Electronic Colloquium on Computational Complexity, Revision 1 of Report No. 90 (2008)

Pointer Programs and Undirected Reachability*

Ulrich Schöpp† Martin Hofmann‡

October 1, 2008

Abstract

We study pointer programs as a model of structured computation within LOGSPACE. Pointer programs capture the common description of LOGSPACE algorithms as programs that take as input some structured data (e.g. a graph) and that store in memory only a constant number of pointers to the input (e.g. to the graph nodes). Starting from pure pointer programs, in which only abstract pointers without any internal structure are allowed, we consider pointer programs with constructs for iterating over the input structure and for counting. We classify with which of these constructs it is possible to write a program for solving **s**-**t**-reachability in undirected graphs.

The main result of this paper is a new lower bound on undirected **s**-**t**-reachability. We show that while pointer programs with counting can decide this problem using Reingold's algorithm, the problem cannot be decided by pointer programs with iteration. As a corollary we obtain that Deterministic Transitive Closure logic on locally ordered graphs cannot express undirected **s**-**t**-reachability.

1 Introduction

Pointer programs are a useful abstraction for LOGSPACE. Most LOGSPACE-algorithms are intended to take as input some structured data and the logarithmic space that they have available is just enough to store a constant number of pointers into the input structure. A typical example of an input structure would be a graph; in logarithmic space one can store a constant number of references to graph nodes. This observation leads to the useful idealisation of LOGSPACE-algorithms as pointer programs that access a structured read-only input by means of abstract pointers and that compute their result by pointer manipulation alone. Many typical LOGSPACE-algorithms are presented informally in this way, see e.g. [1].

Pointer programs are useful not only for presenting LOGSPACE-algorithms, but also to gain insight into the nature of LOGSPACE-computation. In this paper we consider a natural class of pointer-programs that formalises the idealisation of typical LOGSPACE-algorithms as programs that 'use a constant number of pure pointers.' Although this class can express typical LOGSPACE-complete algorithms, such as a test for graph acyclicity, it is strictly contained within

*ECCC Report TR08-90 Revision 1. This revision fixes a few typographical errors.

†Ludwig-Maximilians-Universität München, Germany, schoepp@tcs.ifi.lmu.de

‡Ludwig-Maximilians-Universität München, Germany, mhofmann@tcs.ifi.lmu.de

ISSN 1433-8092

LOGSPACE [5]. It is therefore an interesting question, which typical LOGSPACE-algorithms do fall into this class, that is, which LOGSPACE-algorithms can be expressed by programs that 'use a constant number of pure pointers' only. In this paper we consider the problem of **s**-**t**-reachability in undirected graphs and show that it *cannot* be expressed by a program of this form.

Before we come to undirected reachability, let us discuss how the informal notion of pointer programs with a constant number of pure pointers may be captured precisely. Many informal pointer programs are *pure*, in that they treat pointers as abstract values without internal structure and that they manipulate them using only the operations of the input structure. In the case of graphs the pointers to nodes are typically manipulated by operations like 'move pointer x along the i-th edge from its current position'. In these operations the edges are often referred to by natural numbers. This is important for implementing algorithms like depth-first search, in which the neighbours of a node need to be visited in a particular order. We are therefore led to using *locally ordered graphs*, in which the outgoing edges of each node are numbered successively starting from 1. Moreover, virtually all pointer programs rely on being able to compare pointers for equality, i.e. to check if they point to the same graph node.

These considerations lead us to a simple model of *minimal pure pointer programs* for locally ordered graphs that forms a minimal core of what one should expect pure pointer programs in logarithmic space to be able to do. A minimal pure pointer program has finite control and a constant number of pointer variables that may be compared for equality and that may be manipulated with operations of the form: move pointer x along the i-th edge. This may be formalised in terms of a `while`-language with programs generated by the grammar

$$M \ ::= \ \texttt{skip} \mid M; M \mid x := t \mid \texttt{if } t \texttt{ then } M \texttt{ else } M \mid \texttt{while } t \texttt{ do } M \ ,$$

where in addition to pointer variables there are variables of boolean type to store the finite control state. The terms of boolean type consist of the usual boolean operations like $\wedge$ and $\neg$, in addition to the equality test $t = t'$ for pointer terms. Pointer terms are built from the pointer variables using the operation $t.succ(i)$, which denotes the node obtained by moving along edge number i from the node denoted by t.

This notion of minimal pure pointer program is essentially equivalent to (a uniform version of) the *Jumping Automata on Graphs* (JAGs) of Cook & Rackoff [2]. Previous work on JAGs shows that while the class of minimal pure pointer programs is far from trivial, it is fairly weak [2, 4]. For example, it is impossible to write a program to check the input graph for acyclicity. The reason for this weakness is that JAGs cannot in general visit all nodes of the graph. Since they can only reach new graph nodes by traversing graph edges, they cannot reach disconnected components of the graph.

To obtain more expressive classes of pointer algorithms, one may make a number of reasonable extensions to minimal pure pointer programs. One well-known approach is to add a total ordering on the nodes of the input graph. In this approach, however, one leaves the realm of *pure* pointer algorithms, as an ordering on the pointers allows one to encode data in the pointers themselves. In programming-language terms, adding a total order amounts to allowing pointer-arithmetic, that is moving from the `Java`-like pointers to `C`-like ones. With a total ordering on the graph nodes it is possible to encode the tapes of a

LOGSPACE-Turing Machine, so that any LOGSPACE-algorithm can be encoded. Since our aim is to study the expressibility of LOGSPACE-algorithms by pure pointer algorithms, we thus find a total ordering to be too strong an addition.

A way to allay the weakness of minimal pure pointer programs, while still remaining within the realm of pure pointer programs, is to allow pointer programs to iterate over the whole input graph. The idea is to add to the minimal pointer programming language a `forall`-loop for iteration over all nodes of the graph. The loop (`forall` x `do` M) represents the algorithm that iterates over all graph nodes by setting x successively to each graph node and executing the loop body M after each such setting. Importantly, there are no guarantees on the order in which x iterates over the graph and this order may be different in each run of the `forall`-loop. The program must return the correct result *independently of the order* chosen in the `forall`-loops, i.e. for any choice of iteration order the program must give the correct result. This independence from the evaluation order ensures that pointers remain pure and that their internal representation cannot be inspected by the program. We have shown in [5] that pure pointer programs with iteration, while being able to express interesting problems, do in general only have a very limited ability to count and therefore are not able to encode the tapes of a Turing Machine in the pointers.

A simple example of a pointer program with iteration is the program

$$b := \texttt{false};\ \texttt{forall}\ x\ \texttt{do}\ b := \neg b\ ,$$

which computes the parity of the input graph. More substantial examples for pointer programs with iteration can be found in [5], for example that the well-known algorithm of Cook & McKenzie [1] that decides acyclicity of graphs can be programmed easily. Moreover, it is not hard to see that Deterministic Transitive Closure logic (DTC-logic) on locally ordered graphs (of constant degree), see e.g. [3, 4], is subsumed by the pure pointer language with iteration in the sense that DTC-logic queries can be evaluated in this language.

Having introduced pure pointer programs, we can now state the main objective of this paper. We consider the problem of **s**-**t**-reachability in undirected graphs and study its expressibility by pure pointer programs. Results of Cook & Rackoff [2] imply that this problem cannot be expressed by minimal pure pointer programs. The main result of this paper is that undirected **s**-**t**-reachability cannot be solved by pure pointer programs with iteration either. As a corollary we obtain that this problem cannot be expressed in DTC-logic on locally ordered graphs. This answers an open question by Etessami & Immerman [4].

Our result identifies undirected **s**-**t**-reachability as a problem that appears to require impure pointer operations in order to be expressed by a pointer program. With Reingold's algorithm [9], it *is* possible to solve this problem by a minimal pure pointer program that has access to a constant number of counting registers of logarithmic size [8], e.g. encoded by a logarithmic number of boolean variables. Counting registers are an impure addition to the language, since they can be used to encode Turing-Machine tapes, as with the addition of a total ordering.

2 Locally Ordered Graphs

A *locally ordered graph* of degree d consists of a set of nodes V and a function $succ\colon V \times \{1, \ldots, d\} \to V$. The intention is that the i-th edge leaving node v

goes to node $succ(v, i)$. The graph Γ is undirected if, for any two nodes v and w, there is an edge from v to w and one from w to v. An **s**-**t**-*graph* is a locally ordered graph Γ with two distinguished nodes $\mathbf{s}$ and $\mathbf{t}$.

We use letter Γ to range over locally ordered graphs and write (Γ, s, t) for the **s**-**t**-graph with $\mathbf{s} := s$ and $\mathbf{t} := t$. We often write $v \in \Gamma$ to indicate that v is a node in Γ. We write $deg(\Gamma)$ for the degree of Γ.

The *distance* of two nodes v and w is the length of the shortest path connecting them. We write $d(v, w)$ for it. A *ball of radius* r around a node v is the set

$$\mathrm{B}_\Gamma(v, r) = \{w \mid d(v, w) \leq r\}.$$

We extend this notation to sets of nodes W and write $\mathrm{B}_\Gamma(W, r)$ for the neighbourhood of radius r around the nodes in W.

$$\mathrm{B}_\Gamma(W, r) = \bigcup_{v \in W} \mathrm{B}_\Gamma(v, r)$$

We write $diam(\Gamma)$ for the *diameter* of Γ, that is the largest distance of two nodes in Γ.

3 Pure Pointer Programs with Iteration

In this section we define a PURe Pointer LanguagE (PURPLE) to formalise the notion of pure pointer program with iteration that we discussed in the Introduction. We concentrate on giving the definitions that are needed for this paper and refer to [5] for an in-depth discussion of PURPLE with examples and a proof that PURPLE is indeed a *pure* pointer language.

Terms. There are two types of terms, one for booleans and one for pointers to the graph nodes. Fix countably infinite sets $Vars^{\mathtt{bool}}$ and $Vars^{\Gamma}$ of boolean variables and graph pointer variables. We make the convention that b, c denote boolean variables and x, y denote graph pointer variables. The terms are then generated by the following grammars.

$$\begin{aligned} t^{\mathtt{bool}} &::= \ \mathtt{true} \mid \mathtt{false} \mid b \mid \neg t^{\mathtt{bool}} \mid t^{\mathtt{bool}} \wedge t^{\mathtt{bool}} \mid t^{\mathtt{bool}} \vee t^{\mathtt{bool}} \mid t^{\Gamma} = t^{\Gamma} \\ t^{\Gamma} &::= \ x \mid \mathbf{s} \mid \mathbf{t} \mid t^{\Gamma}.succ(i) \qquad (\text{where } i \in \mathbb{N}) \end{aligned}$$

The intention is that pointer terms denote graph nodes and that the term $t.succ(i)$ denotes the node reached by following edge number i from the node denoted by t. The only direct observation about pointers is the equality test $t = t'$.

Programs. The set of PURPLE programs is defined inductively by the grammar below.

$$\begin{aligned} M ::= \ & \mathtt{skip} \mid M; M \mid x := t^{\Gamma} \mid b := t^{\mathtt{bool}} \\ & \mid \mathtt{if}\ t^{\mathtt{bool}}\ \mathtt{then}\ M\ \mathtt{else}\ M \mid \mathtt{while}\ t^{\mathtt{bool}}\ \mathtt{do}\ M \mid \mathtt{forall}\ x\ \mathtt{do}\ M \end{aligned}$$

The intended behaviour of ($\mathtt{forall}\ x\ \mathtt{do}\ M$) is that the pointer variable x iterates over the nodes of the input graph in some unspecified order, visiting each element exactly once, and M is executed after each setting of x.

We write $Vars^{\texttt{bool}}(M)$ for the boolean variables appearing in program M and $Vars^{\Gamma}(M)$ for the graph variables in M. We define the `forall`-depth of a program M to be the nesting depth of the `forall`-loops in M. A program without `forall`-loops has `forall`-depth 0.

3.1 Operational Semantics

The operational semantics of PURPLE is parameterised over a finite locally ordered **s**-**t**-graph Γ. It is formulated in terms of a small-step transition relation $\rightarrow_\Gamma$. To define this transition relation, we define a set of *extended programs* that have annotations for keeping track of which variables have already been visited in the computation of the `forall`-loops. The set of extended programs consists of PURPLE programs in which the `forall`-loops are not restricted to an iteration over all graph nodes, but where (`for` $x \in W$ `do` M) is allowed for any set of nodes W. We identify (`forall` x `do` M) with (`for` $x \in V$ `do` M), where V is the node set of Γ.

The transition relation $\rightarrow_\Gamma$ is a binary relation on configurations. A *configuration* is a triple (M, q, ρ) consisting of an extended program M, an assignment q that maps the boolean variables in M to boolean values, and an assignment ρ that maps the graph variables in M to nodes in Γ. For assignments q and ρ that are defined on more than the variables in M, we will usually write just (M, q, ρ) for $(M, q|_{Vars^{\texttt{bool}}(M)}, \rho|_{Vars^{\Gamma}(M)})$, making the restriction implicit. The relation $\rightarrow_\Gamma$ is defined in Figure 1. There we write $[\![t]\!]_{q,\rho}$ for the evident interpretations of terms with respect to the variable assignments q and ρ:

Boolean Terms	Graph Pointer Terms
$[\![\texttt{true}]\!]_{q,\rho} = \texttt{true}$	$[\![x]\!]_{q,\rho} = \rho(x)$
$[\![\texttt{false}]\!]_{q,\rho} = \texttt{false}$	$[\![\mathbf{s}]\!]_{q,\rho} = \mathbf{s}$
$[\![b]\!]_{q,\rho} = q(b)$	$[\![\mathbf{t}]\!]_{q,\rho} = \mathbf{t}$
$[\![\neg t]\!]_{q,\rho} = \neg[\![t]\!]_{q,\rho}$	$[\![t.succ(i)]\!]_{q,\rho} = succ\,([\![t]\!]_{q,\rho}, i)$
$[\![t_1 \wedge t_2]\!]_{q,\rho} = [\![t_1]\!]_{q,\rho} \wedge [\![t_2]\!]_{q,\rho}$	
$[\![t_1 \vee t_2]\!]_{q,\rho} = [\![t_1]\!]_{q,\rho} \vee [\![t_2]\!]_{q,\rho}$	
$[\![t_1 = t_2]\!]_{q,\rho} = ([\![t_1]\!]_{q,\rho} = [\![t_2]\!]_{q,\rho})$	

We make the convention that $succ(v, i)$ denotes v if i exceeds the degree of Γ. As usual, we write $\rightarrow_\Gamma^*$ for the reflexive transitive closure of $\rightarrow_\Gamma$.

We say that a configuration (M, q, ρ) is *strongly terminating* if there is no infinite reduction sequence of $\rightarrow_\Gamma$ starting from it. We say that a program M is *strongly terminating* if any configuration (M, q, ρ) on any graph is strongly terminating.

To define what it means for a set of graphs to be recognised by a program, we choose a distinguished boolean variable *result* that indicates the outcome of a computation.

Definition 1 (Recognition)**.** A set X of finite graphs is *recognised* by a program M, if M is strongly terminating and, for all graphs Γ and all ρ, ρ', q and q' satisfying

$$(M,\ q,\ \rho) \rightarrow_\Gamma^* (\texttt{skip}, q',\ \rho')\ ,$$

one has $q'(\mathit{result}) = \texttt{true}$ if and only if $\Gamma \in X$.

Assignment

$$(x := t^{\Gamma}, q, \rho) \to_{\Gamma} (\texttt{skip}, q, \rho[x := [\![t^{\Gamma}]\!]_{q,\rho}])$$
$$(b := t^{\texttt{bool}}, q, \rho) \to_{\Gamma} (\texttt{skip}, q[b := [\![t^{\texttt{bool}}]\!]_{q,\rho}], \rho)$$

Composition

$$(\texttt{skip}; M,\ q,\ \rho) \to_{\Gamma} (M,\ q,\ \rho) \qquad \frac{(M,\ q,\ \rho) \to_{\Gamma} (M',\ q',\ \rho')}{(M; N,\ q,\ \rho) \to_{\Gamma} (M'; N,\ q',\ \rho')}$$

Conditional

$$(\texttt{if } t \texttt{ then } M \texttt{ else } N,\ q,\ \rho) \to_{\Gamma} (M,\ q,\ \rho) \text{ if } [\![t]\!]_{e,\rho} = \texttt{true}$$
$$(\texttt{if } t \texttt{ then } M \texttt{ else } N,\ q,\ \rho) \to_{\Gamma} (N,\ q,\ \rho) \text{ if } [\![t]\!]_{e,\rho} = \texttt{false}$$

`while`-loop

$$(\texttt{while } t \texttt{ do } M,\ q,\ \rho) \to_{\Gamma} (\texttt{skip},\ q,\ \rho) \text{ if } [\![t]\!]_{q,\rho} = \texttt{false}$$
$$(\texttt{while } t \texttt{ do } M,\ q,\ \rho) \to_{\Gamma} (M;\ \texttt{while } t \texttt{ do } M,\ q,\ \rho) \text{ if } [\![t]\!]_{q,\rho} = \texttt{true}$$

`for`-loop

$$(\texttt{for } x \in \emptyset \texttt{ do } M,\ q,\ \rho) \to_{\Gamma} (\texttt{skip},\ q,\ \rho)$$

$$\frac{v \in W}{(\texttt{for } x \in W \texttt{ do } M,\ q,\ \rho) \to_{\Gamma} (M;\ \texttt{for } x \in W \setminus \{v\} \texttt{ do } M,\ q,\ \rho[x := v])}$$

Figure 1: Operational Semantics

Our notion of recognition should not be confused with the usual definition of acceptance for existentially (nondeterministic) or universally branching Turing machines; in contrast to those concepts it is completely symmetrical in X vs. $\overline{X}$. If the input is in X then all runs must accept; if the input is not in X then all runs must reject. In particular, not even for strongly terminating M can we *in general* define 'the language of M'. A program whose result depends on the traversal order does not recognise any set at all.

Definition 2 (Reachability)**.** A PURPLE program M *decides* **s**-**t**-*reachability* for a set X of finite graphs if M recognises the following set of **s**-**t**-graphs

$$\{(\Gamma, s, t) \mid \Gamma \in X \text{ and } s \text{ and } t \text{ are connected by a path in } \Gamma\}.$$

For a more detailed discussion of PURPLE and example programs, we refer to [5].

3.2 Elimination of While-Loops

A property of PURPLE that is useful for studying its expressivity is that all `while`-loops can be eliminated. Since `while`-loops are the only source of non-termination, we therefore need not study the termination behaviour of programs.

Proposition 1. *For any program M, there exists a* `while`*-free program M' that recognises the same sets of graphs.*

Proof sketch. If M is not strongly terminating then it does not recognise any set of graphs, and we can take M' to be, e.g., $((\texttt{forall}\ x\ \texttt{do}\ \texttt{skip}); \mathit{result} := (x = y))$.

For the case where M is strongly terminating, note that a program with m boolean variables and n graph variables can assume no more than $2^m \cdot |V|^n$ distinct states, where V is the set of nodes of the input graph. Therefore, any `while`-loop that is executed more often than this number must go into an infinite loop. Hence, we can transform each program into a `while`-free program by unrolling each `while`-loop into t nested `forall`-loops where $|V|^t \geq 2^m \cdot |V|^n$ that execute the loop body up to $2^m \cdot |V|^n$ times. □

3.3 Relation to Other Models

3.3.1 Jumping Automata on Graphs

We have already mentioned in the Introduction that PURPLE-programs without `forall`-loops are essentially the same as the Jumping Automata on Graphs (JAGs) of Cook & Rackoff [2]. Since we build on existing results about JAGs, we include a precise definition of JAGs in this section.

Definition 3. A *Jumping Automaton on Graphs* (JAG) consists of a finite set of states Q, a subset $F \subseteq Q$ of final states, a finite set of *pebbles* P and a transition function $\delta\colon Q \times \Sigma(P) \to Q \times ((P \times P) + (P \times \mathbb{N}))$, where $\Sigma(P)$ is the set of all equivalence relations on P.

A JAG $J = (Q, F, P, \delta)$ works on a locally ordered graph $\Gamma = (V, \mathit{succ})$ in the following way. The *configurations* of the computation are pairs $(q, \rho) \in Q \times V^P$ of a state and a placement of the pebbles on the graph. In each step, the automaton J can see of the configuration only the state q and the incidence relation of ρ, i.e. the equivalence relation $[\rho] \in \Sigma(P)$ defined by $x[\rho]y \iff \rho(x) = \rho(y)$. Then, the behaviour of J on Γ is defined by the following binary transition relation $\rightarrow_J$ on configurations.

$$
\begin{aligned}
&(q, \rho) \rightarrow_J (q', \rho[x := y]) && \text{if } \delta(q, [\rho]) = (q', \mathrm{inl}(x, y)) \text{ and } q \notin F \\
&(q, \rho) \rightarrow_J (q', \rho[x := \mathit{succ}(x, i)]) && \text{if } \delta(q, [\rho]) = (q', \mathrm{inr}(x, i)) \text{ and } q \notin F
\end{aligned}
$$

Here, we make the convention that $\mathit{succ}(x, i) = x$ holds whenever i exceeds the degree of Γ.

With these definitions, it should be evident that each `forall`-free PURPLE-program can be compiled into a JAG and vice versa.

3.3.2 Deterministic Transitive Closure Logic

A second well-known class of pure pointer programs is captured by *Deterministic Transitive Closure (DTC) logic* on locally ordered graphs, see e.g. [4]. This logic captures strictly more algorithms than JAGs, since it allows for first-order quantification. For example, acyclicity in undirected graphs can be expressed in locally ordered DTC-logic, but not by JAGs.

More evidence for the claim that PURPLE captures the intuitive notion of being computable with a constant number of pure pointers, is given by the fact that PURPLE subsumes DTC-logic on locally ordered graphs. In DTC-logic on locally ordered graphs, one has, in addition to the binary edge relation $E(-,-)$, a ternary relation $F(-,-,-)$, such that $F(v,-,-)$ is a total ordering on $\{w \mid$

$E(v,w)\}$, for any v, see [4]. This representation of graphs in the logic is also called a one-way local ordering (1LO) [4], since it consists of an ordering of all the outgoing edges of each node. It is also possible to study a two-way local ordering (2LO), where in addition the incoming edges of each node are ordered as well. Since in an undirected graph a one-way local ordering automatically induces a two-way local ordering, we shall not need to study two-way local orderings in this paper: showing that DTC-formulae with a one-way local ordering cannot express **s**-**t**-reachability in undirected graphs is enough for showing the same result for formulae with a two-way local ordering.

Each DTC-formula on locally ordered graphs can easily be evaluated by a PURPLE-program, see [5]. For the sake of completeness we include a proof sketch for the case of bounded degree graphs. The general case uses the encoding of unbounded degree graphs given in [5].

Proposition 2. *For each closed* DTC *formula φ for locally ordered graphs there exists a program M_φ such that, for any finite locally ordered graph Γ of degree d, one has $\Gamma \models \varphi$ if and only if M_φ recognises Γ.*

Proof sketch. One constructs for each DTC-formula φ with free variables among $x_1, \ldots, x_n$ a strongly terminating program M_φ such that for all q, q', ρ, ρ' satisfying $(M_\varphi, q, \rho) \to_\Gamma^* (\texttt{skip}, q', \rho')$ one has $q'(\mathit{result}) = \texttt{true}$ if and only if $\Gamma, \rho \models \varphi$. Specialising to $n = 0$ yields the proposition. The construction of M_φ proceeds by induction on φ. We give a representative selection of cases using self-explanatory syntactic sugar as appropriate.

For atomic formulae we put

$$\begin{aligned} M_{E(x,y)} &\equiv \mathit{result} := \textstyle\bigvee_{1\le i\le d} y{=}x.\mathit{succ}(i) \ , \\ M_{F(x,y,z)} &\equiv \mathit{result} := \textstyle\bigvee_{1\le i<j\le d} y{=}x.\mathit{succ}(i) \wedge z{=}x.\mathit{succ}(j) \ . \end{aligned}$$

Quantifiers are dealt with using a `forall`-loop, e.g. $M_{\forall x.\varphi}$ is

$$\vec{z} := \vec{x};\ b := \texttt{true};\ \texttt{forall}\ x\ \texttt{do}\ (M_\varphi;\ \vec{x} := \vec{z};\ b := b \wedge \mathit{result});\ \mathit{result} := b \ ,$$

where $\vec{z}$ is a list of fresh graph variables used to restore the initial values of $\vec{x}$ after each execution of M_φ.

For deterministic transitive closure we use a number of nested `forall`-loops to cycle through all tuples of elements in order to find the next tuple. Writing `forall` $\vec{x}$ `do` M as a shorthand for `forall` x_1 `do` $\cdots$ `forall` x_n `do` M and using similar notation for assignments to vectors of variables and equality test we get the following definition of $M_{\mathrm{DTC}_{\vec{x},\vec{y}}(\varphi)(\vec{u},\vec{v})}$. We assume that φ is the graph of a partial function. If it is not we can replace it by $\varphi \wedge (\forall \vec{z}.\, \varphi[\vec{z}/\vec{y}] \Rightarrow \vec{y}{=}\vec{z})$.

$\vec{x} := \vec{u}$;
$\mathit{found} := \texttt{false}$;
`forall` $\vec{t}$ `do` /* Execute body $|\Gamma|^n$ times */
  $\vec{z} := \vec{x}$;
  `forall` $\vec{y}$ `do` /* Try to find next step */
    $\vec{x} := \vec{z}$; M_φ; `if` result `then` $\vec{a} := \vec{y}$ `else skip`;
  $\mathit{found} := \mathit{found} \vee \vec{v}{=}\vec{a}$;
  $\vec{x} := \vec{a}$;
$\mathit{result} := \mathit{found}$

□

The converse of this proposition is not true, since DTC-logic cannot express that the number of nodes in the input graph is even. We have observed in the Introduction that it is possible to write such a program in PURPLE.

4 Undirected Reachability

We now turn to proving the main result of this paper, that undirected **s**-**t**-reachability cannot be expressed in PURPLE. Since the proof takes up the rest of the paper, we start by sketching the main idea and giving a general overview.

4.1 Overview

The general idea is to show a locality property of PURPLE programs on a certain class of graphs. A PURPLE program without any loops is local in the sense that it can move its graph variables only within a certain radius of their initial positions. This radius is trivially bounded by the size of the program. Hence, a loop-free program is confined to a certain neighbourhood around the initial positions of the graph variables in the start configuration. General PURPLE programs with `forall`-loops are not local; the `forall`-loop allows one to place graph variables at an arbitrary distance from the positions in the start configuration. For example, it is easy to write a program that places a graph variable on some node having a self-loop. Since a self-loop can be arbitrarily far away from the start configuration, this shows that `forall`-loops cannot in general be local.

In this section we show that nevertheless there exist graphs, on which each PURPLE program satisfies a locality property. We will show that for each PURPLE program M there exists a program L without `forall`- or `while`-loops that implements M on a particularly constructed graph in the sense that from each start configuration both M and L can reach the same end-configuration. Notice that the end configuration of the `forall`-free program L is uniquely determined by the start configuration. The program M, on the other hand, may have more than one run, so the assertion is that L simulates one particular run of M. This is appropriate for the goal of proving that M cannot decide undirected reachability, since, by the definition of 'recognises', M must obtain the correct answer on *all* runs.

While L will be much larger than M, we will see that we can choose the input graph such that the radius r up to which L can explore it remains much smaller than the graph itself. This will allow us to show that L cannot distinguish between the situation where $\mathbf{s}$ and $\mathbf{t}$ lie on different connected components from the situation where they have distance greater than $2r$ on the same connected component. This then implies that L cannot decide undirected reachability and as a result we obtain that M cannot decide undirected reachability either, since it does not compute the correct result on the runs implemented by L.

The main difficulty in carrying out this proof idea is implementing `forall`-loops by local programs. After the execution of a `forall`-loop, the graph variables may point to nodes that are arbitrarily far away from the nodes in the start configuration. Note, however, that we can influence the positions in the final configuration by an appropriate choice of the iteration order. Our strategy for showing locality of `forall`-loops is then to construct graphs on which we can choose the iteration order for the `forall`-loops so as to force the graph variables

to be in a small enough neighbourhood around the nodes in the start configuration. With such a choice we can implement the `forall`-loop by a local program that has enough states to exhaustively explore this neighbourhood. The graphs that we construct fall into the class of *action graphs*, where the nodes form a G-set for a finite group G and where edges are given by multiplication with elements from a fixed generating set of G. By using action graphs, we can reuse existing results on the expressivity of JAGs [2, 10] (equivalently: `forall`-free PURPLE programs) and the development in this paper relies crucially on these results.

We define action graphs in Section 4.2. In Section 4.3 we then define *lamplighter graphs*, a particular class of action graphs whose underlying group is obtained by iterated application of the lamplighter construction (wreath product with $\mathbb{Z}/2\mathbb{Z}$) to a group $\mathbb{Z}/m\mathbb{Z}$. It is on these graphs that PURPLE programs cannot decide undirected reachability. In Section 4.4 we define precisely what we mean by a *local* program. We identify conditions for action graphs that are sufficient for showing that local programs are closed under composition and `forall`-loops. In Section 4.5 we then show that the lamplighter construction from Section 4.3 does indeed allow one to construct action graphs satisfying these conditions. In Section 4.6, finally, we combine these results to show that PURPLE cannot decide undirected reachability.

4.2 Action Graphs

We study the behaviour of programs on *action graphs*, which are locally ordered graphs defined by a group action on a set. Action graphs are closely related to *Cayley graphs* and the particular action graphs that are important for this paper consist of two disjoint copies of the Cayley graph of some group. In this section we define action graphs and fix group-theoretic notation.

All groups in this paper are finite. We write them multiplicatively and denote the unit element of group G by e_G or just e. The *exponent* of a group G is the least common multiple of the orders of all its elements. We write $\mathbb{Z}/m\mathbb{Z}$ for the cyclic group of integers modulo m.

A *group action* of a group G on a set V is a function $(-)\cdot(-)\colon V \times G \to V$ that satisfies $v \cdot e = v$ and $(v \cdot g) \cdot h = v \cdot (g \cdot h)$ for all $v \in V$ and $g, h \in G$. A group action if *free*[1] if $v \cdot g = v$ implies $g = e_G$ for all $v \in V$ and $g \in G$.

Definition 4 (Action graph)**.** Let V be a non-empty finite set, G be a group generated by a set S that is closed under inverses, i.e. $S^{-1} = S$ holds, and let $(-)\cdot(-)\colon V \times G \to V$ be a group action of G on V. The *action graph* $A(V, G, S, \cdot)$ is the undirected graph with node set V and an edge from v to $v \cdot s$ for each $v \in V$ and each $s \in S$.

As usual, we shall often not distinguish notationally between an action graph and its set of nodes. We call G the *edge group* of $A(V, G, S, \cdot)$ and S its *generators*. Note that we have $deg(A) = |S|$. We say that an action graph has exponent m if its edge group does.

Any total ordering on S induces a local ordering on this action graph: if s is the i-th element of S then $(v, v \cdot s)$ is the i-th outgoing edge from v. In this

[1] Free group actions are also often called semiregular group actions. We use the term 'free' to avoid confusion with the graph-theoretic meaning of 'regular'.

paper we assume that all action graphs are locally ordered in this way. Since the particular choice of the ordering on the generating sets is not important for our results, we will not show it explicitly.

A *free action graph* is an action graph with a free group action. For any two nodes v and w in the same connected component of a free action graph there exists a unique element $(w/v) \in G$ such that $v \cdot (w/v) = w$ holds.

Cayley graphs are prominent examples of free action graphs. If G is a group and S is a generating set for G that is closed under inverses ($S^{-1} = S$), then the *Cayley graph* $C(G, S)$ is the undirected graph whose nodes are the elements of G and whose edges are $\{(g, gs) \mid g \in G, s \in S\}$. It is the free action graph $A(G, G, S, \cdot)$, where $(-) \cdot (-)$ is the free action given by group multiplication.

In fact, any free action graph with edge group G and generators S is isomorphic to a number of disjoint copies of $C(G, S)$.

We note that the elements of a ball $\mathrm{B}_A(v, r)$ in $A = A(V, G, S, \cdot)$ can be addressed by elements of the ball $\mathrm{B}_C(e_G, r)$ in $C = C(G, S)$. That is, for each $\pi \in \mathrm{B}_C(e_G, r)$ the node $v \cdot \pi \in V$ is an element of $\mathrm{B}_A(v, r)$ and each element of $\mathrm{B}_A(v, r)$ appears in this way. Relying on this fact, we will work with elements of $\mathrm{B}_C(e_G, r)$ instead of paths of length at most r. In free action graphs, each ball $\mathrm{B}_A(v, r)$ is in fact isomorphic to $\mathrm{B}_C(e_G, r)$.

Action graphs are interesting for the purpose of this paper because of the following result that gives a bound on how many nodes a JAG can visit in them.

Theorem 3. *There exists a constant c, such that in any free action graph A with exponent m, any* JAG *with Q states and P pebbles can visit at most $(Qm)^{c^P}$ nodes from any start configuration.*

This theorem is proved by a direct generalisation of an argument of Cook & Rackoff [2]. They prove the special case of the theorem where A is the Cayley graph for the commutative group $(\mathbb{Z}/m\mathbb{Z})^d$ with pointwise addition $(x_1, \ldots, x_d) + (y_1, \ldots, y_d) = (x_1 + y_1, \ldots, x_d + y_d)$ and the generating set $\delta_1, \ldots, \delta_d$, where δ_i is the vector with a 1 in the i-th component and a 0 in all other components. The proof of Cook & Rackoff can be generalised directly to yield the Theorem above [10]. It has been formalised and verified in the theorem prover Coq; see [10] for details.

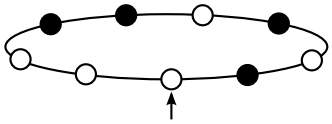
4.3 Lamplighter Graphs

In order to prove that pure pointer programs with iteration cannot decide undirected reachability, we construct graphs of small degree in which JAGs can only visit very small neighbourhoods of their start configuration compared to the overall size of the graph. We obtain such graphs by iterating a construction of *lamplighter graphs*, which we define in this section.

Lamplighter graphs get their name from the following intuition. Before the invention of automatic street lights, towns used to employ lamplighters who would go around town to light all street lamps at night and put them out again at dawn. Lamplighting was not a very varied job: at any time the lamplighter could only choose to put on/off a lamp or to move to a neighbouring lamp. Nevertheless, the lamplighter could choose different routes and so he might have entertained himself (and the town's citizens, no doubt) by lighting the lamps in ever different orders. Lamplighter graphs describe the different options the

lamplighter had. The nodes are the possible situations the lamplighter might be in at any time. A node is given by the lighting state of all lamps, i.e. whether they are on or off, and the position of the lamplighter, i.e. at which lamp he currently stands. There is an edge between two nodes in this graph, if the lamplighter can get from the situation described in one node to that in the other node by lighting or extinguishing the lamp in his current position or by moving to a neighbouring lamp.

Pictured below is a node of the lamplighter graph with nine lamps in a circular arrangement. The balls represent street lamps, the white ones being lit, and the arrow indicates the position of the lamplighter.



In the rest of this section we define precisely lamplighter graphs for any arrangement of lamps that is given by a Cayley graph. In the above picture the lamp arrangement is the Cayley graph of the cyclic group $\mathbb{Z}/9\mathbb{Z}$. Because we want to use Theorem 3, we define lamplighter graphs as Cayley graphs of a lamplighter group.

Let G be a group that should represent the lamp arrangement. The set $|G| \to |\mathbb{Z}/2\mathbb{Z}|$ of all functions from the underlying set of G to that of $\mathbb{Z}/2\mathbb{Z}$ becomes a commutative group when addition is defined pointwise by $(f+g)(x) = f(x) + g(x)$ and $0(x) = 0$. We write δ_x for the function in $|G| \to |\mathbb{Z}/2\mathbb{Z}|$ defined by

$$\delta_x(y) = \begin{cases} 1 & \text{if } x = y, \\ 0 & \text{otherwise.} \end{cases}$$

Furthermore, there is a left-action of G on the set $|G| \to |\mathbb{Z}/2\mathbb{Z}|$ with definition $(x \cdot f)(y) = f(x^{-1} \cdot y)$. For example, we have $\delta_x = x \cdot \delta_e$.

Using this notation, we define $L(G)$ – the *lamplighter group on* G – to be the group with underlying set $(|G| \to |\mathbb{Z}/2\mathbb{Z}|) \times |G|$ and with group multiplication

$$(f, x) \cdot (g, y) = (f + x \cdot g,\, x \cdot y) \ .$$

An element (f, x) of this group corresponds to a node of a lamplighter graph, where f represents the lighting state of the lamps and x represents the position of the lamplighter. Neutral element and inverses in $L(G)$ are $e = (0,\, e_G)$ and $(f, x)^{-1} = (-x^{-1} \cdot f,\, x^{-1})$. This definition of the lamplighter group is an example of a *wreath product* of groups, i.e. $L(G) = (\mathbb{Z}/2\mathbb{Z}) \wr G$.

Lemma 4. *If S is a generating set for G then $\{(0, x) \mid x \in S\} \cup \{(\delta_e, e)\}$ is a generating set for the lamplighter group $L(G)$.*

Proof. For any f and x, we can write $(f + \delta_y, x)$ as the product $(f, x) \cdot (0, x^{-1}y) \cdot (\delta_e, e) \cdot (0, y^{-1}x)$. Since each f can be written as a finite sum of δ_ys, we can use this observation repeatedly to write (f, x) as a product of elements of the form $(0, z)$ and (δ_e, e). These elements can evidently be formed from the claimed generating set. □

Lemma 5. *If G has exponent m then $L(G)$ has exponent $2m$.*

Proof. For any k, the power $(f,x)^k$ clearly has the form (f_k, x^k) for some f_k. It follows easily by induction on k that f_k can in fact be written as $f_k = f + x \cdot f + \cdots + x^{k-1} \cdot f$. Since m is the exponent of G, we have $x^m \cdot f = f$, which implies $f_{2m} = 2(f + x^1 \cdot f + \cdots + x^{m-1} \cdot f)$. Since the group $|G| \to |\mathbb{Z}/2\mathbb{Z}|$ has exponent 2, this implies $f_{2m} = 0$ and thus the required $(f,x)^{2m} = (0,e)$. □

Definition 5. (Lamplighter Graph) For any group G with generating set S, the *lamplighter graph on* G is the Cayley graph of $L(G)$ with respect to the generating set from Lemma 4. We denote this graph by $\Lambda(G,S)$, or $\Lambda(G)$ if S is clear from the context.

The degree and the cardinality of lamplighter graphs obey the following laws.

$$deg(\Lambda(G,S)) = deg(C(G,S)) + 1 \tag{1}$$

$$|\Lambda(G,S)| = |C(G,S)| \cdot 2^{|C(G,S)|} \tag{2}$$

We shall use the lamplighter construction to construct graphs in which JAGs are confined to areas of small radius. These graphs take the form

$$\Lambda^i(m) := \underbrace{\Lambda(\ldots\Lambda(}_{i \text{ times}} \mathbb{Z}/m\mathbb{Z}, S_m)\ldots)$$

of an iteration of the lamplighter construction on the cyclic group of order $m > 2$. For the generating set S_m of $\mathbb{Z}/m\mathbb{Z}$ we choose the canonical set of generators that is closed under inverses and has two elements, e.g. $\{1, m-1\}$.

Depending on i, the graph $\Lambda^i(m)$ can get very large, having at least

$$\exp_2^i(m) = 2^{2^{\cdot^{\cdot^{\cdot^{2^m}}}}}$$

nodes, where the tower has height i (precisely: $\exp_2^0(m) = m$ and $\exp_2^{i+1}(m) = 2^{\exp_2^i(m)}$). In contrast, the exponent of $\Lambda^i(m)$ is $m \cdot 2^i$ only. Theorem 3 then tells us that any JAG with Q states and P pebbles can visit no more than $\left(Q \cdot m \cdot 2^i\right)^{c^P}$ nodes in this graph. The point is that this term grows much slower than the size of the graph when we increase i. This allows us to find for any given JAG a graph in which the JAG is confined to as small a portion of the graph as we like. Moreover, the degree of the graph $\Lambda^i(m)$ is $i+2$ only. This follows from equation (2) and the choice of S_m.

4.4 Abstract Local Programs on Free Action Graphs

Our approach to showing that PURPLE programs cannot decide undirected reachability is to show that on certain free action graphs one run of each PURPLE program can be implemented by a program that is local in the sense that it depends upon and affects only a certain neighbourhood of the initial graph variable positions. In this section we define local programs on free action graphs and analyse their behaviour. We formulate a variant of Theorem 3 for such local programs in Theorem 10 below. The proof, that each PURPLE program can be implemented by a local one, relies crucially on this theorem.

We define local programs in an abstract, syntax-free way, similar to the definition of JAGs. In this syntax-free presentation we will use the term 'pebble' instead of the term 'graph variable' that we use in the context of PURPLE.

Abstract local programs are local in the sense that they can inspect the graph only up to a certain radius r around the initial positions of their pebbles, and that they can move pebbles only within this radius. Due to the uniform structure of free action graphs, the information that can be obtained by looking at the neighbourhood of radius r around the pebbles amounts to the relative pebble displacements of all pairs of pebbles with distance at most $2r$. For any configuration $\rho \in A^P$ on any free action graph A with edge group G and generators S, we capture this information about relative pebble displacements by a partial function $[\rho]_r$ of type $P \times P \rightharpoonup \mathrm{B}_{C(G,S)}(e_G, 2r)$. The function $[\rho]_r$ is defined such that, for all x and y with $d(\rho(x), \rho(y)) \leq 2r$, $[\rho]_r(x, y)$ is the unique value satisfying $\rho(y) = \rho(x) \cdot [\rho]_r(x, y)$, and $[\rho]_r(x, y)$ is undefined for all other x and y. The relative displacement map $[\rho]_r$ is the only information an abstract local program with local radius r can obtain about its start configuration ρ.

We next define the set $\Sigma_{G,S}(P, r)$ of all relative pebble displacement maps that arise as $[\rho]_r$ for some $\rho \in A^P$ and some free action graph A with edge group G and generators S. This set $\Sigma_{G,S}(P, r)$ consists of all partial functions $c\colon P \times P \rightharpoonup \mathrm{B}_{C(G,S)}(e_G, 2r)$ that satisfy

$$\begin{gathered} c(x, x) = e_G, \\ c(x, y) = c(y, x)^{-1}, \\ c(x, y) \cdot c(y, z) \in \mathrm{B}_{C(G,S)}(e_G, 2r) \implies c(x, z) = c(x, y) \cdot c(y, z) \end{gathered}$$

for all $x, y, z \in P$. Note, in particular, that $\Sigma_{G,S}(P, 0)$ is isomorphic to the set $\Sigma(P)$ of all equivalence relations on P.

We note the following bound on the size of $\Sigma_{G,S}(P, r)$.

$$\begin{aligned} |\Sigma_{G,S}(P, r)| &\leq (|\mathrm{B}_{C(G,S)}(e_G, 2r)| + 1)^{|P|^2} && \text{number of partial functions} \\ &\leq \big((|S| + 1)^{2r} + 1\big)^{|P|^2} && \text{by } |\mathrm{B}_{C(G,S)}(e_G, 2r)| \leq (|S| + 1)^{2r} \end{aligned}$$

The following lemma makes precise the intuition that $\Sigma_{G,S}(P, r)$ classifies what the r-neighbourhood around configurations can look like. We omit the routine proof.

Lemma 6. *Let A be a free action graph with edge group G and generators S. For any two $\rho, \tau \in A^P$, we have $[\rho]_r = [\tau]_r$ if and only if there exists a bijection $\varphi\colon \mathrm{B}_A(\rho, r) \to \mathrm{B}_A(\tau, r)$ that preserves pebble positions, i.e. $\tau = \varphi \circ \rho$ holds, and that preserves the local graph structure, i.e. $\varphi(v \cdot \pi) = \varphi(v) \cdot \pi$ holds for all $v \in A$ and $\pi \in G$ such that $v \cdot \pi$ and v are both in the domain of φ.*

Having defined $\Sigma_{G,S}(P, r)$, we can now define abstract local programs.

Definition 6 (Abstract local program)**.** Let G be a finite group and S be a set of generators that is closed under inverses. An *abstract local program* over G and S with state set Q, pebble set P and local radius r is a function

$$f\colon Q \times \Sigma_{G,S}(P, r) \longrightarrow Q \times P^P \times \mathrm{B}_{C(G,S)}(e_G, r)^P.$$

We write $L_{G,S}(Q, P, r)$ for the set of all such abstract local programs. If A is an action graph with edge group G and generators S, then we will also write $L_A(Q, P, r)$ for $L_{G,S}(Q, P, r)$.

The intention of this definition is that an abstract local program with local radius r looks at the state in Q and the r-ball around its start configuration. With this input, the function that defines the abstract local program then yields a final state in Q and two functions $j \in P^P$ and $m \in \mathrm{B}_{C(G,S)}(e_G, r)^P$ that describe pebble moves in the following sense: each pebble x is first jumped to the position of pebble $j(x)$ in the start configuration and then moved along the path $m(x)$. The intention is that the abstract local program makes these pebble moves and halts.

Formally, this behaviour of f on an action graph A with edge group G and generators S is captured by a function $[\![f]\!]_A \colon Q \times A^P \to Q \times A^P$ that is defined in such a way that $[\![f]\!]_A(q, \rho)$ is the pair (p, τ) defined by

$$\begin{aligned}(p, j, m) &:= f(q, [\rho]_r),\\ \tau(x) &:= \rho(j(x)) \cdot m(x).\end{aligned}$$

We will usually just write $[\![f]\!]$ for $[\![f]\!]_A$ when the subscript is clear from the context.

It is clear that any abstract local program $f \in L_{G,S}(Q, P, r)$ can only make moves within an r-ball of its start configuration, i.e. for all $(q, \rho) \in Q \times A^P$, if we let $(p, \tau) := [\![f]\!](q, \rho)$ then we have $\tau(P) \subseteq \mathrm{B}_A(\rho, r)$.

The local character of abstract local programs leads to the following lemma, which states that if we apply an abstract local program with local radius r to two configurations whose $r + k$-neighbourhoods look the same, then the k-neighbourhoods around the two end-configurations will also look the same.

Lemma 7. *Let A be a free action graph with edge group G and generators S. Then, for all $f \in L_{G,S}(Q, P, r)$, $k \in \mathbb{N}$, $q \in Q$ and $\rho, \tau \in A^P$, if we have $[\rho]_{r+k} = [\tau]_{r+k}$ and define $(q', \rho') := [\![f]\!](q, \rho)$ and $(q'', \tau') := [\![f]\!](q, \tau)$, then we also have $q' = q''$ and $[\rho']_k = [\tau']_k$. Moreover, if $\rho'(x)/\rho(y) \in \mathrm{B}_{C(G,S)}(e, k)$ then $\rho'(x)/\rho(y) = \tau'(x)/\tau(y)$.*

Proof. First we note that $[\rho]_{r+k} = [\tau]_{r+k}$ implies $[\rho]_r = [\tau]_r$. If we then set $(p, j, m) := f(q, [\rho]_r)$, then by definition of $[\![f]\!]$ we have $p = q' = q''$ and $\rho'(x) = \rho(j(x)) \cdot m(x)$ and $\tau'(x) = \tau(j(x)) \cdot m(x)$.

It therefore remains just to show $[\rho']_k = [\tau']_k$. To this end it suffices to show that whenever one of $[\rho']_k(x, y)$ and $[\tau']_k(x, y)$ is defined then so is the other and the values are equal.

Assume without loss of generality that $[\rho']_k(x, y)$ is defined. Then we have $d(\rho'(x), \rho'(y)) \leq 2k$ by definition of $[\rho']_k(x, y)$. Since m maps to $\mathrm{B}_{C(G,S)}(e_G, r)$, this implies $d(\rho(j(x)), \rho(j(y))) \leq 2(k + r)$. Then we have

$$\rho(j(y)) = \rho(j(x)) \cdot [\rho]_{k+r}(j(x), j(y))$$

by definition of $[\rho]_{k+r}$. Using the properties of group actions, we get

$$\rho(j(y)) \cdot m(y) = \rho(j(x)) \cdot m(x) \cdot m(x)^{-1} \cdot [\rho]_{k+r}(j(x), j(y)) \cdot m(y),$$

and thus

$$\rho'(y) = \rho'(x) \cdot m(x)^{-1} \cdot [\rho]_{k+r}(j(x), j(y)) \cdot m(y).$$

But this implies $[\rho']_k(x, y) = m(x)^{-1} \cdot [\rho]_{k+r}(j(x), j(y)) \cdot m(y)$, since the group action is free.

Since from $[\rho]_{k+r} = [\tau]_{k+r}$ and $d(\rho(j(x)), \rho(j(y))) \leq 2(k+r)$ we obtain $d(\tau(j(x)), \tau(j(y))) \leq 2(k+r)$, we can use the same argument to infer $[\tau']_k(x,y) = m(x)^{-1} \cdot [\tau]_{k+r}(j(x), j(y)) \cdot m(y)$. But now we have proved $[\rho']_k(x,y) = [\tau']_k(x,y)$, as required.

For the second part we note that under the given assumptions the element $\pi := \rho'(x)/\rho(y) \cdot m(x)^{-1}$ satisfies $\pi \in \mathrm{B}_{C(G,S)}(e, r+k)$ and $\rho(y) \cdot \pi = \rho(j(x))$, so $\pi = [\rho]_{k+r}(y, j(x))$. Since $[\rho]_{k+r} = [\tau]_{k+r}$ we conclude $\tau(y) \cdot \pi = \tau(j(x))$ and $\tau(y) \cdot \pi \cdot m(x) = \tau'(x)$ and hence the claim. □

Lemma 8. *For all $f \in L_{G,S}(Q, P, r)$ and $g \in L_{G,S}(Q, P, s)$ there exists an abstract local program $comp(f, g) \in L_{G,S}(Q, P, r+s)$ such that $[\![g]\!]_A \circ [\![f]\!]_A = [\![comp(f,g)]\!]_A$ holds for any free action graph A with edge group G and generators S.*

Proof. Lemma 7 shows that for all $c \in \Sigma_{G,S}(P, r+s)$ there exists a unique $c_f \in \Sigma_{G,S}(P, s)$, such that $[\![f]\!](q, \rho) = (p, \tau)$ and $[\rho]_{r+s} = c$ implies $[\tau]_s = c_f$. Hence, we can set $comp(f,g)(q,c)$ to be the triple (q_g, j, m) defined by $j(x) = j_f(j_g(x))$ and $m(x) = m_f(j_g(x)) \cdot m_g(x)$, where $f(q,c) = (q_f, j_f, m_f)$ and $g(q_f, c_f) = (q_g, j_g, m_g)$. □

Any PURPLE program without `while` or `forall`-loop can be compiled directly into an abstract local program. We show this in the following lemma, in which we write $|M|$ for the length of PURPLE program M, i.e. the number of nodes in the abstract syntax tree.

Lemma 9. *Let A be a free action graph with edge group G and generators S. For each* PURPLE *program M without* `while`*-loops or* `forall`*-loops there exists an abstract local program*

$$f \in L_{G,S}\left(Q(M), P(M), |M|\right),$$

with $Q(M) = (Vars^{\mathtt{bool}}(M) \to 2)$ and $P(M) = Vars^{\Gamma}(M) \uplus \{\mathbf{s}, \mathbf{t}\}$, such that, for all $q, p \in Q(M)$ and all $\rho, \tau \in A^{P(M)}$, $[\![f]\!](q,\rho) = (p,\tau)$ holds if and only if $(M, q, \rho) \to^*_{(A, \rho(\mathbf{s}), \rho(\mathbf{t}))} (\mathtt{skip}, p, \tau)$ does.

Proof. The proof goes by structural induction on program M. To simplify the proof, we first observe that if the variables of a program M' are contained in those of M then any $f' \in L_{G,S}(Q(M'), P(M'), r)$ induces a canonical $f \in L_{G,S}(Q(M), P(M), r)$ whose behaviour on the variables of M' is as prescribed by f' and which leaves all other variables unchanged. Because of this fact, we can in the following work only with abstract programs in $L(r) := L_{G,S}(Q(M), P(M), r)$.

The cases of the structural induction are as follows.

- **Case M is `skip`.** Take $f(q,c) := (q, id, \lambda x.\, e_G)$.
- **Case M is $M'; M''$.** Use the induction hypothesis and the above observation to obtain abstract local programs $f' \in L(|M'|)$ and $f'' \in L(|M''|)$. The program $comp(f', f'') \in L(|M'| + |M''|)$ has the required property by Lemma 8. Because of $|M| = |M'| + |M''| + 1$ we can consider $comp(f', f'')$ an element of $L(M)$ and thus complete this case.

- **Case M is $x := t^\Gamma$.** We have $|M| = 1 + |t^\Gamma|$. The term t^Γ has the form $y.succ(i_1).\ldots.succ(i_n)$ for suitable $y \in P(M)$ and $i_1, \ldots, i_n \in \mathbb{N}$ with $n \geq 0$. Let G be the edge group of A and let $s_1, \ldots, s_k$ be the list of generators with respect to which the action graph is formed. For $l > k$, set $s_l := e_G$. Then $\pi := s_{i_1} \cdot s_{i_2} \cdots s_{i_n}$ defines an element of $\mathrm{B}_{C(G,S)}(e_G, |t^\Gamma|)$. Define now $f \in L(|M|)$ by $f(q, c) = (q, id[x/y], m)$, where $m(x) = \pi$ and $m(z) = e_G$ for all $z \neq x$.

- **Case M is $x := t^{\mathtt{bool}}$.** We note that $[\![t^{\mathtt{bool}}]\!]_{q,\rho}$ depends only on $[\rho]_{|M|}$. Hence we can define an abstract local program f with the required property by letting $f(q, c) := (q[x := [\![t]\!]_{q,c}], id, \lambda x.\, e_G)$.

- **Case M is `if` $t^{\mathtt{bool}}$ `then` M' `else` M''.** We define f by

$$f(q, c) = \begin{cases} f'(q, c) & \text{if } [\![t^{\mathtt{bool}}]\!]_{q,c} = \mathtt{true} \\ f''(q, c) & \text{if } [\![t^{\mathtt{bool}}]\!]_{q,c} = \mathtt{false}, \end{cases}$$

  where $f' \in L(|M'|)$ and $f'' \in L(|M''|)$ are the abstract local programs given by the induction hypothesis.

□

4.4.1 Range of Abstract Local Programs

We have seen that PURPLE-programs without any loop can be implemented easily by abstract local programs. The difficulty in showing that all PURPLE-programs can be implemented by abstract local programs lies in the treatment of `forall`-loops (recall that `while`-loops can be eliminated). In this section, we develop the main technical tools for the implementation of `forall`-loops by abstract local programs on certain action graphs.

We study how far the pebbles can be moved on a free action graph if we execute a given abstract local program an arbitrary number of times. We establish an upper bound on this distance – which we call the *range* of the program – in Theorem 10.

Definition 7 (Range)**.** Let G be a finite group and S be a set of generators that is closed under inverses. The *range* over G and S of abstract local programs with state set Q, pebble set P and local radius r is the smallest number $range_{G,S}(Q, P, r)$ such that, for all $f \in L_{G,S}(Q, P, r)$ and all $k \in \mathbb{N}$ there exists $f^k \in L_{G,S}(Q, P, range_{G,S}(Q, P, r))$ satisfying $[\![f]\!]_A^k = [\![f^k]\!]_A$ for any free action graph A with edge group G and generators S.

If A is an action graph with edge group G and generators S, then we will also write $range_A$ for $range_{G,S}$.

There always exists a number $range_{G,S}(Q, P, r)$, since the diameter of a free action graph A depends only on its edge group G and generators S and *any* function $h\colon Q \times A^P \to Q \times A^P$ can trivially be realised as $[\![g]\!]$ for some local program g whose local radius is the diameter of the graph A.

The following theorem gives a non-trivial upper bound on the range of abstract local programs.

Theorem 10. *There exists a constant c, such that for any group G with exponent m we have*

$$range_{G,S}(Q,P,r) \leq ((|S|+1)^r \cdot |Q| \cdot m)^{c^{|P|}} \quad .$$

We prove this theorem by reducing it to Theorem 3 with the help of the following lemma.

Lemma 11. *There is a constant a, such that each abstract local program $f \in L_{G,S}(Q,P,r)$ can be implemented by a* JAG *J with pebble set $P+P$ and state set $Q \times \big(\{1,\ldots,a\} \times \mathrm{B}_{C(G,S)}(e_G,r) \times \{1,\ldots,r\} \times \Sigma_{G,S}(P,r)\big)$ in the following sense: There exists q_0 such that for any free action graph A with edge group G and generators S and all $\rho,\tau \in A^{P+P}$ we have $[\![f]\!](q,\rho \circ \mathrm{inl}) = (p,\tau \circ \mathrm{inl})$ if and only if the run of J that starts with configuration $((q,q_0),\rho)$ ends in a configuration of the form $((p,p'),\tau)$.*

Proof. For each $f \in L_{G,S}(Q,P,r)$, we construct a machine J according to the pseudo-code below. The pebbles of J come from the set $P+P$. The elements of the left summand correspond to the pebbles of f, while the elements of the right summand are fresh pebbles that we are free to use. We therefore write simply x for $\mathrm{inl}(x)$ and we write $\overline{x}$ for $\mathrm{inr}(x)$, so that the pebble set of J is $\{x,\overline{x} \mid x \in P\}$. In the pseudo-code below, we furthermore use variables $q \in Q$, $\pi \in \mathrm{B}_{C(G,S)}(e_G,r)$, $n \in \{1,\ldots,r\}$ and $c \in \Sigma_{G,S}(P,r)$ to denote the finite state of J.

Save the pebble positions:
1. for all $x \in P$, jump pebble $\overline{x}$ to x.

Compute $c \in \Sigma_{G,S}(P,r)$:
2. set $c(x,y)$ to be undefined for all $x,y \in P$
3. for all $x \in P$ and $\pi \in \mathrm{B}_{C(G,S)}(e_G,r)$ do:
   - 3.1 move pebble x to $x \cdot \pi$
   - 3.2 set $c(x,y) := \pi$ for each $y \in P$ that lies on the same node as x
   - 3.3 jump pebble x to $\overline{x}$

Look up $f(q,c)$ in transition table, denote the components by (p,j,m).
Move the pebbles to their destination:
4. for all $x \in P$, move pebble x to $\overline{j(x)} \cdot m(x)$
5. set $q := p$ and halt

The statement 'move x to $y \cdot \pi$' is implemented by first jumping pebble x to the location of pebble y and then moving it step-by-step along the path π. Since π represents a path of length at most r, we can implement this step-by-step traversal by using the variable n as a counter.

It is clear that the pseudo-code can be encoded as the transition table of a JAG with state set $Q \times \big(\{1,\ldots,a\} \times \mathrm{B}_{C(G,S)}(e_G,r) \times \{1,\ldots,r\} \times \Sigma_{G,S}(P,r)\big)$, where a is chosen suitably, so that a program counter can be encoded as an element of $\{1,\ldots,a\}$. The other components of the state set are used to store the variables q, π, n and c described above. Then, q_0 can be chosen to be the internal state where the program counter points to the beginning of the program. □

Proof of Theorem 10. We can assume $|S| \geq 1$ and $m \geq 2$, since otherwise all free action graphs with edge group G and generators S have at most self-loops and the result is trivial. Similarly, we can assume that neither Q nor P is empty.

Let $f \in L_{G,S}(Q,P,r)$ and $k \in \mathbb{N}$. Consider the JAG J from Lemma 11 that simulates $[\![f]\!]$. It is easy to construct from J another JAG J' that repeats J eternally, i.e. J' works just like J until J stops, say with a configuration $((p,p'),\tau)$. Then J' moves to configuration $((p,q_0),\tau)$ and behaves like J again, etc. (q_0 is given by Lemma 11). Clearly, J' can be implemented with pebble set $P+P$ and state set $Q' := Q \times \big(\{1,\dots,b\} \times \mathrm{B}_{C(G,S)}(e_G,r) \times \{1,\dots,r\} \times \Sigma_{G,S}(P,r)\big)$, for some constant b.

By Theorem 3, there exists a constant c_0 such that J' can visit at most $s := (|Q'| \cdot m)^{c_0^{2|P|}}$ nodes from any starting configuration on any free action graph A with exponent m. In particular, from a starting configuration $((q,q_0),\rho)$, the machine J' can only reach configurations $((p,p'),\tau)$ with $\tau(P+P) \subseteq \mathrm{B}_A(\rho,s)$. Hence, the moves that J' makes can depend only on q and $[\rho]_s$.

We now define $f^k \in L_{G,S}(Q,P,s)$ such that $[\![f^k]\!] = [\![f]\!]^k$ holds. The definition of $f^k(q,c)$ is non-trivial only if there exist a free action graph A with edge group G and generators S and a configuration $(q,\rho) \in Q \times A^P$ with $[\rho]_s = c$; otherwise we can define $f^k(q,c)$ arbitrarily. Given A and (q,ρ) thus, let $\rho' \in A^{P+P}$ be the unique configuration satisfying $\rho' \circ \mathrm{inl} = \rho' \circ \mathrm{inr} = \rho$. Consider the run on J' with start configuration $((q,q_0),\rho')$. Since J always halts, this run contains infinitely many configurations whose state is a final state of J. Let $((p,p'),\tau')$ be the k-th configuration of this form. It appears after k complete executions of J and by construction of J and J' we have $[\![f]\!]^k(q,\rho) = [\![f]\!]^k(q,\rho' \circ \mathrm{inl}) = (p,\tau' \circ \mathrm{inl})$. Since s bounds the range of J', the moves of J' in this computation can be described by two functions $j\colon (P+P) \to P$ and $m\colon (P+P) \to \mathrm{B}_{C(G,S)}(e_G,s)$, such that $\tau'(x) = \rho(j(x)) \cdot m(x)$ holds for all x. We can define j with codomain P instead of $P+P$, since we have defined ρ' to satisfy $\rho' \circ \mathrm{inl} = \rho' \circ \mathrm{inr}$. Since the moves of J' depend only on q and $[\rho]_s$, the values p, j and m also depend only on q and $[\rho]_s$, in particular they neither depend on the choice of ρ nor that of A. We can therefore define $f^k(q,[\rho]_s) = (p, j \circ \mathrm{inl}, m \circ \mathrm{inl})$ and obtain $[\![f]\!]^k = [\![f^k]\!]$, as required.

This shows $\mathit{range}_{G,S}(Q,P,r) \le s$. To show the assertion of the theorem, it therefore just remains to show $s \le ((|S|+1)^r \cdot |Q| \cdot m)^{c^{|P|}}$ for some constant c.

$$
\begin{aligned}
s = (|Q'| \cdot m)^{c_0^{2|P|}} &= \big(|Q| \cdot b \cdot |\mathrm{B}_{C(G,S)}(e_G,r)| \cdot r \cdot |\Sigma_{G,S}(P,r)| \cdot m\big)^{c_0^{2|P|}} \\
&\le \Big(|Q| \cdot b \cdot (|S|+1)^r \cdot r \cdot ((|S|+1)^{2r}+1)^{|P|^2} \cdot m\Big)^{c_0^{2|P|}} \\
&\le \Big(|Q| \cdot (|S|+1)^{6r \cdot |P|^2} \cdot m^b\Big)^{c_0^{2|P|}} \\
&\le (|Q| \cdot (|S|+1)^r \cdot m)^{(6 \cdot b \cdot c_0)^{2|P|}}
\end{aligned}
$$

In this calculation we have used the assumptions $|S| \ge 1$ and $m \ge 2$ to obtain $r \le (|S|+1)^{r \cdot |P|^2}$ and $m \cdot b \le m^b$. By letting $c := (6 \cdot b \cdot c_0)^2$, we now obtain the result required to complete the proof. □

Definition 8 (Modulus)**.** Let G be a finite group and S be a set of generators that is closed under inverses. The *modulus* over G and S of abstract local programs with state set Q, pebble set P and local radius r is the least number $\mathit{modulus}_{G,S}(Q,P,r)$ such that, for all $f \in L_{G,S}(Q,P,r)$, all free action graphs A with edge group G and generators S and all pairs $(q,\rho) \in Q \times A^P$, the set $\{[\![f]\!]^k(q,\rho) \mid k \in \mathbb{N}\}$ has cardinality at most $\mathit{modulus}_{G,S}(Q,P,r)$.

Again, if A is an action graph with edge group G and generators S, we also write $\mathit{modulus}_A$ for $\mathit{modulus}_{G,S}$.

An upper bound on $\mathit{modulus}_{G,S}(Q,P,r)$ can be given using $\mathit{range}_{G,S}(Q,P,r)$, since we can bound the number of possible configurations that place all pebbles in a neighbourhood of radius $\mathit{range}_{G,S}(Q,P,r)$ around the start configuration.

$$\begin{aligned}\mathit{modulus}_{G,S}(Q,P,r) &\leq |Q| \cdot \max_{A=A(V,G,S,\cdot),\ \rho\in A^P} \left|\mathrm{B}_A(\rho, \mathit{range}_{G,S}(Q,P,r))^P\right| \\ &\leq |Q| \cdot \left(|P| \cdot |\mathrm{B}_{C(G,S)}(e_G, \mathit{range}_{G,S}(Q,P,r))|\right)^{|P|} \\ &\leq |Q| \cdot |P|^{|P|} \cdot (|S|+1)^{\mathit{range}_{G,S}(Q,P,r)\cdot|P|}\end{aligned} \tag{3}$$

4.4.2 Locality of Iteration

Our aim is to show that for each PURPLE-program there exists an abstract local program that implements (one run of) the PURPLE program. In this section we do this for the case of `forall`-loops.

Suppose we have a PURPLE program `forall` x `do` M and we have already compiled the loop body M into an abstract local program $f \in L_{G,S}(Q,P,r)$. We would then like to find a local program g that implements some run of the whole program `forall` x `do` M. To do this, it suffices to implement a `forall`-like iteration `forall` x `do` f of the abstract local program f. This is made precise in the following definition.

Definition 9. Let A be a free action graph. Let $f \in L_A(Q,P,r)$ be an abstract local program and $\boldsymbol{z} \in P$ be a pebble. We say that $g \in L_A(Q,P,l)$ *implements* `forall` $\boldsymbol{z}$ `do` f *on* A, if for all $(q,\rho) \in Q \times A^P$ there exists an enumeration $v_1,\ldots,v_n$ of all nodes in A, so that if we define a sequence of configurations $(q_0,\rho_0),\ldots,(q_n,\rho_n)$ by $(q_0,\rho_0) = (q,\rho)$ and $(q_{i+1},\rho_{i+1}) = [\![f]\!](q_i, \rho_i[\boldsymbol{z} := v_{i+1}])$, then we have $[\![g]\!](q,\rho) = (q_n,\rho_n)$.

In this section we identify conditions on action graphs under which it is always possible to find a local program g with small local radius that implements `forall` $\boldsymbol{z}$ `do` f. In the next section we will then show that iterated lamplighter graphs satisfy these conditions.

Throughout this section we let Q range over sets of states, P over sets of pebbles and r over local radii.

We first introduce some terminology.

Definition 10. A set of nodes U in a graph Γ is *r-sparse* if $\mathrm{B}_\Gamma(u,r) \cap \mathrm{B}_\Gamma(v,r) = \emptyset$ holds for all $u \neq v \in U$.

Definition 11. For any graph Γ and any set of nodes U, the *r-size of U* is the size of the largest r-sparse subset of U.

Definition 12. A *sightseer enumeration with radius r and memory n* is an enumeration $v_1, v_2, \ldots, v_k$ of the nodes in a graph such that for all i the set $\{v_j \mid i-n \leq j \leq i \text{ and } 1 \leq j \leq k\}$ is r-sparse.

The terminology in the last definition is motivated by analogy with a sightseer who wants to choose his route so that he does not see any place (node) twice. If his range of vision is r and he can see all nodes in an r-ball around his position then he would like to choose his route such that it forms an r-sparse

set. However, with memory n the sightseer cannot remember all the nodes he has already visited, but only the last n nodes he visited. Thus, he will be satisfied with an enumeration in which his position and the n previously visited nodes always form an r-spare set.

The main result of this section is Theorem 14, which gives conditions under which a `forall`-like iteration of an abstract local program can be implemented by a single abstract program. Moreover, this theorem provides a good enough bound on the local radius of the program that simulates the `forall`-loop.

To simulate a `forall`-like iteration by a local program, we construct an ordering of the graph nodes such that if we present the graph nodes to the `forall`-loop in this order, then at the end of the computation all pebbles will lie in a small enough neighbourhood around the initial pebble positions. Even though the `forall`-loop will have temporarily placed pebbles outside of this neighbourhood, this is not visible anymore in the final configuration and the move of the `forall`-loop from the start to the end configuration looks like the move of a local program that can only make moves within this neighbourhood. Thus we show that the `forall`-loop can be implemented by an abstract local program whose radius is large enough to contain this neighbourhood.

To realise this simple plan, we need an understanding of where the pebbles will lie at the end of a `forall`-like iteration. Clearly, the pebble moves that the program can make during a `forall`-like iteration are such that the final position of a pebble can be arrived at either (a) by moving along some path from the position of some pebble in the start configuration; or (b) by moving along some path from some of the nodes that are being jumped to in the `forall`-loop. This tautological statement becomes interesting when we can bound the length of the path that the pebbles are being moved along and if we know that the final pebble position depends only on certain select nodes of those that are being jumped to by the `forall`-loop. In fact, we shall see that it is not possible to for a pebble to cling to a node played in the middle of the iteration: the final positions of the pebbles will be contained in a small neighbourhood of the initial pebble positions and the first few and the final few jump destinations.

We begin with a combinatorial helper lemma that is the reason for this latter phenomenon.

Lemma 12. *Let $n, l \in \mathbb{N}$, $\boldsymbol{z} \in P$ and $f\colon P \to \{0, \dots, l\}$. Let $m_0, \dots, m_{n-1}$ be a list of endofunctions on P. Define a sequence $f_0, f_1, \dots$ of functions from P to $\mathbb{N}$ by $f_0 = f$ and $f_{i+1} = f_i[\boldsymbol{z} := l + i + 1] \circ m_{i \bmod n}$. Then for all $k \in \mathbb{N}$ and $y \in P$ we have $f_k(y) \leq l$ or $f_k(y) > l + k - |P| \cdot n$.*

Proof. Let $k \in \mathbb{N}$ and $y \in P$. We may assume $k > |P| \cdot n$, since the assertion is trivial otherwise.

Define a sequence $(y_i \in P)_{i \geq 0}$ by $y_0 = y$ and $y_{i+1} = m_{k-(i+1) \bmod n}(y_i)$. With this definition we have $f_k(y) = f_{k-i}(y_i)$ whenever we have $i \leq k$ and $\boldsymbol{z} \notin \{y_1, \dots, y_i\}$. To show the assertion, it therefore suffices to show that either $\boldsymbol{z}$ does not appear at all in $(y_i)_{i \geq 1}$, in which case we have $f_k(y) = f_0(y_k) = f(y_k) \in \{0, \dots, l\}$, or that the least $i \geq 1$ for which $y_i = \boldsymbol{z}$ holds satisfies $i \leq |P|n$, since in that case we have $f_k(y) = f_{k-(i-1)}(y_{i-1}) = l+k-(i-1) > l+k-i \geq l+k-|P|n$ by definition of $f_{k-(i-1)}$.

Since each y_i is in the finite set P, there exist i and j with $0 \leq i < j \leq |P|$ and $y_{i \cdot n+1} = y_{j \cdot n+1}$. It follows that $y_{i \cdot n+1+l} = y_{j \cdot n+1+l}$ holds for all l, by definition of the sequence $(y_i)_i$ and because $i \cdot n + 1 + l$ and $j \cdot n + 1 + l$ leave

the same remainder modulo n. This gives us $y_{i\cdot n+1+r} = y_{i\cdot n+1+r-(j-i)\cdot n}$ for all $r \geq (j-i)\cdot n$, which can be seen by substituting $r-(j-i)\cdot n$ for l in $y_{i\cdot n+1+l} = y_{j\cdot n+1+l}$. Repeated use of this fact then gives us $y_{i\cdot n+1+l} = y_{i\cdot n+1+(l \bmod (j-i)\cdot n)}$ for all l. As a result we obtain

$$\{y_1, y_2, \dots\} \subseteq \{y_1, \dots, y_{i\cdot n+1+(j-i)\cdot n-1}\} \subseteq \{y_1, \dots, y_{|P|\cdot n}\}.$$

Hence, if $\boldsymbol{z}$ appears at all in the sequence $(y_i)_{i\geq 1}$, then the least $i \geq 1$ such that $y_i = \boldsymbol{z}$ holds must satisfy $i \leq |P|n$. But we have observed above that this is sufficient to prove the lemma. □

The following lemma gives a characterisation of the final configuration for those `forall`-like iterations in which the nodes are presented such that they have a large enough distance from all pebbles at the time they are being jumped to.

Lemma 13. *Let $f \in L_{G,S}(Q, P, r)$ be an abstract local program, $\boldsymbol{z} \in P$ be a pebble and A be a free action graph with edge group G and generators S. Assume*

$$R \geq \mathit{range}_{G,S}\left(Q \times (|P|+2)^P, 2\times P+1, r\right),$$
$$M \geq \mathit{modulus}_{G,S}\left(Q \times (|P|+2)^P, 2\times P+1, r\right).$$

Then, for each $q \in Q$, each $c \in \Sigma_{G,S}(P, 2R)$ and each $k \in \mathbb{N}$, there exist a state $p \in Q$ and two functions $j\colon P \to (P+\mathbb{N})$ and $m\colon P \to \mathrm{B}_{C(G,S)}(e, R)$ with the following properties:

1. *For any list of nodes $v_1, \dots, v_k$ in A and all $\rho \in A^P$ with $[\rho]_{2R} = c$, if the sequence $(q_0, \rho_0), \dots, (q_k, \rho_k)$ defined by*

   $$(q_0, \rho_0) = (q, \rho), \qquad (q_{i+1}, \rho_{i+1}) = [\![f]\!]_A(q_i, \rho_i[\boldsymbol{z} := v_{i+1}])$$

   satisfies $\mathrm{B}_A(v_{i+1}, 3R) \cap (\mathrm{B}_A(\rho_i, 3R) \cup \mathrm{B}_A(\rho, 3R)) = \emptyset$ for all $i < k$, then we have $q_k = p$ and the environment ρ_k can be written as

   $$\rho_k(x) = \begin{cases} \rho(y)\cdot m(x) & \text{if } j(x) = \operatorname{inl} y, \\ v_i \cdot m(x) & \text{if } j(x) = \operatorname{inr} i. \end{cases}$$

2. *If $\operatorname{inr} i$ is in the image of j then we have $1 \leq i \leq M$ or $k - |P|\cdot M < i \leq k$.*

Proof. We show that the essence of a `forall`-like iteration of f can be captured without knowing precisely where $\boldsymbol{z}$ is placed by the `forall` loop, so long as $\boldsymbol{z}$ is always placed far enough away from the other pebbles. This means that the iteration is adequately simulated if we add $|P|+1$ copies of $C(G, S)$ to the graph and always place $\boldsymbol{z}$ on a pebble-free such copy instead of to the location chosen in the `forall`-loop. Such a simulation can then be implemented by an abstract local program operating and therefore be analysed using Theorem 10.

Thus, let B be the free action graph consisting of A and $|P|+1$ disjoint copies of the Cayley graph $C(G, S)$. We denote these copies of $C(G, S)$ in B by $C_1, \dots, C_{|P|+1}$ and write e_i for node e_G in C_i. For each $\rho \in B^P$, we define $I(\rho) \in (|P|+2)^P$ by $I(\rho)(y) = 0$ if $\rho(y) \in A$ and $I(\rho)(y) = i$ if $\rho(y) \in C_i$. For each function $I \in (|P|+2)^P$, we choose a number $\mathit{fresh}(I) \in \{1, \dots, |P|+1\}$ that is not in the image of I.

With these definitions in place we now define a function b, that captures the essence of `forall`-like iteration of f.

$$b\colon Q \times B^P \to Q \times B^P$$
$$b(q,\rho) = [\![f]\!]_B(q, \rho[\boldsymbol{z} := e_{\mathit{fresh}(I(\rho))}])$$

Now, given $q \in Q$ and $c \in \Sigma_{G,S}(P, 2R)$ and $k \in \mathbb{N}$ pick $\rho^c \in A^P$ with $[\rho^c]_{2R} = c$ (if none exists the assertion of the Lemma is trivial and it is easy to choose p, j and m appropriately).

Define a sequence of configurations $(q_0^c, \rho_0^c), \ldots, (q_k^c, \rho_k^c) \in Q \times B^P$ by

$$(q_0^c, \rho_0^c) = (q, \rho^c), \qquad (q_{i+1}^c, \rho_{i+1}^c) = b(q_i^c, \rho_i^c).$$

The pebble jumps in each step of this sequence are captured by the functions $j_0, \ldots, j_{k-1} \in P^P$ defined by $(-, j_i, -) = f(q_i^c, [\rho_i^c[\boldsymbol{z} := e_{\mathit{fresh}(I(\rho_i^c))}]]_r)$. Using these functions, we further define a sequence $l_0, \ldots, l_k \in \mathbb{N}^P$ by

$$l_0(y) = 0, \qquad l_{i+1} = (l_i[\boldsymbol{z} := i+1]) \circ j_i.$$

Intuitively, for pebble x the value $l_i(x)$ tells whether at time i pebble x is close to one of the original pebbles (case $l_i(x) = 0$) or close to one of the jump destinations, to wit the $l_i(x)$-th one when $l_i(x) \neq 0$.

We now define the required state $p \in Q$ by $p := q_k^c$ and choose functions $j\colon P \to P + \mathbb{N}$ and $m\colon P \to \mathrm{B}_{C(G,S)}(e_G, R)$ so as to satisfy

- if $l_k(x) > 0$ and $\rho_k^c(x) \in B \setminus A$ then $j(x) = \mathrm{inr}\ l_k(x)$ and $m(x) = \rho_k^c(x)$ (note that nodes in $B \setminus A$ are just group elements); and
- if $l_k(x) = 0$ and $\rho_k^c(x) \in \mathrm{B}_A(\rho^c, R)$ then $j(x) = \mathrm{inl}\, y$ and $m(x) = \rho_k^c(x)/\rho_0^c(y)$ for some y with $\rho_k^c(x) \in \mathrm{B}_A(\rho^c(y), R)$.

To show that such a choice is always possible, we next show that, for any $x \in P$, the precondition of one of these two points must be satisfied.

To analyse the above sequences defined using b, we implement b by an abstract local program

$$g \in L_{G,S}\left(Q \times (|P|+2)^P, P + \{x_1, \ldots, x_{|P|+1}\}, r\right)$$

in the following sense: For all $(q,\rho) \in Q \times B^P$ we have that $b(q,\rho) = (p,\tau)$ implies $[\![g]\!]((q, I(\rho)), \rho[\vec{x} := \vec{e}]) = ((p, I(\tau)), \tau[\vec{x} := \vec{e}])$, where we write $\rho[\vec{x} := \vec{e}]$ for the environment obtained from ρ by placing the $|P|+1$ additional pebbles $\vec{x}$ on the nodes $\vec{e} = e_1, \ldots, e_{|P|+1}$. We omit the technical details of the straightforward definition of g.

Using this implementation of b, we can show well-definedness of j and m. First, it is straightforward to show by induction on i that $l_i(x) = 0$ implies $\rho_i^c(x) \in A$, and $l_i(x) > 0$ implies $\rho_i^c(x) \in B \setminus A$. Second, since g implements b and, by definition, R is an upper bound on the range of g, we have

$$\rho_i^c(P) \subseteq \mathrm{B}_B(\rho_0^c(P) \cup \{e_1, \ldots, e_{|P|+1}\}, R)$$

for all $i \leq k$. From these two facts we get that for any $x \in P$ we either have $l_k(x) > 0$ and $\rho_k^c(x) \in B \setminus A$ or $l_k(x) = 0$ and $\rho_k^c(x) \in \mathrm{B}_A(\rho^c, R)$, i.e. one of the two cases in the above definition of j and m always applies. In the first case

we moreover have $\rho_k^c(x) \in \mathrm{B}_{C(G,S)}(e_n, R)$ for some n, which shows that m does indeed map to $\mathrm{B}_{C(G,S)}(e_G, R)$.

It now remains to show that p, j and m have the required properties. To this end let $\rho \in A^P$ and $v_1, \ldots, v_k$ be given as in the lemma and consider the ensuing sequence $(q_0, \rho_0), \ldots, (q_k, \rho_k)$ by

$$(q_0, \rho_0) = (q, \rho), \qquad (q_{i+1}, \rho_{i+1}) = [\![f]\!]_A(q_i, \rho_i[\boldsymbol{z} := v_{i+1}]),$$

as in the statement of the lemma.

We use the implementation of b by g to relate (q_i, ρ_i) to (q_i^c, ρ_i^c). For all $i \leq k$ we prove the following four facts by induction on i.

(a) $q_i^c = q_i$;

(b) if $0 < l_i(x) < l_i(x')$ then $d(v_{l_i(x)}, v_{l_i(x')}) > 5R$ and $d(\rho_i^c(x), \rho_i^c(x')) = \infty$;

(c) if $l_i(x) = 0$ then $\rho_i(x) = \rho(y) \cdot \rho_i^c(x)/\rho^c(y)$ whenever $d(\rho_i^c(x), \rho^c(y)) \leq R$;

(d) if $l_i(x) > 0$ then $\rho_i(x) = v_{l_i(x)} \cdot \rho_i^c(x)$.

The base case $i = 0$ is clear from the assumptions.

Assume now that facts (a), (b), (c) and (d) hold for a fixed $i < k$.

We claim that $[\rho_i]_R = [\rho_i^c]_R$. To see this, suppose first $d(\rho_i(x), \rho_i(x')) \leq 2R$. In this case we must have $l_i(x) = l_i(x')$. Suppose this were not the case and without loss of generality $l_i(x) < l_i(x')$. If $l_i(x) = 0$ then by the above analysis of g there exists a y with $d(\rho_i^c(x), \rho^c(y)) \leq R$. The induction hypothesis then yields $d(\rho_i(x), \rho(y)) \leq R$ by item (c) and $d(\rho_i(x'), v_{l_i(x')}) \leq R$ by item (d). This implies $d(\rho(y), v_{l_i(x')}) \leq 4R$, contradicting the assumption $d(\rho(y), v_{l_i(x')}) > 6R$. If $l_i(x) > 0$ then items (b) and (d) yield a contradiction. We conclude that $l_i(x) = l_i(x')$.

Now, if $l_i(x) = l_i(x') = 0$ then we can pick y and y' with $d(\rho_i^c(x), \rho^c(y)) \leq R$ and $d(\rho_i^c(x'), \rho^c(y')) \leq R$. With item (c) we get $d(\rho(y), \rho(y')) \leq 4R$. With the assumption $[\rho]_{2R} = [\rho^c]_{2R}$, this implies $\rho^c(y)/\rho^c(y') = \rho(y)/\rho(y')$. We obtain $\rho_i^c(x)/\rho_i^c(x') = \rho_i(x)/\rho_i(x')$. If $l_i(x) = l_i(x') > 0$ then the equality $\rho_i^c(x)/\rho_i^c(x') = \rho_i(x)/\rho_i(x')$ follows directly from item (d).

A similar (and in fact easier) argument shows that $d(\rho^c(x), \rho^c(x')) \leq 2R$ implies $\rho_i^c(x)/\rho_i^c(x') = \rho_i(x)/\rho_i(x')$.

We have thus shown $[\rho_i]_R = [\rho_i^c]_R$ and in particular $[\rho_i]_r = [\rho_i^c]_r$.

From this we obtain

$$[\rho_i[\boldsymbol{z} := v_{i+1}]]_r = [\rho_i^c[\boldsymbol{z} := e_{\mathit{fresh}(I(\rho_i^c))}]]_r \ ,$$

so that we have

$$\begin{aligned} q_{i+1} &= q' \\ q_{i+1}^c &= q' \\ \rho_{i+1}(x) &= \rho_i[\boldsymbol{z} := v_{i+1}](j'(x)) \cdot m'(x) \\ \rho_{i+1}^c(x) &= \rho_i^c[\boldsymbol{z} := e_{\mathit{fresh}(I(\rho_i^c))}](j'(x)) \cdot m'(x) \ , \end{aligned}$$

where

$$(q', j', m') = f(q_i, [\rho_i[\boldsymbol{z} := v_{i+1}]]_r).$$

Moreover, j' equals the function j_i featuring in the definition of l_i.

We immediately get item (a) for index $i+1$.

For item (b), assume $0 < l_{i+1}(x) < l_{i+1}(x')$. We consider two cases. First, if $l_{i+1}(x') = l_i(j'(x'))$ then we must have $l_{i+1}(x) = l_i(j'(x))$, too, by $l_{i+1}(x) < l_{i+1}(x')$, and the claim follows directly from the induction hypothesis. Second, if $l_{i+1}(x') = i+1$, then, again, we have $l_{i+1}(x) = l_i(j'(x))$ and by item (d) of the induction hypothesis also $d(\rho_i(j'(x)), v_{l_{i+1}(x)}) \leq R$. We have $d(\rho_i(j'(x)), v_{l_{i+1}(x')}) > 6R$ by assumption, so that the first claim $d(v_{l_{i+1}(x)}, v_{l_{i+1}(x')}) > 5R$ follows by triangle inequality. The other claim is immediate by the choice of *fresh*.

For item (c) suppose $l_{i+1}(x) = 0$. From the definition of l_{i+1} it then follows that $j_i(x) \neq \boldsymbol{z}$ and $l_i(j_i(x)) = 0$. We then have

$$\begin{aligned}\rho_{i+1}(x) &= \rho_i(j_i(x)) \cdot m'(x)\\ \rho^c_{i+1}(x) &= \rho^c_i(j_i(x)) \cdot m'(x)\ .\end{aligned}$$

As before, let y be a pebble such that $\rho^c_{i+1}(x)/\rho^c(y)$ is defined and contained in $\mathrm{B}_{C(G,S)}(e,R)$. Likewise, pick a pebble y' such that $\rho^c_i(j_i(x))/\rho^c(y')$ is defined and contained in $\mathrm{B}_{C(G,S)}(e,R)$.

With $\rho^c_{i+1}(x) = \rho^c_i(j_i(x)) \cdot m'(x)$ it follows that the distance of $\rho^c(y)$ and $\rho^c(y')$ is at most $2R+r$, so that this distance must be recorded in $[\rho_c]_{2R}$. The induction hypothesis combined with the assumption $[\rho]_{2R} = [\rho^c]_{2R}$ then yields the result.

For item (d) assume $l_{i+1}(x) > 0$. Then either $j_i(x) = \boldsymbol{z}$ and the claim is obvious or else $j_i(x) \neq \boldsymbol{z}$ and $l_i(j_i(x)) = l_{i+1}(x) > 0$. We then have $\rho^c_{i+1}(x) \in \mathrm{B}_{C(G,S)}(e,R)$ and we are done.

The assertion 1. of the Lemma is now a direct consequence of the case $i = k$. For assertion 2. we argue as follows.

The set $\{(q_0, \rho^c_0), \ldots, (q_k, \rho^c_k)\}$ has cardinality at most M, since M bounds the modulus of g by definition and g implements b.

Hence, there exists $n \leq M$ such that we have $j_{M+i} = j_{M+(i \bmod n)}$ for all $i \leq k - M$. Moreover, the image of l_M is contained in $\{0, \ldots, M\}$. Lemma 12 therefore tells us that we have $l_k(y) \leq M$ or $l_k(y) > M + (k-M) - |P| \cdot n$ (note that we take $f = l_M$ in Lemma 12, so that we have $l_k = f_{k-M}$). We then calculate $l_k(y) > M + (k-M) - |P| \cdot n = k - |P| \cdot n \geq k - |P| \cdot M$ and thus obtain the bounds required in the statement of the lemma. □

Theorem 14. *Let $f \in L_{G,S}(Q,P,r)$ be an abstract local program, $\boldsymbol{z} \in P$ be a pebble and A be a free action graph with edge group G and generators S. Assume*

$$\begin{aligned}R &\geq 3 \cdot \mathit{range}_{G,S}\left(Q \times (|P|+2)^P, 2 \times P + 1, r\right),\\ M &\geq \mathit{modulus}_{G,S}\left(Q \times (|P|+2)^P, 2 \times P + 1, r\right)\end{aligned}$$

and further that $u \geq 3R$ is a natural number such that the following two conditions hold:

1. *A has a sightseer enumeration with radius u and memory $|P| \cdot (M+1)$.*
2. *For any $v \in A$ the $3R$-size of $\mathrm{B}_A(v, u-3R)$ is at least $|P| \cdot (M+1)$.*

Then there exists an abstract local program

$$g \in L_{G,S}\left(Q, P, u + r \cdot |P| \cdot |\mathrm{B}_{C(G,S)}(e_G, u)|\right)$$

that implements `forall` $\boldsymbol{z}$ `do` f *on A.*

Proof. Let C be the Cayley graph $C(G,S)$. Let $l := u + r \cdot |P| \cdot |\mathrm{B}_C(e_G, u)|$. We need to construct an abstract local program

$$g\colon Q \times \Sigma_{G,S}(P,l) \longrightarrow Q \times P^P \times \mathrm{B}_C(e_G,l)^P.$$

Given $q \in Q$ and $c \in \Sigma_{G,S}(P,l)$ we choose $\rho^c \in A^P$ with $[\rho^c]_l = c$ (if none exists then define $g(q,c)$ arbitrarily) and define $k = |A \setminus \mathrm{B}_A(\rho^c, u)| + (|P|+1)M$.

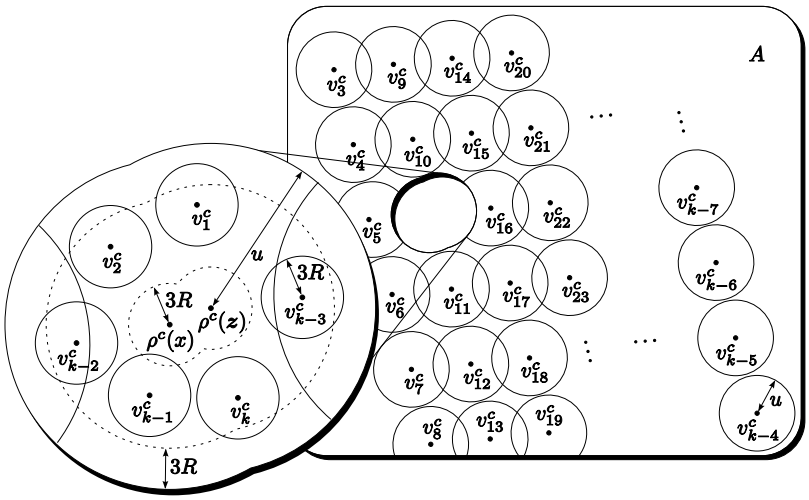
We define an enumeration $v_1^c, \ldots, v_n^c$ of the nodes in A as follows:

- $v_1^c, \ldots, v_M^c, v_{k-|P|\cdot M+1}^c, \ldots, v_k^c$ form a $3R$-sparse subset of $\mathrm{B}_A(\rho^c, u-3R) \setminus \mathrm{B}_A(\rho^c, 3R)$ of size $|P| \cdot M$. Such a set can be obtained from a $3R$-sparse subset of $\mathrm{B}_A(\rho^c, u-3R)$ of size $|P|(M+1)$ by deleting all members contained in $\mathrm{B}_A(\rho^c, 3R)$, of which there are at most $|P|$.

- $v_{M+1}^c, \ldots, v_{k-|P|\cdot M}^c$ form a sightseer enumeration of $A \setminus \mathrm{B}_A(\rho^c, u)$ with radius u and memory $|P| \cdot M$. To construct one such we start with a sightseer enumeration $u_1, u_2, \ldots, u_n$ of A with radius u and memory $|P| \cdot (M+1)$ and delete all nodes in $\mathrm{B}_A(\rho^c, u)$ from that sequence. If u_i and u_j become $|P| \cdot M$ steps close to each other after the deletion then $j - i \leq |P|(M+1)$ because the sequence $u_i, \ldots, u_{i+|P|(M+1)}$ contains at most $|P|$ elements from $\mathrm{B}_A(\rho^c, u)$ and therefore must contain u_j. It follows that the sequence after the deletion is a sightseer enumeration with radius u and memory $|P| \cdot M$.

- $v_{k+1}^c, \ldots, v_n^c$, finally, are an arbitrary enumeration of the remaining nodes, i.e. those in $\mathrm{B}_A(\rho^c, u) \setminus \{v_1^c, \ldots, v_M^c, v_{k-|P|\cdot M+1}^c, \ldots, v_k^c\}$.

The figure below illustrates the choice of the sequence $v_1^c, \ldots, v_k^c$ for the case where $P = \{x, \boldsymbol{z}\}$ and $M = 2$.



Consider now the sequence $(q, \rho^c) = (q_0^c, \rho_0^c), (q_1^c, \rho_1^c), \ldots, (q_n^c, \rho_n^c)$ defined by $(q_{i+1}^c, \rho_{i+1}^c) = [\![f]\!](q_i^c, \rho_i^c[\boldsymbol{z} := v_{i+1}^c])$.

Now, we pick $j \in P^P$ and $m \in \mathrm{B}_{C(G,S)}(l, e_G)^P$ in such a way that $\rho_n^c(x) = \rho^c(j(x)) \cdot m(x)$ holds for all $x \in P$ and put $g(q,c) = (q_n^c, j, m)$. If such j and m do not exist we take an arbitrary default value instead. This concludes the definition of the announced abstract local program g.

We will now show that g has the claimed properties and that in particular j and m as above always exist. So let $q \in Q$ and $c \in \Sigma_{G,S}(P, l)$ be given and let $\rho \in A^P$ satisfy $[\rho]_l = c$. Let $\rho^c \in A^P$ be the configuration with $[\rho^c]_l = c$ that is chosen in the definition of $g(q, c)$, let $v_1^c, \ldots, v_n^c$ be the corresponding enumeration of A and let $(q_i^c, \rho_i^c)_{i=0,\ldots,n}$ be the computation used there.

Let $\varphi : \mathrm{B}_A(\rho^c, l) \to \mathrm{B}_A(\rho, l)$ be a bijection that satisfies $\varphi(\rho^c(x)) = \rho(x)$ for all $x \in P$ and that preserves the local graph structure. Such a bijection exists by Lemma 6.

We have

$$k = |A \setminus \mathrm{B}_A(\rho^c, u)| + (|P| + 1)M = |A \setminus \mathrm{B}_A(\rho, u)| + (|P| + 1)M.$$

We define an enumeration $(v_i)_i$ of A as follows:

- For $i = 1, \ldots, M, k - |P| \cdot M + 1, \ldots, n$ we put $v_i = \varphi(v_i^c)$.
- We choose the sequence $v_{M+1}, \ldots, v_{k-|P|\cdot M}$ to form a sightseer enumeration of $A \setminus \mathrm{B}_A(\rho, u)$ with radius u and memory $|P| \cdot M$.

Consider the corresponding computation sequence $(q_0, \rho_0), (q_1, \rho_1), \ldots, (q_n, \rho_n)$ defined by $(q_0, \rho_0) = (q, \rho)$ and $(q_{i+1}, \rho_{i+1}) = [\![f]\!](q_i, \rho_i[\boldsymbol{z} := v_{i+1}])$.

Clearly, we have $\mathrm{B}_A(\rho^c, R) \cap \mathrm{B}_A(v_i^c, R) = \emptyset$ and $\mathrm{B}_A(\rho, R) \cap \mathrm{B}_A(v_i, R) = \emptyset$ for all $i \in \{1, \ldots, k\}$.

We show now using Lemma 13 that both $\mathrm{B}_A(\rho_i^c, R) \cap \mathrm{B}_A(v_{i+1}^c, R) = \emptyset$ and $\mathrm{B}_A(\rho_i, R) \cap \mathrm{B}_A(v_{i+1}, R) = \emptyset$ hold for all $i < k$. The argument goes by induction on i, the base case being trivial. For the induction step, Lemma 13 implies that in the configuration ρ_i^c, the pebbles must lie in an R-neighbourhood around the nodes $V = \rho \cup \{v_1^c, \ldots, v_M^c\} \cup \{v_{i-|P|\cdot M+1}^c, \ldots, v_i^c\}$. Now, the construction of the sequence $v_1^c, \ldots, v_k^c$ is such that the u-balls around v_{i+1}^c and any $v \in V$ are disjoint. Hence, the distance of v_{i+1}^c and v is more than $3R$. But this implies that v_{i+1}^c has distance larger than $2R$ from any node in $\mathrm{B}_A(V, R)$. Hence we have shown $\mathrm{B}_A(\rho_i^c, R) \cap \mathrm{B}_A(v_{i+1}^c, R) = \emptyset$. The argument for ρ_i is analogous.

Lemma 13 now implies $q_i^c = q_i$ for all $i \leq k$ and furnishes two functions $j_0 \colon P \to P + \mathbb{N}$ and $m_0 \colon P \to \mathrm{B}_C(e_G, R)$ that satisfy

$$\rho_k^c(x) = \begin{cases} \rho^c(y) \cdot m_0(x) & \text{if } j_0(x) = \mathrm{inl}\, y \\ v_i^c \cdot m_0(x) & \text{if } j_0(x) = \mathrm{inr}\, i \end{cases} \tag{4}$$

$$\rho_k(x) = \begin{cases} \rho(y) \cdot m_0(x) & \text{if } j_0(x) = \mathrm{inl}\, y \\ v_i \cdot m_0(x) & \text{if } j_0(x) = \mathrm{inr}\, i \end{cases} . \tag{5}$$

By item 2 of Lemma 13, we know that $j_0(x) = \mathrm{inr}\, i$ implies $i \leq M$ or $i > k - |P| \cdot M$. By the choice of the enumerations of A, this further implies $v_i^c \in \mathrm{B}_A(\rho^c, u - 3R)$ and $v_i \in \mathrm{B}_A(\rho, u - 3R)$ and therefore $v_i = \varphi(v_i^c)$.

From this we get $\rho_k(x) = \varphi(\rho_k^c(x))$ by means of equations (4) and (5) and the fact that φ preserves the local graph structure. Note, though, that $\rho_i(x) = \varphi(\rho_i^c(x))$ will in general not hold for $M < i \leq k - |P| \cdot M$.

We also note $\rho_k^c \subseteq \mathrm{B}_A(\rho^c, u - 2R)$ and likewise for ρ_k.

Now, in each step from (q_k^c, ρ_k^c) to (q_n^c, ρ_n^c) we have first placed pebble $\boldsymbol{z}$ on some node in $\mathrm{B}_A(\rho^c, u)$ and then applied the function $[\![f]\!]$. Since f has local radius r and all assignments of $\boldsymbol{z}$ went to $\mathrm{B}_A(\rho^c, u)$, we have $\rho_{k+i}^c \subseteq$

$\mathrm{B}_A(\rho^c, u + i \cdot r)$ for all $i \leq n - k$. We have $n - k \leq |\mathrm{B}_A(\rho^c, u)|$, since all nodes in $v^c_{k+1}, \ldots, v^c_n$ are taken from $\mathrm{B}_A(\rho^c, u)$, so that

$$\begin{aligned}\rho^c_n &\subseteq \mathrm{B}_A(\rho^c, u + |\mathrm{B}_A(\rho^c, u)| \cdot r) \\ &\subseteq \mathrm{B}_A(\rho^c, l). \qquad \text{since } |\mathrm{B}_A(\rho^c, u)| \leq |P| \cdot |\mathrm{B}_C(e_G, u)|\end{aligned}$$

This implies that if $g(q, c) = (q^c_n, j, m)$ then indeed $\rho^c_n(x) = \rho^c(j(x)) \cdot m(x)$, i.e. the case of arbitrary default values for j and m does not occur as promised. Furthermore, since $q_k = q^c_k$ and $\rho_k = \varphi \circ \rho^c_k$ and since the entire computation from time k to n remains within an l-ball around ρ^c we obtain $q_n = q^c_n$ and $\rho_n = \varphi \circ \rho^c_n$ and thus $\rho_n(x) = \rho(j(x)) \cdot m(x)$, as required. □

4.5 Iterated Lamplighter Graphs

In Theorem 14 we have given conditions for action graphs under which `forall`-like iterations of abstract local programs can be implemented by abstract local programs. In this section we show that, for large enough i, the iterated lamplighter graph $\Lambda^i(m)$ enjoys the required properties.

The conditions on the free action graph in Theorem 14 pertain the existence of a certain sightseer enumeration and the minimum size of certain neighbourhoods around the pebble positions in configurations.

To construct a sightseer enumeration with radius r and memory n in an iterated lamplighter graph, we make use of the simple observation that the nodes of iterated lamplighter graphs are binary tuples and the Hamming distance of such tuples is a lower bound on their distance as graph nodes. We therefore construct an enumeration of binary words of a certain length in which any n consecutive elements have pairwise Hamming distance r.

For $k \in \mathbb{N}$ denote by B_k the binary vectors (bitvectors) of length k. They form a k-dimensional $\mathrm{GF}(2)$ vector space with addition (and subtraction!) being pointwise exclusive-or.

For $x, y \in B_k$ the *Hamming distance* $d_H(x, y)$ is defined as the number of positions at which they are different. Hamming distance satisfies the triangle inequality

$$d_H(x, y) \leq d_H(x, z) + d_H(z, y) \tag{6}$$

or, equivalently, $d_H(x, z) \geq d_H(x, y) - d_H(z, y)$. Furthermore, we have

$$d_H(x, y) = d_H(x + z, y + z) \tag{7}$$

because addition of z toggles the same bits. Since $x + y = x - y$ in B_k we also have that

$$d_H(x + u, y + v) = d_H(x, y + u + v).$$

Proposition 15. *If $k \geq nr$ and $n \geq 12$ then there exists an enumeration $(v_i)_i$ of B_k such that $(i - j) \bmod 2^k \leq n$ implies $d_H(v_i, v_j) \geq r$.*

Proof. We may assume without loss of generality that $k - nr < r$, for otherwise we can enlarge n and get an even stronger statement. For $i = 1, \ldots, n$ let

$$e_i = 0^{(i-1)r} 1^r 0^{k-ir}$$

and put $S = \mathrm{span}(e_1, \ldots, e_n)$. We have $|S| = 2^n$ and for any u, v in S we have $d_H(u, v) \geq r$. Note that S consists of bitvectors which when read from left to right toggle only at positions $ri + 1$ for $i = 0, \ldots, n - 1$.

Now choose vectors $v_1, \ldots, v_{2^{k-n}}$ such that each vector $x \in B_k$ can be uniquely written as $x = u + v_i$ for some $u \in S$ and $1 \leq i \leq 2^{k-n}$. This can be done by choosing representatives for the 2^{k-n} classes of the equivalence relation $u \sim v \iff u - v \in S$.

Notice that the Hamming distance of two distinct vectors in $S + v_i$ for some i is at least r. This then suggests to enumerate first the vectors in $S + v_1$ then the vectors in $S + v_2$ and so forth in such a way that the last n vectors of the previous block are sufficiently distant from the first n vectors of the next block (previous/next understood modulo 2^k).

Let us call a vector $v \in S$ *lean* if it can be written as the sum of at most 2 basis vectors $e_1, \ldots, e_n$. The number of lean vectors exceeds $2n$. Conversely, call a vector $v \in S$ *fat* if it is a sum of $n-1$ distinct basis vectors. There are n fat vectors. The Hamming distance between a lean and a fat vector is at least $r(n-3)$.

We will now define for each i an enumeration of $S + v_i$ which is such that (a) its first n vectors have distance at least r from any vector of the form $u + v_{i-1 \bmod 2^k}$ with u lean and (b) its last n vectors are of the form $u + v_i$ with u lean. It is clear that we then obtain an enumeration of B_k with the desired properties by concatenating the enumerations of the blocks $S + v_i$.

To get such an enumeration of $S + v_i$ put $v = v_i + v_{i-1 \bmod 2^k}$ and choose $s \in S$ so that $d_H(s, v) \leq nr/2 + 2r$. If v has more 0s than 1s then $s = 0$ works, otherwise we put $s = 1^{nr}0^{k-nr}$. Recall that $k - nr < r$. Now choose $t_1, \ldots, t_n$ in S, so that $t_j + s$ is fat for $j = 1, \ldots, n$. We take the n vectors $t_1 + v_i, \ldots, t_n + v_i$ as the first n vectors of the enumeration of the block $S + v_i$. If u is lean then

$$
\begin{aligned}
& d_H(u + v_{i-1 \bmod 2^k}, t_j + v_i) \\
&= d_H(u, t_j + v) && \text{def. of } v \text{ and (7)} \\
&\geq d_H(u, t_j + s) - d_H(t_j + s, t_j + v) && \text{triangle inequality} \\
&= d_H(u, t_j + s) - d_H(s, v) && \text{eqn. (7)} \\
&\geq r(n-3) - nr/2 - 2r && \text{lean vs. fat; constr. of } s \\
&\geq r && \text{since } n \geq 12 \ .
\end{aligned}
$$

Thus, requirement (a) is satisfied. Since at most n of the vectors $t_1, \ldots, t_n$ are lean we can find another n lean vectors $u_1, \ldots, u_n \in S$ allowing us to put $u_1 + v_i, \ldots, u_n + v_i$ at the end of the enumeration so as to satisfy requirement (b). The enumeration of the remaining vectors in the middle can be performed in an arbitrary order. □

Lemma 16. *The lamplighter graph $\Lambda(G, S)$ has a sightseer enumeration of radius r and memory n whenever $|G| \geq (2r+1) \cdot \max(n, 12)$ holds.*

Proof. Nodes of $\Lambda(G, S)$ are pairs (f, x) with $f \in 2^{|G|}$ and $x \in G$. We view such f as an element of $B_{|G|}$. The distance in $\Lambda(G, S)$ of two nodes (f, x) and (f', x') is then at least the Hamming distance $d_H(f, f')$ of f and f'. Under the given assumptions Proposition 15 furnishes an enumeration $f_1, \ldots, f_{2^{|G|}}$ of $2^{|G|}$ such that any two vectors who are at most n steps apart modulo $2^{|G|}$ have Hamming distance at least $2r+1$. We can then choose an arbitrary enumeration $(x_j)_j$ of G and enumerate $\Lambda(G, S)$ as $\left(f_{(k \bmod 2^{|G|})+1}, x_{\lceil k/2^{|G|} \rceil}\right)_{k=1,\ldots,2^{|G|} \cdot |G|}$. □

The next corollary follows by equation (2).

Corollary 17. *If* $\exp_2^{i-1}(m) \geq (2r+1) \cdot \max(n, 12)$ *then* $\Lambda^i(m)$ *has a sightseer enumeration with radius* r *and memory* n*.*

Next we analyse the size of neighbourhoods in lamplighter graphs, as required to satisfy the second precondition of Theorem 14.

Lemma 18. *Let* n *and* r *be positive natural numbers. A ball* $\mathrm{B}_{\Lambda(G,S)}(v, l)$ *in a lamplighter graph* $\Lambda(G, S)$ *has* r*-size at least* n *if from any node in the Cayley graph* $C(G, S)$ *there is a cycle-free path of length* $r \cdot n$ *and if* $l \geq (n+1)r$ *holds.*

Proof. By definition, v is a pair (f, x) with $f \in 2^{|G|}$ and $x \in G$. Let $x_1, \ldots, x_{rn}$ be a cycle-free path in $C(G, S)$ with $x_1 = x$. Define $f_i, d_{i,j} \in 2^{|G|}$ for $i = 0, \ldots, n-1$ and $j = 1, \ldots, r$ by

$$f_i = f + d_{i,1} \ , \qquad\qquad d_{i,j} = \sum_{l=ir+j}^{ir+r} \delta_{x_l} \ .$$

We claim that the set $\{(f_i, x_{ir+r}) \mid 0 \leq i < n\}$ consists of n nodes in $\mathrm{B}_{\Lambda(G,S)}(v, l)$ having pairwise distance greater than $2r$. That their pairwise distance is greater than $2r$ follows because for $i \neq j$ we have constructed f_i and f_j to have Hamming distance $2r$ and x_{ir+r} and x_{jr+r} have distance at least one. It remains to show that these nodes are indeed in $\mathrm{B}_{\Lambda(G,S)}(v, l)$.

Define $b_{i,j} = (x_{ir+j}^{-1} \cdot d_{i,j},\ x_{ir+j}^{-1} \cdot x_{ir+r})$ and observe

$$(f_i, x_{ir+r}) = (f, x_1) \cdot (0,\ x_1^{-1} \cdot x_{ir+1}) \cdot b_{i,1} \ . \tag{8}$$

Now we note that $(0,\ x_j^{-1} \cdot x_{j+k})$ has distance at most k from $e_{L(G,S)}$. Next we show that $b_{i,j}$ has distance at most $2(r-j)+1$ from $e_{L(G,S)}$. This follows because we have $d(e_{L(G,S)}, b_{i,r}) = 1$ and $d(e_{L(G,S)}, b_{i,j}) \leq 2 + d(e_{L(G,S)}, b_{i,j+1})$ for all $j \in \{1, \ldots, r-1\}$. The first fact can be seen by observing $b_{i,r} = (\delta_e, e_G)$, while the second one follows from $b_{i,j} = (\delta_e, e_G) \cdot (0, x_{ir+j}^{-1} \cdot x_{ir+j+1}) \cdot b_{i,j+1}$.

But with (8) we have thus shown that the distance of (f_i, x_{ir+r}) from $v = (f, x_1)$ is no more than $ir + 2(r-1) + 1 < (i+2)r \leq (n+1)r = l$, as required. □

With the lemma, we now have a lower bound on the r-size of balls in lamplighter graphs. The lemma is preconditioned on the existence of long cycle-free paths in $C(G, S)$. Since we will work with iterated lamplighter graphs, we will be interested in the case where $C(G, S)$ is a lamplighter graph itself. In order to apply the above lemma in this situation, we need a lower bound on the length of cycle-free paths in lamplighter graphs.

Lemma 19. *In the lamplighter graph* $\Lambda(G, S)$*, there exists from any node a cycle-free path of length* $2^{|G|}$*.*

Proof. Any vertex of $\Lambda(G, S)$ is a pair of a binary word f of length $|G|$ and an element x of G. We construct a path of length $2^{|G|}$ starting from an arbitrary vertex (f, x). There exists a Gray-code enumeration $f = f_1, \ldots, f_{2^{|G|}}$ of the binary words of length $|G|$, i.e. an enumeration in which subsequent words differ in exactly one position, see e.g. [6]. Write $x_k \in G$ for the position in which f_k and f_{k+1} differ. Clearly, there exists cycle-free path π_k from (f_{k+1}, x_k) to (f_{k+1}, x_{k+1}) for all $k \in \{1, \ldots, 2^{|G|} - 2\}$, because there are cycle-free paths from x_k to x_{k+1} in the connected graph $C(G, S)$. Likewise, there is a (possibly empty) path π_0 from (f, x) to (f_1, x_1). But then $\pi_0 \pi_1 \pi_2 \ldots \pi_{2^{|G|}-2}(f_{2^{|G|}}, x_{2^{|G|}-1})$ is a cycle-free path in $\Lambda(G, S)$ of length at least $2^{|G|}$, as required. □

We can combine the preceding lemmas to yield:

Corollary 20. *Let A be the iterated lamplighter graph $\Lambda^i(m)$ and $v \in A$ be a node. The r-size of $\mathrm{B}_A(v,l)$ is at least n if $\exp_2^i(m) \geq rn$ and $l \geq r(n+1)$.*

4.6 Locality of Programs on Iterated Lamplighter Graphs

In this section we put together the results from the previous sections to show that PURPLE cannot decide undirected **s**-**t**-reachability. Define $\Delta^i(m)$ to be the action graph consisting of two disjoint copies of the iterated lamplighter graph $\Lambda^i(m)$. In this section we show that each PURPLE program can be implemented by an abstract local program on $\Delta^i(m)$, for some large enough i and m, such that the radius of the abstract local program is smaller than half of the diameter of $\Delta^i(m)$. Since the abstract local program then cannot distinguish whether **s** and **t** lie on different connected components or just have large distance on the same component, we can conclude from this that PURPLE cannot decide undirected reachability.

We start by instantiating Theorem 14 to the graphs $\Delta^i(m)$ with the help of Corollaries 17 and 20.

Lemma 21. *There exist numbers a and b, such that whenever $f \in L_{\Delta^i(m)}(Q,P,r)$ where $m \geq \max(|Q|,|P|)$ and $\exp_2^i(m) \geq \exp_2^a(i+m+r)$ then for any $\boldsymbol{z} \in P$ there exists an abstract local program $g \in L_{\Delta^i(m)}\left(Q,P,\exp_2^b(i+m+r)\right)$ that implements* `forall` $\boldsymbol{z}$ `do` f *on $\Delta^i(m)$.*

Proof. We introduce the class of *polynomial functions with exponentiation*, which is the smallest class of functions that contains constants and variables and that is closed under addition e_1+e_2, multiplication $e_1 \cdot e_2$ and exponentiation $e_1^{e_2}$. It is a standard result that for each polynomial with exponentiation e with variables $x_1,\ldots,x_k$, there exists a number l such that $e \leq \exp_2^l(x_1+\cdots+x_k)$ holds for any valuation of $\vec{x}$. See e.g. [7] for a proof.

Define the following polynomials with exponentiation with variables q, p, m, r and i, in which c is the constant from Theorem 10.

$$\begin{aligned} \mathit{rng}(q,p,r) &:= \left((i+3)^r \cdot q \cdot m \cdot 2^i\right)^{c^p} \\ \mathit{mod}(q,p,r) &:= q \cdot p^p \cdot (i+3)^{p \cdot \mathit{rng}(q,p,r)} \\ R &:= 3 \cdot \mathit{rng}(q \cdot (p+2)^p, 2p+1, r) \\ M &:= \mathit{mod}(q \cdot (p+2)^p, 2p+1, r) \\ u &:= 3 \cdot R \cdot p \cdot (M+1) + 6 \cdot R \end{aligned}$$

We now show that there exist natural numbers a and b such that whenever $m \geq \max(q,p)$ and $\exp_2^i(m) \geq \exp_2^a(i+m+r)$ then

$$\exp_2^{i-1}(m) \geq (2u+1) \cdot (p \cdot (M+1) + 12) \tag{9}$$

$$\exp_2^i(m) \geq 3R \cdot p \cdot (M+1) \tag{10}$$

$$\exp_2^b(i+m+r) \geq u + r \cdot p \cdot |\mathrm{B}_{\Lambda^i(m)}(e,u)| \ . \tag{11}$$

To see this, note that by the assumption $m \geq \max(q,p)$, we can replace both q and p by m on the right-hand sides of the inequations. The right-hand sides of the first two inequations then become polynomials with exponentiation with

variables i, m and r. As noted above, we can bound them from above by terms of the form $\exp_2^l(i+m+r)$ for appropriate l. Hence, we can choose a such that the first two inequations are implied by $\exp_2^i(m) \geq \exp_2^a(i+m+r)$. Furthermore, since the degree of $\Delta^i(m)$ is $i+2$, we have $(i+3)^u \geq |\mathrm{B}_{\Lambda^i(m)}(e,u)|$. Hence, we can find a polynomial with exponentiation that is an upper bound for the right-hand side of the last inequation. Then, by the same argument used to find a, we obtain a number b making the last inequality true.

We now show the assertions of the lemma with this choice of a and b. Let $f \in L_{\Delta^i(m)}(Q,P,r)$ and $\boldsymbol{z} \in P$ and assume $m \geq \max(|Q|,|P|)$ and $\exp_2^i(m) \geq \exp_2^a(i+m+r)$. We obtain the result using Theorem 14, whose premises we satisfy as follows.

Notice that for any set of states Q' and any set of pebbles P', we have

$$\begin{aligned} \mathit{rng}(|Q'|,|P'|,r) &\geq \mathit{range}_{\Delta^i(m)}(Q',P',r) \\ \mathit{mod}(|Q'|,|P'|,r) &\geq \mathit{modulus}_{\Delta^i(m)}(Q',P',r) \end{aligned}$$

by Theorem 10, the estimation in (3) and because the graph $\Delta^i(m)$ has exponent $m \cdot 2^i$ and degree $i+2$. It follows that if in the above polynomials with exponentiation we assign the variables q and p the values $|Q|$ and $|P|$ respectively, then the numbers R and M satisfy the requirements of Theorem 14.

The remaining premises of Theorem 14 follow from the choices of u and a by Corollaries 17 and 20.

Theorem 14 therefore shows that there exists an abstract local program $g \in L_{\Delta^i(m)}\left(Q,P,u+r\cdot|P|\cdot|\mathrm{B}_{\Lambda^i(m)}(e,u)|\right)$ that implements `forall` $\boldsymbol{z}$ `do` f on $\Delta^i(m)$. By (11), $\exp_2^b(i+m+r)$ is an upper bound for the local radius of g. Hence, we may consider g as a program in $L_{\Delta^i(m)}(Q,P,\exp_2^b(i+m+r))$, which completes the proof. □

With this lemma, we can show that each `while`-free PURPLE program can be simulated by an abstract local program on any graph $\Delta^i(m)$ with large enough i and m. To formulate this simulation precisely, we use the following notation. For a PURPLE program M, we write $Q(M)$ for $2^{\mathit{Vars}^{\mathtt{bool}}(M)}$ and write $P(M)$ for $\mathit{Vars}^{\Gamma}(M) \cup \{\mathbf{s},\mathbf{t}\}$. We write $|M|$ for the length of program M, that is the number of nodes in the abstract syntax tree.

Theorem 22. *For all k there exist i_0 and r with the following property. For each $i \geq i_0$, each* `while`*-free* PURPLE *program M of* `forall`*-depth k and each $m \geq \max(|Q(M)|,|P(M)|)$, there exists an abstract local program*

$$f \in L_{\Delta^i(m)}\left(Q(M),P(M),\exp_2^r(i+m+|M|)\right),$$

such that $[\![f]\!](q,\rho) = (p,\tau)$ implies $(M,q,\rho) \to^*_{(\Delta^i(m),\rho(\mathbf{s}),\rho(\mathbf{t}))} (\mathtt{skip},p,\tau)$ for all $q,p \in Q(M)$ and all $\rho,\tau \in (\Delta^i(m))^{P(M)}$.

Proof. The proof goes by induction on k.

Base case $k=0$. Define $i_0 = r = 0$. Let $i \geq i_0$. The result then follows, since for any PURPLE program M can be compiled into an abstract local program with local radius $|M|$, as shown in Lemma 9.

Induction step. Assume that we have numbers i_0 and r that work for k in the sense that for all $i \geq i_0$, all `while`-free PURPLE programs M with `forall`-depth k and all $m \geq \max(|Q(M)|, |P(M)|)$, there exists an abstract local program $f \in L_{\Delta^i(m)}(Q(M), P(M), \exp_2^r(i + m + |M|))$ with the property that $(M, q, \rho) \to^*_{(\Delta^i(m), \rho(\mathbf{s}), \rho(\mathbf{t}))} (\mathtt{skip}, p, \tau)$ holds whenever $[\![f]\!](q, \rho) = (p, \tau)$ does.

We have to find numbers i_0' and r' that work for $k + 1$.

To define i_0' and r', let a and b be the numbers from Lemma 21. We choose $i_0' \geq i_0$ and $r' > 0$, in such a way that the inequations

$$\exp_2^i(x) \geq \exp_2^a(i + x + \exp_2^r(i + x)), \tag{12}$$

$$\exp_2^{r'}(i + x) \geq \exp_2^b(i + x + \exp_2^r(i + x)) \tag{13}$$

hold for all $i \geq i_0'$ and all x. This can be done by bounding the right-hand sides from above by $\exp_2^d(i + x)$ for some d, as in the proof of Lemma 21, and then choosing $r' := d$ and $i_0' := 2(d+2)$, which works since $i + x \leq \exp_2^{i/2+2}(x)$ holds for all $i, x \geq 0$.

Let $i \geq i_0'$. We now show the required property by induction on the `while`-free PURPLE programs M of `forall`-depth $k + 1$.

- **Case M is `forall` x `do` M'.** Let $m \geq \max(|Q(M)|, |P(M)|)$.

  Since M has `forall`-depth $k + 1$, the program M' has `forall`-depth k, so that we can apply the outer induction hypothesis to obtain an abstract local program

  $$f' \in L_{\Delta^i(m)}(Q(M'), P(M'), \exp_2^r(i + m + |M'|))$$

  with the property that $(M', q, \rho) \to^*_{(\Delta^i(m), \rho(\mathbf{s}), \rho(\mathbf{t}))} (\mathtt{skip}, p, \tau)$ holds whenever $[\![f']\!](q, \rho) = (p, \tau)$.

  By the choice of i_0' and r' and with (12), Lemma 21 shows that there exists an abstract local program

  $$f \in L_{\Delta^i(m)}(Q(M'), P(M'), \exp_2^b(i + m + \exp_2^r(i + m + |M'|)))$$

  that implements `forall` x `do` f' on $\Delta^i(m)$.

  Now we have

  $$\begin{aligned}
  &\exp_2^b(i + m + \exp_2^r(i + m + |M'|)) \\
  &\leq \exp_2^b(i + m + |M| + \exp_2^r(i + m + |M|)) && \text{by monotonicity} \\
  &\leq \exp_2^{r'}(i + m + |M|) && \text{by (13) with } m + |M| \text{ for } x
  \end{aligned}$$

  Hence, we may consider f an element of

  $$L_{\Delta^i(m)}(Q(M), P(M), \exp_2^{r'}(i + m + |M|)).$$

  Combining the properties of f and f', we get that $[\![f]\!](q, \rho) = (p, \tau)$ implies

  $$(\mathtt{forall}\ x\ \mathtt{do}\ M, q, \rho) \to^*_{(\Delta^i(m), \rho(\mathbf{s}), \rho(\mathbf{t}))} (\mathtt{skip}, p, \tau).$$

  Thus, f is as required to complete this case.

- **Case M is $M'; M''$.** Let $m \geq \max(|Q(M)|, |P(M)|)$. This number m is large enough to apply the inner induction hypothesis to M' and M''. We obtain abstract local programs

$$f' \in L_{\Delta^i(m)}(Q(M'), P(M'), \exp_2^{r'}(i + m + |M'|))$$
$$f'' \in L_{\Delta^i(m)}(Q(M''), P(M''), \exp_2^{r'}(i + m + |M''|))$$

so that $(M', q, \rho) \to^*_{(\Delta^i(m), \rho(\mathbf{s}), \rho(\mathbf{t}))} (\mathtt{skip}, p, \tau)$ holds whenever $[\![f']\!](q, \rho) = (p, \tau)$ does, and that $(M'', q, \rho) \to^*_{(\Delta^i(m), \rho(\mathbf{s}), \rho(\mathbf{t}))} (\mathtt{skip}, p, \tau)$ holds whenever $[\![f'']\!](q, \rho) = (p, \tau)$ does. We can view both f' and f'' as abstract local programs with state set $Q(M)$ and pebble set $P(M)$ in the canonical way.

By setting $f := \mathit{comp}(f', f'')$, we obtain an element of

$$L_{\Delta^i(m)}\left(Q(M), P(M), \exp_2^{r'}(i + m + |M'|) + \exp_2^{r'}(i + m + |M''|)\right)$$

such that $(M, q, \rho) \to^*_{(\Delta^i(m), \rho(\mathbf{s}), \rho(\mathbf{t}))} (\mathtt{skip}, p, \tau)$ holds whenever $[\![f]\!](q, \rho) = (p, \tau)$ does, cf. Lemma 8. To show that f specifies an element of

$$L_{\Delta^i(m)}\left(Q(M), P(M), \exp_2^{r'}(i + m + |M|)\right)$$

it then suffices to show

$$\exp_2^{r'}(i + m + |M|) \geq \exp_2^{r'}(i + m + |M'|) + \exp_2^{r'}(i + m + |M''|).$$

But this follows because the inequality $\exp_2^r(x + y + z) \geq \exp_2^r(x + y) + \exp_2^r(x + z)$ holds for all natural numbers x, y, z and r with $r, y, z \geq 1$.

- **Case M is $\mathtt{if}\ t^{\mathtt{bool}}\ \mathtt{then}\ M'\ \mathtt{else}\ M''$.** Let $m \geq \max(|Q(M)|, |P(M)|)$. This number m is large enough to apply the inner induction hypothesis to M' and M''. We obtain abstract local programs

$$f' \in L_{\Delta^i(m)}\left(Q(M'), P(M'), \exp_2^{r'}(i + m + |M'|)\right),$$
$$f'' \in L_{\Delta^i(m)}\left(Q(M''), P(M''), \exp_2^{r'}(i + m + |M''|)\right).$$

We can consider f' and f'' as abstract local programs with state set $Q(M)$ and pebble set $P(M)$. We then define

$$f(q, c) = \begin{cases} f'(q, c) & \text{if } [\![t^{\mathtt{bool}}]\!]_{q,c} = \mathtt{true}, \\ f''(q, c) & \text{if } [\![t^{\mathtt{bool}}]\!]_{q,c} = \mathtt{false}, \end{cases}$$

as in the proof of Lemma 9. The local radius of f is at most the maximum of the local radii of f' and f''. We can therefore bound it from above by $\exp_2^{r'}(i + m + |M|)$, which is sufficient to complete this case.

- All other cases are excluded by the assumption that M has `forall`-depth $k + 1$, since in these cases the program would have to have `forall`-depth 0.

□

Theorem 23. *For all k there is a number d such that no* `while`*-free* PURPLE *program with* `forall`*-depth k decides reachability on undirected graphs of degree d.*

Proof. Given k, let i_0 and r be the numbers provided by Theorem 22. Choose $i \geq i_0$ to be large enough such that $\exp_2^r(i+2x) < \exp_2^i(x)$ holds for all x. Such an i can be found as in the proof of Theorem 22. With this choice of i, we define $d = i + 4$, the degree of $\Delta^{i+2}(m)$ for any $m > 2$.

We have to show that with this choice of d no abstract program M with `forall`-depth k can decide reachability on undirected graphs of degree d. By Theorem 22 any such program M program can be implemented on $\Delta^{i+2}(m)$ by an abstract local program f with local radius $\exp_2^r(i + m + |M|)$ for $m := \max(|Q(M)|, |P(M)|) + |M|$ and any choice of nodes for $\mathbf{s}$ and $\mathbf{t}$. Now, by the choice of i, we can bound the local radius of f as follows.

$$\exp_2^r(i + m + |M|) < \exp_2^r(i + 2m) \leq \exp_2^i(m)$$

The diameter of a lamplighter graph $\Lambda(G, S)$ is at least $|G|$, since the nodes $(\lambda x.0,\, e)$ and $(\lambda x.1,\, e)$ have distance at least $|G|$. Therefore, we have

$$\begin{aligned} \mathit{diam}\left(\Delta^{i+2}(m)\right) &= \mathit{diam}\left(\Lambda^{i+2}(m)\right) \\ &\geq |\Lambda^{i+1}(m)| \geq 2^{|\Lambda^i(m)|} \geq 2 \cdot |\Lambda^i(m)| \geq 2 \cdot \exp_2^i(m). \end{aligned}$$

Hence, the local radius of f is less than $\mathit{diam}\left(\Delta^{i+2}(m)\right)/2$.

We now consider two ways of making $\Delta^{i+2}(m)$ into an $\mathbf{s}$-$\mathbf{t}$-graph. First, we choose for $\mathbf{s}$ and $\mathbf{t}$ two nodes on the same connected component with distance $\mathit{diam}(\Delta^{i+2}(m))$. Second, we choose $\mathbf{s}$ and $\mathbf{t}$ to be on different connected components. Consider now starting configurations in which all pebbles are placed on either $\mathbf{s}$ or $\mathbf{t}$. Since the abstract local program f has local radius smaller than $\mathit{diam}\left(\Delta^{i+2}(m)\right)/2$, it cannot distinguish whether the nodes $\mathbf{s}$ and $\mathbf{t}$ are on different connected components or whether they are far apart on the same connected component. Since f implements M, the same holds for M, which implies that M cannot decide either whether or not $\mathbf{s}$ is reachable from $\mathbf{t}$. □

Corollary 24. *There is no* PURPLE *program that decides undirected reachability on graphs of degree* 3*.*

Proof. The proof is by the straightforward transformation of degree d graphs to degree 3 graphs that replaces nodes by length d cycles.

In detail, suppose for a contradiction that M decides undirected reachability on graphs of degree 3. In view of Proposition 1 we may assume that M is `while`-free. Let k be the `forall`-depth of M and let d be such that no `while`-free PURPLE-program can decide undirected reachability for graphs of degree d. Such d exists by Theorem 23.

From M we will now construct a program M' of the same `forall`-depth k that does decides undirected reachability on graphs of degree d, which is a contradiction.

Intuitively, M' will replace each node of the input graph by a cycle of length d and present the resulting graph as input to M. To do that in the PURPLE-formalism we introduce for each pointer variable (pebble) x a variable d_x with range $\{1, \ldots, d\}$ (concretely implemented using $\lceil \log d \rceil$ boolean variables). The

idea is that $x = v$, $d_x = i$ means that the variable x contains the i-th node in the cycle representing node v of the input graph. Thus, we obtain M' from M by replacing the instructions of M as follows:

$$
\begin{array}{lcl}
y := x & \mapsto & y := x;\ d_y := d_x \\
y := \mathbf{s} & \mapsto & y := \mathbf{s};\ d_y := 0 \\
y := \mathbf{t} & \mapsto & y := \mathbf{t};\ d_y := 0 \\
y := x.succ(0) & \mapsto & y := x;\ d_y := d_x + 1 \bmod d \\
y := x.succ(1) & \mapsto & y := x;\ d_x := d_x - 1 \bmod d \\
y := x.succ(2) & \mapsto & y := x.succ(d_x) \\
\texttt{forall}\ x\ \texttt{do}\ M & \mapsto & \texttt{forall}\ x\ \texttt{do}\ (y := x; \\
 & & \quad \texttt{for}\ i = 1, \ldots, d\ \texttt{do}\ (x := y;\ d_x := i;\ M))
\end{array}
$$

Here we have used some pseudocode instructions like '`for` $i = 1, \ldots, d$ `do`' or '$y := x.succ(d_x)$' that must be expanded into official PURPLE-code in the obvious way.

It is then clear that M' decides undirected reachability on a given degree d graph. Notice that the simulation of a `forall`-loop presents the nodes of the degree 3 graph in some particular order. But M must give correct results for any order including the ones chosen in the simulation. To prove this formally, one shows that for any run r of M' on some degree d graph G there is a run r' of M on the degree 3 graph G' obtained by replacing the nodes of G by length d cycles. Moreover, if (q, ρ) is a configuration in r immediately before or after the completion of the translation of an instruction of M then the corresponding configuration (q', ρ') in r' satisfies $q'(b) = q(b)$ for all boolean variables of M and $\rho'(x) = (\rho(x), d_x)$. Here we assume that the nodes of G' are pairs of the form (v, i) with v a node of G and $i \in \{1, \ldots, d\}$.

As already mentioned, no such M' exists by Theorem 23. □

Corollary 25. *There is no* DTC*-formula for locally ordered graphs (one- or two-way-ordered) that expresses* **s**-**t***-reachability for undirected graphs of degree* 3*.*

This corollary follows immediately from Corollary 24 and Proposition 2. It strengthens the results of Etessami & Immerman, who prove that undirected reachability cannot be expressed by a DTC-formula of the form $\mathrm{DTC}(\varphi)$, where φ is a first-order formula. Without a total ordering, not every DTC-formula is equivalent to one of this form.

References

[1] S.A. Cook and P. McKenzie. Problems complete for deterministic logarithmic space. *Journal of Algorithms*, 8(3):385–394, 1987.

[2] S.A. Cook and C. Rackoff. Space lower bounds for maze threadability on restricted machines. *SIAM Journal of Computing*, 9(3):636–652, 1980.

[3] H.D. Ebbinghaus and J. Flum. *Finite Model Theory*. Springer-Verlag, 1995.

[4] K. Etessami and N. Immerman. Reachability and the power of local ordering. *Theoretical Computer Science*, 148(2):261–279, 1995.

[5] M. Hofmann and U. Schöpp. Pure pointer programs with iteration. In *CSL*, 2008.

[6] D.E. Knuth. *The Art of Computer Programming 4 Fascicle 2: Generating All Tuples and Permutations*. Addison-Wesley, 2005.

[7] H. Levitz. An ordinal bound for the set of polynomial functions with exponentiation. *Algebra Universalis*, 8:233–243, 1978.

[8] 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.

[9] O. Reingold. Undirected st-connectivity in log-space. In *STOC05*, pages 376–385, 2005.

[10] U. Schöpp. A formalised lower bound on undirected graph reachability, 2008. Manuscript.
`http://www.tcs.ifi.lmu.de/~schoepp/formalcr.html`.

ECCC ISSN 1433-8092

http://eccc.hpi-web.de/