Semantic Search for Quantity Expressions

Bachelor Thesis DRAFT* EdN:1

Tom Wiesing
Supervisor: Michael Kohlhase
Co-supervisor: Tobias Preusser
Jacobs University, Bremen, Germany

May 19, 2015

Abstract

In this proposal we describe how to introduce units to MathWebSearch. The aim of the project is build an extensible semantics-aware system that searches a corpus of documents for quantity expressions. The project will be based on the existing MathWebSearch system and related technologies. [2][3] EdN:2 EdN:3

*EdNote: Remove draft status
[2]EdNote: Physics search
[3]EdNote: Re-write abstract

Contents

1 Introduction

1.1 The Problem Of Units

Units are everywhere. We encounter units and quantity expressions in everyday life wherever we go. When driving on the road we see a speed limits on signs (for example $30\frac{\text{km}}{s}$). When we go shopping there are different shoe sizes. When we buy something, we pay a currency of 30€. Everything is being quantified. This is also the case in science papers. Many, if not all, papers have 1 or more quantity expressions in them. Approximatly 1% of all letters in scienficic papers belong to a quantity expression[4]. EdN:4

This in itself is not a problem. The problem occurs when different units are used to describe the same quantity. In most of the world, the metric system is used to describe most quantities. However some countries still use other units which often leads to accidents.

These different units sometimes make it very difficult to talk about Quantity Expressions. One notable example for this is the *Mars Climate Orbiter* which was destroyed in 1999 when it entered the atmosphere of Mars because it received the non-standard units of *pound-seconds* instead of the expected *newton-seconds* [Boa].

1.2 State Of The Art: Unit Converters And The SI System

Out of convenience new units ones are constantly being invented. Hence there are many hundred units that already exist. Just converting between them is easy, googleing "unit converter" reveals about 12000000 results[5]. Even Google itself has implemented a unit converter into its search engine. EdN:5

However, all of these tools have a problem. They only convert units directly, meaning they require the user to enter the original units and desired units. This reuires the user to identity that there is a problem with units in the first place.

There is no tool which directly incorperates this translation of units into search results. Such a tool should find quantities independently of which units they are expressed in. It would make search efforts much easier, as fewer user input is required. It would no longer be required for the user to recognise that there is a problem.

In addition to this problem unit converters are usually very restricted in the units they support. They commonly store translation formulae for any pair of units to convert between. Hence they are difficult to extend with new units.

Their superficial handling of units usually does not take the underlying meaning, the semantics, into accoun On the other hand, The SI specification [dPeM] provides a very good insight into how units can be handled. It is precisely defined what each unit means and what kind of quantities can be expressed. It is a very formal approach. This approach does not take into account that it is sometimes less practical to user general units instead of specfic ones. Furthermore notation of SI units is not always exact[6]. EdN:6

Neither this very formal nor the previous approach with unit converters can be efficiently used to solve the problem.

[4] EdNote: Verify this
[5] EdNote: Perhaps quote the google result page here?
[6] EdNote: More explanation

1.3 Our Approach: A Semantic Quantity Expression Search

That is why we want to build a search engine that unifies these approaches. It should be capable to find occurrences of quantity expressions within documents, no matter of their representation. For this we need several components: (1) an extensible system flexible enough to convert between units when needed, (2) a so-called spotter that finds occurrences of quantity expressions within documents, (3) a search algorithm that can take a quantity expression from the user and find equivalent ones in the results from the spotter and (4) a front-end that allows the user to enter a quantity expression and receive the results.

For (1) we want to use a meta-mathematical theories approach. With the help of MMT this allows us to build an extendible unit system. This uses a concept of a theory graph in which units are related via so-called views and imports. These can be used to translate between them. Additionally, we can define a new unit and easily link it to any of the ones which have been defined previously. Point (2) will be taken on by Stiv Sherko in a seperate effort [She14]. The spotter finds quantity expressions inside documents which can almost directly be used by our system. For our search algorithm (3) we use a simple trick: When finding units, we normalise them to a normal form. This normal form is then used to efficiently index the harvest delivered from the previous step. Finally for (4) we built a frontend which allows the user to enter quantity expressions. It is deployed at [7]. EdN:7

[8] EdN:8

1.4 Overview

This thesis is organised as follows: In section 2 we start by giving an introduction to mathematical theory modeling. We proceed in section 3 to talk about how quantity expressions can be formalised and in section 4 we apply these insights in order to start building in our search engine. In section 5 we present the implementation in detail by describing how a search query is processed and how it is presented to the end user. After discussing the limits of this implementation as well as future work in section 6 we conclude in section 7.

[7] EdNote: Link the deploy site here.

[8] EdNote: Make a system diagram and refer to it here.

2 The Structure Of Mathematics: Theories, Views And Imports

Before we start looking at Quantity Expressions and how to build a search engine for them, we want to give an introduction to meta-mathmatical structure. We will use this knowledge later to build a better search engine. For this we first need to take a look at the concept of theories.

2.1 Modeling Mathematics With The Help Of Theories

Theories, in this sense, are simply a set of symbols. Each of the symbols optionally can have a type and a definition. Within each theory, we can then use these symbols to write down terms (or expressions) within this theory. Types and definitions of these symbols are terms themselves[1]. As a simple example of this, let us consider the theory of semigroups as seen in 1

Semigroup		
G	:	type
$\circ$	:	$G \to G \to G$
assoc	:	$\mathrm{ded}\,(\forall x \in G.\forall y \in G.\forall z \in G.(x \circ y) \circ z = x \circ (y \circ z))$

Figure 1: The Theory Of Semigroups.

In this theory, we define 3 symbols: G, $\circ$ and assoc. In the first line we define a type G. Next we define a function $\circ$ that takes 2 arguments of type G and returns another term of type G. In the last line, we make the statement that associativity holds[9]. [10] EdN:9 EdN:10

2.2 Extending Theories Using imports

Sometimes we want to extend theories without having to define everything again. For example, we want to say that a Monoid is a semi-group along with an identity element. In the semi-group example above, we have also used terms from other theories to define G as a type.

We can model this concept by using imports. An import from one theory into another makes symbols of the imported theory available in the target theory. In Figure 2 we can easily define a monoid.

Monoid		
import Semigroup		
e	:	G
id	:	$\mathrm{ded}\,(\forall x : G.x \circ e = e \circ x = x)$

Figure 2: The Theory Of Monoids Which Imports The Theory Of Semigroups As Defined In Figure 1.

[1] They are not terms over the same theory however.

[9] EdNote: possibly explain / mention Curry–Howard isomorphism

[10] EdNote: Explain what types are and why we are not using the pair constructor here

2.3 Views As Mappings Between Theories

However imports are not the only way theories can be related. If we have 2 theories, we sometimes want to have a map between them. In addition to the theory of monoids above we define the theory of non-negative integers in Figure 3.

Non-negative integers		
$\mathbb{Z}_0^+$	:	type
0	:	$\mathbb{Z}_0^+$
$+$	:	$\mathbb{Z}_0^+ \to \mathbb{Z}_0^+ \to \mathbb{Z}_0^+$
assoc	:	ded $\left(\forall x : \mathbb{Z}_0^+ . \forall y : \mathbb{Z}_0^+ . \forall z : \mathbb{Z}_0^+ . (x \circ y) \circ z = x \circ (y \circ z)\right)$
id	:	ded $\left(\forall x \in \mathbb{Z}_0^+ . x + 0 = 0 + x = x\right)$

Figure 3: The Theory Of Non-Negative Integers.

A map from the theory of monoids to the theory of positive integers should map all symbols from the theory of monoids to symbols from the theory of positive integers. Furthermore, such a map should be truth preserving, i. e. if I write down a true statement as a term over the theory of monoids and translate this term, it should still be true in the theory of positive integers. Such a mapping is called a *View* from the theory of monoids to the theory of Positive integers. Such a view ϕ could look as follows:

$$\phi = \left\{ \begin{array}{l} G \mapsto \mathbb{Z}_0^+ \\ e \mapsto 0 \\ \circ \mapsto + \\ \mathsf{assoc} \mapsto \mathsf{assoc} \\ \mathsf{id} \mapsto \mathsf{id} \end{array} \right\}$$

If we take a closer look at this view, we notice that we also have to map the imported symbols. This is needed so that we can translate any term or statement in theory of monoids to a term or statment into the theory of non-negative integers.

2.4 Building Theory Graphs

We have seen in the examples above that we can model mathematics with the help of theories, views and imports. To make this structure even more obvious, we can represent it in a graph, a so-called theory graph. We consider the theories as vertices of such a graph and the views and imports as edges. An example can be found in Figure 4.

2.5 Using MMT To Write Down Terms And Theories

MMT is a **M**odule system for **M**athematical **T**heories [RK13]. With the help of MMT we can represent theories, views and imports in *.mmt* files. It is easy to write these files and anyone without programming knowledge can easily extend existing ones. The objects defined in these files can then be used via an API to write down terms and transform these using definitions and views.

This furthermore allows us to easily model an extensible system for units for use within a search engine. We will come back to this later in section 4.1.

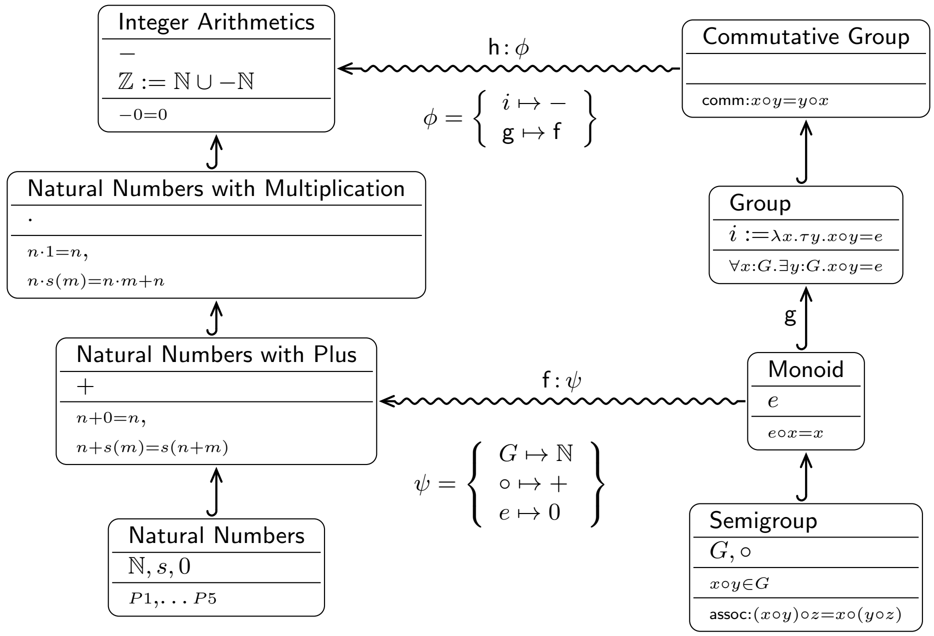


Figure 4: A Simple Theory Graph. Imports Are Represented As Solid Edges And Views As Squiged Edges.

3 The Structure Of Quantity Expressions

The first step in developing a good search engine for Quantity Expressions is to take a closer look at Quantity Expression.

3.1 Compositional Behaviour Of Quantity Expressions

[11] For this purpose let us take a look at: EdN:11

$$x = 25\ \frac{\mathrm{m}}{\mathrm{s}}$$

We notice that x consists of 2 parts, a scalar (25) and a scalar-free $\frac{\mathrm{m}}{\mathrm{s}}$. Furthermore, the unit consists of two primitive units m and s. Since they are divided by one another, we can conclude that x describes a velocity.

While m and s are certainly primitive units (they can not be decomposed further), it is not easy to define a unit as a composition of simple units. Consider the following example:

$$y = \frac{\mathrm{L}}{100\ \mathrm{km}}$$

y is certainly a unit and also a quantity expression. It consists of 2 sub-expressions, L and 100 km. The first one is a primitive unit and the second one a multiplication of a number and the unit km. It is thus reasonable to define the following 6 types of quantity expressions:

[11] EdNote: Rewrite this and the following section to explain notation vs. meaning (semantics); quote the SI report

1. A primitive unit, such as m (meter). This is the most obvious one.
2. the Multiplication of a quantity expression with a scalar.
3. the Division of a quantity expression by a scalar.
4. The multiplication $\cdot$ which takes 2 existing quantity expression and generates a new one, for example $\cdot(100, \mathrm{m}) = 100$ m
5. The division \which again takes 2 quantity expressions and generates a new one, for example $\backslash(\mathrm{m}, \mathrm{s}) = \frac{\mathrm{m}}{\mathrm{s}}$
6. The addition of two quantity expressions

This allows us to easily generate the quantity expressions x and y from primitive units m, s, L and km.

3.2 Dimensions Of Quantity Expressions

Now let us briefly examine the dimensions of Quantity Expressions. The dimension of a quantity expression is the type of quantity it expresses. For example 5 m describes some length. According to the International System of Units[12] there are seven basic dimensions: EdN:12

1. length
2. mass
3. time
4. electric current
5. temperature
6. luminous intensity
7. amount of substance.

In addition to these dimensions there are 2 more special dimensions: The *count* dimension (used for counting of objects) and the *none* dimension for dimensionless quantities. [13] EdN:13

Similar to the compositional behaviour of quantity expressions, dimensions can be multiplied and divided. Unlike quantity expressions however they can not be multiplied with numbers. Furthermore, when multiplying to quantity expressions of dimensions a and b their resulting dimension is $a \cdot b$, the multiplication of the dimensions. The same goes for division. In this regard the dimension of a quantity expression behaves like a type.

3.3 Mathematical Theory Of Quantity Expressions

The realisation that dimensions are types of quantity expressions leads us to our first formalisation of quantity expressions. We first define a theory of dimensions in Figure 5 and then import it to define a theory of Quantity Expressions in Figure 6.

[12] EdNote: Quote something properly here
[13] EdNote: Explain these more

Dimension		
dimension	:	type
none	:	dimension
count	:	dimension
length	:	dimension
mass	:	dimension
time	:	dimension
current	:	dimension
temperature	:	dimension
luminous	:	dimension
amount	:	dimension
$\cdot$	:	dimension $\rightarrow$ dimension $\rightarrow$ dimension
$\backslash$	:	dimension $\rightarrow$ dimension $\rightarrow$ dimension

Figure 5: A Formalisation Of The Theory Of Dimensions.

Quantity Expression		
import Dimension		
import Number		
QE	:	dimension $\rightarrow$ type
QENMul	:	$\forall x$: dimension.number $\rightarrow$ QE $(x) \rightarrow$ QE (x)
QENDiv	:	$\forall x$: dimension.QE $(x) \rightarrow$ number $\rightarrow$ QE (x)
QEMul	:	$\forall x$: dimension.$\forall y$: dimension.QE $(x) \rightarrow$ QE $(y) \rightarrow$ QE $(\cdot\,(x, y))$
QEAdd	:	$\forall x$: dimension.QE $(x) \rightarrow$ QE $(x) \rightarrow$ QE (x)
QEDiv	:	$\forall x$: dimension.$\forall y$: dimension.QE $(x) \rightarrow$ QE $(y) \rightarrow$ QE $(\backslash\,(x, y))$

Figure 6: The Choosen Formalisation Of The Theory Of Quantity Expressions.

Figure 5 defines the 9 basic dimensions and then dimension compositon via multiplication and division. Then we move on in 6 to define quantity expressions. Each quantity expression has a dimension (via the *QE* constant). This allows us to define basic units (which we will actually do in the next Figure.). With the *QENMul* and *QENDiv* symbols we can multiply and divide quantity expressions by numbers (for this case we actually need to import some theory of numbers to allow us actually write this down as a Term). Then we define multiplication and division of quantity expressions in such a way that dimensions multiply and divide appropriatly.

Now we need to introduce some basic units. Let us start by just defining meter in Figure 7. We can now write a term in this theory that expresses any number of meters.

Meter		
import QuantityExpression		
Meter	:	QE (length)

Figure 7: A Theory Defining The Primitive Unit Meter.

3.4 Transforming Quantity Expressions From One Form Into Another

This is a very nice start of a unit system[14]. Let us know define a few more units of length. EdN:14
In Figure 8 we show a few non-si units. Here we first define thoug as a quanity expression of length and then one-by-one define more units in terms of the previous one.

Non SI Lengths		
import QuantityExpression		
Thou	:	$\mathsf{QE}\,(\mathsf{length})$
Foot	:	$\mathsf{QE}\,(\mathsf{length}) = \mathsf{QENMul}\,(1000, \mathsf{Thou})$
Yard	:	$\mathsf{QE}\,(\mathsf{length}) = \mathsf{QENMul}\,(3, \mathsf{Foot})$
Chain	:	$\mathsf{QE}\,(\mathsf{length}) = \mathsf{QENMul}\,(22, \mathsf{Yard})$
Furlong	:	$\mathsf{QE}\,(\mathsf{length}) = \mathsf{QENMul}\,(10, \mathsf{Chain})$
Mile	:	$\mathsf{QE}\,(\mathsf{length}) = \mathsf{QENMul}\,(8, \mathsf{Furlong})$

Figure 8: A Theory Of Some Non-SI Units Of Length.

We now want to relate quantity expressions with units from the *Meter* theory to units from the *Non SI Lengths* theory. It is known that $1\mathsf{Thou} = 0.0000254\mathsf{Meter}$. This can be easily expressed with a view ψ between these to theories:

$$\psi = \left\{\ \mathsf{Thou} \mapsto \mathsf{QENMul}\,(0.0000254, \mathsf{Meter})\ \right\}$$

Even though the view just maps the symbol *Thou* to some Term in the Meter theory, we can also use it to transform any other term from the Thou Theory. Since all units are defined in terms of the previous one, we can just expand all definitions to get an expression containing only numbers and the unit *Thou*. Then we can use the view as normal to get a Quantity Expression in the *Meter* theory.

[14]EdNote: Better formulation

4 Making Quantity Expressions Searchable

Now that we know what quantity expressions are and how we can convert between equivalent representations, it is time to start about concrete algorithms for our search engine.

When querying this search engine with a quantity expression, we want to be able find occurrences of equivalent quantity expressions. For this we need several components:

1. a component that allow the user to enter quantity expressions,
2. a software to find quantity expressions within documents
3. a way to send quantity expressions from the user to the search engine
4. an algorithm to convert between different forms of a quantity expressions (or at least determine if 2 quantity expressions are equivalent) and finally
5. a unit system that is flexible enough to handle different kinds of units.

4.1 An Idea For An Extensible System

Let us start with point (5), a unit system that is flexible enough to handle all different of units. In Figure 9 you can find a small part of the unit graph we developed. At first, in the gray area, we define a basic version of our system as defscribe in section 3.3. Next we continue in the top left corner to define all the basic SI units. For the following we only look at the theories descrobing length and area units (however there are also some for all of the other dimensions).

We start with defining the non-SI unit *Thou* in the *Imperial Lengths A1*. In order to be able to convert between these units we apply what we learned in section 3.4 and create a view that maps this unit back to standard SI units. We then expand into *Imperial Lengths A2* and define a few more units incrementally based on *Thou* (we use an import to be able to do this).

Next, we define a new unit *Link* in the *Imperial Lengths B1* theory and define the unit *Rod* in *Imperial Lengths B2*. We then also make a view from *Imperial Lengths B1* to *Imperial Lengths A2*. Since *Imperial Lengths B2* imports *Imperial Lengths B1* this allows us to easily transform both of the new units back to SI if needed.

Next we want to define some units with the dimension Area. For this we first collect all known lengths and make them available in a *Imperial Lengths* theory via imports. Then we import this theory into a new *Imperial Area* theory where we can then define *Perch*, *Rod* and *Acre*.

But why do we define so many different theories? Why do we not simply define all the units of length in a single theory? Obviously, this one length theory would be very big. Furthermore, it would require anyone who wants to add more units to the system to add to this one theory, which can cause conflicts if there are 2 units with the same name (such as *Mile* and *(nautical) Mile*).

Defining units in the way we do it allows us to incrementally add new units to the system. If we have a view back to already known theories, it is easy to compare them back to the existing units.

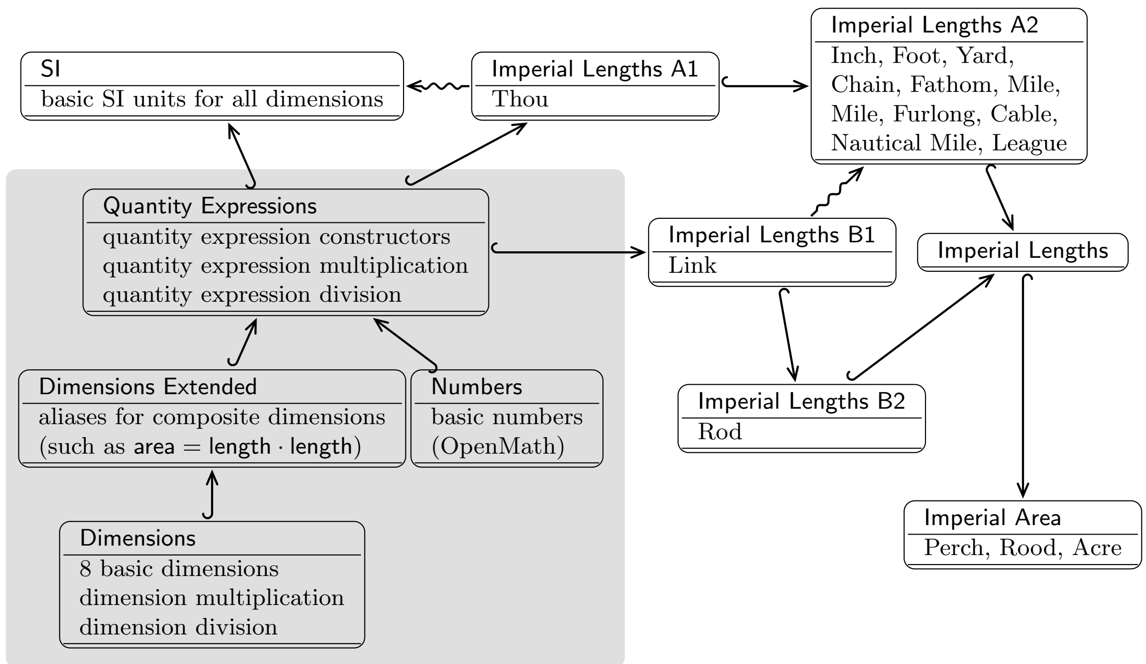


Figure 9: A Small Part Of The Graph Containing Unit Theories. Imports Are Represented As Solid Edges And Views As Squiged Edges. The Gray Area Contains Basic Theories That Are Used To Define Quantity Expressions.

4.2 Normalisation Of Quantity Expressions

To build a search engine however, it is inefficient to try to directly compare a search query with known quantity expressions. One way to compare quantity expressions is to normalise them to a suitable normal form. The normal form we came up with works in 2 steps.

We first turn a Term representing a Quantity Expression into a Term over the theory containing only standard units. In general we can choose any theory we want for this. We choose the *SI* theory for this so that the normal forms can be understood easily.

In order to transform a term to standard units, we just expand each unit (represented by constants) into a Term over the standard theory. This is achieved by translating it along a path in the theory graph. A path in the theory graph can consist out of 2 types of edges, Views and Imports. For views we can obviously use the mappings to translate the unit from the source to the target theory of the respective view. For imports however, it is not that simply. We can translate terms along them via definition expansion. This takes us to the imported theory from the theory that imports. We can use these edges as described and use standard graph theory algorithms to find a path for a unit to be recursively translated upon.

In the second step, we seperate the scalar from the units. For this, we first compute the scalar value of a quantity expression and then cancel out units which occur both in numerator and denominator of the quantity expression. This step is much easier then the first one, as we no longer have to use the theory graph. For this reason, we compute the normalisation of each unit only once by caching the result. Then we no longer need to look at the graph for our normalisation procedure at all.

Let us illustrate the process of normalisation a bit more. We will try to normalise the quantity expression

$$q = 42\frac{\text{Furlong}}{\text{Fortnight}}$$

First, we need to normalise *Furlong* (a quantity expression of length) to *Meter*. For this we first do definitional expansion:

$$\text{Furlong} = 10 \text{ Chain} = 10\,(22 \text{ Yard}) = \cdots = 10\,(22\,(3\,(12\,(1000 \text{ Thou}))))$$

Next, we need to use a view to turn this into meters:

$$10\,(22\,(3\,(1000 \text{ Thou}))) = 10\,(22\,(3\,(12\,(1000\,(0.0000254 \text{ Meter})))))$$

Similary, we use definitional expansion to find that:

$$\text{Fortnight} = (2\,(7\,(24\,(60\,(60 \text{ Second})))))$$

Substituting into q gives us the normal form of the quantity expression after the first step:

$$42\frac{10\,(22\,(3\,(1000\,(0.0000254 \text{ Meter}))))}{2\,(7\,(24\,(60\,(60 \text{ Second}))))}$$

Next, we want to extract the scalar component and compute it:

$$42\frac{10\,(22\,(3\,(12\,(1000\,(0.0000254)))))}{2\,(7\,(24\,(60\,(60))))} = 0.006985$$

We also extract the unit component:

$$\frac{\text{Meter}}{\text{Second}}$$

Finally we compose these again to get the normal form:

$$0.006985\ \frac{\text{Meter}}{\text{Second}}$$

4.3 Serialising Quantity Expressions To XML

Now that we have a normal form of units, we want to be able to exchange quantity expressions between the user, the search engine and the harvesting component. The W3C has written a note on how to handle Units in MathML [Gro]. MathML is an XML serialisiation of Mathmatical expression. We orient ourselves on the format. For this serialisation we start with a term over some unit theory. The serialisation of the term

$$0.006985\ \frac{\text{Meter}}{\text{Second}}$$

can be found in Figure 10. Let us take a closer look to understand this serialisation. We translate each component of the Term individually and then assemble them together.

We translate the unit (constant) *Meter* to `<csymbol cd='SIBase'>Meter</csymbol>`. This XML Tag expresses that we are talking about the symbol *Meter*. The cd attribute stands for content directory.In our case this is the name of the theory were the symbol was declared.

```
<apply>
  <times />
  <cn type='float'>0.006985</cn>
  <apply>
    <divide />
    <csymbol cd='SIBase'>Meter</csymbol>
    <csymbol cd='SIBase'>Second</csymbol>
  </apply>
</apply>
```

Figure 10: Serialisation Of A Simple Quantity Expression.

We translate multiplication and division by using `<apply><times />...</apply>` and `<apply><divide />...</apply>` respectively. We can use quantity expressions directly here. If we want to multiply with or divide by numbers, we use `<cn type='float'>...</cn>` instead of the `<csymbol>` tags.

This format allows us to easily exchange quantity expressions. It is also easy to send quantity expressions from the user to the backend (the core component of the search engine.)

4.4 Finding Units Inside Documents

Apart from the frontend, which we will talk about in section **??**, there is only one major component left: Finding and matching quantity expressions inside documents. As we have outlined in the previous sections, we can easily compare them with the help of normalisation. If we have a list of quantity expressions and where they occur, we can use the following algorithm to easily find them:

1. First find quantity expressions inside documents and store them in a list along with their origin.
2. Normalise each of them as described before.
3. Store the normalised forms in an efficient index along with their original versions and origins.
4. When a query is sent, normalise the query.
5. Next look in the index for the normalised query.
6. Then deliver the original versions and their origins from the index.

This algorithm allows us to easily and efficiently find quantity expression in documents after we have built up an index. In order to build up this index we need a list of quantity expressions and occurrences inside documents. A software that fullfills this task is called a Spotter. In our case the spotter is developed by Stiv Sherko in a seperate effort [She14].

5 The Implementation: The Search Process

Now that we have discussed all the theoretical components of the search engine, it is time to move on to the actual implementation. In this section we describe in detail how a search query is processed and how it is presented to the end user.

5.1 Entering a Query

As to give an overview of what exactly happens during a typical search, we will show what happens during a normal search process. From the user perspective, everything happens inside a web browser. When the user visits the demo page [Wie], they are presented with a frontend as shown in Figure 11.

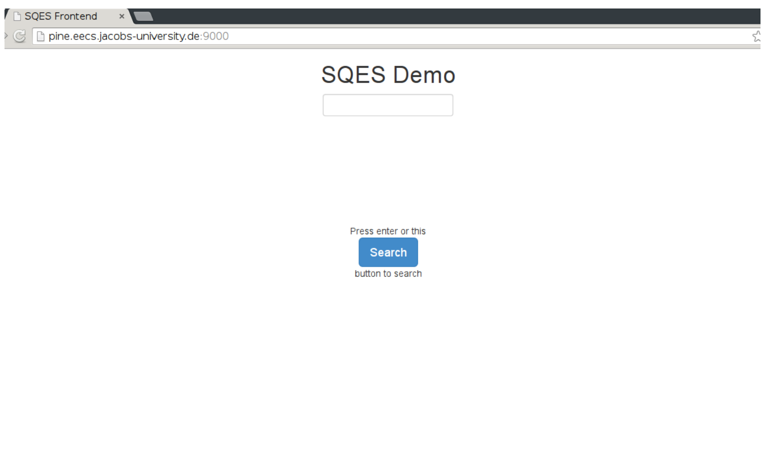


Figure 11: The main page of the frontend.

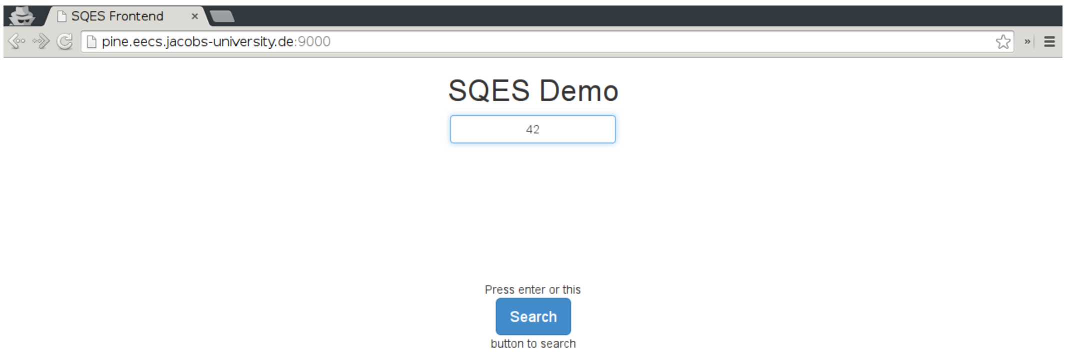
Because the frontend is running inside a Webbrowser it is written using HTML, CSS and JavaScript. Furthermore it uses the frameworks jQuery [jQu] and Bootstrap [Twi]. As seen in the figure, the main components are a text field to enter a quantity expression as well as a search button.

With the text field it is easily possible to enter composed quantity expression.

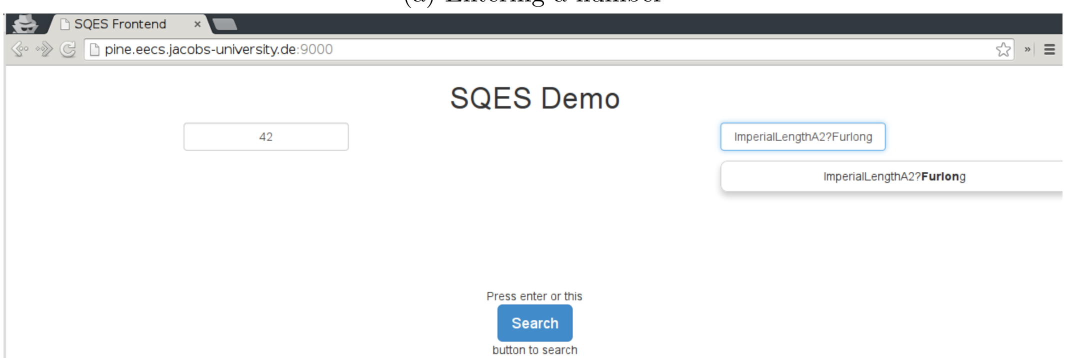
When searching for a certain unit it is neccessary to tell the backend which *unit theory* this unit comes from. In order to make this easier, the GUI has an autocompletion feature. It is only required to enter the name of the unit itself and the possible theories this unit can come from will be suggested automatically.

Apart from entering a simple unit it is possible for the user to compose quantity expressions by multiplication and division. If the keys "*" and "/" are pressed, the text field splits into two different fields. In each of them another quantity expression can be entered.

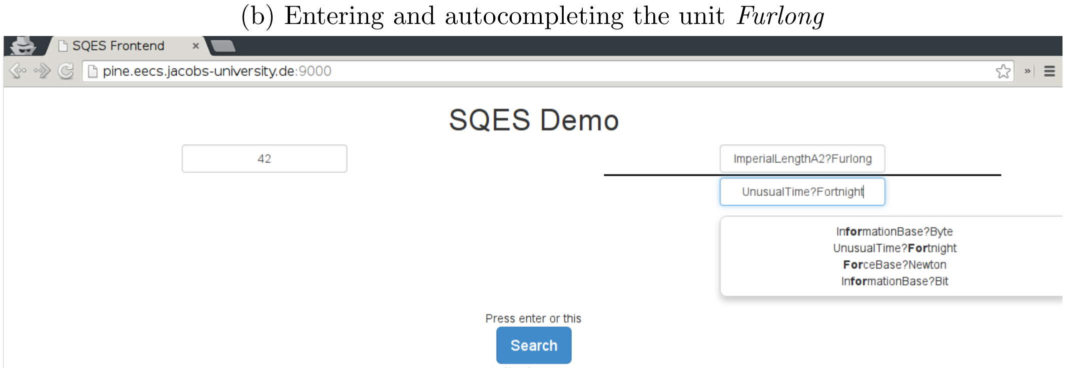
In order to facilitate entering division and multiplication by numbers, it is also possible to enter numbers into these fields. With this input unit it is also possible to enter all the quantity expressions as described by the formalism above. In Figure 12 the process of entering the query $42\frac{\text{Furlong}}{\text{Fortnight}}$ is shown.



(a) Entering a number



(b) Entering and autocompleting the unit *Furlong*



(c) Entering the final component.

Figure 12: Process of entering the quantity expression $42\frac{\text{Furlong}}{\text{Fortnight}}$ into the frontend.

5.2 Processing a typical query

After the quantity expression has been entered, the search can be started by pressing the *Search* button or hitting the *Enter* key. The query inputted into the text field is then serialised to XML as shown in in section 4.3.

5.3 Presenting the results

6 Czrrent Limits and Future work

[15] EdN:15

6.1 Relative vs absolute units

* type equalities * requirements of the graph * (in)stabilities of MMT * davenport units (relative units)

6.2 Extension Of The Theory Graph Of Units

[16] EdN:16

6.3 Integration With MathWebSearch

[17] EdN:17

6.4 Enhancement of the Spotter

[18] EdN:18

[15] EdNote: Write this section
[16] EdNote: Write this section
[17] EdNote: Write this section
[18] EdNote: Write this section

7 Conclusion

[19] EdN:19

[19]EdNote: Write this section

* what did we want to do? * what did we achieve? * were we successfull * end with: there is still a lot to be done

References

[Boa] Mishap Investigation Board. Mars Climate Orbiter, Phase I Report. ftp://ftp.hq.nasa.gov/pub/pao/reports/1999/MCO_report.pdf. 10 November 1999.

[dPeM] Bureau International des Poids et Mesures. The International System of Units (SI). http://www.bipm.org/utils/common/pdf/si_brochure_8_en.pdf. 8th edition, 2006.

[Gro] W3C Working Group. Units in MathML. http://www.w3.org/TR/2003/NOTE-mathml-units-20031110/. Note 10 November 2003.

[jQu] jQuery Foundation. jQuery. https://jquery.com/.

[RK13] Florian Rabe and Michael Kohlhase. A scalable module system. *Information & Computation*, 0(230):1–54, 2013.

[She14] Stiv Sherko. Extracting quantity and unit semantics from technical documents. Bachelor thesis proposal, Computer Science, Jacobs University, Bremen, 2014.

[Twi] Twitter, Inc. Bootstrap. http://getbootstrap.com/.

[Wie] Tom Wiesing. SQES Frontend. http://pine.eecs.jacobs-university.de:9000/.