A Semi-group of Hashing Functions & Hash Tables

Arthur Hughes
University of Dublin,
Trinity College, Dublin, Ireland
e-mail: Arthur.P.Hughes@cs.tcd.ie

January 19, 1997

Abstract

A model of a hash table is reviewed and problems with the model and operations on the model are identified. A collection of isomorphic monoids is found, in each of which will be a semi-group, represented hashing functions and hash tables. One of the semi-groups is used to redefine the previously examined operations. Finally, some algebraic questions arising from the model are answered.

1 Introduction

Searching for a particular object in a collection of objects is fundamental to computer science. A number of efficient searching procedure exist. One such technique is that of hashing with 'overflow chaining'. Hashing is used in the UNIX file system, to buffer data block of opened files. This report re-examines a model of a hash table and presents a new one. The models are presented using the Irish School of Constructive Mathematics, see Mac an Airchinnigh [2, 3, 4]. In which algebraic structures and morphisms are used to specify and develop software systems.

2 A Model of a Hash Table

A model of a hash table is developed in Mac an Airchinnigh [2, pages 386 – 397] as a sequence of sequences of words, $(W^\star)^\star$. The type of hash table modelled is one with 'overflow chaining' to deal with collisions when words are hashed, see Knuth [8, pages 506 – 549]. In most implementations of this kind of hash table the 'overflow chains' of words are sorted and do not contain duplicities. Thus a more appropriate choice of space would be, $(W^\star_{\leq !})^\star$, where $W^\star_{\leq !}$ denotes the space of sorted unique sequences of words. This choice of space is still too large because when a hash table is implemented the 'overflow chains' should be disjoint, that is, a word can be on one and only one 'overflow chain'. The right choice of domain will make the modelling process simpler and with also allow the hidden algebraic structures underlying the system to filter through. This choice of space is partially solved by the introduction of an invariant.

2.1 The Invariant

The invariant on the space of sequences of sequences of words introduced is one which ensures

- the number of 'overflow chains' in a hash table is a prime number greater than one hundred,
- each 'overflow chain' in a hash table is different and finally,
- each 'overflow chain' in a hash table does not contain duplicate words.

This is stated formally for a hash table $\tau \in (W^\star)^\star$ as

$$\begin{aligned}&\mathtt{is} - \mathtt{prime}\,(\mathtt{len}\,\tau) \wedge \mathtt{len}\,\tau > 100 \wedge \\ &\mathtt{len}\,\tau = \mathtt{card\ elems}\ \tau \wedge {}^{\wedge}\!/(\mathtt{len} = \mathtt{card\ elems}\,)^{\star}\tau\end{aligned}$$

This invariant does not ensure the following properties normally associated with a hash table

- the 'overflow chains' in a hash table are normally sorted,
- the 'overflow chains' in a hash table are normally disjoint and finally,
- a hash table should be a 'reflection' of a hashing function.

Thus the given invariant is not sufficient. The required space will be developed in this report. Clearly identifying the space to be modelled has previously led to the development of new monoids, see Hughes and Mac an Airchinnigh [7], this will also be true of this report. The operations on a hash table must also be modelled.

2.2 The Operations

Operations on a hash table are developed in Mac an Airchinnigh [2, pages 388 – 289]. The definitions of the operations make use of a hashing function $h \in \mathbb{Z}_p^W$ which is a total function from words to the set $\{0, \ldots, p\}$ where p is a prime. This function on given a word will return the number of the 'overflow chain' which the word should be on. A hashing function is a projection of words onto their corresponding 'overflow chain' number, thus each hashing function partitions the space of words, see Goldblatt [6, pages 88 – 90]. Mac an Airchinnigh [2, page 392] refers to this function as a 'magic' function. The hashing function is not 'magic', it is the key to the semantics of modelling the hash table, not just in the definition of the operations on the hash table. The hashing functions will be use to define the space which will model hashing functions and their corresponding hash tables. The operations as defined have a number of problems associated with them. The first operation dealt with is the creation of a new hash table.

2.2.1 The New Operation

This operation must create a new hash table. The number of 'overflow chains' in the hash table to be created must be given and then the operation will create a sequence of empty sequences each representing an empty 'overflow chain'.

$$\begin{array}{l} New : \mathbb{N} \to (W^\star)^\star \\ New(p) \triangleq \langle s_i : s_i = \Lambda \wedge 0 \leq i \leq p-1 \rangle \end{array}$$

This definition is not adequate because

- when a hash table is created a hashing function for the table should be chosen, no decision on a hashing function is made in the above definition of the new operation,
- the quantification in the definition is not aesthetically appealing.

Entering a word into a given hash table is the nexted operation examined.

2.2.2 The Enter Operation

Given a word and a hash table the enter operation places the word on the appropriate 'overflow chain' in the hash table. This is done using a hashing function $h \in \mathbb{Z}_p^W$, by splitting the hash table into three parts

- τ_l the begin of the hash table where the word should not appear,
- $\langle \tau_{h(w)} \rangle$ the 'overflow chain' which the word should be placed on,
- τ_r the end of the hash table where the word should also not appear

and then adding the word to the head of the 'overflow chain' which it should be on. Finally the table is reformed by concatenating the three parts together again.

$$\begin{array}{ll} Ent : W \to ((W^\star)^\star \to (W^\star)^\star) & \\ Ent[\![w]\!]\tau \triangleq & \mathbf{let}\, \tau = \tau_l \mathbin{^\frown} \langle \tau_{h(w)} \rangle \mathbin{^\frown} \tau_r \\ & \mathbf{in}\, \tau_l \mathbin{^\frown} \langle \langle w \rangle \mathbin{^\frown} \tau_{h(w)} \rangle \mathbin{^\frown} \tau_r \end{array}$$

Subject to the pre-condition which ensures that the word to be entered in the table is not already on the 'overflow chain' which it should be on

$$\begin{array}{ll} \textit{pre-Ent} : W \to ((W^\star)^\star \to \mathbb{B}) & \\ \textit{pre-Ent}[\![w]\!]\tau \triangleq & \mathbf{let}\, \tau = \tau_l \mathbin{^\frown} \langle \tau_{h(w)} \rangle \mathbin{^\frown} \tau_r \\ & \mathbf{in}\, \neg\chi[\![w]\!]\tau_{h(w)} \end{array}$$

This definition of the enter operation with its precondition has a number of problems

- the hashing function used to split the hash table is pulled out of nowhere,
- the hash table may not be a ‘reflection’ of this hashing function,
- the splitting of the hash table into three parts is difficult to grasp, this has been learned by teaching this model of a hash table to undergraduate students,
- two enter operations one after the other with different hashing functions could place a word on two ‘overflow chains’ and thus violate the condition that the ‘overflow chains’ should be disjoint,
- the operation has a pre-condition associated with it, which could be removed by redefining the operation,
- the uses of the **let** and **in** keywords are not aesthetically appealing and inconsistent with other uses of these keywords.

The lookup operation is examined finally.

2.2.3 The Lookup Operation

A hash table was developed to improve searching performance. The lookup operation should model this searching process. Again a hashing function $h \in \mathbb{Z}_p^W$ is used to find which ‘overflow chain’ the word should be on. This ‘overflow chain’ is then searched for the word.

$$\begin{array}{ll} Lkp : W \rightarrow ((W^\star)^\star \rightarrow \mathbb{B}) & \\ Lkp[\![w]\!]\tau \triangleq & \mathbf{let}\, \tau = \tau_l \mathbin{^\wedge} \langle \tau_{h(w)} \rangle \mathbin{^\wedge} \tau_r \\ & \mathbf{in}\, \chi[\![w]\!]\tau_{h(w)} \end{array}$$

This definition has similar problems to the definition of the enter operation

- the hashing function used to find the ‘overflow chain’ is pulled out of nowhere,
- the hash table may not be a ‘reflection’ of this hashing function,
- as the ‘overflow chains’ may not be sorted, thus $\chi[\![w]\!]$ must represent a linear search on the ‘overflow chain’. The characteristic function should represent a binary search on a sorted ‘overflow chain’,
- if a hashing function is used which hash the word to a ‘overflow chain’ which it is not on then lookup will perform incorrectly,
- the uses of the **let** and **in** constructs are not aesthetically appealing.

After examining the operations and the problems associated with them, it is clear that a hash table and its hashing function should be kept together at all times; a hash table on its own is useless as no word can be looked up efficiently for example.

3 Towards a Space

The key to the development of a space of hash tables is recognising the fact that a hash table is not one object, but two objects, a hashing function and a table which is a 'reflection' of the hashing function. This section will develop a number of spaces which will approximate a hash table space.

The first space is $\mathcal{M}_0 \subset \mathcal{P}W \times W \to \mathbb{Z}_p$, an element of this space is a tuple containing a set of words and a partial mapping from words to integers modulo p, where the set of words is a subset of the the domain of the partial mapping, $(S, \eta) \in \mathcal{M}_0$ iff $S \subset \texttt{dom}\,\eta$. The subset $\mathcal{H}_0 = \mathcal{P}W \times \mathbb{Z}_p^W$ of the space $\mathcal{M}_0$ is the first approximation to a space of hash tables. An element of $\mathcal{H}_0$ is a tuple containing a set of words (representing the table) and a total mapping from words to integers modulo p (representing the hashing function).

$$\mathcal{H}_0 \subset \mathcal{M}_0 \subset \mathcal{P}W \times W \to \mathbb{Z}_p$$

The next space $\mathcal{M}_1$ is formed by applying the mapping

$$< \pi_2, \lhd >: \mathcal{M}_0 \to \mathcal{M}_1 \subset W \to \mathbb{Z}_p \times W \to \mathbb{Z}_p$$

to the space $\mathcal{M}_0$. An element of the space $\mathcal{M}_1$ is a tuple containing two partial mapping where the second mapping is a restriction of the first mapping. The subset $\mathcal{H}_1$ of the space $\mathcal{M}_1$ is formed by applying the above mapping to the set $\mathcal{H}_0$. The space $\mathcal{H}_1$ is the next approximation to a space of hash tables. An element of this space is a tuple containing a total mapping from words to integers modulo p (representing the hashing function) and a restriction of this total mapping (representing the table).

$$\mathcal{H}_1 \subset \mathcal{M}_1 \subset W \to \mathbb{Z}_p \times W \to \mathbb{Z}_p$$

Using the mapping

$$(\mathcal{I} \times (\_)^{-1}) : \mathcal{M}_1 \to \mathcal{M}_2 \subset W \to \mathbb{Z}_p \times \mathbb{Z}_p \to \mathcal{P}'W$$

the space $\mathcal{M}_2$ is formed as the image of the space $\mathcal{M}_1$. An element of the space $\mathcal{M}_2$ is a tuple containing two partial mapping where the second mapping is an inverse image of a restriction of the first mapping. Again the space of hash tables is approximated by a space $\mathcal{H}_2 \subset \mathcal{M}_2$ formed by applying the above mapping to the space $\mathcal{H}_1$. An element of the space $\mathcal{H}_2$ is a tuple containing a total mapping from words to integers modulo p (representing the hashing function) and a mapping from integers modulo p to sets of words (representing the table).

$$\mathcal{H}_2 \subset \mathcal{M}_2 \subset W \to \mathbb{Z}_p \times \mathbb{Z}_p \to \mathcal{P}'W$$

The `elems` mapping from sorted unique sequences to sets is a one-to-one mapping and hence has an inverse map $\texttt{elems}^{-1}$, from sets to sorted unique sequences. The next space formed is $\mathcal{M}_3$ which is the image of the space $\mathcal{M}_2$ under the mapping

$$(\mathcal{I} \times (\mathcal{I} \rightarrow \texttt{elems}^{-1})) : \mathcal{M}_2 \rightarrow \mathcal{M}_3 \subset W \rightarrow \mathbb{Z}_p \times \mathbb{Z}_p \rightarrow W^{+}_{\leq !}$$

An element of the space $\mathcal{M}_3$ is a tuple containing two partial mappings where the second mapping is related to the first, by the construction. The space $\mathcal{H}_3 \subset \mathcal{M}_3$ formed by applying the above mapping to the space $\mathcal{H}_2$ will approximate a space of hash tables. An element of the space $\mathcal{H}_3$ will be a tuple containing a total mapping from words to integers modulo p (representing the hashing function) and a mapping from integers modulo p to sorted unique sequences of words (representing the table), this mapping does not map to any null sequences (representing empty 'overflow chains').

The final space $\mathcal{M}_4$ is formed by applying the mapping

$$< \pi_1, \Lambda^{\texttt{rng}\ \pi_1} \dagger \pi_2 >: \mathcal{M}_3 \rightarrow \mathcal{M}_4 \subset W \rightarrow \mathbb{Z}_p \times \mathbb{Z}_p \rightarrow W^{\star}_{\leq !}$$

to the space $\mathcal{M}_3$. An element of the space $\mathcal{M}_4$ is a tuple containing two mappings where again the second mapping is related to the first mapping. The space $\mathcal{H}_4$ is formed by applying the above mapping to the space $\mathcal{H}_3$, an element of this space is again a tuple containing a total mapping from words to integers modulo p (representing the hashing function) and a mapping from integers modulo p to sorted unique sequences of words, were the sequence may be empty (representing the table with empty 'overflow chains'). The space $\mathcal{H}_4$ is the final approximation of a space of hash tables. Binary operations will now be placed on the space developed.

3.1 Some Monoids, Semi-groups and Morphisms

This section will develop monoids from the spaces defined in the previous section, in such away that the mapping between the spaces will become morphisms. A binary operation is placed first on the space $\mathcal{M}_0$ such that a monoid will be formed. Then a binary operation is placed on the space $\mathcal{M}_1$ in such away that the mapping $< \pi_2, \lhd >$ between the spaces becomes a morphism. This process is then continued for the remaining spaces.

The binary operator placed on the space $\mathcal{M}_0$ is the direct product operator of the monoid sets under union and the monoid of maps under override. Note that $\mathcal{M}_0$ is not the entire direct product space. What will this operator do to the space $\mathcal{H}_0$ which is representing a space of hash tables? The operator must have a meaningful interpretation with respect to hash tables.

Lemma 1 *If $(S_1, \eta_1), (S_2, \eta_2) \in \mathcal{M}_0$ then with respect to the operation defined by*

$$(S_1, \eta_1)(S_2, \eta_2) = (S_1 \cup S_2, \eta_1 \dagger \eta_2)$$

the space $\mathcal{M}_0 \subset \mathcal{P}W \times W \rightarrow \mathbb{Z}_p$ becomes a monoid with identity $(\emptyset, \theta)$, also the space $\mathcal{H}_0 \subset \mathcal{M}_0$ becomes a semi-group.

What interpretation can be given to this operation in terms of hash tables? This operation is taking two hash tables and forming a new hash table, where the new hash table contains the words from both hash tables, hashed with respect to a new hashing function, which is the hashing function from the first hash table override by the hashing function from the second hash table.

Next a binary operation is placed on the space $\mathcal{M}_1$. This operation will take two tuples each containing two mappings, where the second mapping is a restriction of the first mapping, and combine them to form a new tuple where the second mapping is a restriction of the first mapping. This is useful for combining hash tables, as the words in the first hash table are rehashed by the second hashing function and then joined with the second hash table.

Lemma 2 *If $(\eta_1, \mu_1), (\eta_2, \mu_2) \in \mathcal{M}_1$ then with respect to the operation defined by*

$$(\eta_1, \mu_1)(\eta_2, \mu_2) = (\eta_1 \dagger \eta_2, \lhd[\![\mathtt{dom}\,\mu_1]\!](\eta_1 \dagger \eta_2) \cup \mu_2)$$

the space $\mathcal{M}_1 \subset W \to \mathbb{Z}_p \times W \to \mathbb{Z}_p$ becomes a monoid with identity (θ, θ), also the space $\mathcal{H}_1$ becomes a semi-group.

The lemma uses the gluing operation, $\cup$, on maps, see [5]. This operation incorporates the map extend operation, $\sqcup$, for joining two disjoint maps together. The operation also joins two maps which agree on a common domain together.

The spaces $\mathcal{M}_0$ and $\mathcal{M}_1$ are now monoids. Is the mapping $< \pi_2, \lhd >$: $\mathcal{M}_0 \to \mathcal{M}_1$ between them a morphism?

Lemma 3 *The mapping $< \pi_2, \lhd >$: $\mathcal{M}_0 \to \mathcal{M}_1$ between the monoids $\mathcal{M}_1$ and $\mathcal{M}_1$ is an isomorphism.*

Proof If $(S_1, \eta_1), (S_2, \eta_2) \in \mathcal{M}_0$, so $S_1 \subset \mathtt{dom}\,\eta_1$ and $S_2 \subset \mathtt{dom}\,\eta_2$, then

$$\begin{aligned}
&< \pi_2, \lhd > (S_1, \eta_1)(S_2, \eta_2) \\
&\quad = \quad < \pi_2, \lhd > (S_1 \cup S_2, \eta_1 \dagger \eta_2) \\
&\quad = \quad (\eta_1 \dagger \eta_2, \lhd[\![S_1 \cup S_2]\!](\eta_1 \dagger \eta_2)) \\
&\quad = \quad (\eta_1 \dagger \eta_2, \lhd[\![S_1]\!](\eta_1 \dagger \eta_2) \cup \lhd[\![S_2]\!](\eta_1 \dagger \eta_2)) \\
&\quad = \quad (\eta_1 \dagger \eta_2, \lhd[\![\mathtt{dom} \lhd [\![S_1]\!]\eta_1]\!](\eta_1 \dagger \eta_2) \cup \lhd[\![S_2]\!]\eta_2) \\
&\quad = \quad (\eta_1, \lhd[\![S_1]\!]\eta_1)(\eta_2, \lhd[\![S_2]\!]\eta_2) \\
&\quad = \quad < \pi_2, \lhd > (S_1, \eta_1) < \pi_2, \lhd > (S_2, \eta_2)
\end{aligned}$$

So the mapping is a morphism between the spaces. This proof displays a hidden lemma: given a mapping $\mu \in X \to Y$ and sets $S_1, S_2 \in \mathcal{P}X$ then

$$\lhd[\![S_1 \cup S_2]\!]\mu = \lhd[\![S_1]\!]\mu \cup \lhd[\![S_2]\!]\mu$$

This equality will take the form of a morphism under the change of notation $\lhd_\mu(S) = \lhd[\![S]\!]\mu$. This sort of notation change has occurred before in relation to the characteristic function $\chi_S(x) = \chi[\![x]\!]S$. The above equality is rewritten below in the new notation

$$\lhd_\mu(S_1 \cup S_2) = \lhd_\mu(S_1) \cup \lhd_\mu(S_2)$$

It clearly has the form a morphism. More will be said about this in the future work section.

Does the mapping $(\pi_2, \lhd)$ preserve the identities of the monoids?

$$\begin{aligned} & < \pi_2, \lhd > (\emptyset, \theta) \\ = \; & (\theta, \lhd[\![\emptyset]\!]\theta) \\ = \; & (\theta, \theta) \end{aligned}$$

Finally, the mapping is one-to-one and onto between the spaces $\mathcal{M}_0$ and $\mathcal{M}_1$, hence the mapping is an isomorphism. ∎

The remaining spaces are turned into monoids by placing binary operations on them. The binary operations are chosen so as the mapping defined in the previous section become isomorphisms. The space $\mathcal{M}_2$ is turned into a monoid by the lemma below.

Lemma 4 *If $(\eta_1, \beta_1), (\eta_1, \beta_2) \in \mathcal{M}_2$ then with respect to the operation defined by*

$$(\eta_1, \beta_1)(\eta_2, \beta_2) = (\eta_1 \dagger \eta_2, (\mathcal{I} \rightarrow \lhd[\![^{\cup}/\texttt{rng}\ \beta_1]\!])'(\eta_1 \dagger \eta_2)^{-1} \circledcirc \beta_2)$$

where the prime denotes removal of entries of the form $z \mapsto \emptyset$, the space $\mathcal{M}_2 \subset W \rightarrow \mathbb{Z}_p \times \mathbb{Z}_p \rightarrow \mathcal{P}'W$ becomes a monoid with identity (θ, θ), also the space $\mathcal{H}_2 \subset \mathcal{M}_2$ becomes a semi-group.

This lemma uses an indexed monoid operation, $\circledcirc$, see [4, pages 28 – 29]. This is the gluing operator in an inverted world. The operator will union two relations (represented as mappings) together, forming a new relation (represented as a mapping). Next the the merge sort operator, $\bigtriangledown$, see [2, page 112], is indexed, $\bigcirc\!\!\!\!\bigtriangledown$. This operator is used in the definition of the binary operation placed on the space $\mathcal{M}_3$ in the lemma below.

Lemma 5 *If $(\eta_1, \sigma_1), (\eta_2, \sigma_2) \in \mathcal{M}_3$ then with respect to the operation defined by*

$$(\eta_1, \sigma_1)(\eta_2, \sigma_2) = (\eta_1 \dagger \eta_2, (\mathcal{I} \rightarrow \lhd[\![^{\cup}/\mathcal{P}\texttt{elems rng}\ \sigma_1]\!])'(\mathcal{I} \rightarrow \texttt{elems}^{-1})(\eta_1 \dagger \eta_2)^{-1} \bigcirc\!\!\!\!\bigtriangledown\ \sigma_2)$$

where the prime denotes removal of entries of the form $z \mapsto \Lambda$, the space $\mathcal{M}_3 \subset W \rightarrow \mathbb{Z}_p \times \mathbb{Z}_p \rightarrow W^{+}_{\leq !}$ becomes a monoid with identity (θ, θ), also the space $\mathcal{H}_3 \subset \mathcal{M}_3$ becomes a semi-group.

Finally, an operation is placed on the space $\mathcal{M}_4$. This operation is similar to the binary operation introduced in the lemma above, except the priming is removed. This allows the forming empty ‘overflow chains’.

Lemma 6 *If* $(\eta_1, \tau_1), (\eta_2, \tau_2) \in \mathcal{M}_4$ *then with respect to the operation defined by*

$$(\eta_1, \tau_1)(\eta_2, \tau_2) = (\eta_1 \dagger \eta_2, (\mathcal{I} \to \triangleleft [\![^{\cup}/\mathcal{P}\texttt{elems rng } \tau_1]\!])(\mathcal{I} \to \texttt{elems}^{-1})(\eta_1 \dagger \eta_2)^{-1} \bigtriangledown \tau_2)$$

the space $\mathcal{M}_4 \subset W \to \mathbb{Z}_p \times \mathbb{Z}_p \to W^{\star}_{\leq !}$ *becomes a monoid with identity* (θ, θ) *, also the space* $\mathcal{H}_4 \subset \mathcal{M}_4$ *becomes a semi-group.*

This final semi-group $\mathcal{H}_4$ is used in this report to model a space of hash tables and the operations on hash tables. The collection of monoids and semi-groups are related by the mappings defined in the previous section - these mappings become morphisms because of the choice of operators placed on each space.

Lemma 7 *The monoids* $\mathcal{M}_1, \mathcal{M}_2, \mathcal{M}_3, \mathcal{M}_4$ *are isomorphic, also the semi-groups* $\mathcal{H}_1, \mathcal{H}_2, \mathcal{H}_3, \mathcal{H}_4$ *are isomorphic, the isomorphisms being the mappings between the spaces defined above.*

The proof of this Lemma is left as an exercise. The modelling of the operations on a hash table is now re-addressed.

3.2 The Operations

The operations of creating a new hash table, entering a word into a hash table and looking up a word in a hash table are now re-expressed, also the operation of rehashing a hash table is addressed. In the operations below a hash table will be an element of the semi-group $\mathcal{H}_4$. The enter and rehash operations will be defined in terms of the binary operation on the semi-group. In finding this model of a hash table, we see that the binary operation introduced models both enter and rehash. To find a semi-group of hash tables the issues of enter and rehash had to be resolved, the algebra introduced in the previous section demonstrated this.

3.2.1 The New Operation

To create a new hash table a hashing function must be provided or chosen. This is modelled by passing the chosen hashing function as a parameter to the new operation.

$$\begin{array}{l} New : \mathbb{Z}_p^W \to \mathcal{H}_4 \\ New(h) \triangleq (h, \Lambda^{\texttt{rng } h}) \end{array}$$

The new operation forms a tuple containing the given hashing function and an empty hash table.

3.2.2 The Enter Operation

The operation takes a word and a tuple containing a hashing function and a hash table as parameters. The operation creates a tuple containing the hashing function and a mostly empty hash table (the table just contains the word to be hashed) and then combines this tuple with the tuple passed in, using the binary operation of the semi-group $\mathcal{H}_4$.

$$\begin{array}{l} Ent : W \to (\mathcal{H}_4 \to \mathcal{H}_4) \\ Ent[\![w]\!](h, \tau) \triangleq (h, \tau)(h, \Lambda^{\mathtt{rng}\ h} \dagger [h(w) \mapsto \langle w \rangle]) \end{array}$$

3.2.3 The Lookup Operation

The lookup operation must return whether a word is in a given hash table. The operation uses the hashing function which comes with the hash table to find which 'overflow chain' the word should be on, then this 'overflow chain' is searched for the word.

$$\begin{array}{l} Lkp : W \to (\mathcal{H}_4 \to \mathbb{B}) \\ Lkp[\![w]\!](h, \tau) \triangleq \chi[\![w]\!]\tau(h(w)) \end{array}$$

The characteristic function here denotes the binary search of the sorted unique 'overflow chain' for the word.

3.2.4 The Rehash Operation

If the 'overflow chains' in a given hash table are growing to large, the words in the hash table can be rehashed by a new hashing function, so that the lengths of the 'overflow chains' are reduced. The rehash operation takes a new hashing function and a hash table (a tuple containing a hashing function and a table) and then combines the tuple with a new tuple containing the new hashing function and an empty hash table, using the binary operation of the semi-group $\mathcal{H}_4$.

$$\begin{array}{l} Reh : \mathbb{Z}_p^W \to (\mathcal{H}_4 \to \mathcal{H}_4) \\ Reh[\![g]\!](h, \tau) \triangleq (h, \tau)(g, \Lambda^{\mathtt{rng}\ g}) \end{array}$$

4 Future Work

The restriction of a map by a set is a mapping from tuples containing a set and a map to a map,

$$\begin{array}{l} \lhd : \mathcal{P}X \times X \to Y \longrightarrow X \to Y \\ \lhd : (S, \mu) \mapsto \lhd(S, \mu) \end{array}$$

This mapping can be curried in two different ways

- first with respect to the set yielding a mapping from maps to maps restricted by the set,

  $$\begin{array}{l} \lhd : \mathcal{P}X \longrightarrow (X \to Y) \longrightarrow (X \to Y) \\ \lhd_S : \mu \mapsto \lhd(S, \mu) \end{array}$$

  for all maps $\mu \in X \to Y$.

- second with respect to the map yielding a mapping between sets and the map restricted by these sets,

$$\begin{array}{l} \lhd : (X \to Y) \longrightarrow \mathcal{P}X \longrightarrow (X \to Y) \\ \lhd_\mu : S \mapsto \lhd(S, \mu) \end{array}$$

  for all sets $S \in \mathcal{P}X$.

Each curried mapping prompt two algebraic questions,

- firstly, are the curried mappings $\lhd_S, \lhd_\mu$ morphisms?
- secondly, does the collection of all mappings $\lhd_S$ for $S \in \mathcal{P}X$ form a monoid? This can also be asked of the collection of all mappings $\lhd_\mu$ for $\mu \in X \to Y$.

Both of these questions have been answered for the curried mapping $\lhd_S$, in [2, pages 123 – 127]. The results contained there are repeated here for completeness. The answers for the curried mapping $\lhd_\mu$ are new and are introduced below.

Is the mapping $\lhd_S$ a morphism? The mapping $\lhd_S$ sends maps to maps. The space of maps forms a monoid $(X \to Y, \dagger, \theta)$, the curried mapping $\lhd_S$ is an endomorphism of this monoid.

$$\begin{array}{l} \lhd_S : (X \to Y, \dagger, \theta) \longrightarrow (X \to Y, \dagger, \theta) \\ \lhd_S(\mu_1 \dagger \mu_2) = \lhd_S(\mu_1) \dagger \lhd_S(\mu_2) \\ \lhd_S(\theta) = \theta \end{array}$$

Is the collection of mappings $\lhd_S$ for $S \in \mathcal{P}X$ a monoid? This collection of mappings is denoted by $\lhd_{\mathcal{P}X}$. As $\lhd_{S_1}, \lhd_{S_2} \in \lhd_{\mathcal{P}X} \subset (X \to Y)^{(X \to Y)}$ they can be combined by map composition, $\lhd_{S_1} \circ \lhd_{S_2}$, but is the composed mapping in $\lhd_{\mathcal{P}X}$? The composed mapping is

$$\lhd_{S_1} \circ \lhd_{S_2} = \lhd_{S_1 \cap S_2}$$

thus a monoid of endomorphisms is formed $(\lhd_{\mathcal{P}X} \subset (X \to Y)^{(X \to Y)}, \circ, \lhd_X)$.

Is the mapping $\lhd_\mu$ a morphism? This curried mapping sends sets to maps. The space of sets forms a monoid $(\mathcal{P}X, \cup, \emptyset)$. The image of the mapping $\lhd_\mu$, denoted $\lhd_\mu(\mathcal{P}X) \subset X \to Y$, is the space of all submaps of μ, that is $\nu \in \lhd_\mu(\mathcal{P}X)$ iff $\lhd_\mu(\mathtt{dom}\,\nu) = \nu$. The space $\lhd_\mu(\mathcal{P}X)$ becomes a monoid with the gluing operator, $(\lhd_\mu(\mathcal{P}X) \subset X \to Y, \cup, \theta)$. The curried mapping $\lhd_\mu$ is a homomorphism between these monoids,

$$\begin{array}{l} \lhd_\mu : (\mathcal{P}X, \cup, \emptyset) \longrightarrow (\lhd_\mu(\mathcal{P}X) \subset X \to Y, \cup, \theta) \\ \lhd_\mu(S_1 \cup S_2) = \lhd_\mu(S_1) \cup \lhd_\mu(S_2) \\ \lhd_\mu(\emptyset) = \theta \end{array}$$

Is the collection of mappings $\lhd_\mu$ for $\mu \in X \to Y$ a monoid? This collection of mappings is denoted $\lhd_{X \to Y}$. Before this questions is addressed a direct power monoid must be introduced, see [1, pages ? – ?]. Taking the monoid of maps under override, $(X \to Y, \dagger, \theta)$, the $\mathcal{P}X$ direct power monoid can be formed, $((X \to Y)^{\mathcal{P}X}, \dagger, \theta^{\mathcal{P}X})$. Now the space $\lhd_{X \to Y}$ is contained in the space $(X \to Y)^{\mathcal{P}X}$, thus the morphisms $\lhd_{\mu_1}, \lhd_{\mu_2} \in \lhd_{X \to Y}$ can be combined using the direct power operator, $\lhd_{\mu_1} \dagger \lhd_{\mu_2}$, but is the result another element of the space $\lhd_{X \to Y}$? If $S \in \mathcal{P}X$ is a set then

$$\begin{array}{ll}
(\lhd_{\mu_1} \dagger \lhd_{\mu_2})(S) & \\
\quad = & \lhd_{\mu_1}(S) \dagger \lhd_{\mu_2}(S) \\
\quad = & \lhd_S(\mu_1) \dagger \lhd_S(\mu_2) \\
\quad = & \lhd_S(\mu_1 \dagger \mu_2) \\
\quad = & \lhd_{\mu_1 \dagger \mu_2}(S)
\end{array}$$

Thus,

$$\lhd_{\mu_1} \dagger \lhd_{\mu_2} = \lhd_{\mu_1 \dagger \mu_2}$$

Hence a monoid of morphisms is formed, $(\lhd_{X \to Y} \subset (X \to Y)^{\mathcal{P}X}, \dagger, \lhd_\theta)$. This monoid is a submonoid of the above direct power monoid.

Much work remains to be done here, such as

- apply the above process to the mapping which removes a set from a map.
- the interrelations between these.
- the implications of the above results on the semantics of hashing.

These will be examined in future work.

5 Summary

This report introduced a number of isomorphic monoids, each of which contains a semi-group, which represents a space of hashing function and hash tables. The operations of enter and rehash are defined using the binary operation of one of the introduced semi-groups. Some new algebra is also introduced.

I wish to thank Andrew Farrell, who endured and corrected many of my errors. Dr. Mícheál Mac an Airchinnigh inspired the above work by his previous writings. I thank for this. I also wish to wish to thank the other members of the Irish School of Constructive (Applied) Mathematics, Dr. Andrew Butterfield, Dr. Hugh Gibbons, Alexis Donnelly, John Walsh, Colman Reilly and Dara Gallagher for their continued support of my work.

References

[1] Dara Gallagher Alexis Donnelly and Arthur Hughes. On the inheritance of monoid properties in indexed structures, a tale of three proofs. Technical report, Department of Computer Science, Trinity College Dublin, March 1996.

[2] Mícheál Mac an Airchinnigh. *Conceptual Models and Computing.* PhD thesis, Department of Computer Science, Trinity College Dublin, 1990.

[3] Mícheál Mac an Airchinnigh. Tutorial lecture notes on the irish school of the vdm. In S. Prehn and W. J. Toetenel, editors, *VDM'91:Formal Software Development Methods*, volume 2 of *Lecture Notes in Computer Science*, pages 141 – 237. Springer-Verlag, 1991.

[4] Mícheál Mac an Airchinnigh. Formal methods and testing. In *Tutorial Notes:*6th *International Software Quality Week*, Software Research Institute, 625 Third Street, San Fancisco, CA 94107-1997, 1993.

[5] Mícheál Mac an Airchinnigh. Grounded bills of materials. The glueing operator is introduced in this reports, 1996.

[6] R. Goldblatt. *Topoi:The Categtorical Analysis of Logic*, volume 98 of *Studies in Logic and the Foundations of Mathematics.* North-Holland, 1984.

[7] Arthur Hughes and Mícheál Mac an Airchinnigh. The inverse map monoid. This report develops an inverse map monoid isomorphic to the monoid of maps under override, August 1995.

[8] Donald E. Knuth. *Sorting and Searching*, volume 3 of *The Art of Computer Programming.* Addison-Wesley, 1973.