

A Formalised Lower Bound on Undirected Graph Reachability

Ulrich Schöpp

Ludwig-Maximilians-Universität München
Oettingenstraße 67, D-80538 München, Germany

Abstract. We study the expressivity of Jumping Automata on Graphs (JAGs), an idealised model of computation with logarithmic space. JAGs operate on graphs by using finite control and a constant number of pebbles that are placed on graph nodes. In each step, a JAG can observe, for any two pebbles, whether they lie on the same graph node and it may use this information to update its finite state and move a pebble along an edge or jump it to the location of another one.

We revisit the proof of Cook & Rackoff that JAGs cannot decide s - t -reachability in undirected graphs. Cook & Rackoff prove this result by constructing, for any given JAG, a finite graph that cannot be traversed exhaustively by the JAG. In this paper we generalise this construction to a general class of group action graphs, by establishing a bound on the number of nodes that a JAG can visit on these graphs. This generalisation allows us to strengthen the result of Cook & Rackoff to the existence of a graph of small degree whose diameter (rather than its number of nodes) is larger than the number of nodes the JAG can visit.

The main result has been formalised in the theorem prover Coq, using Gonthier’s tactic language SSREFLECT.

1 Introduction

Pointer programs are a useful abstraction for algorithms that run in logarithmic space. Many LOGSPACE-algorithms can be viewed as taking some structured input, e.g. a graph, on which they operate by means of a constant number of pointers, e.g. to the nodes of the graph. Describing LOGSPACE-algorithms as pointer programs is not only usually easier than working directly with Turing Machines, it is also useful for studying the nature of computation with logarithmic space. With current techniques only very few results can be proven about what cannot be done in logarithmic space. Computation models of idealised pointer algorithms are more accessible and may serve as a starting point for obtaining insight into the general problem.

A classic example of a class of pointer algorithms introduced for this purpose are the *Jumping Automata on Graphs* (JAGs) of Cook & Rackoff [1]. JAGs operate on graphs with a local ordering, which means that the outgoing edges of each node are numbered consecutively starting from 1. A JAG is an automaton with finite control that can place a constant number of pebbles on the input graph. In each step the automaton may observe the incidence relation of the pebbles, i.e. it can tell which of its pebbles happen to lie on the same graph node. Using this information it may then change its state and move a pebble of its choice along a graph edge (identified by a number) or jump it to the location of another one.

With these operations, JAGs capture a minimal core of what one would expect pointer programs to be able to do. Although JAGs do not capture all reasonable pointer programs, a good knowledge of their properties may be helpful in analysing the expressivity of extensions that capture larger classes of pointer programs. For example, JAGs lack the ability to decide if all nodes of the input graph have some property, since they cannot reach an isolated component of the input graph if the start configuration does not place a pebble on it. One way of removing this unnatural limitation is to study extensions of JAGs with universal iteration, where it is possible for the machine to iterate with one pebble over all graph nodes. Deterministic Transitive Closure (DTC) logic on locally ordered graphs features first-order quantifiers and may be viewed as an extension of JAGs of this kind. Since the computation of an extended JAG with universal iteration amounts to a sequence of ordinary JAG-computations interrupted by iteration-jumps, we believe that a good knowledge of ordinary JAG-computations should be helpful in analysing the expressivity of the extended JAG-model.

One interesting problem to consider when studying the expressivity of idealised computation models for deterministic LOGSPACE is that of s - t -reachability in undirected graphs. Reingold has recently shown that this problem can be solved in logarithmic space [10]. His algorithm may be expressed as a pointer program that has access to a constant number of counting registers of logarithmic size [9]. However, many models of pointer programs that have been studied before do not feature such counting registers and are not able to encode them by pointer arithmetic. Examples of such models include the above-mentioned Jumping Automata on Graphs and Deterministic Transitive Closure logic. Indeed, it has been proven by Cook & Rackoff [1] that JAGs cannot decide undirected reachability. For DTC-logic it is not known if undirected reachability for locally ordered graphs can be expressed, but Etessami & Immerman [2] have shown that the result of Cook & Rackoff can be used to obtain a partial negative result. This evidence raises the interesting question if counting registers are necessary for pointer programs to decide s - t -reachability in undirected graphs.

The study of this question has led us to revisit the proof of Cook & Rackoff that JAGs cannot decide reachability in undirected graphs. Cook & Rackoff obtain this result by constructing, for any JAG J , a graph that has more nodes than J can visit. In this paper we show that this proof can be generalised to yield the stronger result that there exists a graph whose *diameter* is larger than the number of nodes that J can visit. We formulate this result in a general way, so that it can be used for other graphs, without having to adapt the proof again. We expect the strengthening of Cook & Rackoff's result to be useful for analysing the expressivity of a JAG-model with universal iteration.

We formalise the main results of this paper in the theorem prover Coq, thus giving the highest confidence in the correctness of Cook & Rackoff's result, the work that builds on it, e.g. [2], and the generalisation we make in this paper. We show that with the existing theorem proving technology it is possible to formalise non-trivial results from complexity theory with surprisingly low formalisation overhead. The result we have formalised, like many results from complexity theory, is concerned with computation on finite structures. We have found the method of *small scale reflection*, as proposed by Gonthier [5], to be a very good approach for working efficiently with finite structures. We discuss these matters after presenting the proof that has been formalised.

2 Outline

We start with an informal outline of Cook & Rackoff's proof leading up to the generalisations in this paper.

Cook & Rackoff study the behaviour of JAGs on the d -dimensional torus of side-length m . The nodes of this graph are d -tuples of numbers from 0 to $m - 1$. To define the edges, define an addition $+$ on nodes by pointwise addition modulo m and for each $i \in \{1, \dots, d\}$ define g_i to be the vector having the number 1 in the i -th component and 0 in all other components. Then, for each i , there is an undirected edge between x and $x + g_i$. The examples of the one-dimensional and the two-dimensional torus of side-length three are depicted below.



The reason for studying these graphs is their cyclicity in all directions. If the pebble-moves of a JAG become periodic after a certain amount of time then, by the cyclic nature of the graph, there will be a repeat of pebble configurations not long after. The aim is now to show that, given a JAG J , if we choose m and d are large enough, then such a repeat of configurations will appear before J has had the time to visit all nodes of the graph. The key point for doing this is to derive a good upper bound on the time when the moves of J become periodic.

A bound on the time when the pebble moves of a JAG J become periodic can be obtained by showing that J can only behave in a limited number of ways. The behaviour of J from a certain starting configuration is the sequence of moves it makes together with the sequence of states it assumes. If any computation is longer than the number of possible behaviours then two configurations from which J behaves the same must appear in the sequence, so that the pebble moves between those configurations will be repeated periodically.

To analyse in how many different ways a JAG may behave, it is useful to introduce a notion of extended state. Since we must analyse the behaviour of any JAG, we do not know, in general, what information the JAG stores in its state. To be able to say something about all machines, it is useful to study what information a machine could maximally obtain about the graph and the position of the pebbles. To see what a JAG could learn about the torus, notice first that the neighbourhoods of all nodes in this graph are isomorphic. Hence, the only thing a JAG J can learn about this graph is the relative position of pebbles. In a start configuration J knows only which pebbles lie on the same node, but otherwise has no information about their relative positions. During the course of the computation, the JAG could now (if it had enough states) keep track of the relative distance of any two pebbles that did coincide on the same node at some point earlier in the computation. For any two pebbles the relative distance can be described as a d -vector z , such that if x and y are the positions of the pebbles then $x = y + z$ holds. In each step of the JAG, the known pebble distances can be updated according to the move, for example by adding g_i in the case of a pebble move along edge i .

We call the *extended state* the state of the JAG together with the maximal information of relative pebble distances recorded as just outlined. An important property of extended states is that many pebble-collisions can be predicted from the extended state. In particular, if the distance of two pebbles is known in an extended state, then one can tell by looking at the extended state whether or not they will collide in the next computation step. The pebbles will collide if and only if their distance-vector becomes the zero vector. However, it may still happen that two pebbles with unknown distance collide unexpectedly in an edge move. In this case their distance becomes known. Cook & Rackoff call this event a *coalition*, since if we consider the equivalence relation on pebbles of “having known distance,” two equivalence relations are united at this point. If $|P|$ denotes the number of pebbles then in each computation sequence there can appear only $|P|$ coalitions, as an equivalence relation on $|P|$ elements can have at most $|P|$ equivalence classes. Notice furthermore that in steps with a coalition, the extended state after the step is determined by the previous extended state together with the information which pebbles collided unexpectedly. In all other steps, the new extended state after the step is completely determined by the extended state before the step.

The bounds on the number of possible different behaviours of a JAG can now be established by induction on the number of coalitions. In computations without any coalition, the behaviour of the JAG is uniquely determined by the extended state at the start of the computation. A bound can then be given easily. It depends only on the number of states and pebbles of the JAG. For the induction step, consider the behaviour of a JAG in computation sequences with up to $k + 1$ coalitions. By induction hypothesis, we know how many behaviours there are in computations with at most k coalitions. Now it is not hard to show that a JAG behaves the same from two start configurations in computations with $(k + 1)$ coalitions if in both computations it behaves the same before the $(k + 1)$ -th coalition, if the $(k + 1)$ -th coalition appears at the same time in both computations, and if the same pebbles collide in these coalitions. Because of this observation and the induction hypothesis, we can give a bound on the number of possible behaviours of the JAG, if we can find an upper bound on the time when a $(k + 1)$ -th coalition occurs. This can be done as follows. A $(k + 1)$ -th coalition can only appear if the computation has not gone into a loop before reaching it. Using the cyclicity of the torus and the induction hypothesis, we can get a bound on the number of different configurations that may appear in any computation sequence with up to k coalitions. This bound may be used to find an upper bound on the length of computation sequences that have neither reached a loop nor a $(k + 1)$ -th coalition. This then allows one to give an upper bound on the time of a $(k + 1)$ -th coalition and therefore can be used to give a bound on the number of possible behaviours of the JAG in computations with at most $(k + 1)$ coalitions.

The result one obtains from this estimation is that there exists a constant c such that any JAG with $|P|$ pebbles and $|Q|$ states can visit at most $(|Q|m)^{c^{|P|}}$ nodes in the d -dimensional torus of side-length m . If we choose $d > 2c^{|P|}$ and $m = |Q|$ then any such machine can visit at most $(|Q|m)^{c^{|P|}} = m^{2c^{|P|}} < m^d$ nodes. Since the torus has m^d nodes, the machine can therefore not traverse it exhaustively.

Examination of this proof shows that only a few properties of the d -dimensional torus of side-length m are used in it: (i) the neighbourhoods of all nodes are isomorphic; (ii) by recording the distance of two pebbles, one can predict when two pebbles with

known distance will collide; and (iii) the graph has exponent m , meaning that if we repeat the same edge moves m times then we return to the place we started from. Based on this observation, we generalise the proof of Cook & Rackoff to a class of *action graphs* that have these three properties. We show that this generalisation allows us to strengthen the result of Cook & Rackoff to the existence of a graph in which the JAG can visit less nodes than the diameter of the graph, rather than its number of nodes.

3 Action Graphs

The requirement of properties (i)–(iii) leads us to using action graphs, which we define in this section. We also give example graphs that are relevant for the results in this paper.

Action graphs are given by a set with a group action. We write groups multiplicatively and write e_D or just e for the unit element of group D . The *exponent* of a group (D, \cdot, e) is the smallest number m such that $x^m = e$ holds for all $x \in D$. An *action* \odot of a group (D, \cdot, e) on a set V is a function $\odot: V \times D \rightarrow V$ satisfying $v \odot e = v$ and $v \odot (x \cdot y) = (v \odot x) \odot y$ for all $v \in V$ and $x, y \in D$. A group action \odot is *free* if, for all $v \in V$ and $x \in D$, $v \odot x = v$ implies $x = e_D$.

Definition 1. *An action graph is a tuple $(V, D, g_1, \dots, g_d, \odot)$ consisting of a finite set of nodes V , a finite group (D, \cdot, e) with elements g_1, \dots, g_d , and a group action \odot of D on V . This action graph is a free action graph if the group action is free. We say that this graph has exponent m if the group D has exponent m .*

Each action graph $(V, D, g_1, \dots, g_d, \odot)$ defines a locally ordered undirected graph with node set V , in which there are undirected edges between v and $v \odot g_i$ for each $v \in V$ and each $i \in \{1, \dots, d\}$. In this graph any two nodes look the same, i.e. have the same isomorphism type. This addresses requirement (i) above. To address point (ii), we use free action graphs. In such graphs we can use group elements to keep track of pebble distances: a group element x represents the distance of pebble locations v and w , if $w = v \odot x$ holds. We can then predict pebble collisions from known distances, since if x represents the distance of v and w then we have $v = w$ if and only if $x = e$ holds. Finally, for (iii) we will consider action graphs of (small) exponent m .

Although we distinguish the node set V from the edge group D for conceptual reasons, an important example of action graphs is obtained by letting $V = D$ and $v \odot x = v \cdot x$. In this case, the resulting action graph is the Cayley graph of D with respect to $\{g_1, \dots, g_d\}$. The d -dimensional torus of side-length m used by Cook & Rackoff is an example of this case.

Example 2. For all $m, d > 0$, the d -dimensional torus of side-length m is a free action graph $G_{m,d} = (V_{m,d}, D_{m,d}, g_1, \dots, g_d, \odot)$. The group $D_{m,d}$ is $(\mathbb{Z}/m\mathbb{Z})^d$, the commutative group of d -tuples of the cyclic group of order m with pointwise addition. We write the group $D_{m,d}$ additively. The set of nodes $V_{m,d}$ is the underlying set of this group and the action \odot is the group addition itself, i.e. $v \odot d = v + d$. The vectors g_1, \dots, g_d are the unit-vectors, where g_i is a tuple with a 1 in the i -th position and a 0 elsewhere.

Seen as an undirected graph, $G_{m,d}$ has d -tuples of numbers in $\{0, \dots, m-1\}$ as nodes and from each node x there is an edge to $x \pm g_i$ for each i . This graph has degree $2d$ and exponent m . It has m^d nodes and its diameter is at most md . \square

Having defined action graphs, we can now state the main result we aim to show in this paper in the following proposition. This proposition gives a bound on the number of nodes a JAG can visit in a free action graph.

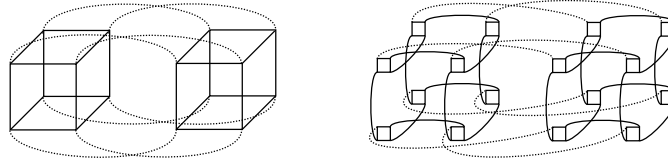
Proposition 3. *There exists a constant c , such that, for any free action graph G with exponent m and any JAG J with $|Q|$ states and $|P|$ pebbles, the number of nodes that J can visit from any start configuration on G is at most $(m|Q|)^{c|P|}$.*

We give a proof of this proposition in the next section. In the rest of the present section, we discuss how this result generalises that of Cook & Rackoff by instantiating it with different free action graphs.

With the family of graphs $G_{m,d}$, the proposition is just the result of Cook & Rackoff. If $d > 2c^{|P|}$ and $m = |Q|$ then no JAG J with $|P|$ pebbles and $|Q|$ states can visit all nodes in $G_{m,d}$. We note that the degree d depends only on the number of pebbles and not on the number of states. This is important, for example for proving the corollary that no JAG can decide reachability of graphs of degree three [1].

One motivation for generalising the result of Cook & Rackoff was to construct graphs of small degree, in which the number of nodes a JAG can visit is not only smaller than the number of nodes in the graph, but also than the length of the shortest path between two nodes. The degree of these graphs should again depend only on the number of pebbles. This rules out the graphs defined by Cook & Rackoff. A family of graphs $H_{m,d}$ with the desired property can nevertheless be constructed as the Cayley graph of the wreath product of two groups.

The undirected graph $H_{m,d}$ can be understood as arising from G_{m,m^d} by replacing each node with a copy of $G_{m,d}$. This is illustrated for $m = d = 2$ in the figure depicting $G_{2,2^2}$ and $H_{2,2}$ below. With this definition, $H_{m,d}$ has a large diameter like G_{m,m^d} , but unlike G_{m,m^d} the degree of $H_{m,d}$ exceeds that of $G_{m,d}$ only by two.



Formally, $H_{m,d}$ has node set $(\mathbb{Z}/m\mathbb{Z})^{V_{m,d}} \times V_{m,d}$, where $V_{m,d}$ is the node set of $G_{m,d}$. There are edges between $\langle f, x \rangle$ and $\langle f, x + g_i \rangle$ for all $i \in \{1, \dots, d\}$ and also edges between $\langle f, x \rangle$ and $\langle (\lambda i. \text{if } i = x \text{ then } f(i) + 1 \text{ else } f(i)), x \rangle$. Notice how in the last edge, the second component of the pair acts as an address for the increment.

The graph $H_{m,d}$ may be described as the Cayley graph of a *wreath product* of two groups. We recall (a special case of) this concept here. Let A and B be groups. The wreath product $A \wr B$ is a group with carrier set $B^A \times A$, where B^A denotes the set of all functions from the carrier set of A to that of B . The multiplication is defined by

$$\langle f, x \rangle \cdot \langle g, y \rangle = \langle \lambda i. f(i) \cdot g(x^{-1} \cdot i), x \cdot y \rangle.$$

If g_1^A, \dots, g_n^A and g_1^B, \dots, g_m^B are generators of the groups A and B respectively, then

$$\{ \langle \lambda i. e_B, g_i^A \rangle \mid 1 \leq i \leq n \} \cup \{ \langle \lambda i. \text{if } i = e_A \text{ then } g_j^B \text{ else } e_B, e_A \rangle \mid 1 \leq j \leq m \}$$

is a set of generators for $A \wr B$. Furthermore, if A has exponent m_A , B has exponent m_B and B is commutative then $A \wr B$ has exponent $m_A \cdot m_B$.

Example 4. For all positive natural numbers m and d define a free action graph $H_{m,d}$ with group $D_{m,d} \wr (\mathbb{Z}/m\mathbb{Z})$. The node set of $H_{m,d}$ is the underlying set of the group and the action is given by the group multiplication itself. For the elements g_1, \dots, g_{d+1} that define the edges of $H_{m,d}$, we take generators of $D_{m,d} \wr (\mathbb{Z}/m\mathbb{Z})$. These generators can be defined from those of $D_{m,d}$ and $\mathbb{Z}/m\mathbb{Z}$ as described above.

The choice of g_1, \dots, g_{d+1} as generators makes the graph $H_{m,d}$ connected. Its exponent is m^2 . The diameter of $H_{m,d}$ is at least m^d , since for all nodes x and all $i \in \{1, \dots, d+1\}$ the nodes x and $x \odot g_i$ are vectors that differ in at most one component, so that a path from $\langle \lambda i, 0, 0 \rangle$ to $\langle \lambda i, 1, 1 \rangle$ must have length at least m^d . \square

If we apply Prop. 3 to $H_{m,d}$, then we obtain that any JAG J with $|P|$ pebbles and $|Q|$ states can visit only $(m^2|Q|)^{c^{|P|}}$ nodes in it. By choosing $d > 3c^{|P|}$ and $m = |Q|$, we know that J can visit $(m^2|Q|)^{c^{|P|}} = m^{3c^{|P|}} < m^d$ nodes in $H_{m,d}$. Since the diameter of $H_{m,d}$ is at least m^d , Prop. 3 thus shows that J must visit fewer nodes than the diameter of the graph.

We note that one can further widen the gap between the nodes that J can visit and the diameter of the graph by iterating the construction of $H_{m,d}$ from $G_{m,d}$, that is by studying the Cayley graph for the wreath product $((D_{m,d} \wr (\mathbb{Z}/m\mathbb{Z})) \wr (\mathbb{Z}/m\mathbb{Z}))$ etc.

4 Reachability

In this section we give a proof of Prop. 3, which we have formalised in Coq. We start by giving a proper definition of Jumping Automata on Graphs.

Definition 5 (Jumping Automaton on Graphs). A jumping automaton on graphs is a triple (Q, P, δ) consisting of a finite set Q of states, a finite set P of pebbles and a function $\delta: Q \times \Sigma_P \rightarrow Q \times M_P$, where the set Σ_P of observations is the set of equivalence relations on P and $M_P \stackrel{\text{def}}{=} (P \times P) \cup (P \times \mathbb{N})$ is the set of moves.

Thus a JAG is a finite state machine that in each step obtains an input of type Σ_P and makes an output of type M_P . The intention is that an equivalence relation in Σ_P contains the information which two pebbles lie on the same graph node. The intention for the moves is such that a move $\langle x, y \rangle \in P \times P$ represents the jump of pebble y to x and a move $\langle x, i \rangle \in P \times \mathbb{N}$ represents the move of pebble x along edge number i . We use the suggestive notation $[x:=y]$ and $[x:=succ_i(x)]$ for the two kinds of moves.

We have formulated JAGs so that their inputs and outputs are given abstractly by the sets of observations Σ_P and moves M_P . This will be useful in Sections 4.1 and 4.2 for analysing the behaviour of JAGs in the presence of abstracted knowledge about the input graph.

Concretely, a JAG can operate on a (Σ_P, M_P) -state space.

Definition 6 (State Space). A (Σ, M) -state space $(S, [-], \cdot)$ consists of a state set S , an observation function $[-]: S \rightarrow \Sigma$ and a move function $(-) \cdot (-): S \times M \rightarrow S$.

We introduce the notion of state space not with the aim of generalising JAGs, but in order to make it clear which properties depend on the concrete construction of the state space. In the end, we are only interested in state spaces generated by graphs.

A locally ordered graph G with vertex set V gives rise to a (Σ_P, M_P) -state space $S_P(G)$ whose state set consists of all functions $\rho \in P \rightarrow V$ that place the pebbles on the graph nodes. The observation function $[-]$ returns the incidence relation, that is $\langle x, y \rangle \in [\rho] \iff \rho(x) = \rho(y)$, so that a JAG can see whether or not any two pebbles lie on the same graph node. The move function $(-) \cdot (-)$ formalises pebble jumps and edge moves in the evident way.

A *configuration* of a JAG $J = (Q, P, \delta)$ on a state space $(S, [-], \cdot)$ is a pair $\langle q, \rho \rangle \in Q \times S$ of a machine state $q \in Q$ and an external state $\rho \in S$. Write $Conf_{J,S}$ for the set of configurations. The transition relation $\longrightarrow_{J,S}$ on configurations is the least relation such that $\delta(q, [\rho]) = \langle q', m \rangle$ implies $\langle q, \rho \rangle \longrightarrow_{J,S} \langle q', \rho \cdot m \rangle$. Write $\longrightarrow_{J,S}^*$ for the reflexive transitive closure of $\longrightarrow_{J,S}$. Since $\longrightarrow_{J,S}$ is a deterministic relation, we have, for any configuration C , a unique configuration that is reached from C in k steps. We write $C(k)$ for it.

4.1 Abstraction

We now work towards the proof of Prop. 3, which in its essence is that of Cook & Rackoff [1]. In this section, we define a framework for stating abstractly the properties that are needed in this proof.

As outlined in Sect. 2, the proof of Prop. 3 goes by giving a bound on the time when the moves of a JAG become periodic and showing that on a free action graph such a periodicity must soon lead to a configuration of the JAG being repeated.

To give a bound on when the moves of a JAG become periodic, we formalise what a JAG could possibly learn about the positions of its pebbles on a free action graph. Since we shall see that the maximum knowledge a JAG can obtain is quite small, this will allow us to find an upper bound on the time when the behaviour of a JAG starts to be repeated. We capture the knowledge a JAG can obtain with the notion of an abstraction.

Definition 7. An abstraction of a (Σ, M) -space $(S, [-], \cdot)$ is a set X together with

- functions $init: S \rightarrow X$, $\omega_0: X \times M \rightarrow X$ and $\omega_1: X \times M \times \Sigma \rightarrow X$; and
- relations $\Vdash \subseteq X \times S$ and $predictable \subseteq X \times M \times \Sigma$,

such that for all $\rho, \rho' \in S$, $\xi \in X$ and $m \in M$ the following properties all hold:

1. $init(\rho) \Vdash \rho$.
2. $\xi \Vdash \rho$ implies $\omega_1(\xi, m, [\rho \cdot m]) \Vdash \rho \cdot m$.
3. $\xi \Vdash \rho$ and $\langle \xi, m, [\rho \cdot m] \rangle \in predictable$ implies $\omega_1(\xi, m, [\rho \cdot m]) = \omega_0(\xi, m)$.
4. $\xi \Vdash \rho$ and $\xi \Vdash \rho'$ implies $[\rho] = [\rho']$.

We use two step functions ω_0 and ω_1 in order to distinguish between predictable and non-predictable steps. In a predictable step, the successor abstract state depends only on the previous abstract state and the move, while a non-predictable step may further depend on the observation in the state-space after the move. This reflects the distinction

in Cook & Rackoff's proof between steps without a coalition and those with a coalition. When no coalition occurs then one can determine the known distances after the step from those before the step, while in the event of a coalition, one additionally needs to know which pebbles collided unexpectedly.

Extremal examples of abstractions can be obtained by letting $X = \Sigma$ and $X = S$, where in the former case no step is predictable, while in the latter all are.

For a JAG $J = (Q, P, \delta)$ and an abstraction X for state-space S , we define *extended configurations* to be configurations together with an abstract value of the state. The set of extended configurations is given by $Conf_{J,S}(X) = \{\langle q, \rho, \xi \rangle \in Q \times S \times X \mid \xi \Vdash \rho\}$.

The transition relation of J can now be partitioned into predictable and non-predictable steps, as given by the two rules below.

$$\text{(PREDICTABLE)} \frac{\delta(q, [\rho]) = \langle q', m \rangle \quad \langle \xi, m, [\rho \cdot m] \rangle \in \text{predictable}}{\langle q, \rho, \xi \rangle \xrightarrow{J,S} \langle q', \rho \cdot m, \omega_0(\xi, m) \rangle}$$

$$\text{(NON-PREDICTABLE)} \frac{\delta(q, [\rho]) = \langle q', m \rangle \quad \langle \xi, m, [\rho \cdot m] \rangle \notin \text{predictable}}{\langle q, \rho, \xi \rangle \xrightarrow{J,S} \langle q', \rho \cdot m, \omega_1(\xi, m, [\rho \cdot m]) \rangle}$$

Clearly, both $E \xrightarrow{=} F$ and $E \xrightarrow{>} F$ imply $\pi_C E \longrightarrow \pi_C F$, where we write $\pi_C E$ for the projection from an extended configuration to the ordinary configuration it contains. We also write $\pi_Q E \in Q$, $\pi_S E \in S$ and $\pi_X E \in X$ for the evident projections.

Definition 8. Define equivalence relations on extended configurations by

$$\begin{aligned} E \simeq_{QX} F &\stackrel{\text{def}}{\iff} \pi_Q E = \pi_Q F \wedge \pi_X E = \pi_X F, \\ E \simeq_{Q\Sigma} F &\stackrel{\text{def}}{\iff} \pi_Q E = \pi_Q F \wedge [\pi_S E] = [\pi_S F]. \end{aligned}$$

Note that we have $\simeq_{Q\Sigma} \subseteq \simeq_{QX}$ by item 4 in Def. 7. Furthermore, if $E \simeq_{Q\Sigma} F$ holds then the machine J will make the same moves from these two configurations, i.e. $\delta(\pi_Q E, [\pi_S E]) = \delta(\pi_Q F, [\pi_S F])$ holds.

Lemma 9 (Determinacy of abstract values).

1. If $E \simeq_{QX} F$, $E \xrightarrow{=} E'$ and $F \xrightarrow{=} F'$ hold, then so does $E' \simeq_{QX} F'$.
2. If $E \simeq_{QX} F$, $E \xrightarrow{>} E'$, $F \xrightarrow{>} F'$ and $[\pi_S E'] = [\pi_S F']$, then also $E' \simeq_{QX} F'$.

4.2 Reachability

In this section we prove Prop. 3. Fix a free action graph $G = (V, D, g_1, \dots, g_n, \odot)$ with exponent m and a JAG $J = (Q, P, \delta)$. We omit subscripts J, P and G where possible.

First we define the instance of an abstraction that formalises the 'maximum possible knowledge' about a free action graph outlined above. The following lemma contains all the properties that we need of it.

Lemma 10. *There exists an abstraction X of the state-space $S(G)$ such that:*

1. *The function $init: S(G) \rightarrow X$ can be written as the composition of two functions $init_1: S(G) \rightarrow \Sigma$ and $init_2: \Sigma \rightarrow X$.*
2. *There exists a function $[-]: X \rightarrow Eq(P)$, such that if $\xi \Vdash \rho$, $\xi' \Vdash \rho$ and $[\xi] = [\xi']$ all hold then so does $\xi = \xi'$.*
3. *There exists a measure function $\mu: Conf(X) \rightarrow \{1, \dots, |P|\}$ satisfying the following three properties for all $E, F \in Conf(X)$, $C \in Conf$, $k \in \mathbb{N}$ and $\xi \in X$.*
 - (a) *$E \xrightarrow{=}_{S(G)} F$ implies $\mu(F) = \mu(E)$.*
 - (b) *$E \xrightarrow{>}_{S(G)} F$ implies $\mu(F) = \mu(E) - 1$.*
 - (c) *$\mu(\langle C, init(\pi_S C) \rangle(k)) \geq \mu(\langle C, \xi \rangle(k))$.*

Proof. The abstraction X is the set of partial functions $P \times P \rightarrow D$ whose domain is an equivalence relation. The functions describe partial knowledge about the relative distance of any two pebbles. This is formalised by the relation \Vdash :

$$\xi \Vdash \rho \stackrel{\text{def}}{\iff} \forall x, y \in P. (\xi(x, y) \neq \perp \implies \rho(y) = \rho(x) \odot \xi(x, y)) \\ \wedge (\rho(x) = \rho(y) \in V \implies \xi(x, y) = e_D)$$

The functions ω_0 and ω_1 return the updated knowledge of the relative distances after a move. The two functions ω_0 and ω_1 differ only in their handling of edge moves $[x:=succ_i(x)]$. In an edge move, it is possible that two pebbles collide whose relative distance is not yet known. In the function ω_0 it is assumed that such a collision does not happen, while ω_1 handles this possibility. Note that in order to tell whether a collision has occurred, it is necessary to observe the state-space after the move, so that collisions can only be accounted for in ω_1 .

$$\omega_0(\xi, [x:=y])(u, v) \stackrel{\text{def}}{=} \begin{cases} e_D & \text{if } u = v = x \\ \xi(y, v) & \text{if } u = x \\ \xi(u, y) & \text{if } v = x \\ \xi(u, v) & \text{otherwise} \end{cases} \\ \omega_1(\xi, [x:=succ_i(x)])(u, v) \stackrel{\text{def}}{=} \begin{cases} e_D & \text{if } u = v = x \\ g_i^{-1} \cdot \xi(x, v) & \text{if } u = x \\ \xi(u, x) \cdot g_i & \text{if } v = x \\ \xi(u, v) & \text{otherwise} \end{cases}$$

Here, we follow the convention that any expression containing $\xi(u, v)$ is undefined if $\xi(u, v)$ is. We omit the precise definition of ω_1 , which extends that of ω_0 by a special case to update the known relative distances in case of an unpredicted collision.

The function $init$ is defined such that $init(\rho)(x, y) = e_D$ holds when $x[\rho]y$ does and $init(\rho)(x, y)$ is undefined for all other $x, y \in P$. For $\mu(q, \rho, \xi)$ we take the number of equivalence classes in the domain of ξ . Finally, we define the set *predictable* to contain those triples $\langle \xi, m, \sigma \rangle$ for which the measure of ξ is the same as that of $\omega_1(\xi, m, \sigma)$.

The required properties then follow by a straightforward (but tedious and lengthy) calculation from the properties of free action graphs. \square

Having proved this lemma, it remains to observe that the proof of Cook & Rackoff can be carried out with just the abstract structure defined in the lemma.

Let $k \in \mathbb{N}$. In this section we first establish a bound on the number of configurations that appear in computations of up to $k + 1$ steps. Since k is arbitrary and the bound will not depend on k , this shall be enough to bound the number of configurations in computation sequences of arbitrary length.

The proof of the bound goes by induction on how often in a computation sequence the measure from Lemma 10 strictly decreases. The points of decrease are called *coalitions*, in reference to the fact that in the abstract values from the proof of Lemma 10 two equivalence classes are united at these points.

Definition 11 (Coalition). *Let $E \in \text{Conf}(X)$ and $n \in \mathbb{N}$. The n -th coalition is the smallest number $\text{coal}_n E \leq k+1$ satisfying $\mu(E(\text{coal}_n E)) \leq P-n$ or $\text{coal}_n E = k+1$.*

We aim to give an upper bound on the number A_n defined by

$$A_n \stackrel{\text{def}}{=} \max_{C \in \text{Conf}} |\{C(i) \mid i < \text{coal}_{n+1} \langle C, \text{init}(C) \rangle\}|.$$

To do so, we also need to give an upper bound on the number of different possible behaviours of J . Similarity of behaviour up to the n -th coalition is captured by the equivalence relation $\sim_n \subseteq \text{Conf}(X) \times \text{Conf}(X)$ defined by

$$E \sim_n F \stackrel{\text{def}}{\iff} \forall i \leq n. \text{coal}_i E = \text{coal}_i F \wedge E(\text{coal}_i E) \simeq_{QX} F(\text{coal}_i F).$$

If E and F are \sim_n -related then the JAG J makes the same moves in the computation sequences starting with E and F , up to any time before the $(n + 1)$ -th coalition in either sequence. This follows from the next lemma, which is an easy consequence of Lemma 9, together with the fact that J always makes the same moves $\simeq_{Q\Sigma}$ -related configurations.

Lemma 12. *If $E \sim_n F$ and $i < \text{coal}_{n+1} E$ and $i < \text{coal}_{n+1} F$ all hold then so does $E(i) \simeq_{QX} F(i)$ and $E(i) \simeq_{Q\Sigma} F(i)$.*

The following necessary condition for $E \sim_{n+1} F$ also follows using Lemma 9.

Lemma 13. *To show $E \sim_{n+1} F$ it suffices to show that $E \sim_n F$, $\text{coal}_{n+1} E = \text{coal}_{n+1} F$ and $[\pi_S E(\text{coal}_{n+1} E)] = [\pi_S F(\text{coal}_{n+1} F)]$ all hold.*

Define an equivalence relation \approx_n for similar behaviour on ordinary configurations:

$$C \approx_n C' \stackrel{\text{def}}{\iff} \langle C, \text{init}(\pi_S C) \rangle \sim_n \langle C', \text{init}(\pi_S C') \rangle.$$

Let R_n be the number of equivalence classes of \approx_n .

We can now bound the numbers R_n and A_n by induction on n . We start with the bound on R_n . The bound on A_n appears in Lemma 17 below.

Lemma 14. *For all $n \in \mathbb{N}$, the following two inequalities hold.*

$$R_0 \leq |Q| \cdot |P|^{|P|} \tag{1}$$

$$R_{n+1} \leq R_n \cdot (2 + A_n \cdot |P|^{|P|}) \cdot |P|^{|P|} \tag{2}$$

Proof. *Ad (1).* We have $\text{coal}_0(E) = 0$ for all E . We therefore know that $C \approx_0 C'$ is equivalent to $\pi_Q C = \pi_Q C'$ and $\text{init}(\pi_S C) = \text{init}(\pi_S C')$. The result follows since there are only $|P|^{|P|}$ possibilities for $\text{init}(\pi_S C)$, which follows from Lemma 10.1 and because there are at most $|P|^{|P|}$ equivalence classes on P .

Ad (2). We use Lemma 13 to decompose the equivalence classes of \approx_{n+1} . The asserted bound then follows because: (i) The equivalence relation \approx_n has at most R_n equivalence classes. (ii) There are $(2 + A_n \cdot |P|^{|P|})$ possibilities for $\text{coal}_{n+1}(E)$. It can be either $k + 1$ or must be below $1 + A_n \cdot |P|^{|P|}$, since after at most $1 + A_n \cdot |P|^{|P|}$ steps a configuration will be repeated, by definition of A_n and Lemma 10.2, and if a repeat appears then the $(n + 1)$ -th coalition must be $k + 1$. (iii) Finally, $[\pi_S E(\text{coal}_{n+1} E)]$ has at most $|P|^{|P|}$ possible values. \square

For any list of moves $\alpha \in M^*$, an action on pebble assignments $\rho \in S(G)$ can be defined by $\rho \cdot \varepsilon = \rho$ and $\rho \cdot (m\alpha) = (\rho \cdot m) \cdot \alpha$. With this notation we have the following lemma, whose proof is just like that of Lemma 4.6 of [1].

Lemma 15. *Let G be a free action graph with exponent m . Let $\alpha \in M^*$ be a sequence of moves. Then the sequence $(\rho_i \in S(G))_{i \geq 0}$ defined by $\rho_0 = \rho$ and $\rho_{i+1} = \rho_i \cdot \alpha$ has the property $\rho_{|P|} = \rho_{|P|+m \cdot |P|!}$.*

Lemma 16. *If $\pi_C E \approx_n \pi_C F$ and $i < \text{coal}_{n+1} E$ and $i < \text{coal}_{n+1} F$ all hold then so does $E(i) \simeq_{Q\Sigma} F(i)$.*

Proof. Using Lemma 10.3.(c) one gets $i < \text{coal}_{n+1} E \leq \text{coal}_{n+1}(\pi_C E, \text{init}(\pi_S E))$ and likewise for F . This implies $\langle \pi_C E, \text{init}(\pi_S E) \rangle(i) \simeq_{Q\Sigma} \langle \pi_C F, \text{init}(\pi_S F) \rangle(i)$ by Lemma 12. But the assertion follows from this, because it is easily seen that $\pi_C E(i) = \pi_C(\langle \pi_C E, \text{init}(\pi_S E) \rangle(i))$ and a similar equation for F hold and that $\pi_C E(i) = \pi_C F(i)$ implies $E(i) \simeq_{Q\Sigma} F(i)$.

Lemma 17. *For all n the inequality $A_n \leq (1 + |P| + m \cdot |P|!) \cdot R_n$ holds.*

Proof. Consider a computation $E \rightarrow E(1) \rightarrow \dots \rightarrow E(l)$ with $l < \text{coal}_n E$. Suppose, for a contradiction, there are more than $(1 + |P| + m \cdot |P|!) \cdot R_n$ configurations in $\{\pi_C E(0), \dots, \pi_C E(l)\}$. Then, by definition of R_n , there exist i and j with $1 \leq i < j \leq R_n$ and $\pi_C E(i) \approx_n \pi_C E(j)$. Using Lemma 16, this implies $E(r) \simeq_{Q\Sigma} E(r + (j - i))$ for all r with $i \leq r \leq l - (j - i)$. Hence, there is a sequence of moves α of length $(j - i) \leq R_n$ that is repeated from point $E(i)$ onwards. By Lemma 15, it follows that some configuration appears twice in the list $\pi_C E(i), \pi_C E(i + (j - i)), \dots, \pi_C E(i + (|P| + m \cdot |P|!) \cdot (j - i))$. The JAG must therefore go into a loop after at most $(1 + |P| + m \cdot |P|!) \cdot R_n$ steps, which implies the required contradiction.

Prop. 3 now follows easily from Lemmas 14 and 17:

Proof (of Prop. 3). Using Lemmas 14 and 17, it is straightforward to find a constant c , such that $A_{|P|} \leq (Qm)^{c|P|}$ holds. (In Coq we have $c = 2^{32}$ by a crude estimation.) The required assertion follows from this, since c does not depend on k and we have $\text{coal}_{|P|+1} E = k + 1$ by the properties of μ .

Up to this point, the results in this section have been fully formalised in Coq, where we have excluded trivial (and uninteresting) cases by assuming that $0 < |Q|$, $1 < m$ and $1 < |P|$ holds. We discuss the formalisation in the next section.

Corollary 18 (Cook & Rackoff 1980). *For each finite set P there is a d such that no JAG $J = (Q, P, \delta)$ decides undirected reachability on graphs of degree d .*

Proof. Construct a graph of two disjoint copies of $H_{m,d}$ and connect the two nodes given by vectors with $(m-1)$ in all components. Let s and t be the two nodes consisting of all zeros. If m and d are large enough then J , when started with each pebble on one of the two nodes s or t , cannot reach the edge connecting the two copies of $H_{m,d}$, since its distance from s and t in $H_{m,d}$ is larger than the number of nodes that J can visit. Hence, J cannot distinguish this graph from the one with this edge removed.

Corollary 19 (Cook & Rackoff 1980). *There is no JAG that decides reachability of graphs of degree 3.*

This corollary can be proven by a standard degree reduction technique [1].

5 Formalisation in Coq

One of the main contributions of this paper is the formalisation of the results in the previous section, up to and including the proof of Prop. 3 (as remarked above, we elude trivial cases in the formalisation by assuming $0 < |Q|$, $1 < m$ and $1 < |P|$). The formalisation also comprises the construction of the graphs $G_{m,d}$ and $H_{m,d}$ from Sect. 3.

The presentation in the previous section can be understood as an overview of the formalisation¹ in ordinary mathematical notation. Indeed, most of the lemmas in the previous section correspond directly to lemmas in the Coq development. To give a concrete example, Lemma 13 appears in the formalisation in the following ASCII notation.

```
Lemma decomp_sim: forall k n E1 E2,
  (sim k n E1 E2) -> (coal k n.+1 E1 == coal k n.+1 E2)
  -> observe (piS (coale k n.+1 E1))
    = observe (piS (coale k n.+1 E2))
  -> (sim k n.+1 E1 E2).
```

The proofs in the formalisation also follow closely the informal outline in the previous section. The main difficulty in making it possible for the formal development to remain close to the informal proofs has been to make define the basic types in an appropriate way. Notice that in the last section we have used classical logic, which appears to be at odds with Coq's intuitionistic meta-logic, and we have used counting arguments, such as that there are at most $|P|^{|P|}$ equivalence relations on a finite set P , which are not readily available in Coq. In the rest of this section we discuss the basic choices that have allowed us to directly formalise the proofs from the last section.

The formalisation consists of approximately 5000 lines of code. It is written in the tactic language SSREFLECT 1.1 developed by Gonthier, Mahboubi and Théry [7, 6], which has its origin in Gonthier's formalisation of the four-colour theorem [4]. We have chosen this tactic language for its conciseness and expressivity and because it contains an excellent library for working with finite sets.

¹ Available from: <http://www.tcs.ifi.lmu.de/~schoep/fomalcr.html>

5.1 Small-scale Reflection

Based on his experience from formalising the four colour theorem [4], Georges Gonthier has advocated a proof methodology that emphasises the use of computation at the propositional level [5]. One central idea is to express the truth of a predicate φ on some structured data type A that one wants to work with as a computational problem. This means that one writes a function $f: A \rightarrow \text{bool}$, where `bool` is the data type with elements `true` and `false`, such that $f(x)$ is `true` if and only if the logical proposition $\varphi(x)$ holds. The function f may implement a decision procedure, for example. Then, one proves in the logic that $\varphi(x)$ holds if and only if $f(x) = \text{true}$ does. As a result, if one wants to prove $\varphi(M)$ for some particular element M , one may replace this goal by $f(M) = \text{true}$ and have the proof system reduce the function f . If M is a closed value having the property φ then $f(M)$ will reduce to `true`, leaving the trivial goal `true = true`. Even if M is not a closed value, it is in many cases still possible to reduce $f(M)$ partially and so make progress in a proof. In this way, propositions are proved by evaluating functions.

The general idea then is to write predicates as functional programs and to prove their correctness. Since boolean-valued functions are very often used in place of predicates, there is a coercion so that one can just write $f(x)$ instead of $f(x) = \text{true}$. The tactic language `SSREFLECT` provides a convenient environment for carrying out proofs with such boolean-valued predicates in `Coq`. For instance, in our formalisation we have used a function `iforall: $\forall A: \text{finType}. (A \rightarrow \text{bool}) \rightarrow \text{bool}$` of universal quantification on finite sets. `SSREFLECT` provides the convenient concept of views that, after proving the equivalence of `iforall(f)` and $\forall x. f(x)$ once, allows one to treat occurrences of `iforall(f)` in essentially the same way as an ordinary quantification.

The approach of modelling predicates as `bool`-valued functions has been very important for the formalisation in this paper. In particular, since `bool`-valued functions must return either `true` or `false`, they validate the law of excluded middle, which we have used many times to formalise the classical arguments in this paper. Also, our construction of quotients on finite types, described below, relies on predicates being modelled as `bool`-valued functions.

`SSREFLECT` turned out to be very well-suited for working with finite sets and decidable predicates in a way that is similar to classical logic. It contains a library of finite sets, which are being modelled as types with decidable equality and an enumeration of their elements as a (necessarily finite) list. With this library it is possible to prove, for example, the following pigeonhole principle in a few lines and in a way that is very close to a standard informal classical proof.

Lemma pigeon : $\forall d_1 d_2: \text{finType}, f: d_1 \rightarrow d_2.$
 $(\text{card } d_2 < \text{card } d_1) \rightarrow (\exists x, y: d_1. (\text{negb } (x == y)) \ \&\& \ (f \ x == f \ y))$

Here, `==` denotes the decidable equality on finite sets, `negb` stands for negation and `&&` for conjunction.

This example illustrates that by using boolean-valued predicates, we can formalise classical arguments as conveniently as in a theorem prover with classical logic, such as, say, Isabelle/HOL, while at the same time being able to use `Coq`'s convenient dependently typed programming facilities.

5.2 Finite Functions

The formalisation of the proofs from Sect. 4 depends, among other things, on being able to count the number of configurations that appear in a computation sequence. Since configurations are modelled by functions from pebbles to graph nodes, we must therefore be able to count functions with finite domain and codomain.

To do this, we use the intensional representation of finite functions by their graphs proposed by Gonthier et al. [8]. However, we use functions with finite codomain as opposed to just a decidable codomain in [8], since we also need to prove cardinality properties, such as that the finite function space $X \rightarrow Y$ has cardinality $|Y|^{|X|}$.

An intensional representation of functions is furthermore useful to avoid problems with Coq's equality being intensional. Two finite functions $f, g: X \rightarrow Y$ are extensionally equal if and only if their intensional representations are equal in Coq's intensional equality. Since one can write in Coq functions to convert a finite function to its intensional representation and vice versa, one can therefore work with the intensional representation of functions instead of functions themselves. By using implicit coercions, one can treat intensionally represented functions almost like ordinary ones.

It is furthermore not hard to show that the functions between finite sets form themselves a finite set with decidable equality. As a result, finite functions may be used in the construction of finite sets, which is useful for defining finite wreath products, for example. Another example, where decidable equality $==$ on finite functions is useful, is the definition of the function `iforall`, which decides whether a given boolean-valued predicate holds for all elements of a finite set. It may be defined simply by

```
iforall(A: finType)(f: A → bool): bool :=  
  (fgraph_of_fun f) == (fgraph_of_fun (λx. true)),
```

where `fgraph_of_fun` converts a function to its intensional representation.

5.3 Equivalence Relations and Quotients

The formalisation of the proofs in Sect. 4 also requires one to be able to count the number of equivalence classes of the equivalence relation \approx_n and to show that there are at most $|P|^{|P|}$ equivalence relations on the finite set P .

We represent equivalence relations on a finite set A by functions of type $A \rightarrow A \rightarrow \text{bool}$ together with proofs of reflexivity, symmetry and transitivity. Using a boolean-valued function to represent relations allows us to give an intensional representation of equivalence relations, just like for finite functions, and to define quotient types.

To define an intensional representation of equivalence relations, we use a choice function for finite sets provided `SSREFLECT`. This choice function computes, for each finite set A and each predicate $f: A \rightarrow \text{bool}$, an element a of A that satisfies $f(a)$, if such an element exists. In particular, suppose R of type $A \rightarrow A \rightarrow \text{bool}$ is an equivalence relation. The equivalence class of a is given by $R(a): A \rightarrow \text{bool}$. Using the choice function, we can pick a canonical element from this equivalence class. In this way, we can represent the equivalence relation R by a finite function of type $A \rightarrow A$, which maps each element of A to a canonical representative of its equivalence class.

For the intensional representation of equivalence classes, we then simply take the choice functions $A \rightarrow A$ that arise from this construction. In particular, using the results on finite functions, we can define it as a finite data type with decidable equality. It follows immediately that there can be no more than $|A|^{|A|}$ equivalence relations on A .

With this groundwork, it is easy to construct quotient types. We define the quotient A/R simply as the image of the choice function $A \rightarrow A$ that represents R . This being a finite type, we can use it to count the number of equivalence classes, as required to formalise the proof of Cook & Rackoff. All in all, we can work with finite quotients just as we are used to from informal work.

The construction of the intensional representation of equivalence relations relies on relations being modelled as boolean-valued functions $A \rightarrow A \rightarrow \text{bool}$. The construction would not work with `Prop` instead of `bool`, since no choice function would be available then. In the formalisation, the relations from Def. 7 are therefore all modelled as boolean-valued functions.

6 Conclusion

In conclusion, once the right basic definitions had been found, the formalisation of the main results with `Coq` and `SSREFLECT` has been surprisingly smooth. We hope that this positive experience can be repeated in the formalisation of further results from complexity theory. We are aware only of little existing work in this direction, e.g. [3]. Nevertheless, since the correctness of results from complexity theory is fundamental for a number of practical applications, for example in cryptography, it should be very valuable to certify correctness with formal proofs. By formalising a non-trivial result from complexity theory we have given evidence that the current theorem prover technology is ready for this task.

I wish to thank M. Hofmann, L. Beringer and anonymous referees for their comments.

References

1. S.A. Cook and C. Rackoff. Space lower bounds for maze threadability on restricted machines. *SIAM Journal of Computing*, 9(3):636–652, 1980.
2. K. Etessami and N. Immerman. Reachability and the power of local ordering. *Theoretical Computer Science*, 148(2):261–279, 1995.
3. R. Gamboa and J.R. Cowles. A mechanical proof of the Cook-Levin theorem. In *TPHOLS*, pages 99–116, 2004.
4. G. Gonthier. A computer-checked proof of the four-colour theorem. Technical report available from <http://research.microsoft.com/~gonthier/4colproof.pdf>.
5. G. Gonthier. Notations of the four colour theorem proof. Technical report available from <http://research.microsoft.com/~gonthier/4colnotations.pdf>.
6. G. Gonthier and A. Mahboubi. A small scale reflection extension for the COQ system. INRIA Technical Report, December 2007.
7. G. Gonthier and A. Mahboubi and L. Théry. SSREFLECT extension for Coq, Version 1.1 <http://www.msr-inria.inria.fr/Projects/math-components>.
8. G. Gonthier, A. Mahboubi, L. Rideau, E. Tassi, and L. Théry. A modular formalisation of finite group theory. In *TPHOLS*, pages 86–101, 2007.
9. P. Lu, J. Zhang, C.K. Poon, and J. Cai. Simulating undirected *st*-connectivity algorithms on uniform JAGs and NNJAGs. In *ISAAC*, pages 767–776, 2005.
10. O. Reingold. Undirected *st*-connectivity in log-space. In *STOC*, pages 376–385, 2005.