Efficient Algorithms for Highly Compressed Data: The Word Problem in Generalized Higman Groups Is in P

Jürn Laun

Published online: 10 October 2013
© Springer Science+Business Media New York 2013

Abstract This paper continues the 2012 STACS contribution by Diekert, Ushakov, and the author as well as the 2012 IJAC publication by the same authors. We extend the results published there in two ways.

First, we show that the data structure of power circuits can be generalized to work with arbitrary bases \( q \geq 2 \). This results in a data structure that can hold huge integers, arising by iteratively forming powers of \( q \). We show that the properties of power circuits known for \( q = 2 \) translate to the general case. This generalization is non-trivial and additional techniques are required to preserve the time bounds of arithmetic operations that were shown for the case \( q = 2 \).

The extended power circuit model permits us to conduct operations in the Baumslag-Solitar group \( BS(1, q) \) as efficiently as in \( BS(1, 2) \). This allows us to solve the word problem in the generalization \( H_4(1, q) \) of Higman’s group in polynomial time. The group \( H_4(1, q) \) is an amalgamated product of four copies of the Baumslag-Solitar group \( BS(1, q) \) rather than \( BS(1, 2) \) in the original group \( H_4 = H_4(1, 2) \).

As a second result, we allow arbitrary numbers \( f \geq 4 \) of copies of \( BS(1, q) \), leading to an even more generalized notion of Higman groups \( H_f(1, q) \). We prove that the word problem of the latter can still be solved within the \( \mathcal{O}(n^6) \) time bound that was shown for \( H_4(1, 2) \).

Keywords Data structures · Compression · Algorithmic group theory · Word problem
1 Introduction

This work is a sequel to the STACS paper [4] and its journal version [5]. However, we try to keep it self-contained by reiterating everything of importance from the two preceding papers. We extend their results to a more general version of power circuits. As a consequence, we can apply them to larger classes of groups. Details about the results can also be found in the author’s PhD thesis [8].

The group $H_4$ was introduced by Higman in 1951 and served to provide the first known example of a finitely generated infinite simple group [6]. It belongs to a family $H_f$ ($f \geq 1$) of groups with $f$ generators and $f$ relators:

$$H_f = \langle a_1, \ldots, a_f a_i a_{i+1} a_i^{-1} = a_i^2 \mid i \in \mathbb{Z}/f\mathbb{Z} \rangle$$

For $f < 4$ these groups are trivial, which is easy to see for $f \in \{1, 2\}$, but surprisingly hard for $f = 3$. The latter case was proven by Hirsch (unpublished but mentioned in [6]), see also §23 in [12] and [1]. If $f \geq 4$, then $H_f$ is infinite, see [13], Sect. 1.4. Since $H_f$ has no non-trivial normal subgroups of finite index, taking a minimal non-trivial quotient results in a finitely generated infinite simple group.

Until recently, Higman’s Group $H_4$ was a candidate for a “natural” group with non-elementary word problem. This was suggested by the huge compression that this group allows. In fact, there are words of length $n$ over the generators $a_i$ and their inverses, which in the group are equal to $a_i^{	ext{tow}_2(n)}$, where $\text{tow}_2(n)$ is the tower function (also called “tetration”) defined by $\text{tow}_2(0) = 1$ and $\text{tow}_2(n+1) = q^{\text{tow}_2(n)}$. However, in [4] it was shown that the word problem of $H_4$ is decidable in $O(n^6 \cdot \log n)$ time and [5] improved this bound to $O(n^6)$. Both results rely on a data structure called “power circuit” which was introduced by Miasnikov, Ushakov, and Won in [11]. Power circuits had already proven useful in algorithmic group theory. In fact, their invention was entailed by the wish to efficiently solve the word problem in the Baumslag-Gersten group $G_{(1,2)}$ (see [10] and Sect. 3) which shares with $H_4$ the property of huge compression. (In [10] the group $G_{(1,2)}$ is called “Baumslag group”. We use the equally common name “Baumslag-Gersten group” to avoid confusion with the Baumslag-Solitar group.) For $q = 2$, power circuits have been implemented [9] and there is a computer program solving the word problem in $G_{(1,2)}$.

This paper builds on the work of Diekert, Ushakov, and the author conducted in [4] and [5]. Its contributions are twofold: In Sect. 2, we extend power circuits to allow arbitrary bases $q \geq 2$. This necessitates changes in the reduction procedure, the core component of power circuits. In Sect. 4, we generalize Higman’s group $H_4$ by replacing the underlying group $BS(1,2)$ by $BS(1,q) = \langle a, t \mid tat^{-1} = a^q \rangle$. Power circuits with base $q$ are naturally suited for computations in this group.

Furthermore, with the help of rewriting systems, we give a constructive method of treating any group $H_f$ ($f \geq 4$) rather that just $H_4$. Combining this with base $q$ power circuits leads to an algorithm for the word problem in generalized Higman groups $H_f(1,q)$ which retains the $O(n^6)$ time bound proved in [5] for $q = 2$ and $f = 4$.

Notation and Preliminaries

Algorithms and decision problems are classified by their time complexity on a random-access machine (RAM). We use the notion of
amortized analysis with respect to a potential function, see Sect. 17.3 in [3]. We denote time and space complexity using the Landau symbol $\mathcal{O}$ and the derived $\tilde{\mathcal{O}}$ (so-called “soft-Oh” which ignores polylogarithmic factors).

With regard to group theory, we use standard notation and facts that can be found in any textbook on the subject, e.g. [7]. In particular, we apply the technique of Britton reductions for solving the word problem in HNN extensions and amalgamated products.

Rewriting systems are of particular importance for this work. We assume that the reader is familiar with the basic notions of (local) confluence and termination. See for example the textbook [2] for a quick introduction.

2 Power Circuits

Power circuits were introduced by Miasnikov, Ushakov, and Won in [11]. While the underlying ideas presented in this chapter originate from their work, there are some important differences. First, we use a different (and hopefully more accessible) notation, following [4] and [5]. We also allow multiple markings in one circuit. The most important modification, which also distinguishes this paper from [4] and [5], is the generalization from base 2 to arbitrary bases $q \geq 2$.

2.1 Power Circuit and Evaluation

For the rest of this paper, we fix an integer $q \geq 2$ for the base and the interval $D = \{-q + 1, \ldots, q - 1\}$ (the set of digits). We start with a directed acyclic edge-labelled graph without multi-edges, given by $\Pi = (\Gamma, \delta)$. Here, $\Gamma$ is a finite set which will act as the set of nodes (or vertices). The labelled edges (or arcs) are given by the map $\delta : \Gamma \times \Gamma \to D$ where $\delta(u, v) = 0$ means that there is no edge from $u$ to $v$ and $\delta(u, v) = e \neq 0$ implies an edge from $u$ to $v$ labelled with the number $e$. In other words, the edge set is $\text{supp} \delta$, the support of the map $\delta$. In addition, we require that the directed graph $(\Gamma, \text{supp} \delta)$ is acyclic. We shall make this assumption throughout this paper without mentioning it again. For any operation on graphs introduced in this chapter, it will be obvious that acyclicity is preserved.

A marking of $\Pi = (\Gamma, \delta)$ is a mapping $M : \Gamma \to D$. Often we regard $M$ as a labelled subset of the nodes of $\Pi$, where the subset is $\text{supp} M$ and the labels are given by $M|_{\text{supp} M} : \text{supp} M \to D$. In this sense, $M = 0$ (the constant zero marking) and $M = \emptyset$ (the empty marking) are the same.

Each node $u \in \Gamma$ induces in a natural way its successor marking $\Lambda_u$ defined by

$\Lambda_u : \Gamma \to D; \quad v \mapsto \delta(u, v)$.

Intuitively, the successor marking of a node $u$ consists of the target nodes of edges starting at $u$ and their labels are given by those of the edges.

The purpose of markings is to store large numbers. Therefore, we define the evaluation function $\varepsilon$ which assigns a real number to each node and each marking of a
graph $\Pi$. The number $\varepsilon(P)$ (or $\varepsilon(M)$) is called the value of the node $P$ (or of the marking $M$). As $\Pi$ is acyclic we can define $\varepsilon$ by induction:

$$
\varepsilon(\emptyset) = 0 \quad \text{where } \emptyset \text{ is the empty marking,}
$$

$$
\varepsilon(M) = \sum_{u \in \text{supp}(M)} M(u) \cdot \varepsilon(u) \quad \text{for all other markings } M,
$$

$$
\varepsilon(u) = q^{\varepsilon(\Lambda_u)} \quad \text{for each node } u \in \Gamma.
$$

Note that this implies $\varepsilon(u) = 1$ for all sinks $u$ (nodes without outgoing edges). For every node $u \in \Gamma$ we have

$$
\varepsilon(\Lambda_u) = \log_q \varepsilon(u).
$$

**Example 2.1** Figure 1 shows an example of such a graph for $q = 3$. The set of nodes is $\Gamma = \{u_1, u_2, u_3, u_4, u_5\}$ and $\delta$ is given by

$$
\begin{align*}
\delta(u_2, u_1) &= +1, & \delta(u_3, u_1) &= +2, & \delta(u_4, u_1) &= -1, & \delta(u_4, u_2) &= -2, \\
\delta(u_4, u_3) &= +1, & \delta(u_5, u_2) &= +2, & \delta(u_5, u_3) &= +1, & \delta(u_5, u_4) &= -2.
\end{align*}
$$

The nodes evaluate to

$$
\varepsilon(u_1) = 1, \quad \varepsilon(u_2) = 3, \quad \varepsilon(u_3) = 9, \quad \varepsilon(u_4) = 9, \quad \text{and } \varepsilon(u_5) = \frac{1}{27}.
$$

**Lemma 2.2** Let $\Pi = (\Gamma, \delta)$ be as described above. The following statements are equivalent:

(i) $\varepsilon(u) \in q^{\mathbb{N}_0} = \{q^n : n \in \mathbb{N}_0\}$ for every node $u \in \Gamma$

(ii) $\varepsilon(\Lambda_u) \geq 0$ for every node $u \in \Gamma$

(iii) $\varepsilon(M) \in \mathbb{Z}$ for every possible marking $M$ in $\Pi$

**Proof** This is easily seen by noetherian induction with respect to a topological order of $\Gamma$ (i.e., an order compatible with the edges).

**Definition 2.3** A **power circuit** is a finite acyclic edge-labelled graph $\Pi = (\Gamma, \delta)$ without multiple edges that meets the equivalent conditions of Lemma 2.2.
Example 2.4 The graph in Fig. 1 is not a power circuit due to $\varepsilon(u_5) \notin \mathbb{Z}$. In contrast, Fig. 2 depicts a power circuit for $q = 2$. The values of the nodes are given for illustrative purposes only. They are not part of the data structure, since in general these number become too large to be computed or even stored efficiently. The marking $M$ evaluates to $\varepsilon(M) = 29$.

In Corollary 2.15 we will show that it can be efficiently tested whether a given graph is a power circuit. In [11], a power circuit does not have to satisfy the criteria of Lemma 2.2, but if it does, it is called proper. In this sense, we only deal with proper power circuits.

Figure 3 shows that a power circuit of linear size can contain markings with values the magnitude of the tower function.

2.2 Arithmetic Operations

Let $\Pi = (\Gamma, \delta)$ be a power circuit and $u \in \Gamma$ a node. The operation CLONE with result $v = \text{CLONE}(u)$ creates a new node $v$ with the same successor marking as $u$, but no incoming arcs. This means that $v$ has the same number of outgoing edges than $u$ and they go to the same target nodes and have the same labels. We extend this operation to markings $M$, by cloning every single node in $\text{supp} M$. The resulting marking $\text{CLONE}(M)$ is defined as the marking consisting of all these clones, and the signs are copied from $M$:

$$\text{CLONE}(M) : \Gamma \cup \{\text{CLONE}(u) : u \in \text{supp} M\} \to D;$$

$$v \mapsto \begin{cases} M(u) & \text{if } v \text{ is the clone of some } u \in \text{supp} M, \\ 0 & \text{if } v \in \Gamma. \end{cases}$$

Example 2.5 In Fig. 4, the marking $M$ consisting of two nodes is cloned.
Now we can define arithmetic operations. Let $\Pi = (\Gamma, \delta)$ be a power circuit and let $K$ and $M$ be markings in $\Pi$. If the supports of $K$ and $M$ are disjoint, the mapping $K + M$ defined by $(K + M)(u) = K(u) + M(u)$ is a marking with $\varepsilon(K + M) = \varepsilon(K) + \varepsilon(M)$. In general, however, the operands $K$ and $M$ will not be disjoint. In this case we might have nodes $u \in \text{supp } K \cap \text{supp } M$ with $K(u) + M(u) \notin D$, hence $K + M$ is not a valid marking. We solve this problem by cloning: for every node $u$ with $K(u) + M(u) \notin D$, we create a clone $u' = \text{CLONE}(u)$ and modify $K + M$ by putting $(K + M)(u) := K(u)$ and $(K + M)(u') := M(u)$. We obtain a valid marking in the (now enlarged) circuit with value $\varepsilon(K) + \varepsilon(M)$.

Example 2.6 In Fig. 5, $\varepsilon(K) = 7$ and $\varepsilon(M) = 35$ are added. In the resulting marking, the node with value 1 cancels out, whereas both the original node with value 4 and its newly created clone are included.
The second operation that we need is multiplication by a power of $q$, i.e., obtain a marking $K \cdot q^M$ with value $\varepsilon(K) \cdot q^{\varepsilon(M)}$. We observe that

$$\varepsilon(K) \cdot q^{\varepsilon(M)} = \sum_{u \in \text{supp } K} K(u) \cdot q^{\varepsilon(\Lambda u)} \cdot q^{\varepsilon(M)} = \sum_{u \in \text{supp } K} K(u) \cdot q^{\varepsilon(\Lambda u) + \varepsilon(M)} ,$$

so in principle we would just have to introduce new edges from each node in supp $K$ to each node in supp $M$. The label of such an edge would be the value that $M$ assigns to the respective target node. This operation works as long as

1. no cycles are introduced into the circuit,
2. no multi-edges between two nodes are introduced,
3. there are no edges between nodes in supp $K$, and
4. no other marking in the circuit is affected. (Note here, that the original value of $K$ is lost in any case.)

Again, the solution is cloning. Create clones $K' := \text{CLONE}(K)$ and $M' := \text{CLONE}(M)$ and introduce new edges by putting $\delta(u, v) := M(v)$ for all $u \in \text{supp } K'$, $v \in \text{supp } M'$. Being clones, nodes in supp $K'$ and supp $M'$ have no incoming edges, which prevents cycles and multi-edges. Also, no other marking in the circuit depends on $K'$ or $M'$ directly (by containing these nodes) or indirectly (by containing nodes that are topologically above any node in $K'$ or $M'$). The value of $K'$ is exactly $\varepsilon(K) \cdot q^{\varepsilon(M)}$. An example (in which no further cloning is necessary) is shown in Fig. 6.

Finally, the operation $M \mapsto -M$ which negates the value of $M$ is easy to conduct without any complications or the need for cloning.

There is a crucial difference between the power circuits in this paper and those introduced in [11], where only one marking per circuit was allowed. In contrast, we keep several markings in the same circuit. In fact, in our applications, we will use just one single circuit containing all the markings. The fact that we don’t have to merge different circuits for each operation contributes to the improved time complexity of our algorithms. From a programmer’s point of view, the power circuit might be regarded as some kind of memory management system and the markings as (integer) variables stored in this system.
This means that a power circuit usually come with two sets of markings attached to it: first, the successor markings which are implicitly given by the (labelled) graph structure, and second, the “normal” markings (or variables) that carry meaningful data for the algorithm in question. While an operation may change the structure of the circuit (and thus the successor markings), it should not affect markings of the second type (except, maybe, for the operands which might be destroyed, depending on the exact definition of the operation). As we have just seen, cloning helps us do just that in the arithmetic operations discussed in this section.

2.3 Reduction

The operations $K + M$ and $K \cdot q^M$ introduced in the previous section are quite efficient. Assuming that the graph is stored using adjacency lists, the time they take depends only on the size of the markings $M$ and $K$ and the number of their successors, but not the size of the circuit. The price for this efficiency is that the structure of a power circuit can quickly become rather opaque. In particular, it is unclear how (in)equality of the values of two markings can be determined in an arbitrary circuit. Again, note that evaluating the nodes or markings is not an option, due to the vast growth permitted by power circuits. For this reason, we restrict ourselves to a subclass of circuits and augment them with some additional data:

**Definition 2.7** A reduced power circuit is a power circuit $\Pi = (\Gamma, \delta)$ together with a list $(u_1, \ldots, u_n)$ of its nodes and a bit vector $(b_1, \ldots, b_{n-1}) \in \mathbb{B}^{n-1}$ such that

(i) different nodes evaluate to different numbers, i.e., for all $u, v \in \Gamma$ with $u \neq v$, $\varepsilon(u) \neq \varepsilon(v)$,

(ii) the list of nodes is sorted by value, i.e., $\varepsilon(u_1) < \varepsilon(u_2) < \cdots < \varepsilon(u_n)$,

(iii) $b_i = 1$ if and only if $q \cdot \varepsilon(u_i) = \varepsilon(u_{i+1})$.

In particular, a (non-empty) reduced power circuit contains exactly one node with value 1 (a sink).

**Proposition 2.8** (cf. Prop. 5 in [4] and Sect. 2.1 in [11]) Given a reduced circuit and two markings $K$ and $M$, the values $\varepsilon(K)$ and $\varepsilon(M)$ can be compared (yielding $<$, $=$, or $>$ as the result) in $O(|\Gamma|)$ time. The algorithm can also determine whether $|\varepsilon(K) - \varepsilon(M)| = 1$.

**Proof** Assume that we want to determine whether for a sum $\varepsilon = \sum_{i=0}^{n} \delta_i \cdot q^{\gamma_i}$ with integers $|\delta_i| \leq 2q - 2$ and $\gamma_0 < \gamma_1 < \cdots < \gamma_n$ we have $\varepsilon \leq -2$, $\varepsilon = -1$, $\varepsilon = 0$, $\varepsilon = +1$ or $\varepsilon \geq +2$. We can do this inductively using the following procedure:

1. If $\delta_n = 0$, use induction on $\varepsilon = \sum_{i=0}^{n-1} \delta_i \cdot q^{\gamma_i}$.
2. If $|\delta_n| \geq 2$, then $|\varepsilon| \geq 2 \cdot q^{\gamma_n} - \sum_{i=0}^{n-2} (2q - 2) \cdot q^i \geq 2$ and the sign of $\varepsilon$ is the same as the sign of $\delta_n$.
3. If $|\delta_n| = 1$, look at $\delta_{n-1}$. If $\delta_{n-1} = 0$ or if $\gamma_n > \gamma_{n-1} + 1$ or if $\delta_{n-1}$ has the same sign as $\delta_n$, then $|\varepsilon| \geq q^{\gamma_n} - \sum_{i=0}^{\gamma_n-2} (2q - 2) \cdot q^i = q^{\gamma_n-1}(q - 2) + 2 \geq 2$ since $q \geq 2$. Again, $\varepsilon$ has the same sign as $\delta_n$. 

© Springer
If \( \delta_{n-1} \) has the opposite sign of \( \delta_n \) and \( \gamma_n = \gamma_{n-1} + 1 \), use induction on \( \varepsilon = \hat{\delta}_{n-1} \cdot q^{\gamma_{n-1}} + \sum_{i=0}^{n-2} \delta_i \cdot q^{\gamma_i} \), where \( \hat{\delta}_{n-1} = \delta_n \cdot q + \delta_{n-1} \in \{-2q+2, \ldots, 2q-2\} \).

The answer to the original question can be found by applying this algorithm to the mapping \( M - K : \Gamma \to \{-2q+2, \ldots, 2q-2\} \) given by \( (M - K)(u) = M(u) - K(u) \).

Note, that this algorithm does not need the actual values of the \( \gamma_i \). It only asks whether \( \gamma_i = \gamma_{i-1} + 1 \). This information can be obtained from the bit vector provided by the reduced circuit. □

**Corollary 2.9** For two markings \( K \) and \( M \) in a reduced circuit, it can be tested in \( O(|\Gamma|) \) time whether \( q^{\varepsilon(K)} \) divides \( \varepsilon(M) \).

**Proof** Let \( u \) be the node of minimal value in \( \text{supp} \ M \). Then \( q^{\varepsilon(K)} \mid \varepsilon(M) \) if and only if \( q^{\varepsilon(K)} \mid \varepsilon(u) \). Using Proposition 2.8, we can check the equivalent condition \( \varepsilon(K) \leq \varepsilon(\Lambda u) \). □

Power circuits arising from a sequence of arithmetic operations are usually far from being reduced. Every cloning creates a pair of nodes with the same value. Therefore, we need an algorithm that given an arbitrary circuit produces an equivalent reduced circuit. In this context, equivalence means that for each node and each marking in the old circuit, there is one with the same value in the reduced circuit. Before giving the algorithm, we need some preparations.

**Definition 2.10** A list \( u_1, \ldots, u_k \) of nodes in a power circuit is called a chain (starting at \( u_1 \)), if \( q \cdot \varepsilon(u_i) = \varepsilon(u_{i+1}) \) for all \( 1 \leq i < k \). It is called a maximal chain (starting at \( u_1 \)), if it is not part of a longer chain (starting at \( u_1 \)).

Note that chains have nothing to do with paths in the graph. In arbitrary power circuits, chains are difficult to spot. However, in a reduced power circuit, they can be easily identified using the bit vector.

In a reduced circuit, the maximal chain starting at the unique node of value 1 is of particular interest. It is called the base chain of the power circuit. For later use, we define in Algorithm 1 a procedure PROLONGBASECHAIN prolonging this chain by one node without destroying the reducedness property of a circuit. The procedure PROLONGBASECHAIN takes \( O(|\Gamma|) \) time.

Now we can give an algorithm that reduces power circuits. Reduction is done node by node. This means that at any point during the reduction procedure, the circuit consists of a reduced part and a part that is not yet reduced. The nodes in the non-reduced part are processed in topological order. In this way, the procedure only has to work for nodes all of whose successors are already in the reduced part.

This approach allows us to generalize the reduction procedure. Instead of reducing the entire circuit, we can take into account that parts of it might already be reduced. This will turn out to be useful in applications. The procedure EXTENDREDUCTION described in Algorithm 2 takes as input not only the power circuit but also a list \( \mathcal{M} \) of markings that need to be adjusted during reduction in order to preserve their value.

 Springer
Algorithm 1: Procedure PROLONGBASECHAIN

**input**: a reduced power circuit \( \Pi = (\Gamma, \delta) \)

**output**: a reduced power circuit \( \Pi' = (\Gamma \cup \{u\}, \delta') \) which is \( \Pi \) with an additional node \( u \) prolonging the base chain of \( \Pi \)

1. Let \( \Gamma = (v_0, \ldots, v_n) \) be the ordered list of the nodes of the reduced circuit \( \Gamma \).

2. Using this list and the bit vector, find the smallest \( i \geq 0 \) such that \( \varepsilon(v_i) > q^i \) (in other words: find the first 0 in the bit vector);

3. Place \( u \) in the ordered list of nodes between \( v_{i-1} \) and \( v_i \);

4. Check whether \( q \cdot \varepsilon(u) = \varepsilon(v_i) \) by applying Proposition 2.8 to \( \Lambda_u \) and \( \Lambda_{v_i} \) (both are contained in the reduced circuit \( \Pi \)). Set the bit vector for \( u \) accordingly;

5. Insert a new node \( u \) with \( \Lambda_u = M \) into the circuit;

6. For each node \( u_i \in U \), its position in the ordering of \( \Gamma \) has to be found in step 5. Since \( u_i \) is chosen to be topologically minimal, the successor marking \( \Lambda_{v_i} \) is contained in the reduced circuit \( \Gamma, \) so \( u_i \) can be compared to any node \( v \in \Gamma \) in \( O(n) \) time. Using binary search, \( O(\log n) \) comparisons are sufficient, taking \( \tilde{O}(n \cdot |U|) \) time in total.

7. For the insertion of \( u_i \) in \( \Gamma \), we distinguish two cases. In the first one (step 6), there is no node in \( \Gamma \) with the same value as \( u_i \). In this case, \( u_i \) is moved from \( U \) to \( \Gamma \) without any modification. Markings containing \( u_i \) (including successor markings, i.e., edges with target \( u_i \)) are not affected either.

8. The second case (step 10), where there is a node \( v_j \) with the same value as \( u_i \) is more difficult. Figure 7 shows an example. The idea is to delete \( u_i \) and replace it in all markings \( M \) (both markings from \( M \) and successor markings of nodes in \( U \)) by \( v_j \). This may cause \( v_j \) to by “overmarked” by \( M \), i.e., \( M(v_j) \notin D \). For example, in the simplest case \( q = 2 \), if \( M(v_j) = M(u_i) = 1 \), then \( M(v_j) = 2 \) after the replacement. The solution is inspired by the idea of carry digits used when adding two \( q \)-ary numbers: if \( |M(v_j)| \geq q \), subtract the appropriate number \( \alpha \cdot q \) and add \( \alpha \) to the value that \( M \) assigns to the next node \( v_{j+1} \) in the chain, which has \( q \) times the value of \( v_j \). The carry might propagate to the end of the chain, which is why we preventively prolonged it by one node \( v_{k+1} \).

Proposition 2.11 (cf. [4], Thm. 6) The procedure EXTENDREDUCTION is correct and takes \( \tilde{O}((|\Gamma| + |U|) \cdot (|U| + m)) \) time. The circuit growth \( |\Gamma' \setminus \Gamma| \) is bounded by \( 2|U| \).

**Proof** At first, a topological order is computed. This can be done in linear time in terms of the size of the subgraph \( U \) (nodes and edges), which is \( O(|U|^2) \). In the main loop starting at line 2, the nodes are eliminated from \( U \) one by one. Let \( n = |\Gamma| + |U| \) be the initial size of the whole graph. Since \( \Gamma \) grows by \( O(|U|) \) during the procedure (although we keep calling it \( \Gamma \) for convenience), \( O(n) \) is the correct bound for the size of \( \Gamma \).

Since \( u_i \) is chosen to be topologically minimal, the successor marking \( \Lambda_{v_i} \) is contained in the reduced circuit \( \Gamma \), so \( u_i \) can be compared to any node \( v \in \Gamma \) in \( O(n) \) time. Using binary search, \( O(\log n) \) comparisons are sufficient, taking \( \tilde{O}(n \cdot |U|) \) time in total.

For the insertion of \( u_i \) in \( \Gamma \), we distinguish two cases. In the first one (step 6), there is no node in \( \Gamma \) with the same value as \( u_i \). In this case, \( u_i \) is moved from \( U \) to \( \Gamma \) without any modification. Markings containing \( u_i \) (including successor markings, i.e., edges with target \( u_i \)) are not affected either.

The second case (step 10), where there is a node \( v_j \) with the same value as \( u_i \) is more difficult. Figure 7 shows an example. The idea is to delete \( u_i \) and replace it in all markings \( M \) (both markings from \( M \) and successor markings of nodes in \( U \)) by \( v_j \). This may cause \( v_j \) to by “overmarked” by \( M \), i.e., \( M(v_j) \notin D \). For example, in the simplest case \( q = 2 \), if \( M(v_j) = M(u_i) = 1 \), then \( M(v_j) = 2 \) after the replacement. The solution is inspired by the idea of carry digits used when adding two \( q \)-ary numbers: if \( |M(v_j)| \geq q \), subtract the appropriate number \( \alpha \cdot q \) and add \( \alpha \) to the value that \( M \) assigns to the next node \( v_{j+1} \) in the chain, which has \( q \) times the value of \( v_j \). The carry might propagate to the end of the chain, which is why we preventively prolonged it by one node \( v_{k+1} \).
Algorithm 2: Procedure EXTEND REDUCTION

\[ \text{input: a graph } \Pi = (\Gamma \cup U, \delta) \text{ such that } (\Gamma, \delta |_{\Gamma \times \Gamma}) \text{ is a reduced power circuit, a list } M = (M_1, \ldots, M_m) \text{ of markings in } \Pi \]

\[ \text{output: a reduced power circuit } \Pi' = (\Gamma', \delta') \text{ with } \Gamma \subseteq \Gamma' \text{ and } \delta'|_{\Gamma' \times \Gamma'} = \delta|_{\Gamma \times \Gamma}, \text{ a list } M' = (M'_1, \ldots, M'_m) \text{ of markings in } \Pi' \text{ such that } \varepsilon(M_i) = \varepsilon(M'_i) \]

1. Compute a topological order of the reduced part, i.e., $U = (u_1, \ldots, u_k)$ such that $\delta(u_i, u_j) \neq 0$ implies $i > j$;
2. for $i = 1, \ldots, k$ do
   3.   $U := U \setminus \{u_i\}$;
   4.   If $\Gamma = \emptyset$, set $\Gamma := \{u_1\}$ (a circuit with just one node is obviously reduced) and continue with the iteration $i = 2$;
   5.   Let $\Gamma = (v_1, v_2, \ldots)$ be the ordered list of the nodes of the reduced circuit $\Gamma$. Using binary search, find the minimal $j$ such that $\varepsilon(u_i) \leq \varepsilon(v_j)$.
      Comparing $u_i$ to some $v_j$ is done by comparing $\Lambda_{u_i}$ to $\Lambda_{v_j}$, both markings are in the reduced part, so Proposition 2.8 applies;
   6.   if $\varepsilon(\Lambda_{u_i}) < 0 = \varepsilon(\Lambda_{v_1})$ then the graph $\Pi$ is not a power circuit; abort the algorithm. if $\varepsilon(u_i) < \varepsilon(v_j)$ (or no such $v_j$ exists) then
      7.     $\Gamma := \Gamma \cup \{u_i\}$;
      8.     Insert $u_i$ into $\Gamma$’s sorted list of nodes between $v_{j-1}$ and $v_j$;
      9.     Set the bit vector for $u_i$ according to whether $\varepsilon(\Lambda_{u_i}) + 1 = \varepsilon(\Lambda_{v_j})$;
   10. else $\varepsilon(u_i) = \varepsilon(v_j)$
      11. Find the last node $v_k$ of the maximal chain starting at $v_j$ and create $v := \text{CLONE}(v_k)$;
      12. Multiply the value of $v$ by $q$ by adding 1 to $\Lambda_{v}$: Let $v_{\ell}$ be the first node in the base chain with $\Lambda_v(v_{\ell}) < q - 1$. If such $v_{\ell}$ does not exist, call PROLONG BASE CHAIN to create it. Set $\Lambda_v(v_1) = \ldots = \Lambda_v(v_{\ell-1}) = 0$ and increment $\Lambda_v(v_{\ell})$ by one;
      13. Insert $v$ in the ordered list after $v_k$ and set the bit vector for $v$ by comparing $\Lambda_v$ to $\Lambda_{v_{k+1}}$;
   14. foreach $M \in \{\Lambda_u : u \in U\} \cup M$ with $u_i \in \text{supp } M$ do
    15. Replace $u_i$ in $M$ by $v_j$, i.e., set $M(v_j) := M(v_j) + M(u_i)$ and $M(u_i) := 0$. If now $M(v_j) = \alpha \not\in D$, write $\alpha = \beta \cdot q + \gamma$ with $\gamma \in D$, set $M(v_j) := \gamma$ and add $\beta$ to $M(v_{j+1})$. If again $M(v_{j+1}) \not\in D$, repeat. This terminates at the latest at the newly created node $v$ which is not marked by $M$;

Note that the time bound for one execution of step 14 is not $\tilde{O}(|\Gamma|)$, but rather $O(|\Gamma| \cdot \text{#of markings})$. Since this is not sufficient to prove the claimed bound, instead we count the total amount of time spent in step 14 during the whole procedure. The key observation is that for every carry that has to be moved to the next node in the
Fig. 7 One iteration of EXTEND REDUCTION ($q = 2$)

chain, the number $C := \sum_M \sum_{v \in \Gamma \cup U} |M(v)|$ decreases. Initially $C \leq (q - 1) \cdot (n \cdot (|U| + m))$, so the total time complexity of step 14 is $O(n \cdot (|U| + m))$. □

Remark 2.12 Not all existing markings in $\Gamma \cup U$ need to be included in $\mathcal{M}$. Since $\Pi$ remains a subcircuit of $\Pi'$ and the values of nodes in $\Gamma$ do not change, all markings whose support is completely contained in $\Gamma$ are automatically preserved. Only markings using nodes in $U$ have to be put into $\mathcal{M}$. In most applications, $\mathcal{M}$ consists only of a constant number of markings.

Remark 2.13 The bound for the circuit growth given in Proposition 2.11 is a rather crude one. A more detailed analysis shows that calling PROLONG BASE CHAIN is only
necessary once every time $|\Gamma|$ grows by a factor $q$. If one does some “cleaning up” in the circuit (for instance delete unmarked nodes with no incoming edges), [11] shows that the growth during reduction is even bounded by 1. However, this bound is of no importance in our applications since cloning during arithmetic operations increases the size by $O(|U|)$ anyway.

In practice, the circuit size rarely ever increases at all during reduction. Usually, the circuit even shrinks.

**Theorem 2.14** (cf. [4], Cor. 7) There is a procedure \textsc{Reduce} which given a power circuit $\Pi = (\Gamma, \delta)$ and a list $\mathcal{M} = (M_1, \ldots, M_m)$ of markings in $\Pi$, returns a reduced circuit $\Pi' = (\Gamma', \delta')$ and a list $\mathcal{M}' = (M'_1, \ldots, M'_m)$ of markings in $\Pi'$ such that $\varepsilon(M_i) = \varepsilon(M'_i)$ $(1 \leq i \leq m)$. \textsc{Reduce} takes $\tilde{O}(|\Gamma|^2 + |\Gamma| \cdot m)$ time and the size of $\Gamma'$ is bounded by $2|\Gamma|$.

**Proof** Invoke \textsc{ExtendReduction} taking $\emptyset$ as the reduced part and the whole circuit as $U$. \hfill \Box

Step 6 in \textsc{ExtendReduction} tests whether $\varepsilon(\Lambda_u) \geq 0$. This is one of the equivalent conditions specified in Lemma 2.2 for a graph to be a power circuit. Therefore, reduction is at the same time a test whether a graph is a power circuit:

**Corollary 2.15** ([4], Cor. 8) Given a dag $\Pi = (\Gamma, \delta)$ it can be determined in $\tilde{O}(|\Gamma|^2)$ time whether $\Pi$ is a power circuit.

### 2.4 Compactness

In this section, we will show that using a richer data structure for reduced circuits, the time complexity of \textsc{ExtendReduction} can be reduced to $O((|\Gamma| + |U|) \cdot |U|)$. This eliminates the logarithmic factors both for \textsc{Reduce} and for the test presented in Corollary 2.15. We start by taking a closer look at power sums.

#### 2.4.1 Compact Power Sums

A power sum is a formal sum $S = \sum_{i \geq 0} \alpha_i \cdot q^i$ whose coefficients $\alpha_i$ are in $D$ and only finitely many of them are non-zero. The integer value of the power sum $S$ is denoted $\varepsilon(S)$. On the set of all power sums, we define a rewriting system $\mathcal{P}$ generated by the rules

1. $\alpha \cdot q^i + \beta \cdot q^{i+1} \rightarrow (\alpha - q) \cdot q^i + (\beta + 1) \cdot q^{i+1}$ for $\alpha > 0$, $\beta < 0$,
2. $\alpha \cdot q^i + \beta \cdot q^{i+1} \rightarrow (\alpha + q) \cdot q^i + (\beta - 1) \cdot q^{i+1}$ for $\alpha < 0$, $\beta > 0$,

and for $i < j$

3. $\alpha \cdot q^i + (q - 1) \cdot (q^{i+1} + \cdots + q^j) + \beta \cdot q^{i+1}$
   \quad $\rightarrow (\alpha - q) \cdot q^i + (\beta + 1) \cdot q^{i+1}$ for $\alpha > 0$, $\beta < q - 1$,
4. $\alpha \cdot q^i + (-q + 1) \cdot (q^{i+1} + \cdots + q^j) + \beta \cdot q^{i+1}$
   \quad $\rightarrow (\alpha + q) \cdot q^i + (\beta - 1) \cdot q^{i+1}$ for $\alpha < 0$, $\beta > -q + 1$. 

© Springer
None of these rules changes the value of the power sum. We omit the proof for the following lemma, since it consists of a long but simple enumeration of cases.

**Lemma 2.16** The rewriting system $\mathcal{P}$ is locally confluent.

**Lemma 2.17** The rewriting system $\mathcal{P}$ is terminating (noetherian) and hence confluent ([2], Thm. 1.1.13).

**Proof** Let $S_1 \Rightarrow^* S_2 \Rightarrow^* \cdots$ be a sequence of rewritings. Since none of the rules of $\mathcal{P}$ increases the number of non-zero coefficients, this number must eventually reach a minimum. Thus, ignoring a finite number of terms, we can assume that the number of non-zeros is constant within the sequence. No rule in $\mathcal{P}$ moves a non-zero coefficient to the left (in the direction of smaller exponents). As the value of the $S_i$ is fixed, non-zeros cannot be moved indefinitely to the right either. Again, by disregarding a finite prefix of the sequence, we assume that the positions of the non-zero coefficients are fixed. At this point, no application of (3) or (4) is possible any more. Finally, rules of type (1) and (2) move pairs of consecutive coefficients with opposite signs to the left (or remove them), which can also occur only finitely often. Thus, the sequence $S_1, S_2, \ldots$ is eventually constant. \hfill $\square$

We call power sums that are irreducible with respect to $\mathcal{P}$ compact. If $S = \sum_{i \geq 0} \alpha_i q^i$ is compact, then so is $-S = \sum_{i \geq 0} (-\alpha_i) q^i$.

**Proposition 2.18**

(i) For each power sum there is a unique compact power sum of the same value.

(ii) Compact power sums have the minimal number of non-zero coefficients among all power sums of the same value.

(iii) If $S$ and $T$ are compact power sums, then $\varepsilon(T) = \varepsilon(S) + 1$ if and only if

\[
S = U \cdot q^{i+1} + \alpha_i \cdot q^i + \sum_{j=0}^{i-1} \alpha_j \cdot q^j \quad \text{and} \\
T = U \cdot q^{i+1} + \beta_i \cdot q^i + \sum_{j=0}^{i-1} \beta_j \cdot q^j
\]

for some power sum $U$, $\beta_i = \alpha_i + 1$, and for each $0 \leq j < i$ either $\alpha_j = q - 1$ and $\beta_j = 0$ or $\alpha_j = 0$ and $\beta_j = -q + 1$.

(iv) The usual order on $D$ gives rise to a lexicographical order on power sums (where coefficients of higher powers of $q$ are compared before those of lower powers). Restricted to compact power sums, this lexicographical order coincides with the order by values.

**Proof** For (i) it suffices to show that for two power sums $S$ and $T$ of the same value, we have $S \not\Rightarrow^* T$. This is true, since applying the rules of $\mathcal{P}$ forward or backward,
one can turn any power sum into an ordinary \( q \)-ary number with coefficients from \{0, \ldots, q - 1\} (\{-q + 1, \ldots, 0\} if the value is negative). For instance, for positive values of \( S \), use rules of type (1) backward and rules of type (2) forward to push negative coefficients to the right.

Claim (ii) follows from the fact that no rule increases the number of non-zero coefficients.

For the “if” part of (iii), we observe that component-wise subtraction yields

\[
T - S = q^i - (q - 1) \sum_{j=0}^{i-1} q^j,
\]

which evaluates to 1. For the “only if” part, let \( S = \sum_{\ell \geq 0} \alpha_{\ell} \cdot q^{\ell} \) be compact. Consider \( S' = \sum_{\ell \geq 1} \alpha_{\ell} \cdot q^{\ell} + (\alpha_0 + 1) \). If \( S' \) is a valid power sum (i.e. \( \alpha_0 + 1 \in D \)) and \( S' \) is compact, it already has the desired form (for \( i = 0 \)). Otherwise we have one of the following cases:

1. \( S' \) is not a valid power sum, since \( \alpha_0 = q - 1 \). Let \( k \geq 0 \) be the maximum number such that \( \alpha_0 = \cdots = \alpha_k = q - 1 \). We transform \( S' \) into

\[
(\alpha_{k+1} + 1) \cdot q^{k+1} + \sum_{\ell > k+1} \alpha_{\ell} \cdot q^{\ell}
\]

and use induction on \( (\alpha_{k+1} + 1) \cdot q^0 + \sum_{\ell > k+1} \alpha_{\ell} \cdot q^{\ell-k-1} \).

2. A rule of type (1) can be applied. We have \( \alpha_0 = 0 \) and \( \alpha_1 < 0 \). Applying the rule gives

\[
S' \Rightarrow (1) \sum_{\ell \geq 2} \alpha_{\ell} \cdot q^{\ell} + (\alpha_1 + 1) \cdot q + (-q + 1).
\]

Use induction on \( \sum_{\ell \geq 2} \alpha_{\ell} \cdot q^{\ell-1} + (\alpha_1 + 1) \).

3. A rule of type (3) can be applied. We have \( \alpha_0 = 0 \) and \( \alpha_1 = \cdots = \alpha_k = q - 1 \) and \( \alpha_{k+1} < q - 1 \) for some \( k \geq 1 \). This yields

\[
S' \Rightarrow (3) \sum_{\ell > k+1} \alpha_{\ell} \cdot q^{\ell} + (\alpha_{k+1} + 1) \cdot q^{k+1} + (-q + 1)
\]

and induction applies to \( \sum_{\ell > k} \alpha_{\ell} \cdot q^{\ell-k-1} + (\alpha_{k+1} + 1) \).

Finally, (iv) is a consequence of (iii).

The notion of compactness was introduced in [11] for \( q = 2 \) and subsequently used in [5]. Our definition is a generalization that inherits most of the original characteristics.

There is, however, one important difference: it is much less obvious for \( q > 2 \) how to make a power sum compact in linear time. In the case \( q = 2 \) it suffices to apply the rules of \( \mathcal{P} \) from left to right. Yet, if for instance \( q = 3 \), the application of rule (1)
to

\[
1 + q + q^2 + q^3 - 2q^4 \implies 1 + q + q^2 - 2q^3 - q^4
\]

(1)

turns the previously compact prefix \(1 + q + q^2 + q^3\) into the \(\mathcal{P}\)-reducible sum \(1 + q + q^2 - 2q^3\).

**Proposition 2.19** Any power sum \(S = \sum_{i=0}^{n} \alpha_i \cdot q^i\) can be transformed into a compact power sum with the same value in \(O(n)\) time.

**Proof** Any two power sums \(S\) and \(T\) with \(\varepsilon(S) = \varepsilon(T)\) can be transformed into each other using only replacements of the form

\[\alpha \cdot q^{i-1} + \beta \cdot q^i \longrightarrow (\alpha \pm q) \cdot q^{i-1} + (\beta \mp 1) \cdot q^i,\]

(\(\ast_i^\pm\))

i.e., rules (1) and (2) without the restrictions.

Moreover, at most one application of \((\ast_i^+)\) or \((\ast_i^-)\) is needed for each \(i\). In fact, whether \((\ast_i^+)\) or \((\ast_i^-)\) or neither is needed, depends only on \(S\) and \(T\). This can be seen using induction on \(i\).

Let \(T = \sum_{i=0}^{n+1} \beta_i \cdot q^i\) be the compact power sum with \(\varepsilon(S) = \varepsilon(T)\). Define \(J_i \in \{+1, -1, 0\}\) (\(1 \leq i \leq n\)) depending on whether the replacement \((\ast_i^+)\) or \((\ast_i^-)\) or neither of them occurs in the sequence \(S \Longrightarrow^* T\). For notational convenience, we define \(J_0 = J_{n+2} = 0\) and \(\alpha_{n+1} = 0\). Then we have \(\beta_i = \alpha_i + J_i - q \cdot J_{i+1}\) for \(0 \leq i \leq n + 1\).

It remains to compute the values \(J_i\) \((1 \leq i \leq n + 1)\). These are the unique solution of the following system of conditions \((0 \leq i \leq n + 1)\):

\[
\begin{aligned}
(\Diamond_i) \quad &\alpha_{i-1} + J_{i-1} - q \cdot J_i \in D \text{ and } \alpha_i + J_i - q \cdot J_{i+1} \in D \text{ and} \\
&\text{sign}(\alpha_{i-1} + J_{i-1} - q \cdot J_i) \cdot \text{sign}(\alpha_i + J_i - q \cdot J_{i+1}) \neq -1 \text{ and} \\
&\text{if sign}(\alpha_{i-1} + J_{i-1} - q \cdot J_i) = \text{sign}(\alpha_i + J_i - q \cdot J_{i+1}) = \pm 1, \\
&\text{then } \alpha_i + J_i - q \cdot J_{i+1} \neq \pm (q - 1).
\end{aligned}
\]

For \(i = 2, \ldots, n + 1\) and for all \(j, k \in \{-1, 0, +1\}\) we define \(\mathcal{J}_i[j, k]\) to be the set of possible values for \(J_i\), provided that \(J_{i-2} = j\) and \(J_{i-1} = k\):

\[
\mathcal{J}_i[j, k] := \{ J_i : \text{there are } J_1, \ldots, J_{i-3} \text{ such that} \\
J_0 = 0, J_1, \ldots, J_{i-3}, J_{i-2} = j, J_{i-1} = k, J_i \\
\text{satisfy } (\Diamond_0), \ldots, (\Diamond_{i-1}) \}.
\]

Since \((\Diamond_{i-1})\) only depends on \(J_{i-2}, J_{i-1}, J_i, \alpha_{i-2}, \alpha_{i-1}\), the sets \(\mathcal{J}_i[\cdot, \cdot]\) can be computed in constant time using \(\mathcal{J}_{i-1}[\cdot, \cdot]\). After this, the solution \(J_1, \ldots, J_{n+1}\) can be read from right to left. \(\square\)
Example 2.20 Suppose that $q = 3$ and we want to make $S = 1 + q - 2q^2 - 2q^3 + q^4 - q^5$ compact. We get:

$$
\begin{array}{cccccc}
J_i[j,k] & i = 2 & i = 3 & i = 4 & i = 5 & i = 6 & i = 7 \\
\hline
j = -1, k = -1 & \emptyset & \emptyset & \emptyset & \{0\} & \emptyset & \emptyset \\
j = -1, k = 0 & \emptyset & \emptyset & \emptyset & \emptyset & \{-1, 0\} & \emptyset \\
j = -1, k = +1 & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset \\
j = 0, k = -1 & \emptyset & \emptyset & \{-1\} & \emptyset & \emptyset & \emptyset \\
j = 0, k = 0 & \{0\} & \{-1\} & \emptyset & \emptyset & \emptyset & \{0\} \\
j = 0, k = +1 & \{+1\} & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset \\
j = +1, k = -1 & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset \\
j = +1, k = 0 & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset \\
j = +1, k = +1 & \emptyset & \{0\} & \emptyset & \emptyset & \emptyset & \emptyset \\
\end{array}
$$

Since $J_7 = 0$, we deduce from the last column that $J_5 = J_6 = 0$ as well. The only 0 in the column for $i = 6$ with $k = 0$ yields $J_4 = -1$ and so on. We end up with $J_0 = J_1 = J_2 = 0, J_3 = J_4 = -1, J_5 = J_6 = J_7 = 0$ which results in $T = 1 + q + q^2 - q^5$.

2.4.2 Power Circuits and Trees

Property (iv) of Proposition 2.18 is the main motivation for the following definition.

Definition 2.21 (cf. [5], Def. 5) A power circuit $\Pi = (\Gamma, \delta)$ is called treed (a convenient shorthand for “reduced and equipped with additional data in the form of a tree”), if

(i) $\Pi$ is reduced (i.e., different nodes evaluate to different values and it is equipped with an ordered list $\Gamma = (u_1, \ldots, u_n)$ and a bit vector, see Definition 2.7)

(ii) $\Pi$ is equipped with a directed tree $T$ in which each node has up to $|D|$ outgoing edges, labelled with pairwise distinct values from $D$ (and ordered from left to right by increasing values). For each leaf in $T$, the unique path from the root to this leaf must consist of exactly $n = |\Gamma|$ edges. The sequence of labels $\alpha_1, \ldots, \alpha_n$ of such a path (read from the leaf to the root) corresponds to a marking $M : u_i \mapsto \alpha_i$. For all leaves, the power sum $\sum_{i=1}^n \alpha_i \cdot \varepsilon(u_i)$ (which evaluates to $\varepsilon(M)$) must be compact. Any marking represented by a leaf in this way is called compact. (A leaf can represent multiple markings, of course.)

(iii) All successor markings $\Lambda_u (u \in \Gamma)$ are compact.

(iv) For each level in the tree, there is a list containing the nodes of this level.

Example 2.22 Figure 8 shows an example of a treed circuit (for $q = 2$) alongside the tree containing its markings. Note that the order of the nodes is implicitly given by the lowest level of the tree.

The most time consuming step in EXTENDREDUCTION was to find the position of a new node in the sorted list of $\Gamma$. Using binary search, this took $O(n \log n)$ time. If $\Gamma$ is treed and $\Lambda_u$ is compact, we can improve this to $O(n)$: the position of the leaf corresponding to $\Lambda_u$ in the list of leaves already tells the position of $u$ in $\Gamma$. In
order to adjust the bit vector, we have to read the paths from the root of the tree to the respective leaves and check the condition given by Proposition 2.18(iii). Yet, making a circuit treed is more complicated than just reducing it.

For the time analysis, we use amortization with respect to a potential function pot, mapping power circuits to numbers, see 17.3 in [3]. An algorithm on power circuits is said to run in amortized time $t$, if the real running time is bounded by $t + \text{pot}(\Pi) - \text{pot}(\Pi')$, where $\Pi$ is the input and $\Pi'$ is the resulting circuit. Thus, an algorithm may take longer than its indicated amortized time, as long as it decreases the potential by the same amount.

The number of leaves in a treed power circuit depends not only on the circuit itself (i.e. $\Gamma$ and $\delta$) but also on the number of compact markings that are not successor markings. Furthermore, a power circuit usually grows in the course of the execution of an algorithm. In order to keep time analysis as simple as possible, we take $n$ to be an upper bound on the number of nodes and leaves.

**Definition 2.23** For a power circuit $\Pi = (\Gamma, \delta)$, the number of maximal chains is denoted $\text{ch}(\Pi)$. The potential of $\Pi$ is $\text{pot}(\Pi) = \text{ch}(\Pi) \cdot n$.

In a situation where $\Pi = (\Gamma \cup U, \delta)$ is a graph with an embedded power circuit $\Pi' = (\Gamma, \delta|_{\Gamma \times \Gamma})$, we define $\text{ch}(\Pi) := \text{ch}(\Pi')$ and $\text{pot}(\Pi) := \text{pot}(\Pi')$. 
Lemma 2.24 There is a procedure INSERTNODE, which takes a treed power circuit \( \Pi = (\Gamma, \delta) \) and a compact marking \( M \) in \( \Pi \) and which inserts a new node \( \nu \) with \( \Lambda_{\nu} = M \) into \( \Pi \), leaving the circuit treed. If a node with value \( q^{\epsilon(M)} \) already exists in \( \Pi \), then no new node \( \nu \) is created. INSERTNODE runs in amortized time \( O(n) \).

Proof Since \( M \) is compact, the position of \( \nu \) in the ordered list of nodes is determined by the position of the leaf corresponding to \( M \) in the tree. If the same leaf is also associated with some successor marking, a node with value \( q^{\epsilon(M)} \) already exists and the procedure is aborted. Otherwise, the bit vector can be adjusted by comparing \( M \) to the successor markings of the nodes immediately before and after \( \nu \). The tree has to be “stretched” by inserting a new level corresponding to \( \nu \). All edges on this level are labelled 0, since no marking contains the new node \( \nu \). Using the lists of nodes of the same level, this takes \( O(n) \) time.

The insertion of \( \nu \) may increase \( ch(\Pi) \) by one. Thus, the potential grows by up to \( n \).

Incrementing a compact marking by 1 is easier than doing the same with an arbitrary marking. Let \( M \) be a compact marking in a treed circuit, in which \( u_1, u_2 \) are the unique nodes with values 1 and \( q \) respectively (assuming they exist). If \( M(u_1) < q - 1 \), \( M(u_1) \) can be incremented directly. If \( M(u_1) = q - 1 \), then \( M(u_2) < q - 1 \) thanks to compactness. Set \( M(u_1) := 0 \) and increment \( M(u_2) \). However, the resulting marking is not always compact.

Lemma 2.25 There is a procedure COMPACTIFYMARKING which, given an arbitrary marking \( M \) in a treed circuit \( \Pi = (\Gamma, \delta) \), computes in \( O(n) \) time a compact marking \( M' \) in \( \Pi \) with \( \epsilon(M') = \epsilon(M) \), given that the following condition holds:

\[
\text{For each maximal chain } C \subseteq \Gamma, \text{ either the power sum } \epsilon(M|_C) \text{ is compact or } \text{supp } M \text{ does not contain the top node of } C. \tag{1}
\]

Proof This is based on the following observation: if \( S = \sum_{i=k}^{\ell} \alpha_i \cdot q^i \) is a power sum, only the coefficients of \( q^k, \ldots, q^{\ell+1} \) can be non-zero in the corresponding compact power sum. For each maximal chain \( C \subseteq \Gamma \), we can apply the algorithm from Proposition 2.19 to the power sum defined by \( M|_C \) individually. Condition 2.25 ensures that a node for \( q^{\ell+1} \) exists in \( \Pi \), if needed. After that, we create a path for the marking in the tree.

In the case of PROLONGBASECHAIN, Condition (1) is satisfied. The successor marking of the new top node of the base chain only uses nodes further down in that chain (approximately the first \( \log_q i \) nodes) which, by definition, already exist. This shows:

Corollary 2.26 The procedure PROLONGBASECHAIN given in Algorithm 1 can be adapted for treed power circuits. The amortized time complexity is \( O(n) \).
Corollary 2.27 There is a procedure \textsc{INCREMENT MARKING} which, given a compact marking $M$ in a treed power circuit $\Pi = (\Gamma, \delta)$, computes a compact marking $M'$ in a (possibly enlarged) circuit $\Pi'$ with $\epsilon(M') = \epsilon(M) + 1$. \textsc{INCREMENT MARKING} takes $\mathcal{O}(n)$ amortized time and increases the circuit size by $1 + \text{ch}(\Pi) - \text{ch}(\Pi')$.

Proof We invoke \textsc{PROLONG BASE CHAIN} which creates a new node $u$. If $u$ is not the new top node of the base chain, the insertion of $u$ has linked two maximal chains, decreasing $\text{ch}(\Pi)$ by one. We repeat calling \textsc{PROLONG BASE CHAIN} until we have created a new top node of the base chain which is not used by any marking. As discussed above, incrementing $M$ by 1 only affects the two nodes with values 1 and $q$. As a consequence, the compactification process of the incremented marking is limited to the base chain of $\Pi$ and stops at the top node we just created.

Assume we need $k$ calls to \textsc{PROLONG BASE CHAIN}, each taking $n$ time (for simplicity we ignore the constant hidden in the $\mathcal{O}$ notation). The first $k-1$ calls decrease the potential by $n$ each. Thus, we get amortized time $n$ plus another $\mathcal{O}(n)$ for compactification.

For treed circuits, the procedure \textsc{EXTEND TREE} given in Algorithm 3 replaces \textsc{EXTEND REDUCTION}. Just like \textsc{EXTEND REDUCTION}, it preserves all (now compact) markings that are completely contained in $\Gamma$. Only those markings that use nodes from $U$ need to be included in the list $M$.

Proposition 2.28 The procedure \textsc{EXTEND TREE} is correct and runs in amortized time $\mathcal{O}(n \cdot (|U| + m))$. The circuit growth $|\Gamma' \setminus \Gamma|$ is bounded by $4 \cdot |U| + \text{ch}(\Pi) - \text{ch}(\Pi')$.

Proof The basic structure of \textsc{EXTEND TREE} is the same as that of \textsc{EXTEND REDUCTION}, so we focus on the differences. Between cycles of the main loop, we keep up the following invariants for all markings $M \in \mathcal{M} \cup \{\Lambda_u : u \in U\}$:

(a) If the support of $M$ is completely contained in $\Gamma$, $M$ is compact.
(b) For each maximal chain $C \subseteq \Gamma$, either the power sum $\epsilon(M|_C)$ is compact or the top node of $C$ is not contained in supp $M$.

The second invariant is true by assumption, the first one is established in step 1.

The time for finding $v_j$ in step 6 is reduced to $\mathcal{O}(n)$ due to the representation of markings as leaves.

If $\Gamma$ contains no node with the same value as $u_i$, we can insert it as we did in \textsc{EXTEND REDUCTION}. Remember that $\Lambda_{u_i}$ is compact due to (a). The case $\epsilon(u_i) = \epsilon(v_j)$ also resembles \textsc{EXTEND REDUCTION} and has the same (although now amortized) time bound.

Let $C \subseteq \Gamma$ be the maximal chain starting at $v_j$. If a marking $M \in \mathcal{M} \cup \{\Lambda_u : u \in U\}$ had $u_i$ in its support, the power sum $\epsilon(M|_C)$ is probably not compact anymore after replacing $u_i$ in $M$ by $v_j$. In order to restore (b), we create a new top node for $C$ in step 13. As we have seen in Corollary 2.27, this takes amortized time $\mathcal{O}(n)$ and increases the circuit size by $1 + \text{ch}(\Pi) - \text{ch}(\Pi')$. 

© Springer
Algorithm 3: Procedure EXTEND TREE

**input**: a graph \( \Pi = (\Gamma \cup U, \delta) \) with \( \delta|_{\Gamma \times U} = 0 \) and \( (\Gamma', \delta'|_{\Gamma \times \Gamma}) \) a treed power circuit, a list \( \mathcal{M} = (M_1, \ldots, M_m) \) of markings in \( \Pi \). For all \( M \in \mathcal{M} \cup \{ \Lambda_u : u \in U \} \), the power sum \( \varepsilon(M|_{\Gamma}) \) must be compact.

**output**: a treed power circuit \( \Pi' = (\Gamma', \delta') \) with \( \Gamma \subseteq \Gamma' \) and \( \delta'|_{\Gamma \times \Gamma} = \delta|_{\Gamma \times \Gamma} \), a list \( \mathcal{M}' = (M'_1, \ldots, M'_m) \) of compact markings in \( \Pi' \) such that \( \varepsilon(M_i) = \varepsilon(M'_i) \) for each marking \( M \in \mathcal{M} \cup \{ \Lambda_u : u \in U \} \) with \( \text{supp} M \subseteq \Gamma \).

1. **foreach** marking \( M \in \mathcal{M} \cup \{ \Lambda_u : u \in U \} \) with \( \text{supp} M \subseteq \Gamma \) **do**
2.   **COMPACTIFY MARKING** \( (M) \)
3. Compute a topological order \( U = (u_1, \ldots, u_k) \);
4. **for** \( i = 1, \ldots, k \) **do**
5.   \( U := U \setminus \{ u_i \} \);
6.   **if** \( \Gamma = \emptyset \) **then** set \( \Gamma := \{ u_1 \} \), create a tree for \( u_1 \) and insert all markings with support \( \{ u_1 \} \). Continue with \( i = 2 \). Traversing the lowest level of the tree, find the first node \( v_j \) in the ordered list \( \Gamma = (v_1, v_2, \ldots) \) such that \( \varepsilon(u_i) \leq \varepsilon(v_j) \);
7.   **if** \( \varepsilon(\Lambda_{u_i}) < 0 \) **then** abort the algorithm. **if** \( \varepsilon(u_i) < \varepsilon(v_j) \) (or no such \( v_j \) exists) **then**
8.     Insert \( u_i \) into \( \Gamma \) using **INSERT NODE** \( (\Lambda_{u_i}) \);
9. **else** \( \varepsilon(u_i) = \varepsilon(v_j) \)
10.   Prolong the maximal chain \( \ldots, v_j, v_{j+1}, \ldots, v_k \) by a new node by creating a copy of \( \Lambda_{v_k} \) and applying **INCREMENT MARKING** and **INSERT NODE**;
11. **foreach** marking \( M \) with \( u_i \in \text{supp} M \) **do**
12.   Replace \( u_i \) in \( M \) with \( v_j \) by adding \( M(u_i) \) to \( M(v_j) \) and setting \( M(u_i) := 0 \). If, after this, \( M(v_j) \notin D \), add \( \pm q \) to \( M(v_j) \) and \( \pm 1 \) to \( M(v_{j+1}) \). Repeat if \( M(v_{j+1}) \notin D \) and so on.
13. **repeat**
14.   Prolong the maximal chain starting at \( v_j \) using **INCREMENT MARKING** and **INSERT NODE**
15. **until** the newly created node is the top of the chain;
16. **foreach** marking \( M \in \mathcal{M} \cup \{ \Lambda_u : u \in U \} \) for which \( u_i \) was the last node not in \( \Gamma \) **do**
17.   **COMPACTIFY MARKING** \( (M) \)

All markings \( M \) whose support is completely contained in \( \Gamma \) after the processing of \( u_i \) must be made compact in order to maintain (a). This is done in step 16. Invariant (b) ensures that Condition (1) of Lemma 2.25 holds. \( \square \)

**Theorem 2.29** (Treed analogue of Theorem 2.14) There is a procedure MAKE TREE which given a power circuit \( \Pi = (\Gamma, \delta) \) and a list \( \mathcal{M} = (M_1, \ldots, M_m) \) of markings in \( \Pi \), returns a treed circuit \( \Pi' = (\Gamma', \delta') \) and a list \( \mathcal{M}' = (M'_1, \ldots, M'_m) \) of compact...
markings in $\Pi'$ such that $\varepsilon(M_i) = \varepsilon(M'_i)$ ($1 \leq i \leq m$). MAKE_TREE takes $O(n^2 + n \cdot m)$ time and the size of $\Gamma'$ is bounded by $4|\Gamma|$.

2.4.3 Working with Treed Power Circuits

Before we move on to group theory, let us discuss how reduced or treed power circuits can be used in applications.

The usual strategy when solving a problem using power circuits is to create one power circuit and keep all integers as markings in this circuit. The power circuit is kept in treed form in order that comparisons can be done efficiently at any time. For each arithmetic operation, the markings corresponding to the operands are cloned, and the operation (addition or multiplication by a power of $q$, see Sect. 2.2) is performed on the clones. Finally, EXTEND_TREE is called with the set of clones as $U$ to regain a treed circuit. The marking that holds the result of the operation will usually contain nodes from $U$ and must therefore be put into the list $M$. EXTEND_TREE takes $O(n \cdot |U|)$ time and this time bound also absorbs everything else done during the operation. Note that the precondition of EXTEND_TREE that the $\Gamma$ part of all markings be compact, is satisfied. The cloned markings are completely outside $\Gamma$ and their successor markings are unions of compact markings in $\Gamma$ and nodes outside $\Gamma$.

For an estimate of the time complexity of an entire algorithm, we need to keep track of the circuit size $|\Gamma|$ as well as the sum $\omega$ of the sizes of the supports of the markings (those storing data, not the successor markings). The number $\omega$ is usually called the “weight” of the circuit. It is an upper bound on the growth of the circuit during each operation. Quite often, $\omega$ remains constant during the algorithm, e.g. if after each operation only the result is kept and the operands (i.e., their markings) are discarded. If this is the case and if we start with a comparatively small circuit, then after $s$ operations the circuit size is bounded by $O(s \cdot \omega)$. Hence, each call of EXTEND_TREE takes $O((s \cdot \omega) \cdot \omega)$ time which sums up to $O(s^2 \cdot \omega^2)$. In our main application—the solution of the word problem in Higman’s group—$s$ will turn out to be quadratic and $\omega$ linear in the input size $n$, leading to an $O(n^6)$ time algorithm.

Seen from the outside, EXTEND_REDUCED and EXTEND_TREE as well as REDUCE and MAKE_TREE behave very much alike. In applications, all four procedures are used as “black boxes” and of the resulting circuits only the weaker property of reducedness is used. Therefore, in order to simplify nomenclature, we will speak of “reduction” and “reduced” circuits, subsuming both concepts. The reader may then choose whether to use the simpler reduction concept at the cost of logarithmic factors or to go through the more complicated procedures for treed power circuits with better asymptotic time complexity.

3 The Word Problem in Generalized Baumslag-Gersten Groups

Although the main goal of this paper is to solve the word problem in Higman’s groups, we sidetrack briefly to present another generalization that is made possible by power circuits with arbitrary base $q$. 
The Baumslag-Gersten group is defined as
\[ G_{(1,2)} = \langle a, b \mid a^{ab} = a^2 \rangle = \langle a, b \mid (bab^{-1})a(bab^{-1})^{-1} = a^2 \rangle \cong \langle a, b, t \mid tat^{-1} = a^2, bab^{-1} = t \rangle. \]

This is an HNN extension of BS\(_{(1, 2)}\) generated by \(a\) and \(t\). Replacing BS\(_{(1, 2)}\) by BS\(_{(1, q)}\) (for \(q \geq 2\)), we get a family of generalized Baumslag-Gersten groups:
\[ G_{(1, q)} = \langle a, b \mid a^{ab} = a^q \rangle \cong \langle a, b, t \mid tat^{-1} = a^q, bab^{-1} = t \rangle. \]

**Theorem 3.1** The word problem for the generalized Baumslag-Gersten group \(G_{(1, q)}\) is solvable in \(O(n^3)\) time.

The proof of Theorem 3.1 is literally the same as that for \(G_{(1, 2)}\) which was given in [4] and [5], except that the new base \(q\) power circuits from Sect. 2 are used.

### 4 The Word Problem in Generalized Higman Groups

We generalize the groups \(H_f\) defined in the introduction by replacing the underlying Baumslag-Solitar group BS\(_{(1, 2)}\) by BS\(_{(1, q)}\).

**Definition 4.1** The (generalized) Higman group \(H_f(1, q)\) is defined as
\[ H_f(1, q) = \langle a_1, \ldots, a_f \mid a_{i+1}a_i^{-1} = a_i^q (i \in \mathbb{Z}/f\mathbb{Z}) \rangle. \] (2)

While \(H_f = H_f(1, 2)\) \((f > 4)\) retains all the important properties of \(H_4\) (infinite, huge compression, no non-trivial normal subgroup of finite index), this is not entirely true for \(H_f(1, q)\) in general. For example, for all \(f \geq 1\), the homomorphism given by
\[ H_f(1, 3) \twoheadrightarrow \mathbb{Z}/2\mathbb{Z}; \]
\[ a_1 \mapsto 1, \]
\[ a_2, \ldots, a_f \mapsto 0 \]
sends \(H_f(1, 3)\) onto a finite non-trivial group.

In this section, we will prove:

**Theorem 4.2** Let \(q \geq 2, f \geq 4.\) The word problem for the generalized Higman group \(H_f(1, q)\) can be solved in \(O(n^6)\) time.
The key observation for the solution of the word problem is the decomposition of $H_f(1, q)$ into a series of amalgamations of $f$ copies of the Baumslag-Solitar group $BS(1, q) = \langle a, t \mid tat^{-1} = t^q \rangle$, see [13]. We have

$$H_f(1, q) = G_{1,\ldots, f-1} \ast_{F_{1, f-1}} G_{f-1, f, 1},$$

where

$$G_{1,\ldots, f-1} = \langle a_1, \ldots, a_{f-1} \mid a_{i+1}a_i^{-1} = a_i^q (1 \leq i < f - 1) \rangle \quad \text{and} \quad G_{f-1, f, 1} = \langle a_{f-1}, a_f, a_1 \mid a_f a_{f-1} a_f^{-1} = a_f^{-q}, a_1 a_f a_1^{-1} = a_f^q \rangle$$

and in both cases $F_{1, f-1}$ is the subgroup generated by $a_1$ and $a_{f-1}$, which in fact freely generate $F_{1, f-1}$ (if $f \geq 4$). Furthermore, we can break $G_{1,\ldots, f-1}$ and $G_{f-1, f, 1}$ down to

$$G_{1,\ldots, f-1} = G_{1,2} \ast_{F_2} G_{2,3} \ast_{F_3} \ldots \ast_{F_{f-2}} G_{f-2, f-1}$$

and

$$G_{f-1, f, 1} = G_{f-1, f} \ast_{F_f} G_{f, 1},$$

where $G_{i,i+1} = \langle a_i, a_{i+1} \mid a_{i+1}a_i^{-1} = a_i^q \rangle$ and the indices are read in $\mathbb{Z}/f\mathbb{Z}$.

Each group $G_{i,i+1}$ is a copy of the Baumslag-Solitar group $BS(1, q)$ and thus isomorphic to the semidirect product $\mathbb{Z}[1/q] \rtimes \mathbb{Z}$ which consists of pairs $(u, k) \in \mathbb{Z}[1/q] \rtimes \mathbb{Z}$. The isomorphism is given by $a_i \mapsto (1, 0)$ and $a_{i+1} \mapsto (0, 1)$. In $\mathbb{Z}[1/q] \rtimes \mathbb{Z}$, we have the following formulae for multiplication and inversion:

$$(u, k)(v, \ell) = (u + v \cdot q^k, k + \ell)$$

$$(u, k)^{-1} = (-u \cdot q^{-k}, -k)$$

When dealing with more than one group $G_{i,i+1}$, we add $i$ as a subscript to those pairs designating an element of $G_{i,i+1}$.

In order to solve the word problem for $H_f(1, q)$, we first need a solution for the subgroup membership problem of $F_{1,e}$ in $G_{1,\ldots,e}$ (with $e \geq 3$; this covers both $G_{1,\ldots,f-1}$ and $G_{f-1, f, 1}$). Furthermore, we have to do this in an effective way, i.e., given a sequence of pairs $(u, k)_i$ which represents an element of $F_{1,e}$, we have to find a corresponding sequence of pairs of the form $(u, 0)_1$ and $(0, \ell)_{e-1}$.

We start by giving a reduction system $\mathcal{L}$ for $G_{1,\ldots,e}$:

1. $(u, k)_i(v, \ell)_i \rightarrow (u + v \cdot q^k, k + \ell)_i$ for $1 \leq i \leq e$
2. $(u, k)_i(v, 0)_{i+1} \rightarrow (u, k + v)_i$ for $1 \leq i < e$ and $v \in \mathbb{Z}$
3. $(u, 0)_{i+1}(v, \ell)_i \rightarrow (v \cdot q^u, \ell + u)_i$ for $1 \leq i < e$ and $u \in \mathbb{Z}$
4. $(u, k)_{i+1}(0, \ell)_i \rightarrow (u + \ell \cdot q^k, k)_{i+1}$ for $1 \leq i < e$
5. $(0, k)_{i+1}(v, \ell)_{i+1} \rightarrow (k + v, \ell)_{i+1}$ for $1 \leq i < e$

The system $\mathcal{L}$ is not confluent in general, but the following property holds:

**Proposition 4.3** If $w$ is an $\mathcal{L}$-reduced word that equals 1 in $G_{1,\ldots,e}$, then $w$ is the empty word. \( \square \)
This resembles Britton’s Lemma for HNN extensions. In fact, Bass-Serre theory provides a unifying notion (and proof) for both phenomena. We give no further proof here, but instead apply the system $\mathcal{L}$ to the subgroup membership problem.

Let $\mathcal{L}'$ be the system $\mathcal{L}$ extended by the rules

\begin{align*}
(6) \quad & (x_1, -x_2) \cdot (x_2, -x_3) \cdot \ldots \cdot (x_{e-2}, -x_{e-1}) \cdot x_{e-2} \cdot (x_{e-1}, x_e) \cdot x_{e-1} \\
& \quad \quad \quad \rightarrow (x_1, 0) \cdot (x_{e-1}, x_e) \cdot x_{e-1} \quad \text{and} \\
(7) \quad & (-x_{e-1} \cdot q^x, x_e) \cdot x_{e-1} \cdot (-x_{e-2} \cdot q^x, x_e) \cdot x_{e-2} \cdot (-x_2 \cdot q^x, x_3) \cdot x_1 \cdot q^x, x_2) \\
& \quad \quad \quad \rightarrow (0, x_e) \cdot x_{e-1} \cdot (x_1, x_2 - x_2),
\end{align*}

where all $x_i \neq 0$.

The new rules respect the group structure, and hence Proposition 4.3 holds for $\mathcal{L}'$ as well. The new rules are not length-increasing, since $e \geq 3$.

Starting with an arbitrary sequence $w$ of pairs $(u, k)_i$ representing an element in $G_{1, \ldots, e}$, one can compute an equivalent $\mathcal{L}'$-reduced word $\tilde{w}$ with linearly many operations: First, compute an $\mathcal{L}$-reduced word $\tilde{w}$, then apply rules (6) and (7). Note that the latter leave $\tilde{w}$ $\mathcal{L}$-reduced. Only the second pair generated by either of these rules can be part of another application of (6) or (7). Therefore, $\tilde{w}$ can be $\mathcal{L}'$-reduced with one pass from left to right.

**Proposition 4.4** Let $w$ be a sequence of pairs $(u, k)_i$ which represents an element of the subgroup $F_{1,e} \subseteq G_{1,\ldots,e}$. If $w$ is $\mathcal{L}'$-reduced, then $w$ is an alternating sequence of pairs of types $(u, 0)_1$ and $(0, \ell)_{e-1}$.

**Proof** Let $w = (u_1, k_1)i_1(u_2, k_2)i_2 \ldots (u_n, k_n)i_n$. As $w$ is $\mathcal{L}'$-reduced, we have in particular that $i_j \neq i_{j+1}$ $(1 \leq j < n)$. Since $w \in F_{1,e}$, we can write

$$w \sim \tilde{w} = (v_1, 0) \cdot (0, \ell_1) \cdot (v_2, 0) \cdot (0, \ell_2) \cdot \ldots$$

and we may assume that $\tilde{w}$ contains no trivial pairs $(0, 0)_i$ which makes $\tilde{w}$ $\mathcal{L}$-reduced. The case where $\tilde{w}$ starts with $(0, \ell_1) \cdot (v_1, 0) \ldots$ is similar. The sequence

$$\tilde{w}^{-1} = \ldots \cdot (v_2, 0) \cdot (0, -\ell_1) \cdot (v_1, 0) \cdot (u_1, k_1)i_1(u_2, k_2)i_2(u_3, k_3)i_3 \ldots$$

equals 1 in $G_{1,\ldots,e}$ and must therefore $\mathcal{L}$-reduce to the empty sequence. Note that both $\tilde{w}^{-1}$ and $w$ are $\mathcal{L}$-reduced, so any $\mathcal{L}$-reduction can only occur at the border between the two words.

Clearly, we cannot have $i_1 \geq 3$ or else $\tilde{w}^{-1}w$ would be $\mathcal{L}$-reduced. If $i_1 = 2$, then a reduction (of type (2)) is possible if $k_1 = 0$ and $u_1 \in \mathbb{Z}$, in which case we get $(v_1, 0) \cdot (u_1, 0) \overset{(2)}{\rightarrow} (v_1, u_1).$ But after that, the sequence is $\mathcal{L}$-reduced since $k_1 = 0$ and $u_1 \in \mathbb{Z}$ imply $i_2 \geq 3$.

Hence, we are left with $i_1 = 1$. In that case, we get $(v_1, 0) \cdot (u_1, k_1) \overset{(1)}{\rightarrow} (v_1 + u_1, k_1).$ If this is $(0, 0)_1$, we have $(u_1, k_1)i_1 = (v_1, 0)_1$ and we proceed inductively with the remaining sequence. Otherwise, we must have $u_1 = v_1$ in order to continue applying rules (rule (2) is not possible since $k_2 = 0$ would make the same
rule applicable to $w$). If $e \geq 4$, the next rule can only apply to $(0, k_1)(u_2, k_2)_{i_2}$, so $i_2 = 2$ and we get $(0, k_1)(u_2, k_2)_{2} \overset{\text{(5)}}{\to} (k_1 + u_2, k_2)_{2}$. Again, the sequence is $L$-reduced unless $u_2 = -k_1$. We iterate this argument until we arrive at

$$\bar{w}^{-1}w \overset{\ast}{\Rightarrow} \cdots (-v_2, 0)_1(0, -\ell_1)e_{-1}(0, k_{e-2})e_{-2}(u_{e-1}, k_{e-1})_{i_{e-1}} \cdots .$$

On the way, we have found $u_1 = v_1, u_2 = -k_1, u_3 = -k_2, \ldots, u_{e-2} = -k_{e-3}$ (only $u_1 = v_1$ for $e = 3$), and $i_j = j$ for $1 \leq j \leq e - 2$. One further reduction of type (4) brings us to

$$\bar{w}^{-1}w \overset{\ast}{\Rightarrow} \cdots (-v_2, 0)_1(k_{e-2} \cdot q^{-\ell_1}, -\ell_1)e_{-1}(u_{e-1}, k_{e-1})_{i_{e-1}} \cdots .$$

Since $\ell_1 \neq 0$, the next reduction requires $i_{e-1} = e - 1$. Thus, rule (6) of $L'$ can be applied to the prefix $(u_1, k_1) \cdots (u_{e-1}, k_{e-1})_{e-1}$ of the original word $w$. \hfill $\square$

For the amalgamated product $H_f(1, q) = G_1, \ldots, f-1 \ast F_{1,f-1} G_{f-1}, f,1$, a property similar to 4.3 holds:

**Proposition 4.5** Let $w = w_1w_2 \cdots w_s$ be a non-empty sequence with $w_i \in \{(u, k)_i : 1 \leq i < f - 1\}$ or $w_i \in \{(u, k)_i : f - 1 \leq i \leq f\}$, alternatingly. If $w$ equals 1 in $H_f(1, q)$, then $w_i \in F_{1,f-1}$ for some index $i$.

From this proposition, we can derive Algorithm 4 which solves the word problem in $H_f(1, q)$. In this algorithm, the word $w = w_1w_2 \cdots w_s$ is split into $w_1 \cdots w_t$ and $w_{t+1} \cdots w_s$. The first part is an $L'$-reduced alternating sequence of group elements from $G_1, \ldots, f-1$ or $G_{f-1}, f,1$ with no $w_i$ ($1 \leq i \leq t$) being in the subgroup $F_{1,f-1}$. In each loop cycle either $t$ increases, or $t$ decreases by one and at the same time, some $w_i$ is merged with $w_{i+1}$, which means that $s$ decreases. Thus, the loop is executed only linearly often.

In order to get a time bound for Algorithm 4, it remains to show how to perform the tests arithmetic operations on the pairs $(u, k)_i$ efficiently.

Since power circuits are designed to work with integers, we have to avoid fractions for the first components of pairs $(u, k)_i \in G_{i,i+1} \simeq \mathbb{Z}/[1/q] \times \mathbb{Z}$. Therefore, we use the triple notation introduced in [4]. For $u, x, k \in \mathbb{Z}$ with $x \leq 0 \leq k$, let

$$[u, x, k]_i := (u \cdot q^x, x + k)_i \in G_{i,i+1}.$$

If $U, X, K$ are markings in a base $q$ power circuit, we call $T_i = [U, X, K]_i$ a triple marking and define its value by $\varepsilon(T_i) = [\varepsilon(U), \varepsilon(X), \varepsilon(K)]_i \in G_{i,i+1}$.

Any element of $G_{i,i+1}$ can be written as a triple, but not in a unique way. For instance, $[2, 0, 0]_i = (2, 0)_i = [4, -1, 1]_i$, if $q = 2$. The group operations translate to formulae for multiplication and inversion of triples:

$$[u, x, k] \cdot [v, y, \ell] = [u \cdot q^{-y} + v \cdot q^k, x + y, k + \ell]$$

$$[u, x, k]^{-1} = [-u, -k, -x]$$

\hfill $\square$ Springer
Algorithm 4: Procedure for solving the word problem in $H_f(1, q)$

**input**: a word $w$ over $a_i, a_i^{-1} (1 \leq i \leq f)$

**output**: “yes” if $w = 1$ in $H_f(1, q)$, “no” otherwise

1. Rewrite the input $w$ by replacing each $a_i^{\pm 1}$ by $(\pm 1, 0)_i$;
2. Break $w$ into subsequences $w = w_1 w_2 \ldots w_s$ such that in each $w_j$ the subscripts of all pairs are either in $\{1, 2, \ldots, f - 2\}$ or in $\{f - 1, f\}$. This defines the initial value of $s$;
3. Let $t := 0$;
4. while $(t = 0 \land s > 1) \lor (0 < t < s)$ do
   5. if $t = 0 \land s > 1$ then
      6. $\mathcal{L}'$-reduce $w_1$. If $w_1$ becomes empty, remove it (thereby decreasing $s$) and continue with the next iteration;
      7. if $w_1 \in F_{1, f - 1}$ then
         8. Merge $w_1$ and $w_2$. Before doing so, if $w_1$ and $w_2$ are from different groups (one from $G_1, \ldots, f - 1$ and the other one from $G_{f - 1}, f$), swap all the pairs in $w_1$ using the following rules: $(x, 0)_1 \leftrightarrow (0, x)_f$ and $(0, x)_{f - 2} \leftrightarrow (x, 0)_{f - 1}$;
      else
         9. Increment $t$ by one;
   else
      10. $(0 < t < s)$
      11. if $w_t$ and $w_{t + 1}$ are both from $G_1, \ldots, f - 1$ or both from $G_{f - 1}, f$ then
         12. Merge $w_t$ and $w_{t + 1}$;
         13. Decrement $t$ by one;
      else
         14. $\mathcal{L}'$-reduce $w_{t + 1}$. If $w_{t + 1}$ becomes empty, remove it and continue with the next iteration;
      15. if $w_{t + 1} \in F_{1, f - 1}$ then
         16. Perform the replacements $(x, 0)_1 \leftrightarrow (0, x)_f$ and $(0, x)_{f - 2} \leftrightarrow (x, 0)_{f - 1}$ in $w_{t + 1}$, then merge it with $w_t$;
         17. Decrement $t$ by one;
      else
         18. Increment $t$ by one;
   21. return “yes” if $s = 0$, “no” otherwise;

Furthermore, $[u, x, k]_i \in \langle a_i \rangle \leq G_{i, i + 1}$ if and only if $x = -k$ and $u \cdot q^x \in \mathbb{Z}$. Similarly, $[u, x, k]_i \in \langle a_{i + 1} \rangle \leq G_{i, i + 1}$ if and only if $u = 0$, and finally $[u, x, k]_i$ is the group identity if and only if $u = 0$ and $x = -k$.

At the beginning of Algorithm 4, we create a power circuit $\Pi$ with base $q$ consisting of a single node $u$ with $\varepsilon(u) = 1$. We represent each letter $a_i^{\pm 1}$ (which corresponds to the pair $(\pm 1, 0)_i$) by a triple marking with value $[\pm 1, 0, 0]_i$. Of the three markings required for each of these triples, two are zero (empty) and the third has either value
+1 or −1 and can be created using $u$. Let $\omega$ be the sum of the sizes of (the supports of) all these markings. We call $\omega$ the weight of the circuit. From the multiplication formula for triples we see that $\omega$ never increases during the algorithm, keeping in mind that after an operation we can “forget” the operands (i.e., remove their markings) and just keep the resulting marking instead. The initial value of $\omega$ is exactly $n = |w|$.

After step 1, we reduce the circuit, which takes $O(n^2)$ time. From now on, we keep $\Pi$ reduced following the strategy proposed in Sect. 2.4.3. The swapping operation $(x, 0) \leftrightarrow (0, x)$ works in the following way for triples (note that a triple corresponding to $(x, 0)$ has the form $[u, x, k]$ with $x + k = 0$):

$$
[u, x, k] \mapsto \begin{cases} 
[0, 0, u \cdot q^x] & \text{if } x = -k \text{ and } q^x | u \text{ and } u \geq 0 \\
[0, u \cdot q^x, 0] & \text{if } x = -k \text{ and } q^x | u \text{ and } u < 0
\end{cases}
$$

$$
[0, x, k] \mapsto [x + k, 0, 0]
$$

The whole algorithm computes $O(n)$ many times an $L'$-irreducible word. Each of these computations necessitates $O(n)$ arithmetic operations (and subsequent calls to EXTEND REDUCTION). The circuit size remains bounded by $O(n^2 \cdot \omega) \subseteq O(n^3)$. Thus, one call of EXTEND REDUCTION takes $O(n^3 \cdot \omega) \subseteq O(n^4)$ time. We get a total time bound of $O(n^6)$.

This concludes the proof of Theorem 4.2.

5 Conclusion

We have shown that the word problem for the generalized Higman groups $H_f(1, q)$ is decidable in polynomial time. An important ingredient for this result was the extension of the power circuit data structure to arbitrary bases $q \geq 2$. From an algorithmic point of view, this is an interesting result in itself and may provide a useful tool in group theory as well as other areas.

The techniques used in this paper do not apply to the even more general groups $H_f(p, q)$ and $G(p, q)$, where the underlying Baumslag-Solitar group $BS(1, q)$ is replaced by $BS(p, q) = \langle a, t \mid t^p a t^{-1} = a^q \rangle$ for some $p \geq 1$. This is because $BS(p, q)$ is not a semi-direct product when $p > 1$. The word problem for these groups is open. Note that even for $f < 4$ the group $H_f(p, q)$ can be non-trivial if $p > 1$.

Acknowledgements The author gratefully acknowledges the helpful comments and suggestions by the anonymous reviewers.

References

1. Allcock, D.: Triangles of baumslag-solitar groups. Can. J. Math. 64(2), 241–253 (2012)
2. Book, R.V., Otto, F.: String-Rewriting Systems. Springer, Berlin (1993)
3. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms, 3rd edn. MIT Press, Cambridge (2009)
4. Diekert, V., Laun, J., Ushakov, A.: Efficient algorithms for highly compressed data: the word problem in Higman’s group is in P. In: 29th International Symposium on Theoretical Aspects of Computer Science (STACS 2012). Leibniz International Proceedings in Informatics (LIPIcs), vol. 14, pp. 218–229 (2012)
5. Diekert, V., Laun, J., Ushakov, A.: Efficient algorithms for highly compressed data: the word problem in Higman’s group is in P. Int. J. Algebra Comput. 22(8) (2012)
6. Higman, G.: A finitely generated infinite simple group. J. Lond. Math. Soc. 26, 61–64 (1951)
7. Lyndon, R., Schupp, P.: Combinatorial Group Theory. Classics in Mathematics. Springer, Berlin (2001)
8. Laun, J.: Solving Algorithmic Problems in Baumslag-Solitar Groups and their Extensions Using Data Compression. PhD thesis, Universität Stuttgart (2012)
9. Myasnikov, A.G., Ushakov, A.: Cryptography And Groups (CRAG) Software Library
10. Myasnikov, A.G., Ushakov, A., Won, D.W.: The word problem in the baumslag group with a non-elementary Dehn function is polynomial time decidable. J. Algebra 345(1), 324–342 (2011)
11. Myasnikov, A.G., Ushakov, A., Won, D.W.: Power circuits, exponential algebra, and time complexity. Int. J. Algebra Comput. 22(6) (2012)
12. Neumann, B.H.: An essay on free products of groups with amalgamations. Philos. Trans. R. Soc. Lond. Ser. A, Math. Phys. Sci. 246(919), 503–554 (1954)
13. Serre, J.-P.: Trees, 2nd corr. printing edn. Springer, Berlin (2002)