$. The classes NC^i and SAC^i are defined in the same manner as AC^i for bounded fan-in and semi-unbouded fan-in circuits.*

Note that these circuit classes are non-uniform. In fact, there exists an NC^0 circuit family computing a non-recursive set. In roder to consider inclusion relations between circuit classses and classes defined by other computation models, we need uniform version of these classes. So we define the following uniform notion.

Definition 6 *Let $\{C_n\}_{n\in\omega}$ be circuit family. The Direct Connection Language (DCL) is the set of $(g, h, l, 0^n)$ where g receives an input from h in C_n and l is a label of g. The family $\{C_n\}_{n\in\omega}$ is U_{E^*}-uniform if its DCL is DLOGTIME-computable.*

We assume U_{E^*}-uniformity for any circuits class unless otherwise stated.

For the relation between LOGCFL and circuit classes, Venkateswaran [11] proved the following suprising result.

Theorem 1 (Venkateswaran [11]) *The class LOGCFL is equal to U_{E^*}-uniform SAC^1.*

This gives the following inclusions neither of which is known whether it is proper or not:

$$L \subseteq NL \subseteq LOGCFL = SAC^1 \subseteq AC^1.$$

The problem whether LOGCFL is closed under complementation operation is not obvious and was proved by Borodin et. al. The proof is the application of the inductive counting argument by Immerman [6] and Szelepscényi [10].

Theorem 2 (Borodin et.al. [1]) *The non-uniform SAC^1 is closed under complementation.*

(Proof Idea). Let C be an SAC^1 circuit with n inputs whose depth and width are $|\alpha(n)|$ and $\beta(n)$ respectively. We can transform C so that the resulting circuit has at most polynomial increase of width and satisfies the following conditions:

1. synchronicity: all inputs are on row 0, output is on row $2|\alpha(n)|$, and all gates on row k receive inputs from row $k-1$,
2. strict alternation: for $0 < k < |\alpha(n)|$ all gates on row $2k$ are AND gates and all gates on row $2k-1$ are OR gates.

Based on this reformulation of the original circuit, we construct the inductive counting circuit to compute its complement. The general idea of the construction is to add two kinds of gates. The first kind is called the "contingent complement gate". Let g be a gate in C and c be a number less than or equal to the width of C. Then a contingent complement gate $cc(g|c)$ outputs 1 if and only if g outputs 0, provided that c is the number of 1's in the input rows to g.

The second kind is the "counting gate",denoted by $COUNT(c,k)$, which outputs 1 if and only if the number of 1's in the $2k$-th row of C is equal to c.

Thus computing the disjunct of all $cc(g|c) \wedge COUNT(c,k)$ for $0 \leq c \leq width(C)$ computes the complement of the original gate g on row $2k$.

Next let us see how each gates above are defined. The definition of The contingent complement gate $cc(g|c)$ differs according to the row of g. If g is on row $2k$ (that is, g is an AND gate) then

$$cc(g|c) = cc(h_0|c) \vee cc(h_1|c)$$

where h_0 and h_1 are inputs to g. If g is on row $2k-1$ (that is, g is an OR gate) then

$$cc(g|c) = c\text{-}THRES\{h : h \text{ is the non-input of } g \text{ on row } 2k-2\},$$

where c-THRES outputs 1 if and only if the number of 1's in its inputs are more than or equal to c.

The counting gate $COUNT(c,k)$ is inductively defined as

$$COUNT(c,k) = \bigvee_{d=0}^{\beta(n)} AND(COUNT(d,k-1), TH1(c,k), TH0(c,k,d)),$$

where $TH1(c,k)$ is the c-THRES of all original gates (that is gates in C) on row $2k$ and $TH0(c,k,d)$ is the $(c-d)$-THRES of all $cc(g|c)$ where g is the original gate on row $2k$.

Finally, let g_{out} be the output gate of the original circuit. Then the complement can be computed by

$$\bigvee_{0 \le c \le \beta(n)} (cc(g_{out}|c) \wedge COUNT(c, \alpha(n)-1)).$$

Note that throughout the construction, gates like $cc(g|c)$, $TH1$ and $TH0$ are defined using arbitrary threshold gates and replaced by SAC^1 subcircuits which is given by

$$\begin{aligned} &c\text{-}THRES(x_1,\ldots,x_n) \\ &= \bigvee_{k=0}^{c} (k\text{-}THRES(x_1,\ldots,x_{\lfloor \frac{n}{2} \rfloor}) \wedge (c-k)\text{-}THRES(x_{\lfloor \frac{n}{2} \rfloor+1},\ldots,x_n)). \end{aligned}$$

Corollary 1 *The class LOGCFL is closed under complementation.*

This closure property guarantees the robustness of the corresponding function class. Let $\mathcal{F}_{LOGCFL}$ be the class of functions $f : \Sigma^* \to \Sigma^*$ such that

- there exists a polynomial p such that $|f(\vec{x})| \le p(|\vec{x}|)$.
- the predicate "the ith bit of $f(\vec{x})$ is 1" is computed in LOGCFL.

Then we have

Corollary 2 *The class $\mathcal{F}_{LOGCFL}$ is closed under complementation.*

4 Logic for LOGCFL

It is a fundamental problem in comptational complexity to give logical characterizations, or to find logical properties for complexity classes.

4.1 Descriptive characterization

Descriptive complexity can be seen as an application of model theory to complexity theory. The intuitive idea is as follows: given an algorithm A for a problem, we express it by a logical formula φ_A. Similarly an input to the algorithm can be exressed by a structure $\mathcal{A}$ with universe $\{0, 1, \ldots, n-1\}$ where n is the length of the input so that

$$\mathcal{A} \models \varphi_A \Leftrightarrow A \text{ accepts the input expressed by } \mathcal{A}.$$

Generally speaking, the computational complexity of A corresponds to the logical complexity of φ_A.

To formalize this correspondence we define

Definition 7 *Let Φ be a class of formulae and C be a complexity class. We say that Φ captures C (over arithmetical structures) if*

1. *for all $\varphi \in \Phi$ there exists an algorithm in C which checks whether $\mathcal{A} \models \Phi$ where $\mathcal{A}$ is an arithmetical structure.*

2. *for all $L \in C$ there exists $\varphi \in \Phi$ such that*

$$\mathcal{A} \models \varphi \Leftrightarrow bin(\mathcal{A}) \in L$$

where $bin(\mathcal{A})$ is a suitable binary coding of $\mathcal{A}$.

For example following correspondence is known:

- The class of existential second order formulae $SO\exists$ captures the class NP.

- The class of first order formulae FO captures the class AC^0.

A descriptive characterization for the class $LOGCFL$ is first given by Vollmer et. al. by way of generalized quantifiers expressing word problems for finite groupoids. See [12].

A similar characterization using the alternation path in SAC^1 circuits are given in Kuroda [9].

4.2 Satisfiability problem for LOGCFL

Satifiability problems are typical complete problems for several complexity classes. For example, the satisfiability for propositional formulae called SAT is NP-complete. Also restrictions to the form of propositional formulae leads to complete problems: the satisfiability for Horn formula (Horn-SAT) and 2CNF (2SAT) are complete for P and NL respectively.

The satisfiablity for LOGCFL was given by Gottlob et. al. [5] in a slightly different form. Boolean conjunctive queries are regarded as seaching witnesses to given conditions over relational databeases. Consider the following example:

Example 1 *Suppose that we have a relational database with the following relational schema:*

$$\begin{aligned} &\textit{u-tokyo}(\textit{Student}, \textit{Year}, \textit{Faculty}),\\ &\textit{kyoto-u}(\textit{Student}, \textit{Year}, \textit{Faculty}),\\ &\textit{sisters}(\textit{Student}1, \textit{Student}2). \end{aligned}$$

We can consider the following queries over this database.

$$\begin{aligned} Q_1 &: \textit{ans} \leftarrow \textit{u-tokyo}(S1, Y, F) \wedge \textit{kyoto-u}(S2, Y, F') \wedge \textit{sisters}(S1, S2),\\ Q_2 &: \textit{ans} \leftarrow \textit{u-tokyo}(S1, Y, F) \wedge \textit{kyoto-u}(S2, Y', F') \wedge \textit{sisters}(S1, S2), \end{aligned}$$

In general, queries are of the form Q : *ans* $\leftarrow$ *body* and if *body* is a conjunction of atomic formulae then the query is called a *Boolean conjunctive query.*

A Boolean conjunctive query Q is expressed as a hypergraph, that is, the pair $G = (V, E)$ where

$$\begin{aligned} V &= \text{the set of variables occurring in } Q,\\ E &= \{(v_1, \ldots, v_k) : \textit{name}(v_1, \ldots, v_k) \text{ is a conjunct in } Q\}. \end{aligned}$$

A Boolean conjunctive query is called cyclic (acyclic) if the corresponding hypergraph is cyclic (resp. acyclic). Note that Q_1 and Q_2 in example 1 are cyclic and acyclic respectively.

Now let us briefly formalize this argument. A relatinal schema R consists of a name r of a relation and a finite ordered list of attributes. In Example 1, *sister*(*Student*1, *Student*2) is a relational scheme with a name *sister* and attributes *Student*1 and *Student*2.

For an attrubute A, $Dom(A)$ be a countable domain of atomic values.

Let $R = (A_1, \ldots, A_k)$ be a relation. A relation instance is a subset $S \subseteq Dom(A_1) \times \cdots \times Dom(A_k)$ so that $|S| < \omega$. A database schema DS consists of a finite set of relation schemata.

Database **db** over $DS = \{R_1, \ldots, R_m\}$ consists of:

- relation instance $r_1, \ldots, r_m$ for $R_1, \ldots, R_m$, and
- universe $U \subseteq \bigcup_{R_i(A_1^i,\ldots,A_{k_i}^i) \in DS} Dom(A_1^i) \cup \cdots \cup Dom(A_{k_i}^i)$,

such that all data values in **db** are from U.

Now we define conjunctive query on database schema as follows:

Definition 8 *Let $DS = \{R_1, \ldots, R_m\}$ be a database schema. A conjunctive query Q on RS consists of a rule of the form*

$$Q : ans(u) \rightarrow r_1(u_1) \wedge \cdots r_n(u_n)$$

where $r_1, \ldots, r_n$ are relation names in DS, ans is a relation name not in DS, and $u_1, \ldots, u_n$ are terms of appropriate length.

Next we define the satisfaction relation for conjunctive queries. For a conjunctive query Q we define $var(Q)$ to be the set of variables occurring in it and $atom(Q)$ to be the set of atomic formulae occurring in its body.

Definition 9 *The answer of a conjunctive query Q on a database* **db** *with associated universe U consists of a relation ans whose arity is equal to the length of u defined as follows:*

> *ans contains all tuples $u\mathcal{S}$ where $\mathcal{S} : var(Q) \rightarrow U$ is a substitution replacing each variable in $var(Q)$ by a value of U so that $r_i(u_i)\mathcal{S} \in$* **db** *for all $1 \leq i \leq n$.*

We say that Q evaluates to true on **db** *if there exists a substitution $\mathcal{S}$ such that $r_i(u_i)\mathcal{S} \in$* **db** *for all $1 \leq i \leq n$.*

We say that Q is a Boolean conjunctive query if its head does not contain variables.

As stated above, we can associate a hypergraph $H(Q) = (V, E)$ for each conjunctive query Q which is defined as:

- V is the set of variables occurring in Q,
- if $r_i(u_i)$ is an atomic formula occurring in the body of Q then E contains a hypergraph consisting of all variables in u_i.

Q is an acyclic conjunctive query if $H(Q)$ is acyclic.

Finally we define the satisfiability of (acyclic) Boolean conjunctive query as follows:

BCQ: Given a database **db** and a Boolean conjunctive query Q. Decide whether Q evaluates to true on **db**.

ABCQ: Given a database **db** and an acyclic Boolean conjunctive query Q. Decide whether Q evaluates to true on **db**.

Note that *BCQ* was previously known to be NP-complete. So the following result is rather surprising:

Theorem 3 (Gottlob et.al. [5]) *The problem ABCQ is complete for LOGCFL under logspace reduction.*

4.3 Bounded arithmetic

The main idea of bounded arithmetic is to characterize a given function class as provably total functions in a weak subsystems of Peano Arithmetic or its variants. More precisely , let Φ be a class of formulae.. Then a function f is said to be Φ-definable in a system T if there exists a formula $\varphi \in \Phi$ such that

$$T \vdash (\forall x)(\exists! ty)\varphi(x, y),$$
$$\mathbb{N} \models (\forall x)\varphi(x, f(x)).$$

S. Buss [2] was the first to give such systems for levels of the polynomial time hierarchy. Namely, he defined a series of systems S_2^i $(i \geq 0)$ such that

Theorem 4 *For $i \geq 1$, a function is polynomial time computable with an oracle from Σ_{i-1}^p if and only if it is Σ_i^b definable in S_2^i.*

Buss' systems are defined in the language of natural numbers. On the other hand, S. Cook and his students developed a theory of binary strings called two-sort systems and based on this formulation, A. Kolokolova [8] gave a relation between descriptive complexity and two-sort bounded arithmetic.

Furthermore, the author [9] used Kolokolova's argument to define a two-sort system whose provably total functions are exactly those bitwise computable in LOGCFL. In the following we briefly review this work.

The language L_2 of two sort systems has two kinds of variables, namely

- number variables denoted by lower case letters $x, y, z, \ldots$, and
- string variables denoted by upper case letters $X, Y, Z, \ldots$

and contains function symbols $Z(x) = 0$, $s(x) = x + 1$, $x + y$, $x \cdot y$, $|X|$ (the length of string X) and a relation symbol $x \leq y$. Terms and formulae are built up in the usual manner. The standard model of L_2 is the pair $(\mathbb{N}, \Sigma^*)$.

Quantifiers of the form $(\forall x \leq t)$ and $(\exists x \leq t)$ are called bounded number quantifiers where t is a term not including x. Quantifiers of the form $(\forall X \leq t)$ and $(\exists X \leq t)$ are called bounded string quantifiers whose intended meanings are

$$(\forall X \leq t)\varphi \equiv (\forall X)(|X| \leq t \rightarrow \varphi),$$
$$(\exists X \leq t)\varphi \equiv (\exists X)(|X| \leq t \wedge \varphi)$$

where t is a term not containing X.

We define Σ_0^B to be the class of L_2-formulae in which all quantifiers are bounded number quantifiers and Σ_1^B be the class of L_2-formulae in which all quantifiers are either bounded number quantifiers, positive occurrences of bounded existential string quantifiers, or negative appearances of bounded universal string quantifiers.

We augment the language L_2 by introducing an operator which applies to L_2-formulae. Let $\varphi(g, h, z, \bar{x}, \bar{Y})$ be an L_2-formula with free variables as shown and $\alpha(z)$ and $\beta(z)$ be L_2 terms with the only parameter z. Then we introduce a new relation

$$\mathbf{Q}^{SAC}[\varphi(g, h, z, \bar{x}, \bar{Y}), \alpha(z), \beta(z)](t, n, \bar{p}, \bar{Q}, X) \qquad (*)$$

For a class Φ of L_2-formulae we define $QL_2(\Phi)$ to be L_2 extended by $\mathbf{Q}^{SAC}[\varphi]$ for all $\varphi \in \Phi$. Let $\mathbf{Q}^{SAC}(\Phi)$ be the class of formulae of the form $(*)$ where $\varphi \in \Phi$.

The intended meaning of $(*)$ is as follows: Let $C_{\varphi,\alpha,\beta}$ be an SAC^1 circuit with n inputs whose depth and width are $|\alpha(n)|$ and $\beta(n)$ respectively, and its direct connection language is given by $\varphi(g, h, n, \bar{p}, \bar{Q})$, that is, h receives an input from g if and only if $\varphi(g, h, n, \bar{p}, \bar{Q})$. Here g and h denote the positions of gates, and throughout the paper we identify gates with their positions. Then $\mathbf{Q}^{SAC}[\varphi(g, h, z, \bar{x}, \bar{Y}), \alpha(z), \beta(z)](t, n, \bar{p}, \bar{Q}, X)$ holds if and only if there exists an "alternating path" Z from the input X to the gate t which satisfies:

- Z contains no input node whose value is 0,
- Z contains at least one node whose value is 1,
- if g is an AND gate and all its offsprings are in Z then so is g,
- if g is an OR gate and some of its offsprings is in Z then so is g,
- the gate t is at the top of Z.

Note that the circuit $C_{\varphi,\alpha,\beta}$ accepts the input X if and only if there exists such an alternating path Z.

Since we are interested in the FO-uniform SAC^1, we will concentrate on the language $QL_2(\Sigma_0^b)$.

Now we will formalize the above conditions to give the defining axiom for the operator $\mathbf{Q}^{SAC}$. First of all, the defining axiom for $\mathbf{Q}^{SAC}$ must contain formulae saying that φ defines a circuit. Remember that the combinatorial structure of circuits is that of directed acyclic graphs. In our setting, circuits are coded by the tuple $\langle \varphi(g,h,z,\bar{x},\bar{Y}), \alpha(z), \beta(z)\rangle$. This is explained as follows:

- each gate in $C_{\varphi,\alpha,\beta}$ is assigned an unique value $g < |\alpha(n)| \cdot \beta(n)$,
- for $g, h < |\alpha(n)| \cdot \beta(n)$, g is an input of h if $\varphi(g,h,z,\bar{x},\bar{Y})$, and
- $C_{\varphi,\alpha,\beta}$ is divided into $|\alpha(n)|$ rows, each containing $\beta(n)$ gates.

So two gates g and h are in the same row if and only if $\left\lfloor \frac{g}{\beta(n)} \right\rfloor = \left\lfloor \frac{h}{\beta(n)} \right\rfloor$.

For the sake of readability, we define $\mathrm{row}_\beta(g) = \left\lfloor \frac{g}{\beta(n)} \right\rfloor$ and $\mathrm{col}_\beta(g) = Rem(g, \beta(n))$ to denote the vertical and horizontal positions of the gate g respectively. We omit the subscript β if it is clear from the context. Note that these functions are Σ_0^B definable in V^0.

Now φ gives an edge relation for a directed acyclic graph if whenever $g < |\alpha(n)| \cdot \beta(n)$ receives an input from $h < |\alpha(n)| \cdot \beta(n)$ then $\mathrm{row}(g) \leq l(h)$. We define this formally as

$$
\begin{aligned}
&Circuit^{\alpha,\beta}_{\varphi(g,h,z,\bar{x},\bar{Y})}(n,\bar{p},\bar{Q}) \\
&\Leftrightarrow (\forall g < |\alpha(n)| \cdot \beta(n))(\forall h < |\alpha(z)| \cdot \beta(z))(\varphi(g,h,n,\bar{p},\bar{Q}) \\
&\quad \rightarrow \mathrm{row}_\beta(g) < \mathrm{row}_\beta(h)).
\end{aligned}
$$

We will not add formulae to explain what semi-unbounded circuits are. Instead, we may consider that any $C_{\varphi,\alpha,\beta}$ is an SAC^1 circuit by assuming that

- gates with fan-in more than two are OR gates, and
- gates with fan-in less than or equal to two are AND gates.

Note that negations are only applied to the input row for SAC^1 circuits. So we assume that inputs already contain the complement for each bit:

$$
\begin{aligned}
&Input(X,n) \\
&\Leftrightarrow |X| = 2n+2 \wedge (\forall x < n)(X[x] \leftrightarrow \neg X[x+n]) \wedge X[2n] \wedge \neg X[2n+1].
\end{aligned}
$$

This means that the actual input is given by the first n bits and next n bits gives the bitwise complement of the input. For a technical reason, we add two more bits at positions $2n$ and $2n+1$ which always evaluates to 1 and 0 respectively.

Now we give the formal definition of alternating paths on SAC^1 circuits. The formula $APATH_{\varphi}^{\alpha,\beta}(t,n,\bar{p},\bar{Q},X,Z)$ is a conjunction of the following formulae:

- No inputs with the value zero are in the path:

$$(\forall g < |\alpha(n)| \cdot \beta(n))(\mathrm{row}_{\beta}(g) = 0 \wedge \neg X[g] \rightarrow \neg Z[g]),$$

- there exists at least one input with the value one in the path:

$$(\exists g < |\alpha(n)| \cdot \beta(n))(\mathrm{row}_{\beta}(g) = 0 \wedge X[g] \wedge Z[g]),$$

- if g is an OR gate and either one of its offsprings is in the path then so is g:

$$\begin{array}{l}(\forall g < |\alpha(n)| \cdot \beta(n)) \\ (\exists h_0 < |\alpha(n)| \cdot \beta(n))(\exists h_1 < |\alpha(n)| \cdot \beta(n))(\exists h_2 < |\alpha(n)| \cdot \beta(n)) \\ (\varphi(h_i,g,n,\bar{p},\bar{Q}) \wedge h_0 \neq h_1 \wedge h_0 \neq h_2 \wedge h_1 \neq h_2 \\ (\exists h < |\alpha(n)| \cdot \beta(n))\varphi(h,g,n,\bar{p},\bar{Q}) \wedge Z[h] \leftrightarrow Z[g]),\end{array}$$

- if g is an AND gate and both of its offsprings are in the path then so is g:

$$\begin{array}{l}(\forall g < |\alpha(n)| \cdot \beta(n))(\exists h_0 < |\alpha(n)| \cdot \beta(n))(\exists h_1 < |\alpha(n)| \cdot \beta(n)) \\ (\varphi(h_0,g,n,\bar{p},\bar{Q}) \wedge \varphi(h_1,g,n,\bar{p},\bar{Q}) \wedge \\ (\forall h < |\alpha(n)| \cdot \beta(n))(\varphi(h,g,n,\bar{p},\bar{Q}) \rightarrow (h = h_0 \vee h = h_1)) \\ \wedge Z[h_0] \wedge Z[h_1] \rightarrow Z[g]),\end{array}$$

- the gate t is at the top of Z:

$$Z[t] \wedge (\forall g)(g \neq t \wedge Z[g] \rightarrow (\exists h)\varphi(g,h,z,\bar{x},\bar{Y}) \wedge Z[h]).$$

So $APATH_{\varphi}^{\alpha,\beta}(t,n,\bar{p},\bar{Q},X,Z)$ says that there exists an alternating path in $C_{\varphi,\alpha,\beta}$ starting from the gate t.

Definition 10 *Let Ax-*$\mathbf{Q}^{SAC}(\varphi)$ *be the following axiom:*

$$\begin{array}{l}\mathbf{Q}^{SAC}[\varphi(g,h,\bar{x},\bar{Y}),\alpha(z),\beta(z)](t,n,\bar{p},\bar{Q}) \leftrightarrow \\ (Circuit_{\varphi}^{\alpha,\beta}(n,\bar{p},\bar{Q}) \wedge Input^{\alpha,\beta}(X,n) \rightarrow (\exists Z)APATH_{\varphi}^{\alpha,\beta}(t,n,\bar{p},\bar{Q},X,Z)).\end{array}$$

Note that the righthand side of Ax-$\mathbf{Q}^{SAC}(\varphi)$ is $\Sigma_1^B(\varphi)$.

Based on the above formulation, we will now define our system which captures the class LOGCFL.

Definition 11 *The system V-$\mathbf{Q}^{SAC}(\Sigma_0^B)$ is the $QL_2(\Sigma_0^b)$ theory whose axioms are*

- $BASIC_2$,
- *Ax-*$\mathbf{Q}^{SAC}(\varphi)$ *for* $\varphi \in \Sigma_0^B$,
- $\mathbf{Q}^{SAC}(\Sigma_0^B)$*-bit-comprehension:*

$$(\exists P)(\forall k < c)(P[k] \leftrightarrow \mathbf{Q}^{SAC}[\varphi(g, h, z, \bar{x}, y, \bar{Y}), \alpha(z), \beta(z)](|\alpha(n)| \cdot \beta(n) - 1, n, \bar{p}, k, \bar{Q}, X)).$$

Using the technique developed by Kolokolova [8], the author [9] showed that

Theorem 5 (K) *A function is in $\mathcal{F}_{LOGCFL}$ if and only if it is Σ_1^B definable in V-$\mathbf{Q}^{SAC}(\Sigma_0^B)$.*

5 Problems

It is an open problem whether a fragment of second order logic captures the class LOGCFL in the sense of descriptive complexity. Since a second order existential logic $SO\exists$ captures the class NP, it is natural to conjecture that subsets of $SO\exists$ captures complexity classes inside NP.

In fact, Grädel [4] gave a second order existential logic which captures P and NL. A formula is said to be restricted $SO\exists$ if it is of the form

$$(\exists P_1) \cdots (\exists P_k)(\forall \vec{y})\varphi(\vec{P}, \vec{y}), \quad (*)$$

where $\varphi(\vec{P}, \vec{y})$ is quantifier free. Grädel defined

Definition 12 *A formula is $SO\exists$-Horn if it is a $SO\exists$ formula such that its quantifier free part is in conjunction normal form where each conjunct contains at most one negative literal.*

A formula is $SO\exists$-Krom if it is a $SO\exists$ formula such that its quantifier free part is in conjunction normal form where each conjunct contains at most two literals.

Theorem 6 (Grädel [4]) *$SO\exists$-Horn captures P and $SO\exists$-Krom captures NL over arithmetical structures.*

The proof uses the fact that the satisfiability for Horn formulae and 2CNF formulae are complete for P and NL respectively.

Now, turning our attension back to the class LOGCFL, there is a possibility that the acyclic Boolean conjunctive queries can be used to give a class of restricted $SO\exists$ formulae which captures the class LOGCFL. To this end, the formula φ in $(*)$ must express the acyclic Boolean conjunctive queries.

References

[1] A.Borodin, S.A.Cook, P.W.Dymond, W.L.Ruzzo and M.Tompa, Two applications of inductive counting for complementation problems. SIAM J.Comput., vol.18, No.3, pp.559–578 (1989)

[2] Buss. S. R., Bounded Arithmetic. Ph.D thesis. (1985) Published in 1986 by Bibliopolis, Naples.

[3] Fagin, R., Generalized first-order spectra and polynomial-time recognizable sets. In *Comlexity of Computation*, R. Karp, ed., SIAM-AMS Proceedings, 7 (1974), pp.43–73.

[4] Grädel, E., Capturing complexity classes by fragments of second order logic. Theoretical Computer Science, 101 (1992), pp.35–57.

[5] Gottlob, G., N. Leone and F. Scarcello, The Complexity of Acyclic Conjunctive Queries, Journal of ACM, vol.43, No.3, pp.431–498 (2001)

[6] Immerman, N., Nondeterministic space is closed under complementation. SIAM Journal on Computing, 17 (1988), pp.935–938.

[7] Immerman, N., Descriptive Complexity. Guraduate Texts in Computer Science, Springer (1999).

[8] A.Kolokolova, Systems of bounded arithmetic from descriptive complexity. Ph.D.Thesis, University of Toronto (2005)

[9] Kuroda, S., A bounded arithmetic theory for LOGCFL, to appear in Archive for Mathematical Logic. (2007)

[10] Szelepscényi, R., The method of forced enumeration for nondeterministic automata. Acta Informatica, 26 (1988) pp.279–284.

[11] H.Venkateswaran, Properties that characterizes LOGCFL. J. Computer and System Sciences, 42, pp.380–404 (1991)

[12] C.Lautemann, P.McKenzie, T.Schwentick and H.Vollmer, The Descriptive Complexity Approach to LOGCFL. J. Computer and System Sciences, 62(4), pp.629–652 (2001)