Question Answering and Cognitive Automata with Background Intelligence

Jan van Leeuwen

Jiří Wiedermann

Technical Report UU-CS-2016-007
May 2016

Department of Information and Computing Sciences
Utrecht University, Utrecht, The Netherlands
www.cs.uu.nl

ISSN: 0924-3275

Department of Information and Computing Sciences
Utrecht University
Princetonplein 5
3584 CC Utrecht
The Netherlands

Question Answering and Cognitive Automata with Background Intelligence*

Jan van Leeuwen[1] Jiří Wiedermann[2]

[1] Centre for Philosophy of Computer Science, Utrecht University
P.O. Box 80.089, 3508 TB Utrecht, The Netherlands.
j.vanleeuwen@cs.uu.nl

[2] Institute of Computer Science of AS CR, Prague, Czech Republic
jiri.wiedermann@cs.cas.cz

Abstract. Cognitive automata model the human capability of processing information and giving feedback. We augment cognitive automata with a form of intelligence, by giving them the possibility to query a given Turing machine and use the answers from one interaction to the next. We demonstrate that the augmented cognitive automata obtained in this way are equivalent to so-called question-answering Turing machines with advice and persistent learning space.

The result sheds light on the power of question-answering. As a concrete application, we show that polynomially bounded cognitive automata with 'background intelligence' are equivalent to polynomially bounded question-answering Turing machines with advice and logarithmic learning space. This generalizes Pippenger's theorem from 1979 which relates the power of switching circuits to Turing machines with advice in non-uniform complexity theory, linking the latter field to cognitive science.

The computational models involved can be seen as simulating a human player (the cognitive automaton with background intelligence) and a question-answering computer like Watson (a Turing machine with advice and learning space) respectively, giving bounds on the amount of resources needed by one model to match the question-answering abilities of the other. The results reveal why Watson has an advantage.

Keywords: cognitive automata, intelligence, learning space, oracles, persistence, Pippenger's theorem, QA-sessions, Turing machine with advice, Watson.

1 Introduction

In 2011, two experienced human players were beaten by IBM's supercomputer *Watson* at the game of `Jeopardy!` [2, 4]. Can the question-answering capabilities of the human players and those of a machine like Watson be placed in a rational perspective? In particular, can their potential be captured and compared in computational terms? In this paper we attempt to do so.

In our approach to the problem, we make use of theoretical models for the human players and Watson respectively, extending classical automaton-theoretic conceptions for both of them. We then compare the relative capabilities of human players and Watson-type machines in question-answering sessions (QA-sessions). In these sessions, sequences of questions are asked that are coded as words of length (at most) n, for some fixed but arbitrary $n \in \mathbb{N}$. We assume that all questions (input words) and answers (output words) are written over a fixed alphabet Σ with $|\Sigma| = \sigma \geq 2$.

* Version dated May 31, 2016. This research was partially supported by RVO 67985807 and GA ČR grant No. 15-04960S.

In order to compare the qualities of humans and machines in question-answering, we need suitable models that express their different ways of dealing with input, output, memory, discovery, and reasoning. The options for modeling the underlying processes in a fitting architecture are plenty. We will use a combination of abstract computational models that all have a long standing in the philosophy of mind and in cognitive science [14, 16], with extensions as they are studied in modern complexity theory [1].

A guiding principle in the modeling effort is the *duality* between human players and question-answering machines, in the following sense. One may view human players as agents that consist of a knowledge-laden brain and a 'background intelligence' that can answer queries on call. On the other hand, one may view question-answering machines as agents that consist of a computational core to answer queries, and a 'background intelligence' that contains all knowledge that it can use.

Finally, we embed some form of 'learning' in the models. Concretely, we will assume that a suitable facility is in place for preserving some amount of data in passing from one question to a next within a session. The human player could preserve his 'state of mind' after a question, a question-answering machine could preserve the contents of a kind of 'persistent memory'. No data is preserved in either system across session boundaries. The question arises how these facilities enrich the question-answering capabilities of each of the models and how they compare.

1.1 Models: Overview and Intuitions

We give a first impression of the models as we envision them, below. We also preview the notion of QA-sessions that we use.

Modeling the human player Cognitive science often models the information processing abilities of humans by means of finite-state automata with some kind of interactive quality. As processing qualities are expected to evolve as inputs get longer, we will assume that a separate automaton C_n is specified and called into action for each input length $n \geq 1$. 'Intelligence' is provided to each automaton, by allowing each C_n to access the computational support of a shared Turing machine M that acts as a 'knowledge source' (or oracle) to all of them.

With these conceptions, a human player can now be modeled as a *cognitive array* (CA) consisting of the following parts: a certain family $C = \{C(n)\}_{n\geq 1}$ of 2-way finite-state automata, and a single background Turing machine M_C that can be called by the automata. We assume that each CA C comes with a function s such that for each n, $s(n)$ bounds the size of its component $C(n)$.

Given a cognitive array C, a QA-session on inputs of length n proceeds by *iterating* $C(n)$ on the successive inputs of the session. While $C(n)$ is processing an input question it can issue queries to M_C, using an oracle tape that is 'left-to-right write-only' during the process. The oracle tape is erased before a next question in the session is taken up, but $C(n)$ is assumed to be persistent in the sense that it continues in the state it was in after answering its most recent input.

Modeling Watson-type machines In computing, the information processing abilities of question-answering machines such as Watson tend to be modeled by means of augmented Turing machines that have some kind of (potentially infinite) 'knowledge base' at their disposal. As the complexity of questions evolves as inputs get longer, we assume that for each $n \geq 1$ an appropriate string of data $B_n \in \Sigma^*$ is specified which is called into memory from the knowledge base whenever an input of length n is entered. Further data may accumulate during a session in a separate 'learning space' and may be passed on as newly discovered knowledge between successive queries. When a session ends, the learning space is cleared.

With these broad ideas we can define a machine like Watson as a *question answering machine* (QA) consisting of the following parts: a multi-tape Turing

machine W, a tape that serves as a learning space, and a knowledge base $B = \{B(n)\}_{n\geq 0}$. During a QA session, W iterates on a series of questions of some fixed length n. Every time a question has been answered and a next one is presented in the session, W re-initializes but it does so while preserving some information that was gained in the handling of the previous question (in its learning space). This gives the model the capability to 'learn' during a session.

To measure their effect, we assume that every QA machine W comes with functions b and l such that $|B(n)| \leq b(n)$ for all n and learning space in QA-sessions on inputs of length n always remains bounded by $l(n)$.

Modeling Question Answering We aim to compare the human players and their Watson-type competitor by focusing on the essence of their question-answering abilities. We will not consider the mismatch in speed nor any competitive interaction that would allow one party to step in if another one is slow or fails an answer. We will not even check whether questions adhere to a specific domain description or whether the answers that are given are 'correct'. For a background on the theories of question answering see e.g. [13].

We will focus entirely on a comparison of the fundamental question-answering abilities of the different agents. Can one agent (e.g. the human player) *in principle* come up with the same answers as the other agent (Watson)? Can one agent simulate the other and if so, what are the scaling effects involved? To make this precise, several more details have to be agreed upon.

Definition 1. *A QA-session on inputs of length n consists of a finite sequence of inputs $u_1, \cdots$ with $u_i \in \Sigma^n$ $(i \geq 1)$ and corresponding outputs $v_1, \cdots$ with $v_i \in \Sigma^*$ such that for each i, v_i is output by the machine on input u_i after it has completed processing $u_1, \cdots, u_{i-1}$. A QA-session is $h(n)$-bounded, for some non-decreasing function h, if the length of all outputs in the session is bounded by $h(n)$.*

Definition 2. *Given a CA or QA machine, the process of answering any individual question (from input to corresponding output) in a session is called a* run *of the respective machine in the session.*

In order to make sure that CA's and QA machines always answer, we assume for both that the runs on individual questions of length n in a session are implicitly or explicitly bounded by some time-bound $f(n)$. We assume that $f(n) \geq n$, as the entire input must at least be read. If $f(n)$ is bounded by a polynomial, we say that the corresponding CA or QA machine is 'polynomial-time bounded'. We say that a CA or QA machine is *polynomially bounded* if all its respective parameters are polynomially bounded.

1.2 Discussion

The models we use are potentially very powerful, computationally. For example, having a separate finite-state automaton $C(n)$ for each input length n, already gives CA's the processing power of general 'resource-bounded' families of circuits [1]. Also, QA machines build on Turing machines with bounded *advice*, the latter being an oracle facility similar to the notion of knowledge base as we defined it. CA's and QA machines have several more features that we need for our modeling purposes.

Circuits and Turing machines with advice are common to non-uniform complexity theory, and their *duality* is well-known in this theory [1]. For example, a result known as Pippenger's Theorem asserts that a set is 'recognizable' by a polynomial-size bounded family of circuits if and only if the set can be recognized by a polynomial-time bounded Turing machine with polynomial-length bounded advice [15, 10, 1]. (For a more general formulation, see e.g. [18], Section 2.3.) This raises

the interesting question whether a more general duality result may be achieved, notably for the models we have defined.

CA's and QA machines not only have advanced 'hardware features'. The iterative behaviour during QA-sessions gives new computational possibilities and in fact a whole hierarchy of them, as we will show in Section 2. The crucial property in both models is that information may be passed from the answering of previous questions to the next, either by state (CA) or through learning space (QA). This makes the answering of questions dependent on everything that happened in the session before. (Note that no information is passed between different sessions.)

Question-answering scenarios, in combination with the passing of information in between consecutive questions, have been considered before in studies of *persistent Turing machines* [5–8, 11]. In these studies, question-answering and persistence have been considered e.g. as a model of interactive computation and as a model for computing on infinite streams of structured data. See also our studies of *interactive Turing machines with advice* [17] We use question-answering in a different way and in a different architectural context, as a means to compare the relative performance of the different agents in our framework.

1.3 Outline

We compare the capabilities of human players and Watson-type computers in QA-sessions by analyzing the models we designed for them: CA's and QA machines, respectively. The models are described in detail in Section 2.

In Section 2 we first show that CA's and QA machines become strictly more powerful when their 'capabilities for learning' grow. For CA's this corresponds to a growing brain size $s(n)$, and for QA machines to a growing size $l(n)$ of the learning space.

In section 3 we develop the general theoretical results that show that CA's and QA machines can simulate each other, within quantifiable resource bounds. This leads to the following key theorem: *polynomially bounded cognitive arrays realize the* same *QA-sessions as polynomially bounded QA machines with* $O(\log n)$*-size bounded learning space can.* This establishes the desired duality result for cognitive automata with background intelligence.

The main result provides a powerful extension of Pippenger's theorem for plain (acyclic, memoryless) switching circuits to the complex context of cognitive automata with advanced features like persistent memory and background intelligence. The theorem and some further consequences are presented in Section 4. In Section 5 we give some final conclusions.

2 Models in Detail

In order to derive precise simulation results, we need a more complete description of the models we have in mind. Recall that both cognitive arrays and QA machines are designed to perform QA-sessions of successive questions (inputs) of length n, for some n that is fixed throughout a session. In Section 2.1 we describe how cognitive arrays operate, in Section 2.2 we describe QA machines in detail. Finally, in Section 2.3 we comment on the use of QA-sessions for characterizing the strength of question-answering agents in general and prove a fundamental result for it.

2.1 Modelling Human Players: Cognitive Arrays

The premise of cognitive arrays is that question answering is based on the knowledge that is stored in the 'states' of the human brain, with an inference mechanism

provided by a Turing machine that acts as a *computable oracle*. Hence, the inference mechanism is potentially far more powerful than plain finite-state reasoning. The precise architecture of the model is as follows.

A cognitive array C consists of an infinite sequence of components $\{C(n)\}_{n\geq 1}$ and one background multi-tape Turing machine $M = M_C$. Each $C(n)$ is a finite-state machine, with $s(n)$ states and a special capability to create $q(n)$-size queries on an internal oracle tape and pose them to M. Whenever M is queried, we assume that it always responds 'superfast' (i.e. in one step).

If a QA-session starts and its first input of length n appears, C initializes $C(n)$ and delegates the further handling of the session to this automaton. Suppose $C(n)$ has handled zero or more inputs of the session and now receives the next question, say x, on its input tape ($|x| = n$). $C(n)$ continues in the state it was in after completing the answering of the preceding input, but its oracle tape is emptied before it starts its run on x. Now $C(n)$ acts as follows.

First of all, $C(n)$ processes x like a *two-way* automaton, with state transitions determined by the current input or oracle signal and the current state. (In *one-way* CA's, this is limited to a one-way behaviour on x.) The oracle signal can be 'yes' or 'no' and is received from a query request to machine M. (How this works is explained below.) In each action, also some output may be written to its one-way oracle tape and to a one-way output as well.

During its run on an input x, $C(n)$ can query the fixed background Turing machine M. To this end, $C(n)$ has its oracle tape of at most $q(n)$ positions, which is write-only and *written from left-to-right*. (When the oracle tape is full, an end marker will prevent it from overflowing.) When $C(n)$ gets into an oracle state, M is queried with the contents of the oracle tape, and $C(n)$ moves according to the answer it gets ('yes' or 'no') to a new state. The oracle tape is *not* emptied but keeps being written to by adding things to the right. M does not retain any information from one query to a next.

This continues until the machine stops, completing its answer to x on the output. When $C(n)$ is given a new input x' with $|x'| = n$, *it continues in the state it was in* but with an *emptied* oracle tape (i.e. the tape contents does not transfer). This continues input after input, as long as the session lasts. We assume that every run of $C(n)$ on an input in the session n is $f(n)$-time bounded.

When M is queried as an 'oracle' we assume that it gives its answer like an oracle 'in one step'. However, for consistency we require that M gives the signal 'yes' precisely when it halts on the query in an accepting state, 'no' when it halts in a rejecting state or when it does not halt at all. (This conforms to M being a Σ_1-oracle.) We let $r(n)$ be the maximum time M computes in any halting computation on queries of size at most $q(n)$. (Clearly $r(n)$ is generally a non-computable function but we use it purely as a mathematical function here.)

Definition 3. *A cognitive array C is called a $<s(n), q(n), r(n), f(n)>$-array, if C consists of a sequence of 2-way finite-state components $\{C(n)\}_{n\geq 1}$ and a background Turing machine M which conform to the parameters specified above: $s(n)$ bounds the number of states of $C(n)$, $q(n)$ bounds the length of the queries $C(n)$ can ask M, $r(n)$ is the maximum time M needs in any halting computation on queries posed by $C(n)$, and $f(n)$ bounds the time of any run by $C(n)$, all on inputs of length n.*

Note that the representation of $C(n)$ with a full transition table actually requires $O(\log|\Sigma| \cdot s(n) \log s(n))$ bits.

2.2 Modelling Watson-type computers: QA Machines

QA machines are based on the premise that question answering is driven by an inference mechanism modeled as a Turing machine, supported by the retrieval of

information from a large external knowledge base and a learning facility. The architecture of QA machines is as follows.

A QA machine consists of a front-end multi-tape Turing machine W with the following features: (a) W can access a knowledge base B containing strings $B(n) \in \Sigma^*$ of size $b(n)$ (at most one for each $n \geq 1$), and (b) W has a learning space (i.e. tape) of size $l(n)$ on which it can accumulate some information for later use in a given session. In addition, W has its usual set of working tapes as a multi-tape machine.

In a QA-session, W operates on successive inputs x with $|x| = n$ in the expected way, each time waiting until the answer to a question has been completed on its one-way output next input is taken in. While it operates on an input of length n, W can access its knowledge base to retrieve the stored item $B(n)$ from it with $|B(n)| \leq b(n)$. This information acts as an *advice* [10] that stays with the machine for the rest of the session. The string $B(n)$ presumably holds all the relevant information which W can extract from its knowledge base and use in a single session.

W is not just a machine with 'advice'. W also has a special tape of size $l(n)$ that acts as a learning space. The learning space is a *persistent* memory that keeps its contents in between different 'runs' of the machine on queries of length n. The space allows W to accumulate some additional knowledge during a QA-session, e.g. by storing useful facts that it learns from its inputs. All other working tapes are cleared when a new run begins.

A QA machine basically gets all its knowledge from its knowledge base B, and from what it learns from its query processing and can keep in its learning space during a session. Clearly, the QA-session thus depends on the historical information W builds up in its learning space, hence on the particular chain of inputs it is provided during a session. A different chain of inputs might give different answers to identical queries later on, as one expects from a *learning machine*, although the accumulated data is only used during a single session.

We assume that a run of W always halts within a given time-bound $f(n)$, implying that questions are always answered in finite time.

Definition 4. *A QA-machine is called a $<b(n), l(n), f(n)>$-machine, if it consists of a front-end Turing machine W and a knowledge base B with $B(n) \in \Sigma^*$ ($n \geq 1$) which conform to the parameters specified above: $b(n)$ bounds the length of $B(n)$, $l(n)$ bounds the size of the learning space throughout a session, and $f(n)$ bounds the time of any run by the machine, all on inputs of length n.*

We usually 'name' a QA machine by the name of its Turing machine. Note that outputs in a QA-session with a $< b(n), l(n), f(n) >$ machine are necessarily $f(n)$-bounded.

2.3 Question Answering: On the Power of Learning Space

Question answering is a standard type of information retrieval and filtering. We defined QA-sessions abstractly. In a session, questions of a specified complexity (length) are to be supplied by an external party, where next question is supplied only if the answer to the current question has been output. We do not concern ourselves with the particular way in which the external party determines its next question.

The answering party could be any agent, natural or artificial. In our present models we have amplified different options for the way agents may store and act with information. While QA-sessions were designed primarily as a means to compare the strength of different answering agents, one could as well use QA-sessions to characterize the potential of any one answering party in particular. Let $\mathcal{A}$ be any answering agent.

Definition 5. *The* QA-script *of $\mathcal{A}$ is the set $R(\mathcal{A})$ of all QA-sessions that are realized using $\mathcal{A}$ as the answering agent.*

Given a family of answering agents, one may study the class of all QA-scripts that can be realized by these agents, in the same way as classes of formal languages are studied in relation to the automata that accept them [9]. Some aspects of this are studied in [5–7, 10]. We do not pursue this question here but only highlight the effect that various *model parameters* can have on the class of QA-scripts that can or cannot be realized.

The quality of the question-answering during a session depends on two factors: one is the intrinsic computational ability of the question-answering agent by itself, another is the facility by which information can be passed on from the answering of one question to the next. This information can be viewed as a 'residue' from preceding answers that can help in future questions in the same session [12].

Focusing on the latter, for example, it is intuitive that the more information can be passed, the more 'powerful' the question-answering during a session becomes. We give a concrete example for QA-machines. The argument exploits a technique from automata theory involving crossing sequences [9].

Theorem 1. *For any computable function $L : \mathbb{N} \to \mathbb{N}$, there exists a QA-script $\mathcal{S}$ that can be realized by a QA-machine within a learning space of size $L(n)$, but not by any QA-machine with a learning space of size $L'(n)$ with* $\inf_{n\to\infty} \frac{L'(n)}{L(n)} = 0$.

Proof. Let $n \in \mathbb{N}$ be arbitrary, and set $k = \lceil \frac{L(n)}{n} \rceil$. (The value of k can be computed by a QA-machine anytime it needs it.) We will consider sessions with questions of length n. Define a question-block to be any sequence of k questions (of length n each) of the form $uv_0, v_1, \cdots, v_{k-1}$ where u is a fixed default string of length $kn - L(n)$. Obviously $|v_0|+\cdots+|v_{k-1}| = L(n)$. Given a question-block $\alpha = uv_0, v_1, \cdots, v_{k-1}$ we say that $v_0v_1 \cdots v_{k-1}$ is the string stored in it. This gives a unique correspondence between question-blocks and strings of length $L(n)$ over Σ.

View strings of length $L(n)$ over Σ as σ-ary numbers. Let $l = \sigma^{L(n)}$ and let $\alpha_0, \cdots, \alpha_{l-1}$ be the question-blocks corresponding to the complete list of length-$L(n)$ strings over Σ in enumeration order. We use these blocks to define the QA-script $\mathcal{S}$ consisting of sessions which adhere to the following rules. Our aim is to view any session as a sequence of blocks of k questions each. First assume that the number of questions in a session is a multiple of k.

- If the first block of k questions equals α_0, then the corresponding block of outputs consists of $k-1$ times *no* followed by one *yes*, otherwise it consists of k times *no*.
- If the last block of k answers consisted of k times *no*, then every next block of k questions is answered by k times *no*.
- Suppose the last block of k answers consisted of $k-1$ times *no* followed by one *yes* and that this last block of k questions equaled α_i (some $0 \le i \le l-1$). Let $i' = (i+1) \mod l$. If the next block of k questions corresponds to $\alpha_{i'}$, then the corresponding block of outputs consists of $k-1$ times *no* followed by one *yes*, otherwise it consists of k times *no*.

If the number of questions in a session is not a multiple of k, then apply the rules to the largest prefix with a multiple of k questions and answer *no* to the remaining (fewer than k) questions.

Claim. $\mathcal{S}$ is realized by a QA machine with a learning space of size $L(n)$.

Proof. (Sketch.) A QA-machine W can be designed which answers in all sessions exactly according to the rules. The idea is to work through blocks of k consecutive

questions, and pass a copy of the desired α_j from question to question in learning space. By checking that the k consecutive parts of α_j correspond exactly (from left to right) to the k consecutive questions in the input, W can verify that the desired match occurs. The process is initialized by starting with the empty (default) learning space. If a match is made, the contents of learning space is incremented by one as a number (modulo l), and W is ready for the next block of k inputs. If no match is made, then the *no* answer perpetuates from then onward. Note that for passing a question-block in learning space, it suffices to pass only the 'significant' part of the block that corresponds to the size-$L(n)$ string stored in it. By marking segments, one can keep track of which part of the block must match the current input question. This can all be encoded within $L(n)$ workspace. □

Now let $L'(n) : \mathbb{N} \to \mathbb{N}$ be any other function, not necessarily computable, and assume that $\inf_{n\to\infty} \frac{L'(n)}{L(n)} = 0$.

Claim. $\mathcal{S}$ cannot be realized by any QA machine with a learning space of size $L'(n)$.

Proof. Suppose there was a QA-machine W' that realized $\mathcal{S}$ with a learning space of size $L'(n)$. For some n to be chosen later, consider how W' would act in a session with questions $\alpha_0, \alpha_1, \cdots, \alpha_{l-1}, \alpha_0$. Clearly, W' must answer with repeated blocks of $k-1$ times *no* followed by one *yes*.

Let W' have a working alphabet on its learning tape of δ symbols. Then there can be at most $\delta^{L'(n)}$ different learning tape contents when W' passes from some block α_i to the next ($0 \le i \le l-1$). By assumption on L' there must be infinitely many values of n such that $L'(n)/L(n) < \log\sigma/\log\delta$. For any such value of n we have $\delta^{L'(n)} < l$.

Let n be any value with this property. By the pidgeonhole principle, there are indices i, j with $0 \le i < j \le l-1$ such that W' passes from α_i to α_{i+1} and from α_j to $\alpha_{j'}$ with the same learning tape contents, where $j' = (j+1) \mod l$. But, then we can omit the questions from α_{i+1} to α_j and W' must answer consistently with repeated blocks of $k-1$ times *no* followed by one *yes* even though the series of questions has been altered to $\alpha_0, \cdots, \alpha_i, \alpha_{j'}, \cdots$ which clearly should have led to k times *no* when $\alpha_{j'}$ is processed. Contradiction. □

This proves the result. □

A similar result can be shown for cognitive arrays, using the size function $s(n)$ of the state sets as a discriminator: the faster $s(n)$ grows with n, the more powerful the question-answering can be. The results are consistent with the controversial claim that *a bigger persistent memory (or, brain) makes a machine smarter.*

3 Cognitive Arrays versus Question-answering Machines

With the detailed models at our disposal, the key issue of this paper can be phrased in the following concrete terms: *are human players and Watson-type QA-machines able to realize the same QA-scripts*? And, if the answer is positive, are there any appreciable differences in effectiveness between the two, not counting the obvious difference in speed?

We will show that the question-answering power of cognitive arrays corresponds exactly to that of suitably configured QA-machines, and vice versa. We prove this by showing that one model can realize the same QA-sessions as the other, with the right parameter setting.

Throughout this section we consider the performance of the models during QA-sessions on inputs of length n. We assume that all models work with a fixed alphabet Σ with $|\Sigma| \ge 2$. Constant factors will be suppressed in the various bounds.

3.1 Simulating Human Players by QA-Machines

We first show how QA machines can 'simulate' the capabilities of cognitive arrays of any given setting.

Theorem 2. *Let Φ be any QA-session. If Φ is realized by a $<s(n), q(n), r(n), f(n)>$-array, then the session can also be realized by a $<s(n)\log s(n)+q(n), \log s(n), q(n)+f(n)(r(n)+s(n)\log s(n))>$-machine.*

Proof. Let C be an $<s(n), q(n), r(n), f(n)>$-array and let $M = M_C$ be the multi-tape Turing machine it uses as an oracle. Consider the QA-session Φ on inputs of length n. We design a QA machine W that simulates $C(n)$ as follows.

First of all we need to deal with the simulation of M on queries of length up to $q(n)$. Define:

$$\gamma(n) = \text{"the lexicographically least string of length} \leq q(n) \text{ that leads } M \text{ to its longest halting computation over all strings of length} \leq q(n)\text{."}$$

If W knows $\gamma(n)$, then it can simulate M in finite time, using the computation on $\gamma(n)$ as a yardstick to decide when to terminate any computation of M on a query of length $q(n)$ or less. It is immaterial for our purposes that $\gamma(n)$ may be non-computable as a function of n.

We now define W in a number of steps. To begin with, we give it a knowledge base B with

$$B(n) = \text{"the description of } C(n) \text{ followed by } \gamma(n)\text{"}.$$

Given any input x with $|x| = n$, W first retrieves $B(n)$ and unpacks the description of $C(n)$ and $\gamma(n)$ (encoded e.g. in binary) in $s(n)\log s(n) + q(n)$ time and then starts to simulate the execution of $C(n)$ on x. Whenever an oracle call to M is made, W simulates the call in finite time with the computation on $\gamma(n)$ to bound it.

When the simulation of $C(n)$ ends, W stores an encoding of the ending state of $C(n)$ in its learning space. This will enable W to start the simulation of $C(n)$ on a next input in the session correctly, namely in the state which it has recorded in its learning space. Repeating this, W precisely realizes Φ.

We conclude by noting that $|B(n)| = b(n) \leq s(n)\log s(n) + q(n)$ and $l(n) \leq \log s(n)$. The simulation of $C(n)$ takes $O(s(n)\log s(n))$ per step, to retrieve the right move from the encoded transition table, plus an additional $O(r(n))$ time if the oracle is called. □

Note in the given proof that we did not really need to 'know' $r(n)$ or $f(n)$ in the construction but we merely used them for bounding the time-complexity of W.

Corollary 1. *If a QA-session Φ is realized by a $< s(n), q(n), r(n), f(n) >$-array with $r(n) \geq s(n)\log s(n)$, then the session can also be realized by a $<s(n)\log s(n)+f(n), \log s(n), f(n)r(n)>$-machine.*

Proof. By the operation of $C(n)$ we may assume that $q(n) \leq f(n)$. Assuming that $r(n) \geq s(n)\log s(n)$ we obtain the desired result, suppressing constant factors. □

Corollary 2. *If a QA-session Φ is realized by a $<s(n), q(n), r(n), f(n)>$-array with $r(n) \geq s(n)\log s(n)$, then Φ can also be realized by a $<s(n)(nq(n)+\log s(n)), \log s(n), ns(n)q(n)r(n)>$-machine.*

Proof. By the operation of $C(n)$ we may assume that $f(n) \leq n \cdot s(n) \cdot q(n)$. After all, in between any two subsequent writes to its oracle tape, the finite-state machine $C(n)$ cannot make more than $n \cdot s(n)$ moves, otherwise it would repeat an input position/state pair and get into a loop. Now apply Corollary 1 to get the desired bound. □

3.2 Simulating Watson-type QA-Machines by Cognitive Arrays

Next we reduce the question-answering intelligence of QA-machines to that of, suitably dimensioned, cognitive arrays. The simulation shows what parameter settings are needed for the human player, to match the capabilities of a QA-machine.

Theorem 3. *Let Φ be a QA-session. If Φ is realized by a $<b(n), l(n), f(n)>$-machine, then the session can also be realized by a $<b(n) + 2^{O(l(n))}, b(n) + l(n) + f(n), f(n), b(n) + 2^{O(l(n))} + f(n)>$-array.*

Proof. Let W be a $<b(n), l(n), f(n)>$-machine, and let B be the knowledge base it uses. We show that a cognitive array C can be designed such that for every n, the automaton $C(n)$ encodes W's knowledge insofar as it is needed in processing inputs of length n and can simulate W during the session on successive inputs x as processed in Φ, using an suitable multi-tape Turing machine $M = M_C$ as background intelligence.

We develop the description of $C(n)$ while outlining the various phases it goes through in processing a next input x of the session. Each time we unveil a next set of states that $C(n)$ needs in a phase.

Phase I: Preparations.

First we need a representation of the initial contents of the learning tape. Let A be the complete σ-ary tree of depth $l(n)$, with the edges at every node labeled by the different symbols of Σ. Each node of A thus represents a possible contents of W's learning space, by collating the edge labels on the path from the node to the root. With this correspondence we identify each internal node of T with a possible tape contents of the learning space, a so-called A-state. This leads to some $\sigma^{l(n)}$ A-states.

Assume that $C(n)$ starts its new run in state α, corresponding to the contents of W's learning space at the end of the previous (simulation of a) run. $C(n)$ first reads and copies its input x to its oracle tape, symbol by symbol. Next, it switches over to a different mode and moves from state α up the tree towards the root, copying the symbols on the way to the oracle tape as well.

The oracle tape now contains $<x, \overrightarrow{\alpha}, \cdots>$, where $\overrightarrow{\alpha}$ is the string corresponding to state α. Note that the total number of states to execute this phase so far is bounded by $O(\sigma^{l(n)})$.

Next, $C(n)$ needs to 'internalize' $B(n)$. To this end, we hardwire $B(n)$ into $C(n)$, by means of a row of (up to) $b(n)$ states. After completing the previous steps, $C(n)$ starts at the beginning of the row of states and moves from left to right, outputting the corresponding symbols to its oracle tape again. When the last state in the row is passed, $C(n)$ moves to the next phase.

Phase II: Generating output

The oracle tape now contains $<x, \overrightarrow{\alpha}, B(n), \cdots>$. It is time to start generating output. To do so, $C(n)$ moves to a yet another set of special states, the Y-states.

In its Y-states, $C(n)$ repeatedly adds a 1 to its oracle tape and then queries its oracle $\sigma + 1$ times as follows. When the oracle tape contains $<x, \overrightarrow{\alpha}, B(n), 1^y>$, the following oracle queries are asked, by moving through subsequent states:

- for each $\sigma \in \Sigma$: "does W print out a symbol σ in the y-th step of its computation on input x, with $\overrightarrow{\alpha}$ as its initial learning tape contents and $B(n)$ as the data in its knowledge base for this session?"
- and finally: "does W terminate in the y-th step of its computation on input x, with $\overrightarrow{\alpha}$ as its initial learning tape contents and $B(n)$ as the data in its knowledge base for this session?"

Clearly we need $\sigma + 1$ Y-states to perform the queries. Whenever the answer to a σ-query is 'yes', $C(n)$ outputs σ and continues its cycle. If the answer to the termination-query is 'no', $C(n)$ writes another 1 to its oracle tape and goes a new cycle. If the answer is 'yes', $C(n)$ ends the cycling and moves to the next phase.

Note that at this point $C(n)$ has generated the output W would have generated. Note that the oracle machine must be able to answer its queries in the time W takes for its computations, i.e. within $f(n)$ steps.

Phase III: Wrapping up.

Note that the oracle tape now holds $<x, \overrightarrow{\alpha}, B(n), 1^K>$, where K ($K \leq f(n)$) is the number of steps W takes to complete its output on x. To wrap up, $C(n)$ must reconstruct the contents of the final learning tape and move to the corresponding A-state, to prepare for the next input. To this end, $C(n)$ moves to its Z-states.

The Z-states are in fact a replica of the A-states, with transitions between them now corresponding to a level-wise enumeration order of tree A. Moving 'from left to right' through its Z-states, $C(n)$ repeatedly adds yet another 1 to the oracle tape and queries the oracle with $<x, \overrightarrow{\alpha}, B(n), 1^K, 1^z>$ on its tape with the following question:

– "does W terminate with a contents corresponding to the z-th node in the enumeration in its learning space, in the computation on input x, with $\overrightarrow{\alpha}$ as initial learning tape contents and $B(n)$ as the data in its knowledge base for this session?"

Once the oracle says 'yes', we have in fact reached the state in the duplicate A-tree that corresponds to the contents of the learning space that must carry over and $C(n)$ can move to the corresponding equal A-state α', to prepare for a next run in the session. Note that we do not need access to K for these last queries, although we have its value on the oracle tape.

For the final accounting, observe the following. The required oracle is easily seen to be implementable as a uniform TM M for all n. Considering the queries we ask, M always halts and does so within time $f(n)$ by definition. A single run of $C(n)$ takes time in the order of $n+l(n)+b(n)+\sigma f(n)+\sigma^{l(n)} = O(b(n)+2^{O(l(n))}+f(n))$, with oracle queries taking 'one step'. We assume that $f(n) \geq n$. □

Interestingly, the construction automatically yields a one-way cognitive automaton that simulates the QA-machine within the given complexity bounds.

Corollary 3. *Let Φ be a QA-session. If Φ is realized by a $<b(n), l(n), f(n)>$-machine, then it can also be realized by a* one-way $<b(n) + 2^{O(l(n))}, b(n) + l(n) + f(n), f(n), b(n) + 2^{O(l(n))} + f(n)>$*-array.*

Proof. Considering the construction above, we see that Phase I is the only phase in which the (next) input of a session is concretely read by $C(n)$. It does so in a straight one-way manner. □

Note once again that we did not really need $f(n)$ in the construction but merely used it for bounding the time-complexity of $E(n)$. We may clearly assume w.l.o.g. that $f(n)$ dominates both $b(n)$ and $l(n)$, as W must read both the query and the learning tape or can be made to do so. This leads to the following corollary.

Corollary 4. *If a QA-session Φ is realized by a $<b(n), l(n), f(n)>$-machine and $f(n) \geq \max\{b(n), l(n)\}$, then the session can also be realized by a (one-way) $< b(n) + 2^{O(l(n))}, f(n), f(n), f(n) + 2^{O(l(n))}>$-array.*

Proof. The bounds follow from Theorem 3. □

4 Some General Consequences

It is tempting to hypothesize that there are no principal differences in intellectual capability between a human player and a question-answering machine like Watson, except for those caused by the sheer unlimited memory and processing speed of the latter. In the models we present it becomes meaningful to address this question. Can the perceived advantage of question-answering machines as artificial brains be made tangible?

The simulation results proved in Section 3 are tedious but they show that in principle, cognitive arrays and QA-machines are *equally powerful*. The models can be matched to each other on any QA-script, provided they are tuned as in the simulations. We discuss some more detailed aspects of this. For example, do the simulations point to any differences in effectiveness between the two models and thus, between the human players and Watson? In this section we discuss what the quantised analysis of the models tells us.

4.1 Overview

Our comparison between human players and machines focuses entirely on their capabilities in question-answering, in relation to the 'system parameters' we distinguished for each of them. We abstracted away from physical details and allowed for perfect generality, in the scenario of QA-sessions and scripts. This seems to be a new approach to the analysis of question-answering in general.

The automata-theoretic models we use, attempt to approximate the functionalities of the human players and Watson as much as possible. In particular:

- Human players are modeled as $<s(n), q(n), r(n), f(n)>$-arrays. The processing capability of the brain is modeled by an (infinite) series of finite-state automata of increasing complexity. The automata can query a fixed Turing machine by way of background intelligence. During a QA-session, states of the acting 'circuit' persist from question to question.
- Watson-type machines are modeled as $< b(n), l(n), f(n) >$-machines. A fixed Turing machine is used for basic information processing 'up front'. An (infinite) knowledge base provides background knowledge (or 'advice'). During a QA-session, knowledge is preserved and passed on through learning space.

The persistence of states and learning space, respectively, gives an interactive feature to the models which substantially extends the regime of traditional machine models with 'advice' only and no persistent memory ([3, 10]).

The results of Section 3 can be summarized as follows, where 'simulation' refers to the capability of realizing equal QA-sessions. We omit the natural assumptions on $f(n)$ that are implicit in these statements.

- Every cognitive $<s(n), q(n), r(n), f(n)>$-array with $r(n) \geq s(n) \log s(n)$ can be simulated by a $<s(n) \log s(n) + f(n), \log s(n), f(n)r(n)>$-machine.
- Every $< b(n), l(n), f(n) >$-machine can be simulated by a (one-way) cognitive $<b(n) + 2^{O(l(n))}, f(n), f(n), f(n) + 2^{O(l(n))}>$-array.

4.2 Extending Pippenger's Theorem

The models of cognitive arrays and question-answering machines build on circuit families and Turing machines with advice, respectively [1, 18]. The duality between these models is well-studied in non-uniform complexity theory. One of the key observations in this field is Pippenger's theorem which states the following: *a language*

is accepted by a family of circuits of polynomially bounded size if and only if the language is accepted by a polynomial-time bounded Turing machine with polynomially bounded advice. For a more detailed explanation and proof we refer to [1, 10, 15, 18].

The models we study here differ appreciably from plain combinatorial circuits and Turing machines with advice. CA's and QA-machines are based on machine models which extend their basic counterparts in substantial ways. Nevertheless, it is a natural question whether Pippenger's theorem might not in essence be an instance of a more general result, for the more general models we defined. Let $T(n) \geq n$.

Definition 6. *A $<s(n), q(n), r(n), f(n)>$-array is said to be $T(n)^{O(1)}$-bounded if all its parameters are. A $<b(n), l(n), f(n)>$-machine is said $T(n)^{O(1)}$-bounded if its parameters $b(n)$ and $f(n)$ are.*

An immediate consequence of the simulation results from Section 3 is the following result, generalizing Pippenger's theorem.

Theorem 4. *$T(n)^{O(1)}$-bounded QA-machines with $O(\log T(n))$-size learning space are equivalent to (one-way) $T(n)^{O(1)}$-bounded cognitive arrays.*

Proof. The result follows immediately when we choose $l(n) = \log T(n)$ and all other specified parameters $T(n)^{O(1)}$-bounded. □

Corollary 5. *Polynomially bounded QA machines with $O(\log n)$-size learning space are equivalent to (one-way) polynomially bounded cognitive arrays.*

Corollary 6. *$2^{O((g(n))}$-bounded QA-machines with $O(g(n))$-size learning space are equivalent to (one-way) $2^{O(g(n))}$-bounded cognitive arrays.*

These results show how the dual models of question-answering relate, if one insists on a bounding of the parameters as indicated.

4.3 Effectiveness

However, more conclusions may be drawn from the bounds in Section 3. A key connection is apparent between the size-bound $s(n)$ for the finite-state automata $C(n)$ in a cognitive array C, and the size-bounds $b(n)$ and $l(n)$ of a QA machine.

In the simulation of a QA-machine by a CA, the key observation is that $s(n) \approx b(n) + 2^{O(l(n)}$. The term $b(n)$ reflects the fact that the background knowledge of the QA machine must be encoded in the brain automaton of the CA, as expected. However, one also sees that $s(n)$ has a term that is exponential in the size of the learning space of the QA-machine. Examples like the one in Theorem 1 show that this bound is best possible in general, in the worst case. The exponential term reflects the fact that the brain automaton of the CA must have the circuitry in place for any possible contents of the learning space that may arise. (A more detailed argument would show that the term can be replaced by the different number of possible learning space contents that can actually arise in a QA-session of the QA-machine on inputs of length n.)

Thus, whereas the question-answering capabilities of the two models may be made equal by suitable parameter tunings, it cannot be avoided that the size bound of the automata in a simulating CA is exponential in the size of the learning space of the QA-machine, aside from the 'linear' term $b(n)$. This is the unavoidable consequence of having a model in which the brain of the CA must be hard-wired in advance for the knowledge that the given QA-machine has at its disposal during a session, either from its knowledge base or by accumulation in learning space. In

contrast, the simulation of a CA by a QA-machine takes a setting that remains polynomial in all parameters of the CA.

The observations give an indication of the difference in effectiveness between the human players and Watson, aside from the obvious differences in speed. The difference seems to be caused entirely by the advantages of a Watson-type computer in storing vast amounts of potentially useful information, possible exceeding any feasible bound on the information that can be stored in the human brain, either in advance or by learning during a session.

5 Conclusion

In this paper we gave a theoretical model for comparing the qualities of human players and machines at *question answering*. The motivating premise was the bold idea that the human players and the supercomputer Watson may be equally good at the game of `Jeopardy!` in principle, if their capabilities match. Part of the effort in this paper went into identifying suitable models and parameters, to make a comparison meaningful.

In this study we modeled human players as cognitive arrays and Watson-type computers by QA-machines, both enriched with a suitable form of background intelligence and a learning facility during question-answering sessions. Cognitive arrays model the 'circuits of the mind' of the human players, with the option to learn and gain additional information from a Turing machine. QA-machines model the query-answering abilities of Turing machines with pre-defined advice and a persistent learning space. A main outcome is the result that the models are 'equally powerful', provided their parameters are set appropriately.

As a case in point, we show that cognitive automata of polynomial size are equivalent to polynomial-time bounded QA-machine with polynomial advice and logarithmic learning space. This results extends Pippenger's theorem from non-uniform complexity theory, originally relating the computational power of circuits and of Turing machines with advice only. The extension may be of interest as a foundational result in the theory of question-answering machines.

The models are very powerful, if we allow them to be as general as defined. In particular, without any further constraints cognitive arrays and QA-machines can encode c.q. make use of arbitrary knowledge bases. This gives the models potentially 'super-Turing power' as computational devices, unless one insists on uniform definability of the infinite parts of the models. The present analysis has not made use of any such assumption.

A comparison of the models shows that the power of QA machines as we have modeled them can be captured by cognitive arrays, but at the price of having state sets that are large enough so as to encode what a QA-machine stores in its knowledge base and can accumulate in its learning space during a session. This suggests a possible advantage of QA-machines in having more effective means for storing and maintaining information.

The comparison also suggests that the model of cognitive arrays might have to be changed to deal with temporary knowledge in a different way than through fixed circuitry. This seems to lead straightforwardly to the notion of neural nets that have the possibility to modify their states, transitions and even topology [19]. For the time being, QA-machines have the edge.

References

1. J.L. Balcázar, J. Diaz, J. Gabarro, *Structural complexity I*, 2nd ed., Springer-Verlag, Berlin, 1995.

2. BBC News, IBM's Watson supercomputer crowned Jeopardy king, February 17, 2011, `http://www.bbc.co.uk/news/technology-12491688`.
3. C. Damm, M. Holzer, Automata that take advice, in: J. Wiedermann, P. Hájek (Eds), Mathematical Foundations of Computer Science 1995, Proc. 20th International Symposium (MFCS'95), Lecture Notes in Computer Science Vol 969, Springer-verlag, Berlin, 1995, pp. 149-158.
4. D. Ferrucci *et al.*, Building Watson: An overview of the DeepQA project, *AI Magazine* (Fall 2010), pp. 5979.
5. D. Goldin, P. Wegner, Persistence as a form of interaction, Tech. Rep. CS-98-07, Dept. of Computer Science, Brown University, Providence RI, 1998.
6. D.Q. Goldin, Persistent Turing machines as a model of interactive computation, in: K.-D. Schewe, B. Thalheim (Eds.), *Foundations of Information and Knowledge Systems*, Lecture Notes in Computer Science, Vol. 1762, Springer-Verlag, 2000, pp. 116-135.
7. D.Q. Goldin, S.A. Smolka, P.C. Attie, E.L. Sonderegger, Turing machines, transition systems, and interaction *Information and Computation* 194:2 (2004) 101-128.
8. H. Hempel, M. Kimmritz, Persistent computations of Turing machines, in: O.H. Ibarra, B. Ravikumar (Eds.), *Implementation and Application of Automata*, 13th Int. Conference (CIAA 2008), lecture Notes in Computer Science, Vol. 5148, Springer-Verlag, 2008, pp. 171-180.
9. J.E. Hopcroft, J.D. Ullman, *Formal Languages and their Relation to Automata*, Addison-Wesley Publ. Comp., Reading MA, 1969.
10. R.M. Karp, R.J. Lipton, Turing machines that take advice, *L'Enseignement Mathématique* 28 (1982) 191-209.
11. S. Kosub, Persistent computations, Techn. Rep. 217, Institut für Informatik, Julius-Maximilians-Universität Würzburg, Würzburg, 1998.
12. W.G. Lehnert, Paradigmatic issues in cognitive science, in: W. Kintsch, J.R. Miller, P.G. Polson (Eds.), *Methods and Tactics in Cognitive Science*, Psychology Press, New York, 1984, Ch. 2, pp. 21-49.
13. M.T. Maybury (Ed.), *New Directions in Question Answering*, AAAI Series, MIT Press, Cambridge MA, 2004.
14. R.J. Nelson, Machine models for cognitive science, *Philosophy of Science* 54 (1987) 391-408.
15. N. Pippenger, On simultaneous resource bounds, in: *20th Symp. Foundations of Computer Science*, Proceedings, IEEE Press, 1979, pp. 307-311.
16. Z.W. Pylyshyn, Computing in cognitive science, in: M.I.Ṗosner (ed.), *Foundations of Cognitive Science*, MIT Press, Cambridge, MA, 1989, pp. 49-91.
17. J. van Leeuwen, J. Wiedermann, Beyond the Turing limit: Evolving interactive systems, in: L. Pacholski, P. Ružička (Eds.), *SOFSEM 2001: Theory and Practice of Informatics*, Proc. 28th Conference, Lecture Notes in Computer Science, Vol. 2234, Springer-Verlag, Berlin, 2001, pp. 90-109.
18. H. Vollmer, *Introduction to Circuit Complexity - A Uniform Approach*, Springer-Verlag, Berlin, 1999.
19. J. Wiedermann, The computational limits to the cognitive power of the neuroidal tabula rasa, *J. Exp. Theor. Artif. Intell.* 15:3 (2003) 267-279.