A Formal Theory for the Complexity Class Associated with the Stable Marriage Problem

Dai Tri Man Lê, Stephen A. Cook, and Yuli Ye

Department of Computer Science, University of Toronto

Abstract

Subramanian defined the complexity class CC as the set of problems log-space reducible to the comparator circuit value problem. He proved that several other problems are complete for CC, including the stable marriage problem, and finding the lexicographical first maximal matching in a bipartite graph. We suggest alternative definitions of CC based on different reducibilities and introduce a two-sorted theory VCC^* based on one of them. We sharpen and simplify Subramanian's completeness proofs for the above two problems and formalize them in VCC^*.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems; F.4.1 Mathematical logic

Keywords and phrases Bounded arithmetic, complexity theory, comparator circuits

Digital Object Identifier 10.4230/LIPIcs.CSL.2011.381

1 Introduction

Comparator networks were originally introduced as a method of sorting numbers (as in Batcher's even-odd merge sort [2]), but they are still interesting when the numbers are restricted to the Boolean values $\{0,1\}$. A comparator gate has two inputs p, q and two outputs p', q', where $p' = \min\{p,q\}$ and $q' = \max\{p,q\}$. In the Boolean case (which is the one we consider) $p' = p \wedge q$ and $q' = p \vee q$. A comparator circuit (i.e. network) is presented as a set of m horizontal lines in which the m inputs are presented at the left ends of the lines and the m outputs are presented at the right ends of the lines, and in between there is a sequence of comparator gates, each represented as a vertical arrow connecting some wire w_i with some wire w_j as shown in Fig. 1. These arrows divide each wire into segments, each of which gets a Boolean value. The values of wires w_i and w_j after the arrow are the comparator outputs of the values of wires w_i and w_j right before the arrow, with the tip of the arrow representing the maximum.

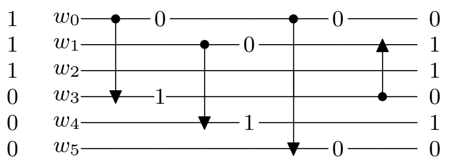


Figure 1

The comparator circuit value problem (Ccv) is: given a comparator circuit with specified Boolean inputs, determine the output value of a designated wire. To turn this into a complexity class it seems natural to use a reducibility notion that is weak but fairly robust. Thus we define CC to consist of those problems (uniform) AC^0 many-one-reducible to Ccv. However Subramanian [9] studied the complexity of Ccv using a stronger notion of reducibility. Thus his class, which we denote $\mathsf{CC}^{\mathsf{Subr}}$, consists of those problems log-space (many-one)-reducible to Ccv. It turns out that a generalization of AC^0 many-one reducibility which we will call AC^0 *oracle reducibility* (called simply AC^0 reducibility in [3]), is also useful. Standard complexity classes such as AC^0, L (log space), NL (nondeterministic log space), NC, and P are all closed under this AC^0 oracle reducibility. We denote the closure of Ccv under this reducibility by CC^*.

We will show that

$$\mathsf{NL} \subseteq \mathsf{CC} \subseteq \mathsf{CC}^{\mathsf{Subr}} \subseteq \mathsf{CC}^* \subseteq \mathsf{P} \tag{1.1}$$

The last inclusion is obvious because Ccv is a special case of the monotone circuit value problem, which is clearly in P. The inclusion $\mathsf{CC} \subseteq \mathsf{CC}^{\mathsf{Subr}}$ follows because AC^0 functions are also log-space functions. The inclusion $\mathsf{CC}^{\mathsf{Subr}} \subseteq \mathsf{CC}^*$ follows from the first inclusion, which in turn is a strengthening of a result in [6] (attributed to Feder) showing that $\mathsf{NL} \subseteq \mathsf{CC}^{\mathsf{Subr}}$. Of course all three comparator classes coincide if it turns out that CC is closed under AC^0 oracle reductions, but we do not know how to show this.

Note that comparator circuits are more restricted than monotone Boolean circuits because each comparator output has fan-out one. This leads to the open question of whether $\mathsf{CC}^* \subsetneq \mathsf{P}$. A second open question is whether CC^* and NC are incomparable. (Here NC is the class of problems computed by uniform circuit families of polynomial size and polylog depth, and satisfies $\mathsf{NL} \subseteq \mathsf{NC} \subseteq \mathsf{P}$.) The answers could be different if we replaced CC^* by $\mathsf{CC}^{\mathsf{Subr}}$ or CC, although $\mathsf{CC} \subseteq \mathsf{NC}$ iff $\mathsf{CC}^* \subseteq \mathsf{NC}$ because NC is closed under AC^0 oracle reductions.

The above classes associated with Ccv are also interesting because they have several disparate complete problems. As shown in [6, 9] both the lexicographical first maximal matching problem (Lfmm) and the stable marriage problem (Sm) are complete for $\mathsf{CC}^{\mathsf{Subr}}$ under log-space reductions[1]. The Sm problem is especially interesting: Introduced by Gale and Shapley in 1962 [4], it has since been used to pair medical interns with hospital residencies jobs in the USA. Sm can be stated as follows: Given n men and n women, each with a complete ranking according to preference of all n members of the opposite sex, find a complete matching of the men and women such that there are no two people of opposite sex who would both rather have each other than their current partners. Gale and Shapley proved that such a 'stable' matching always exists, although it may not be unique. Subramanian [9] showed that Sm treated as a search problem (i.e. find any stable marriage) is complete for CC under log-space reducibility.

Strangely the CC classes have received very little attention since Subramanian's papers [8, 9]. The present paper contributes to their complexity theory by sharpening these early results and simplifying their proofs. For example we prove that the three problems Ccv, Lfmm, and Sm are inter-reducible under AC^0 many-one reductions as opposed to log-space reductions. Also we introduce a three-valued logic version of Ccv to facilitate its reduction to Sm. Our paper contributes to the proof complexity of the classes by introducing a two-sorted formal theory VCC^* which captures the class CC^* and which can formalize the proofs of the above results.

Our theory VCC^* is a two-sorted theory developed in the way described in [3, Chapter 9]. In general this method associates a theory VC with a suitable complexity class C in such a way that a function is in FC, the function class associated with C, if and only if it is provably total in VC. (A string-valued function is in FC iff it is polynomially bounded and its bit-graph is in C.) This poses a problem for us because the provably-total functions in a theory are always closed under composition, but it is quite possible that neither of the function classes FCC and $\mathsf{FCC}^{\mathsf{Subr}}$ is closed under composition. That is why we define the class CC^* to consist of the problems AC^0-oracle-reducible (see Definition 3 below) to Ccv, rather than the problems AC^0 many-one reducible to Ccv, which comprise CC. It is easy to

[1] The second author outlined a proof that Lfmm is complete under NC^1 reductions in unpublished notes from 1983.

see that the functions in FCC^* are closed under composition, and these are the functions that are provably total in our theory VCC^*.

The above paragraph illustrates one way that studying proof complexity can contribute to main-stream complexity theory, namely by mandating the introduction of the more robust version CC^* of CC and $\mathsf{CC}^{\mathsf{Subr}}$. Another way is by using the simple two-sorted syntax of our theories to demonstrate AC^0 reductions. Thus Theorem 1 below states that a simple syntactic class of formulas represents precisely the AC^0 relations. In general it is much easier to write down an appropriate such formula than to describe a uniform circuit family or alternating Turing machine program.

Once we describe our theory VCC^* in Sections 2.1 and 2.2, the technical part of our proofs involve high-level descriptions of comparator circuits and algorithms. We do not say much about formalizing the proofs in VCC^* since this part is relatively straightforward.

2 Preliminaries

2.1 Two-sorted vocabularies

We use two-sorted vocabularies for our theories as described by Cook and Nguyen [3]. Two-sorted languages have variables $x, y, z, \ldots$ ranging over $\mathbb{N}$ and variables $X, Y, Z, \ldots$ ranging over finite subsets of $\mathbb{N}$, interpreted as bit strings. Two sorted vocabulary $\mathcal{L}_A^2$ includes the usual symbols $0, 1, +, \cdot, =, \leq$ for arithmetic over $\mathbb{N}$, the length function $|X|$ for strings ($|X|$ is zero if X is empty, otherwise $1 + \max(X)$), the set membership relation $\in$, and string equality $=_2$ (subscript 2 is usually omitted). We will use the notation $X(t)$ for $t \in X$, and think of $X(t)$ as the t^{th} bit in the string X.

The number terms in the base language $\mathcal{L}_A^2$ are built from the constants $0, 1$, variables $x, y, z, \ldots$ and length terms $|X|$ using $+$ and $\cdot$. The only string terms are string variables, but when we extend $\mathcal{L}_A^2$ by adding string-valued functions, other string terms will be built as usual. The atomic formulas are $t = u$, $X = Y$, $t \leq u$, $t \in X$ for any number terms x, y and string variables X, Y. Formulas are built from atomic formulas using $\wedge, \vee, \neg$ and $\exists x$, $\exists X$, $\forall x$, $\forall X$. Bounded number quantifiers are defined as usual, and bounded string quantifier $\exists X \leq t, \varphi$ stands for $\exists X(|X| \leq t \wedge \varphi)$ and $\forall X \leq t, \varphi$ stands for $\forall X(|X| \leq t \to \varphi)$, where X does not appear in term t.

The class Σ_0^B consists of all $\mathcal{L}_A^2$-formulas with no string quantifiers and only bounded number quantifiers. The class Σ_1^B consists of formulas of the form $\exists \vec{X} < \vec{t}\varphi$, where $\varphi \in \Sigma_0^B$ and the prefix of the bounded quantifiers might be empty. These classes are extended to Σ_i^B (and Π_i^B) for all $i \geq 0$, in the usual way. More generally we write $\Sigma_i^B(\mathcal{L})$ to denote the class of Σ_i^B-formulas which may have function and predicate symbols from $\mathcal{L} \cup \mathcal{L}_A^2$.

Two-sorted complexity classes contain relations $R(\vec{x}, \vec{X})$, where $\vec{x}$ are number arguments and $\vec{X}$ are string arguments. In defining complexity classes using machines or circuits, the number arguments are represented in unary notation and the string arguments are represented in binary. The string arguments are the main inputs, and the number arguments are auxiliary inputs that can be used to index the bits of strings.

In the two-sorted setting, we can define AC^0 to be the class of relations $R(\vec{x}, \vec{X})$ such that some alternating Turing machine accepts R in time $\mathcal{O}(\log n)$ with a constant number of alternations. Then the descriptive complexity characterization of AC^0 gives rise to the following theorem [3, Chapter 4].

▶ **Theorem 1.** *A relation $R(\vec{x}, \vec{X})$ is in AC^0 iff it is represented by some Σ_0^B-formula $\varphi(\vec{x}, \vec{X})$.*

Given a class of relations C, we associate a class FC of string-valued functions $F(\vec{x},\vec{X})$ and number functions $f(\vec{x},\vec{X})$ with C as follows. We require these functions to be p-bounded, i.e., $|F(\vec{x},\vec{X})|$ and $f(\vec{x},\vec{X})$ are bounded by a polynomial in $\vec{x}$ and $|\vec{X}|$. Then we define FC to consist of all p-bounded number functions whose graphs are in C and all p-bounded string functions whose bit graphs are in C.

▶ **Definition 2.** Let C be a complexity class. A relation $R_1(\vec{x},\vec{X})$ is C-many-one-reducible to a relation $R_2(\vec{y},\vec{Y})$ (written $R_1 \leq^{\mathsf{C}}_{\mathsf{m}} R_2$) if there are functions $\vec{f},\vec{F}$ in FC such that

$$R_1(\vec{x},\vec{X}) \leftrightarrow R_2(\vec{f}(\vec{x},\vec{X}),\vec{F}(\vec{x},\vec{X})).$$

A function $H_1(\vec{x},\vec{X})$ is C-many-one-reducible to a function $H_2(\vec{y},\vec{Y})$ if there are functions $G,\vec{f},\vec{F}$ in FC such that

$$H_1(\vec{x},\vec{X}) = G(H_2(\vec{f}(\vec{x},\vec{X}),\vec{F}(\vec{x},\vec{X}))).$$

Here we are mainly interested in the cases that C is either AC^0 or L (log space). We also need a generalization of AC^0 many-one reducibility called simply AC^0 reducibility in [3, Definition IX.1.1], which we will call AC^0 *oracle reducibility.* Roughly speaking a function or relation is AC^0-oracle-reducible to a collection $\mathcal{L}$ of functions and relations if it can be computed by a uniform polynomial size constant depth family of circuits which have unbounded fan-in gates computing functions and relations from $\mathcal{L}$ (i.e. 'oracle gates'), in addition to Boolean gates. Formally:

▶ **Definition 3.** A string function F is AC^0-oracle-reducible to a collection $\mathcal{L}$ of relations and functions (written $F \leq^{\mathsf{AC}^0}_{\mathsf{o}} \mathcal{L}$) if there is a sequence of string functions $F_1,\ldots,F_n = F$ such that each F_i is p-bounded and its bit graph is represented by a $\Sigma^B_0(\mathcal{L},F_1,\ldots,F_{i-1})$-formula.

A number function f is AC^0-oracle-reducible to $\mathcal{L}$ if $f = |F|$ for some string function F which is AC^0-reducible to $\mathcal{L}$. A relation R is AC^0-oracle-reducible to $\mathcal{L}$ if its characteristic function is AC^0-oracle-reducible to $\mathcal{L}$.

We note that standard small complexity classes including AC^0, TC^0, NC^1, NL and P (as well as their corresponding function classes) are closed under AC^0 oracle reductions.

2.2 Two-sorted theories

The theory V^0 for AC^0 is the basis for developing theories for small complexity classes within P in [3]. V^0 has the vocabulary $\mathcal{L}^2_A$ and is axiomatized by the set of *2-BASIC* axioms, which express basic properties of symbols in $\mathcal{L}^2_A$, together with the *comprehension* axiom schema

$$\Sigma^B_0\textit{-COMP}: \qquad \exists X \leq y \forall z < y\big(X(z) \leftrightarrow \varphi(z)\big),$$

where $\varphi \in \Sigma^B_0(\mathcal{L}^2_A)$ and X does not occur free in φ. Although V^0 has no explicit induction axiom, nevertheless, using Σ^B_0-*COMP* and the fact that $|X|$ produces the maximum element of the finite set X, the following schemes are provable in V^0 for every formula $\varphi \in \Sigma^B_0(\mathcal{L}^2_A)$

$$\Sigma^B_0\textit{-IND}: \qquad \big[\varphi(0) \wedge \forall x\big(\varphi(x) \to \varphi(x+1)\big)\big] \to \forall x\varphi(x),$$
$$\Sigma^B_0\textit{-MIN}: \qquad \varphi(y) \to \exists x\big(\varphi(x) \wedge \neg\exists z < x\,\varphi(z)\big).$$

In general, we say that a string function $F(\vec{x},\vec{X})$ is Σ^B_1-definable (or provably total) in a two-sorted theory $\mathcal{T}$ if there is a Σ^B_1 formula $\varphi(\vec{x},\vec{X},Y)$ representing the graph $Y = F(\vec{x},\vec{X})$ of F such that $\mathcal{T} \vdash \forall\vec{x}\forall\vec{X}\exists!Y\varphi(\vec{x},\vec{X},Y)$. Similarly for a number function $f(\vec{x},\vec{X})$.

It was shown in [3, Chapter 5] that V^0 is finitely axiomatizable, and a function is provably total in V^0 if and only if it is in FAC^0.

In [3, Chapter 9], Cook and Nguyen develop a general method for associating a theory VC with certain complexity classes $\mathsf{C} \subseteq \mathsf{P}$, where VC extends V^0 with an additional axiom asserting the existence of a solution to a complete problem for C. In order for this method to work, the class C must be closed under AC^0-oracle-reducibility (Definition 3). The method shows how to define a universal conservative extension $\overline{\mathsf{VC}}$ of VC, where $\overline{\mathsf{VC}}$ has string function symbols for precisely the string functions of FC, and terms for precisely the number functions of FC. Further, $\overline{\mathsf{VC}}$ proves the $\Sigma_0^B(\mathcal{L})$-*IND* and $\Sigma_0^B(\mathcal{L})$-*MIN* schemes, where $\mathcal{L}$ is the vocabulary of $\overline{VC}$. It follows from the Herbrand Theorem that the provably total functions of both VC and $\overline{\mathsf{VC}}$ are precisely those in FC.

Using this framework Cook and Nguyen define specific theories for several complexity classes and give examples of theorems formalizable in each theory. The theories of interest to us in this paper are VTC^0, VNC^1, VNL and VP for the complexity classes TC^0, NC^1, NL and P respectively. All of these theories have vocabulary $\mathcal{L}_A^2$. Let $\langle x, y\rangle$ denote the *pairing function*, which is the $\mathcal{L}_A^2$ term $(x+y)(x+y+1)+2y$. The theory VTC^0 is axiomatized by the axioms of V^0 and the axiom:

$$\textit{NUMONES}: \qquad \exists Z \le 1 + \langle n, n\rangle, \delta_{\mathsf{NUM}}(n, X, Z), \tag{2.1}$$

where the formula $\delta_{\mathsf{NUM}}(n, X, Z)$ asserts that Z is a matrix consisting of n rows such that for every $y \le n$, the y^{th} row of Z encodes the number of ones in the prefix of length y of the binary string X. Thus, the n^{th} row of Z essentially "counts" the number of ones in X. Because of this counting ability, VTC^0 can prove *the pigeonhole principle* $PHP(n, F)$ saying that if F maps a set of $n+1$ elements to a set of n elements, then F is not an injection.

The theory VNC^1 is axiomatized by the axioms of V^0 and the axiom:

$$\textit{MFV}: \qquad \exists Y \le 2n+1, \delta_{\mathsf{MFV}}(n, F, I, Y), \tag{2.2}$$

where F and I encode a monotone Boolean formula with n literals and its input respectively, and the formula $\delta_{\mathsf{MFV}}(n, G, I, Y)$ holds iff Y correctly encodes the evaluation of the formula encoded in F on input I. Recall that the *monotone Boolean formula value* problem is complete for NC^1.

The theory VP is axiomatized by the axioms of V^0 and the axiom *MCV*, which is defined very similarly to *MFV*, but the *monotone circuit value* problem is used instead.

The theory VNL is axiomatized by the axioms of V^0 and the axiom:

$$\textit{CONN}: \qquad \exists U \le \langle n, n\rangle + 1, \delta_{\mathsf{CONN}}(n, E, U), \tag{2.3}$$

where E encodes the edge relation of a directed graph G with n vertices $v_0, \ldots, v_{n-1}$, and the formula $\delta_{\mathsf{CONN}}(n, E, U)$ holds iff U is a matrix of n rows, where the d^{th} row encodes the set of all vertices in G that are reachable from v_0 using a path of length at most d.

Similar to what is currently known about complexity classes, it was shown in [3, Chapter 9] that $\mathsf{V}^0 \subsetneq \mathsf{VTC}^0 \subseteq \mathsf{VNC}^1 \subseteq \mathsf{VNL} \subseteq \mathsf{VP}$.

2.3 The Ccv problem and its complexity classes

A *comparator gate* is a function $C : \{0,1\}^2 \to \{0,1\}^2$, that takes an input pair (p, q) and outputs a pair $(p \wedge q, p \vee q)$. Intuitively, the first output in the pair is the smaller bit among the two input bits p, q, and the second output is the larger bit.

We will use the graphical notation in Fig. 2 to denote a comparator gate, where x and y denote the names of the wires, and the direction of the arrow denotes the direction to which we move the larger bit as shown in the picture.

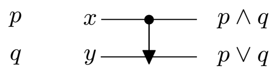


Figure 2

A *comparator circuit* is a directed acyclic graph consisting of: *input nodes* with in-degree zero and out-degree one, *output nodes* with in-degree one and out-degree zero, and *internal nodes* with in-degree two and out-degree two, where each internal node is labelled with a comparator gate. We also require each output computed by a comparator gate has fan-out exactly one. Under this definition, each comparator circuit can be seen as consisting of the wires that carry the bit values and are arranged in parallel, and each comparator gate connects exactly two wires as previously shown in Fig. 1.

The *comparator circuit value* problem (Ccv) is the task of deciding, on a given input assignment, if a designated wire of a comparator circuit outputs one.

▶ **Definition 4.** The complexity class CC (resp. $\mathsf{CC}^{\mathsf{Subr}}$, CC^*) is the class of decision problems (i.e. relations) that are AC^0 many-one-reducible (resp. log space-reducible, AC^0 oracle-reducible) to Ccv. A decision problem R is CC-complete (resp. $\mathsf{CC}^{\mathsf{Subr}}$-complete, CC^*-complete) if the respective class is the closure of R under the corresponding reducibility. We say that R is $\mathsf{CC}_{\mathsf{all}}$-complete if it is complete in all three senses.

The next result is a straightforward consequence of (1.1) and the definitions involved.

▶ **Lemma 5.** *If a decision problem is CC-complete then is is $\mathsf{CC}_{\mathsf{all}}$-complete.*

In the above definition of comparator circuit, each comparator gate can point in either direction, upward or downward (see Fig. 1). However, it is not hard to show the following.

▶ **Proposition 1.** *The Ccv problem with the restriction that all comparator gates point in the same direction is CC-complete.*

2.4 The stable marriage problem

An instance of the stable marriage problem (Sm) is given by a number n (specifying the number of men and the number of women), together with a preference list for each man and each woman specifying a total ordering on all people of the opposite sex. The goal of Sm is to produce a perfect matching between men and women, i.e., a bijection from the set of men to the set of women, such that the following *stability* condition is satisfied: there are no two people of the opposite sex who like each other more than their current partners. Such a stable solution always exists, but it may not be unique. Under this formulation Sm is a *search problem*, rather than a function problem.

However there is always a unique *man-optimal* and a unique *woman-optimal* solution. In the man-optimal solution each man is matched with a woman whom he likes at least as well as any woman that he is matched with in any stable solution. Dually for the woman-optimal solution. Thus both the man-optimal and the woman-optimal versions are function problems (and hence equivalent to decision problems.)

We show here that the search version and the decision versions are computationally equivalent, and each is complete for CC. Section 6.1 shows how to reduce the lexicographical first maximal matching problem (a decision problem complete for CC) to the search version of Sm, and Section 6.2 shows how to reduce both the man-optimal and the woman-optimal function problems of Sm to Ccv.

2.5 Notation

We write the notation "$(T \vdash)$" in front of the statement of a theorem to indicate that the statement is formulated and proved within the theory T.

3 The new theory VCC*

We encode a comparator circuit as a sequence of pairs $\langle i, j\rangle$, where each pair $\langle i, j\rangle$ encodes a comparator gate that swaps the values of the wires i and j if and only if the value on wire i is greater than the value of wire j. We also allow "dummy" gates of the form $\langle i, i\rangle$, which do nothing. We want to define a formula $\delta_{\mathsf{CCV}}(m, n, X, Y, Z)$, where

- X encodes a comparator circuit with m wires and n gates as a sequence of n pairs $\langle i, j\rangle$ with $i, j < m$, and we write $(X)^i$ to denote the i^{th} comparator gate of the circuit.
- $Y(i)$ encodes the input value for the i^{th} wire as a truth value, and
- Z is an $(n+1) \times m$ matrix, where $Z(i, j)$ is the value of wire j at layer i, where each layer is simply a sequence of the values carried by all the wires right after a comparator gate.

Although X encodes a circuit with only n gates, Z actually encodes $n+1$ layers since we use the first layer to encode the input values of the circuit. The formula $\delta_{\mathsf{CCV}}(m, n, X, Y, Z)$ holds iff Z encodes the correct values of the layers computed by the comparator circuit encoded by X with input Y, and thus is defined as the following Σ_0^B-formula:

$$\begin{aligned}
&\forall i < m\big(Y(i) \leftrightarrow Z(0, i)\big) \wedge \forall i < n \forall x < m \forall y < m,\\
&(X)^i = \langle x, y\rangle \to \left[\begin{array}{ll} & Z(i+1, x) \leftrightarrow \big(Z(i, x) \wedge Z(i, y)\big)\\ \wedge & Z(i+1, y) \leftrightarrow \big(Z(i, x) \vee Z(i, y)\big)\\ \wedge & \forall j < m\Big[(j \neq x \wedge j \neq y) \to \big(Z(i+1, j) \leftrightarrow Z(i, j)\big)\Big]\end{array}\right]
\end{aligned} \qquad (3.1)$$

Note that in this paper we index the entries of matrices starting from 0 instead of 1.

▶ **Definition 6.** The theory VCC* has vocabulary $\mathcal{L}_A^2$ and is axiomatized by the axioms of V^0 and the following axiom (the formula $\delta_{\mathsf{CCV}}(m, n, X, Y, Z)$ is defined as in (3.1)):

$$CCV: \qquad \exists Z \le \langle m, n+1\rangle + 1,\ \delta_{\mathsf{CCV}}(m, n, X, Y, Z) \qquad (3.2)$$

There is a technical lemma required to show that VCC* fits the framework described in [3, Chapter 9]. Define $F_{\mathsf{CCV}}(m, n, X, Y)$ to be the Z satisfying $\delta_{\mathsf{CCV}}(m, n.X, Y, Z)$ (with each $Z(i)$ set false when it is not determined). We need to show that the *aggregate* F^*_{CCV} of F_{CCV} is Σ_1^B-definable in VCC*, where (roughly speaking) F^*_{CCV} is the string function that gathers the values of F_{CCV} for a polynomially long sequence of arguments. The nature of CC circuits makes this easy: The sequence of outputs for a sequence of circuits can be obtained from a single circuit which computes them all in parallel: the lines of the composite circuit comprise the union of the lines of each component circuit. Thus the framework of [3, Chapter 9] does apply to VCC*, and in particular the theory $\overline{\mathsf{VCC}^*}$ is a universal conservative extension of VCC* whose function symbols are precisely those in the function class FCC.

It is hard to work with VCC* up to this point since we have not shown whether VCC* can prove the definability of basic counting functions (as in VTC^0). However, we have the following theorem.

▶ **Theorem 7** ($\mathsf{VNC}^1 \subseteq \mathsf{VCC}^*$). *The theory* VCC* *proves the axiom MFV defined in (2.2).*

Proof. Observe that each comparator gate can produce simultaneously an AND gate and an OR gate with the only restriction that each of these gates must have fan-out one. However, since all AND and OR gates of a monotone Boolean formula also have fan-out one, each instance of the Boolean formula value problem is a special case of Ccv. ◀

A corollary of this theorem is that $\mathsf{VTC}^0 \subseteq \mathsf{VCC}^*$, and thus we can use the counting ability of VTC^0 freely in VCC^* proofs.

▶ **Theorem 8** ($\mathsf{VCC}^* \subseteq \mathsf{VP}$). *The theory* VP *proves the axiom CCV defined in (3.2).*

Proof. This follows since Ccv is a special case of the *monotone circuit value* problem. ◀

4 Lexicographical first maximal matching problem is CC-complete

Let $G = (V, W, E)$ be a bipartite graph, where $V = \{v_i\}_{i=0}^{m-1}$, $W = \{w_i\}_{i=0}^{n-1}$ and $E \subseteq V \times W$. The *lexicographical first maximal matching* (lfm-matching) is the matching produced by successively matching each vertex $v_0, \dots, v_{m-1}$ to the least vertex available in W. The lexicographical first maximal matching problem (Lfmm) is to decide if a designated edge belongs to the lfm-matching of G, and 3Lfmm is the restriction of Lfmm to graphs of degree at most three. In this section we give simplified constructions showing that Ccv is AC^0-many-one-reducible to 3Lfmm and Lfmm is AC^0-many-one-reducible to Ccv.

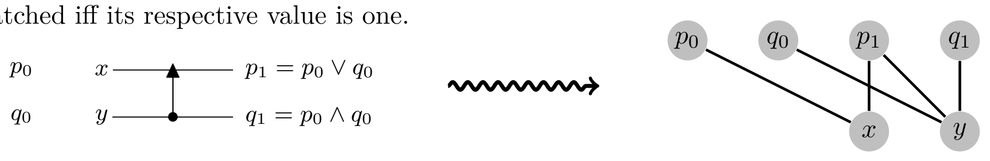
Formally, let $E_{m\times n}$ be a matrix encoding the edge relation of a bipartite graph with m bottom nodes and n top nodes, where $E(i,j) = 1$ iff the bottom node v_i is adjacent to the top node w_j. Let L be a matrix of the same size as E with the following intended interpretation: $L(i,j) = 1$ iff the edge (v_i, w_j) is in the lfm-matching. We can define a Σ_0^B-formula $\delta_{\mathsf{LFMM}}(m, n, X, L)$, which holds iff L properly encodes the lfm-matching of the bipartite graph represented by X as follows:

$$\forall i < m \forall j < n,\ L(i,j) \leftrightarrow \left[\begin{array}{ccc} E(i,j) & \wedge & \forall k < j\, \forall \ell < i\big(\neg L(i,k) \wedge \neg L(\ell,j)\big) \\ & \wedge & \forall k < j\big(\neg E(i,k) \vee \exists i' < i\, L(i',k)\big) \end{array}\right]. \tag{4.1}$$

4.1 Ccv $\le_{\mathsf{m}}^{\mathsf{AC}^0}$ 3Lfmm

By Proposition 1, it suffices to consider only instance of Ccv, where all comparator gates point upward. We will show that these instances of Ccv are AC^0-many-one-reducible to instances of 3Lfmm, which consist of bipartite graphs with *degree at most three.*

The key observation is that a comparator gate on the left below closely relates to an instance of 3Lfmm on the right. We use the top nodes p_0 and q_0 to represent the values p_0 and q_0 carried by the wires x and y respectively before the comparator gate, and the nodes p_1 and q_1 to represent the values of x and y after the comparator gate, where a top node is matched iff its respective value is one.



If nodes p_0 and q_0 are not previously matched, i.e. $p_0 = q_0 = 0$ in the comparator circuit, then edges $\langle x, p_0\rangle$ and $\langle y, q_0\rangle$ are added to the lfm-matching. So the nodes p_1 and q_1 are not matched. If p_0 is previously matched, but q_0 is not, then edges $\langle x, p_1\rangle$ and $\langle y, q_0\rangle$ are added to the lfm-matching. So the node p_1 will be matched but q_1 will remain unmatched. The other two cases are similar.

Thus, we can reduce a comparator circuit to the bipartite graph of an 3Lfmm instance by converting each comparator gate into a “gadget” described above. We will describe our method through an example, where we are given the comparator circuit in Fig. 3.

We divide the comparator circuit into vertical layers 0, 1, and 2 as shown in Fig. 3. Since the circuit has three wires a, b and c, for each layer i, we use six nodes, including three top nodes a_i, b_i and c_i representing the values of the wires a, b and c respectively, and three bottom nodes a'_i, b'_i, c'_i, which are auxiliary nodes used to simulate the effect of the comparator gate at layer i.

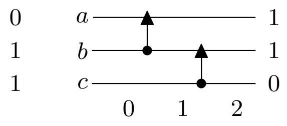
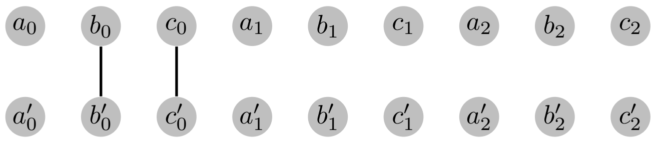
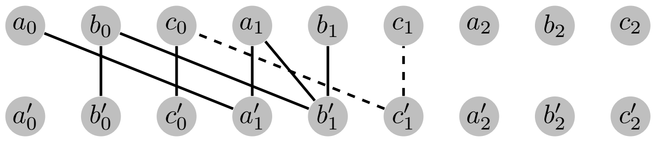


Figure 3

Layer 0: This is the input layer, so we add an edge $\{x_i, x'_i\}$ iff the wire x takes input 1. In this example, since b and c are wires taking input 1, we add the edges $\{b_0, b'_0\}$ and $\{c_0, c'_0\}$.

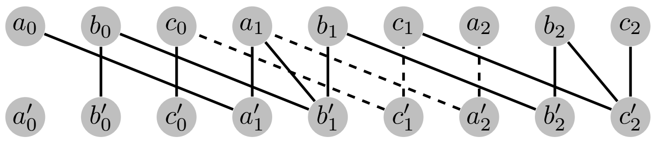


Layer 1: We then add the gadget simulating the comparator gate from wire b to wire a.



Since the value of wire c does not change when going from layer 0 to layer 1, we can simply propagate the value of c_0 to c_1 using the pair of dotted edges in the picture.

Layer 2: We proceed very similarly to layer 1 to get the following bipartite graph.



Finally, we can get the output values of the comparator circuit by looking at the "output" nodes a_2, b_2, c_2 of this bipartite graph. We can easily check that a_2 is the only node that remains unmatched, which corresponds exactly to the only zero produced by wire a of the comparator circuit above.

The construction above is an AC^0 many-one reduction since each gate in the comparator circuit can be reduced to exactly one gadget in the bipartite graph that simulates the effect of the comparator gate. Note that since it can be tedious and unintuitive to work with AC^0-circuits, it might seem hard to justify that our reduction is an AC^0-function. However, thanks to Theorem 1, we do not have to work with AC^0-circuits directly; instead, it is not hard to construct a Σ_0^B-formula that defines the above reduction. The correctness of our construction can be proved in VCC^* by using Σ_0^B induction on the layers of the circuits and arguing that the matching information of the nodes in the bipartite graph can be correctly translated to the values carried by the wires at each layer.

4.2 Lfmm $\leq_m^{\mathsf{AC}^0}$ Ccv

Consider an instance of Lfmm consisting of a bipartite graph on the left of Fig. 4. Recall that we find the lfm-matching by matching the bottom nodes $x, y, \ldots$ successively to the first available node on the top. Hence we can simulate the matching of the bottom nodes to the top nodes using comparator circuit on the right of Fig. 4, where we can think of the moving of a one, say from wire x to wire a, as the matching of node x to node a in the original bipartite graph. Note that we draw bullets without any arrows going out from them in the circuit to denote dummy gates, which do nothing. These dummy gates are introduced for

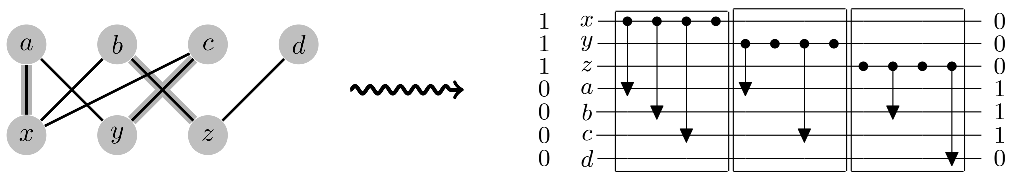


Figure 4

the following technical reason. Since the bottom nodes might not have the same degree, the position of a comparator gate really depends on the number of edges that do not appear in the bipartite graph, which makes it harder to give a direct AC^0-reduction. By using dummy gates, we can treat the graph as if it is a complete bipartite graph, where the missing edges are represented by dummy gates. This can easily be shown to be an AC^0-reduction from LFMM to CCV, and its correctness can be carried out in VCC^*. This together with the reduction from Section 4.1 gives us the following theorem.

▶ **Theorem 9.** ($\mathsf{VCC}^* \vdash$) *The* LFMM *problem is* CC*-complete.*

Since the reduction from CCV to LFMM in Section 4.1 only produces bipartite graphs with degree at most three, we have the following corollary.

▶ **Corollary 10.** *($\mathsf{VCC}^* \vdash$) The* 3LFMM *problem is* CC*-complete.*

5 The theory VCC^* contains VNL

Each instance of the REACHABILITY problem consists of a directed acyclic graph $G = (V, E)$, where $V = \{v_0, \ldots, v_{n-1}\}$, and we want to decide if there is a path from v_0 to v_{n-1}. It is well-known that REACHABILITY is NL-complete. It is also well-known that the REACHABILITY problem still remains NL-complete under the following restriction:

The graph G only has directed edges of the from (v_i, v_j), where $i < j$. (5.1)

We will show how to use comparator circuits to solve the above restricted instances of REACHABILITY. We believe that our new construction is more intuitive than the one in [8, 6]. Moreover, we reduce REACHABILITY to CCV directly without going through some intermediate complete problem, and this was stated as an open problem in [8, Chapter 7.8.1].

We will demonstrate our construction through a simple example, where we have the directed graph in Fig. 5 satisfying the assumption (5.1). We will build a comparator circuit as in Fig. 6, where the wires $\nu_0, \ldots, \nu_4$ represent the vertices $v_0, \ldots, v_4$ of the preceding graph and the wires $\iota_0, \ldots, \iota_4$ are used to feed 1-bits into the wire v_0, and from there to the other wires v_i reachable from v_0. We let every wire ι_i take input one and every wire ν_i take input zero. We next show how to construct the gadget in the boxes. For a graph with n vertices ($n = 5$ in our example), the gadget in the ℓ^{th} box is constructed as follows:

```
1: Add a comparator gate from wire $\iota_\ell$ to wire $\nu_0$
2: for $i = 0, \ldots, n-1$ do
3:   for $j = i+1, \ldots, n-1$ do
4:     Add a comparator gate from $\nu_i$ to $\nu_j$ if $(v_i, v_j) \in E$,
       or a dummy gate on $\nu_i$ otherwise.
5:   end for
6: end for
```

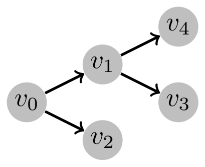


Figure 5

Note that we only use the loop structure to clarify the order the gates are added. The construction can easily be done in AC^0 since the position of each gate can be calculated exactly, and thus all gates can be added independently from one another. Note that for a graph with n vertices, we have at most n vertices reachable from a single vertex, and thus we need n gadgets described above. In our example, there are at most 5 wires reachable from wire ν_0, and thus we utilize the gadget 5 times.

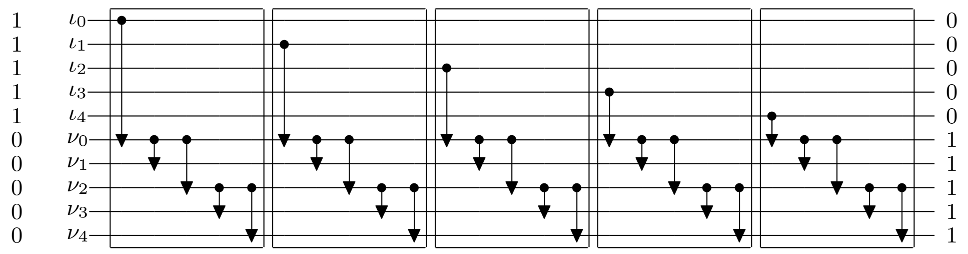


Figure 6 A comparator circuit that solves Reachability. (The dummy gates are omitted.)

Intuitively, the construction works since each gadget from a box looks for the *lexicographical first maximal path* starting from v_0 (with respect to the *natural lexicographical ordering* induced by the vertex ordering $v_0, \ldots, v_n$), and then the vertex at the end of the path will be marked (i.e. its wire will now carry one) and thus excluded from the search of the gadgets that follow. For example, the gadget from the left-most dashed box in Fig. 6 will move a one from wire ι_0 to wire ν_0 and from wire ν_0 to wire ν_1. This essentially "marks" the wire ν_1 since we cannot move the one away from ν_1, and thus ν_1 can no longer receive any new incoming ones. Hence, the gadget from the second box in Fig. 6 will repeat the process of finding the lex-first maximal path from v_0 to the remaining (unmarked) vertices. These searches end when all vertices reachable from v_0 are marked. Note that this has the same effect as applying the *depth-first search* algorithm to find all the vertices reachable from v_0. Thus, we can prove the following theorem.

▶ **Theorem 11** ($\mathsf{VNL} \subseteq \mathsf{VCC}^*$). *The theory* VCC^* *proves the axiom CONN defined in (2.3).*

As a of consequence of Theorem 11, we have the following result.

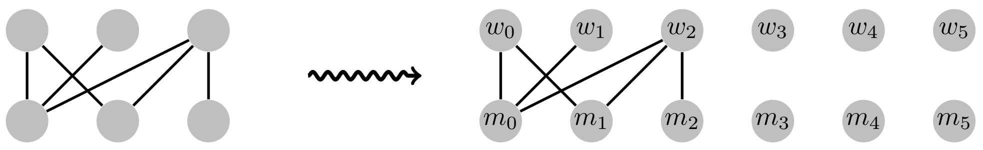
▶ **Theorem 12.** CC^* *is closed under many-one* NL*-reductions, and hence* $\mathsf{CC}^{\mathsf{Subr}} \subseteq \mathsf{CC}^*$.

Proof. This follows from the following three facts: The function class FCC^* is closed under composition, $\mathsf{FNL} \subseteq \mathsf{FCC}$, and a decision problem is in CC^* if and only if its characteristic function is in FCC^*. ◀

6 The Sm problem is CC-complete

6.1 3Lfmm $\leq_m^{\mathsf{AC}^0}$ Sm

Let $G = (V, W, E)$ be a bipartite graph from an instance of 3Lfmm, where V is the set of bottom nodes, W is the set of top nodes, and E is the edge relation such that the degree of each node is at most three (see the example in the figure on the left below). Without loss of generality, we can assume that $|V| = |W| = n$. To reduce it to an instance of Sm, we double the number of nodes in each partition, where the new nodes are enumerated after the original nodes and the original nodes are enumerated using the ordering of the original bipartite graph, as shown in the diagram on the right below. We also let the bottom nodes and top nodes represent the men and women respectively.



It remains to define a preference list for each person in this SM instance. The preference list of each man m_i, who represents a bottom node in the original graph, starts with all the woman w_j (at most three of them) adjacent to m_i in the order that these women are enumerated, followed by all the women $w_n, \ldots, w_{2n-1}$; the list ends with all women w_j not adjacent to m_i also in the order that they are enumerated. For example, the preference list of m_2 in our example is $w_2, w_3, w_4, w_5, w_0, w_1$. The preference list of each newly introduced man m_{n+i} simply consists of $w_0, \ldots, w_{n-1}, w_n, \ldots, w_{2n-1}$, i.e., in the order that the top nodes are listed. Preference lists for the women are defined dually.

Intuitively, the preference lists are constructed so that any stable marriage (not necessarily man-optimal) of the new SM instance must contain the lfm-matching of G. Furthermore, if a bottom node u from the original graph is not matched to any top node in the lfm-matching of G, then the man m_i representing u will marry some top node w_{n+j}, which is a dummy node that does not correspond to any node of G. Thus we have the following theorem.

▶ **Theorem 13.** *($\mathsf{VCC}^* \vdash$) The* 3LFMM *problem is* AC^0*-many-one-reducible to* SM.

6.2 SM $\leq_{\mathsf{m}}^{\mathsf{AC}^0}$ CCV

In this section, we formalize a reduction from SM to CCV due to Subramanian [8, 9]. Subramanian did not reduce SM to CCV directly, but to the *network stability problem* built from the less standard X gate, which takes two inputs p and q and produces two outputs $p' = p \wedge \neg q$ and $q' = \neg p \wedge q$. It is important to note that the "*network*" notion in Subramanian's work denotes the generalization of circuits by allowing connection from output of a gate to input of any gate including itself, and thus a network in his definition might contain cycles. An X-network is a network consisting only of X gates under the important restriction that each X gate has fan-out exactly one for each output it computes. The network stability problem for X gate (XNS) is then to decide if an X-network has a stable configuration, i.e., a way to assign Boolean values to the wires of the network so that the values are compatible with all the X gates of the network. Subramanian showed in his dissertation [8] that SM, XNS and CCV are all equivalent under log space reduction.

We do not work with XNS in this paper since networks are less intuitive and do not have a nice graphical representation as do comparator circuits. By utilizing Subramanian's idea, we give a more direct AC^0-reduction from SM to CCV. For this goal, it turns out to be conceptually simpler to go through a new variant of CCV, where the comparator gates are three-valued instead of Boolean.

6.2.1 THREE-VALUED CCV is CC-complete

We define the THREE-VALUED CCV problem similarly to CCV, except each wire can now take any of the values 0, 1 or $*$. A wire takes value $*$ when its value is not known to be 0 or 1. The two outputs of a three-valued comparator gate on inputs p and q is defined as follows.

$$p \wedge q = \begin{cases} 0 & \text{if } p = 0 \text{ or } q = 0 \\ 1 & \text{if } p = q = 1 \\ * & \text{otherwise.} \end{cases} \qquad p \vee q = \begin{cases} 0 & \text{if } p = q = 0 \\ 1 & \text{if } p = 1 \text{ or } q = 1 \\ * & \text{otherwise.} \end{cases}$$

Every instance of CCV is also an instance of THREE-VALUED CCV. It is also not hard to show that every instance of THREE-VALUED CCV is AC^0-reducible to an instance of CCV by using a pair of Boolean wires to represent each three-valued wire and adding comparator gates appropriately to simulate three-valued comparator gates. Thus, the THREE-VALUED CCV problem is CC-complete.

6.2.2 A fixed-point method for Sm

We formalize a reduction from Sm to Three-valued Ccv based on [8, 9]. Consider an instance of Sm, where preference matrices for men a, b and women x, y are given in Fig. 7. From this instance of Sm, we construct a three-valued comparator circuit in Fig. 7 as follows.

Men:

a	x	y
b	y	x

Women:

x	a	b
y	a	b

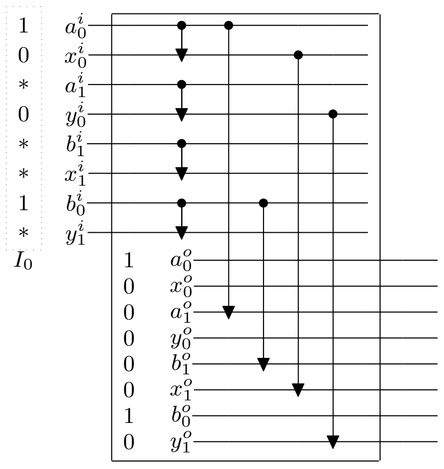


Figure 7

First, since we have two men a, b and two women x, y, we start with four pairs of wires (a_0^i, x_0^i), (a_1^i, y_0^i), (b_0^i, y_1^i), and (b_1^i, x_1^i), connected by four gates $\langle a_0^i, x_0^i \rangle$, $\langle a_1^i, y_0^i \rangle$, $\langle b_0^i, y_1^i \rangle$ and $\langle b_1^i, x_1^i \rangle$ respectively, which represent four possible ways of pairing men a, b to women x, y. The subscripts are important in our construction since the subscript of a person p within a pair indicates the preference of a person about his or her partner in the pair; the superscripts i are less important, and used to indicate that all of these wires are the 'input wires' of this construction. For example, the subscript of b in the pair (b_0^i, y_1^i) indicates that y is a's first choice, and the subscript of y in this pair indicates that b is y's second choice. For convenience, let Pair be a binary predicate such that $\mathsf{Pair}(m_j^i, w_k^i)$ holds iff m is a man and w is a woman and wires m_j^i and w_k^i are paired up, i.e., w is at the j^{th} position of m's preference list and m is at the k^{th} position of w's preference list.

Second, we will introduce four more pairs of 'output wires' (a_0^o, x_0^o), (a_1^o, y_0^o), (b_0^o, y_1^o), and (b_1^o, x_1^o), which are arranged in exactly the same order as input wires, where the subscripts follow the same preference rules as with the input wires. We also define $\mathsf{Pair}(m_j^o, w_k^o)$ to hold iff m is man and w is woman and m_j^o and w_k^o are paired up. Since all subscripts of the wires encode the preference information, they can be used in our construction as follows. Assume that preference lists are of size n, then for every person p, we add a gate from wire p_j^i to p_{j+1}^o for every $j < n - 1$. In our example, we add four gates $\langle a_0^i, a_1^o \rangle$, $\langle b_0^i, b_1^o \rangle$, $\langle x_0^i, x_1^o \rangle$, and $\langle y_0^i, y_1^o \rangle$ as shown in Fig. 7. Note that these gates can be added in any order. It remains to show how to feed inputs to the 'output wires'. We let output wire m_0^o take input one for every man m, and let the rest of output wires have zero inputs.

Given an instance of Sm with n men and n women, define $\mathcal{M} : \{0, 1, *\}^{2n^2} \rightarrow \{0, 1, *\}^{2n^2}$ to be the function computed by the preceding circuit construction, where the inputs of $\mathcal{M}$ are those fed into the input wires, and the outputs of $\mathcal{M}$ are those produced by the output wires. We will use the following notation. Any sequence $I \in \{0, 1, *\}^{2n^2}$ can be seen as an input of function $\mathcal{M}$, and thus we write $I(p_j^i)$ to denote the input value of wire p_j^i with respect to I. Similarly, if a sequence $J \in \{0, 1, *\}^{2n^2}$ is an output of $\mathcal{M}$, then we write $J(p_j^o)$

to denote the output value of wire p^o_j.

Let sequence $I_0 \in \{0, 1, *\}^{2n^2}$ be an input of $\mathcal{M}$ defined as follows: $I_0(m^i_0) = 1$ for every man m, and $I_0(w^i_0) = 0$ for every woman w, and $I_0(p^i_j) = *$ for every person p and every j, $1 \leq j < n$. Note that the number of $*$'s in the sequence I_0 is

$$c(n) = 2n^2 - 2n. \tag{6.1}$$

Our version of Subramanian's method [8, 9] consists of computing

$$I_{c(n)} = \mathcal{M}^{c(n)}(I_0),$$

where $\mathcal{M}^d$ simply denotes the d^{th} power of $\mathcal{M}$, i.e. the function we get by composing $\mathcal{M}$ with itself d times. It turns out that $I_{c(n)}$ is a fixed point of $\mathcal{M}$, i.e. $I_{c(n)} = \mathcal{M}(I_{c(n)})$. To show this, we define a sequence I' to be an *extension* of a sequence I if $I(p) = I'(p)$ for every person p such that $I(p) \in \{0, 1\}$. We can show that $\mathcal{M}(I)$ is an extension of I for every I which extends I_0, and hence $\mathcal{M}^d(I_0)$ extends I_0 for all d. It follows that $\mathcal{M}^{c(n)}(I_0)$ is a fixed point because there are at most $c(n)$ $*$'s to convert to 0 or 1.

Now we can extract a stable marriage from the fixed point $I_{c(n)}$ by letting B be the sequence obtained by substituting zeros for all remaining $*$-values in $I_{c(n)}$. Then B is also a fixed point of $\mathcal{M}$. A stable marriage can then be extracted from B by announcing the marriage of a man m and a woman w if and only if $\mathsf{Pair}(m^o_j, w^o_k)$ and $B(m^o_j) = 1$ and $B(w^o_k) = 0$. Our goal is to formalize the correctness of this method.

In the example in Fig. 7, we can check that the fixed point $I_4 = \mathcal{M}^4(I_0)$ in this case simply consists of Boolean values, where $(I_4(a^o_0), I_4(x^o_0)) = (1, 0)$ and $(I_4(b^o_0), I_4(y^o_1)) = (1, 0)$. Thus, women x, y are married to men a, b respectively, which is a stable marriage.

More formally, given a three-valued sequence I, let $I[* \to v]$ denote the sequence we get by substituting v for all the $*$-values in I. Define G to be an AC^0-function, which takes as input a Boolean fixed point B of $\mathcal{M}$, and returns a marriage M in the way explained above. (Note that since $B = \mathcal{M}(B)$, we have $B(p^i_k) = B(p^o_k)$ for every person p and every $k < n$; however, the superscripts o and i are useful for distinguishing between input and output values of the comparator circuit computing $\mathcal{M}$.) We can prove the following theorem.

▶ **Theorem 14.** *($\mathsf{VCC}^* \vdash$) Let M be a stable marriage of the* Sm *instance $\mathcal{I}$. We let $M_0 = G(I_{c(n)}[* \to 0])$ and $M_1 = G(I_{c(n)}[* \to 1])$. Then M_0 and M_1 are stable marriages, and every man gets a partner in M_0 no worse than the one he gets in M, and every woman gets a partner in M_1 no worse than the one she gets in M. In other words, M_0 and M_1 are the man-optimal and woman-optimal solutions respectively.*

Corollary 10 and Theorems 13 and 14 give us the following corollary.

▶ **Corollary 15.** *($\mathsf{VCC}^* \vdash$) The* Sm *problem is* CC*-complete.*

Proof. Following the above construction, we can write a Σ^B_0-formula defining an AC^0 function that takes as input an instance of Sm with preference lists for all the men and women, and produces a three-valued comparator circuit that computes the three-valued fixed point $I_{c(n)} = \mathcal{M}^{c(n)}(I_0)$, and then extracts the man-optimal stable marriage from $I_{c(n)}[* \to 0]$. Thus the man-optimal (and similarly the woman-optimal) decision versions of Sm are AC^0-many-one-reducible to Three-valued Ccv, and hence also to Ccv. Corollary 10 shows that 3Lfmm is CC-complete, and Theorem 13 shows that 3Lfmm is AC^0-many-one-reducible to Sm. Hence, Sm is CC-complete under AC^0 many-one reductions. ◀

7 Conclusion and future work

Our correctness proof of the reduction from Sm to Ccv is a nice example showing the utility of three-valued logic for reasoning about uncertainty. Since an instance of Sm might not have a unique solution, the fact that the fixed point $I_{c(n)} = \mathcal{M}^{c(n)}(I_0)$ is three-valued indicates that the construction cannot fully determine how all the men and women can be matched. Thus, different Boolean fixed-point extensions of $I_{c(n)}$ give us different stable marriages.

It is worth noting that Subramanian's method is not the "textbook" method for solving Sm. The most well-known is the Gale-Shapley algorithm [4]. In fact, our original motivation was to formalize the correctness of the Gale-Shapley algorithm, but we do not know how to talk about the computation of the Gale-Shapley algorithm in VCC^*. Thus, we leave open the question whether VCC^* proves the correctness of the Gale-Shapley algorithm.

We believe that CC deserves more attention, since on the one hand it contains interesting complete problems, but on the other hand we have no real evidence (for example based on relativized inclusions) concerning whether Ccv is complete for P, and if not, whether it is comparable to NC. The perfect matching problem (for bipartite graphs or general undirected graphs) shares these same open questions with Ccv. However several randomized NC^2 algorithms are known for perfect matching [5, 7], but no randomized NC algorithm is known for any CC-complete problem.

Another open question is whether the three Ccv complexity classes mentioned in (1.1) coincide, which is equivalent to asking whether CC (the closure of Ccv under AC^0 many-one reductions) is closed under AC^0 oracle reductions, or equivalently whether the function class FCC is closed under composition. A possible way to show this would be to show the existence of universal comparator circuits, but we do not know whether such circuits exist.

The analogous question for standard complexity classes such as TC^0, L, NL, NC, P has an affirmative answer. That is, each class can be defined as the AC^0 many-one closure of a complete problem, and the result turns out to be also closed under AC^0 oracle reducibilities. (A possible exception is the function class $\#\mathsf{L}$, whose AC^0 oracle closure is the $\#\mathsf{L}$ hierarchy [1]. This contains the integer determinant as a complete problem.)

References

1 E. Allender and M. Ogihara. Relationships among PL, #L, and the determinant. *RAIRO, Theoretical Informatics and Applications*, 30(1):1–21, 1996.

2 K.E. Batcher. Sorting networks and their applications. In *Proceedings of the AFIPS Spring Joint Computer Conference 32*, pages 307–314. ACM, 1968.

3 S. Cook and P. Nguyen. *Logical foundations of proof complexity.* Cambridge University Press, 2010.

4 D. Gale and L.S. Shapley. College admissions and the stability of marriage. *The American Mathematical Monthly*, 69(1):9–15, 1962.

5 R.M. Karp, E. Upfal, and A. Wigderson. Constructing a perfect matching is in random NC. *Combinatorica*, 6(1):35–48, 1986.

6 E.W. Mayr and A. Subramanian. The complexity of circuit value and network stability. *Journal of Computer and System Sciences*, 44(2):302–323, 1992.

7 K. Mulmuley, U.V. Vazirani, and V.V. Vazirani. Matching is as easy as matrix inversion. *Combinatorica*, 7(1):105–113, 1987.

8 A. Subramanian. *The computational complexity of the circuit value and network stability problems.* PhD thesis, Dept. of Computer Science, Stanford University, 1990.

9 A. Subramanian. A new approach to stable matching problems. *SIAM Journal on Computing*, 23(4):671–700, 1994.