Attributed Range Algebra

Extending Core Range Algebra to Arbitrary Structures

Gavin Thomas Nicol
Chief Technology Officer
Red Bridge Interactive, Inc.
Providence, RI
email: gtn@rbii.com

June 11, 2002

Abstract

Core Range Algebra defines an algebra of sequences and ranges over sequences that can be used to mark structural boundaries in data, without explicit inline markers. In the context of text, Core Range Algebra offers a single model for both structured texts, such as XML documents, and semi-structured texts, such as email and memos. In addition, Core Range Algebra allows structures to overlap, going beyond the expressive power of popular markup languages, such as XML in the structural dimension. The key limitation of Core Range Algebra is in it's inability to model anything other than basic structural boundaries. This paper presents Attributed Range Algebra, an extension of Core Range Algebra that provides greater expressive power such that it can model all structures found in current markup languages.

1. Introduction

In [nicolE2002] is defined Core Range Algebra: an algebra of sequences and ranges over sequences that can be used to mark structural boundaries in data, without explicit inline markers. For text, Core Range Algebra offers a single model for both structured texts, such as XML documents, and semi-structured texts, such as email and memos. While Core Range Algebra goes beyond typical markup in that it also allows overlapping structures, it is also limited in it's abilities to model anything other than basic structural boundaries. Take the following piece of text:

T	**h**	**e**		***q***	***u***	***i***	***c***	***k***		**b**	**r**	**o**	**w**	**n**		**f**	**o**	**x**
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18

Sample text sequence.

In core range algebra, the italic text could be identified, using a RANGE *R* with *R.start* equal to 4 and *R.length* equal to 5. We represent this as *{4,5}* and we shall adopt this form throughout the rest of this paper.

While the range {4,5} tells us that *something* is at that position in the data, it doesn't tell us *what* is there. The following shows some possible ways the text above could be marked up in HTML, LaTeX and Lout.

```
<p>The <i>quick</i> brown fox.</p>

The \emph{quick} brown fox.

The @i{quick} brown fox.
```

At the level of the markup, these all share one trait that cannot be expressed in Core Range Algebra: the ability to *label* the range of characters, or in other words, the ability to associate *attributes* with the sequence. It can be argued that the entire purpose of markup is to annotate regions of text with additional information, and so, Core Range Algebra cannot represent markup as such.

This paper extends Core Range Algebra such that it is also possible to associate attributes with ranges, thereby making it possible to model both the syntax and the metadata associated with structural boundaries within data. This extension is called *Attributed Range Algebra*.

2. Attributed Range Algebra

In Core Range Algebra, a range is defined as a *{start,length}* pair that identify a start and end position within a given sequence. While useful in and of itself, it is beneficial to extend this to allow the range to be attributed, or in other words, to attach metadata to the range. Take for example, the following document.

```
EBISU

Ebisu is the Japanese god of money and good fortune.
```

The range {0,5} identifies the sub-sequence "EBISU"but tells us nothing more. If the range has an added "type" attribute to get {0,5,type:header}, the attributed range obviously carries more information. It is also obvious that this bears some resemblance, in terms of purpose, to XML markup, and if stored, would essentially be a form of external markup[nelsonEmbed]. This additional behavior can be accommodated with a simple extension to the Core Range Algebra data types, described below.

Attributed Range Data Type

Definitions

attributed range (arange)
A range as defined in [nicolE2002] extended to support an arbitrary *set* of associated attributes. Attributes can be accessed via the dot operator, just as *start* and *length* can be. So if an attributed range **`A`** has attribute *foo*, then **`A.foo`** would retrieve the value of the attribute or **`null`** (see *GetAttributeValue* below).

Extended Syntax

Same: ARANGE ARANGE → BOOLEAN
equals: ARANGE ARANGE → BOOLEAN
HasAttribute: ARANGE STRING → BOOLEAN

GetAttributeValue: ARANGE STRING → SEQUENCE
SetAttributeValue: ARANGE STRING SEQUENCE → ARANGE
RemoveAttribute: ARANGE STRING → ARANGE
GetAttributesNames: ARANGE → SEQUENCE
GetAttributes: ARANGE → SEQUENCE

Predicates

Equals(A, B)
Tests whether ARANGE *A* and ARANGE *B* are equivalent by means of deep comparison. This is the same as comparing the *start* and *length* of the range, and comparing the set of attributes for equality.

Same(arange, arange)
Tests whether a given attributed range is the same as another. It is possible for two ranges to be equal, but not the same.

HasAttribute(arange, attr)
Tests whether *arange* has an attribute *attr*, or $arange.attr \neq null$

Operations

GetAttributeValue(R,A)
Given the ARANGE *R*, returns the value of the attribute *A* or *null* if *R* has no such attribute.

SetAttributeValue(R,A,V)
Given the ARANGE *R*, set the attribute *A* to the value *V* provided, and return a new ARANGE. *V* is constrained to not be *null*.

RemoveAttribute(R,A)
Given the ARANGE *R*, remove the ATTRIBUTE with name *A* from it and return a new ARANGE. This is essentially the same as copying the entire range, including all attributes, except the attribute with the name *A*.

GetAttributesNames(R)
Returns a SEQUENCE of STRING containing the names of all attributes of ARANGE *R*. The ordering of names in the sequence is undefined, so two invocations could potentially produce two sequences that may not be equal.

GetAttributes(R)
Returns a SEQUENCE of ATTRIBUTE containing the set of all attributes of ARANGE *R*. The ordering of attributes within the set is undefined.

String Data Type

Definitions

string
A SEQUENCE of INTEGER. The range of values the integers can take corresponds to the range of character codes in ISO10646/Unicode.

Extended Syntax

COMPARE: STRING STRING → INTEGER

Operations

Compare(A, B)
Iterate over each INTEGER in *A* and compare the value to the corresponding INTEGER in *B*. If the code point is less than the corresponding code point in *B*, return -1, if greater, return 1, if equal, continue. If *A* and B are of different lengths, and all comparisons of integers up to the length of the shortest SEQUENCE compare equal, and *A* is the longer sequence, 1 is returned, otherwise -1.

Attribute Data Type

Definitions

attribute
A *{name, value}* pair, where name is the key into the *set* of attributes on a given ARANGE, and value is the associated SEQUENCE representing the value of the ATTRIBUTE. These can be accessed using the dot operator, so for a ATTRIBUTE **`A`**, **`A.name`** represents the name, and **`A.value`** represents the value.

Extended Syntax

SAME: ATTRIBUTE ATTRIBUTE ⟶ BOOLEAN
EQUALS: ATTRIBUTE ATTRIBUTE ⟶ BOOLEAN

GETNAME: ATTRIBUTE ⟶ STRING
GETVALUE: ATTRIBUTE ⟶ SEQUENCE
SETVALUE: ATTRIBUTE SEQUENCE ⟶ ATTRIBUTE

Predicates

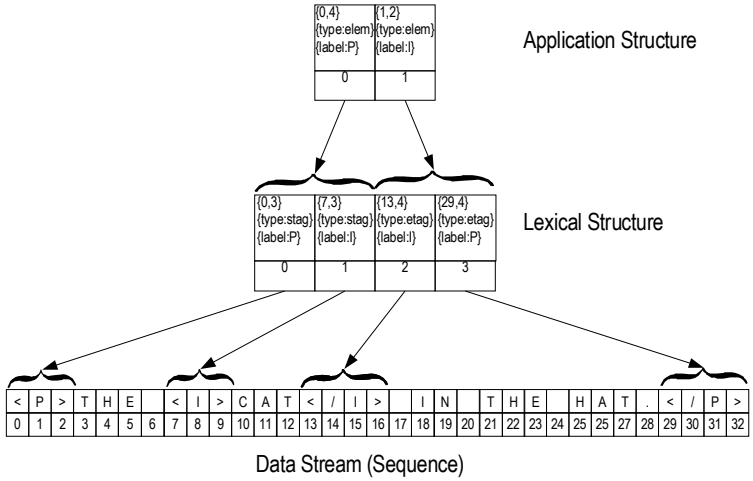
Same(A,B)
Tests to see if *A* and *B* are the same ATTRIBUTE.
Equals(A,B)
Checks to see if *A* and *B* are equivalent. This means that **`A.name`** and **`B.name`** are equal and that **`A.value`** and **`B.value`** are equal as defined by the *Equals* operation for SEQUENCE.

Operations

GetName(A)
This will return the STRING corresponding to **`A.name`**.
GetValue(A)
This will return the SEQUENCE corresponding to **`A.value`**.
SetValue(A,V)
This will create a new ATTRIBUTE with *V* as the value and the name **`A.name`**.

3. Application to Markup

Given the extended data types above, it is possible to represent arbitrary data structures, and especially arbitrary markup structures, in terms of attributes, ranges, and sequences. Typically, there will be 3 or more sequences needed to capture the information in given stream of data. An example follows:



Deriving Structure Via Range Hierarchies

In the above, there are 3 *layers* of sequences, and two layers of ranges over sequences. The lowest SEQUENCE in the hierarchy is simply the sequence making up the data stream. While the above shows it to be a sequence of characters, this is not strictly necessary.

The next layer in the diagram is the *lexical structure* layer. This layer identifies the tokens that are used for delineating the markup in the text fragment shown. In the example above, the start tag and end tags of element are identified, and ranges are created holding information both about the start offset and length in the data stream, but also metadata identifying what the range identifies. This information forms the attributes of the ranges.

The final layer in the above operates over the lexical structure layer, and here is called the *application structure* layer. This layer takes the sequence from the lexical structure layer, and imposes a canonical ordering on it. In the above, the canonical ordering is by the starting offset in the data stream of the ranges (this is the *StartOrder* operation defined in [nicolE2002]), and then identifies ranges within that sequence that form higher-level structures. In the example above, a new sequence of ranges that identifies elements is created. This process of deriving structures is similar to that found in many parsers for data streams, such as those created by `lex` and `yacc`.

Note that while the above has 3 layers, this is also not strictly necessary. It is possible for many languages, to go directly from syntax to application structures, and in other cases, it may well be beneficial to have more than 3 layers. The example above is typical, not canonical.

Deriving Hierarchical Structures

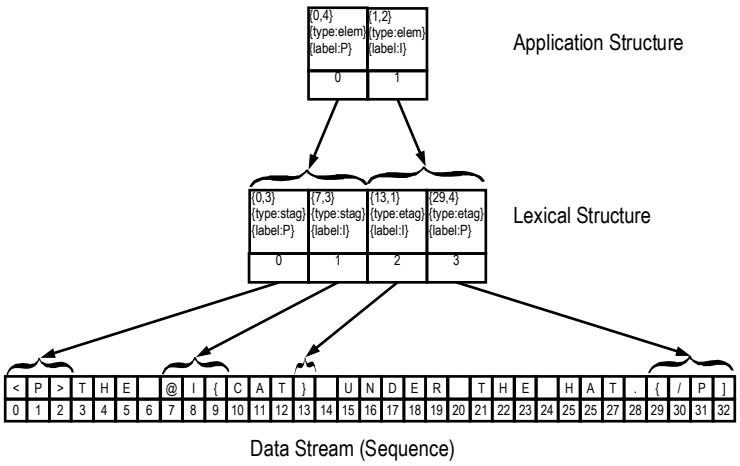
One of the things that may not be immediately obvious in the diagram above is that the containment hierarchy is available to the application. In [nicolE2002] the *Ancestor*, *Descendant*, *Parent* and other operations are defined in terms of range extents, and those operations are applicable to any sequence. In the above, the operators would best operate on the *application structure* layer, where the traditional XML notion that the <p> element is a parent of the <i> element could be derived, such that a DOM-like API could be implemented over top of the underlying operations.

One of the advantages of using ranges in such as way is that it becomes very easy to suppress markup, and thereby hide it from higher-level API's, while maintaining data integrity. For example, to remove all <i> elements from the tree, one need only remove, or suppress the ranges in the SEQUENCE over which the API operates. In the example above, that would leave a single <p> element that spanned the entire document.

While the fragment above shows strictly nested markup, the sequence data type and ranges make it possible to represent overlapping markup structures. Using markup suppression it is then possible to derive arbitrary sets of markup, and in particular, arbitrary well-formed trees. This is very much in line with the work of Patrick Durasu et al. [durasuE2002] who have been doing very interesting work on Just In Time Trees (JITT).

Syntax Independence

Another thing that may not be obvious is that everything from the lexical structure level upward is independent of syntax. Take the following diagram for example.



Mixed Syntax Range hierarchy

In this diagram, a mixture of XML, LaTeX and and invented markup is used at the syntax level, but only small adjustments to the ranges in the lexical structure layer are needed in order to accommodate the difference. Any application that deals in the structures defined at the application structure level will be independent of the syntax changes. This opens up the possibility of using domain-specific languages with a single processing model, and a single data model, relying on *edge transformations* to obtain the information the application wants.

Note that the goal of attributed range algebra is *not* to belittle the value of standardized syntax's, such as XML. For purposes of interchange, the benefit of using a standard syntax can often far outweigh the inconvenience of doing so, even when the use feels forced. Instead, the goal of attributed range algebra is to recognize that there are many cases where a domain-specific syntax might be better, and to provide a common processing framework, independent of the syntax, and a framework for arbitrary transformations between syntax's.

4. Conclusions

Core Range Algebra is a simple algebra that operates over SEQUENCES, RANGES, and SEQUENCES of RANGES. This paper presents and extension to Core Range Algebra that allows arbitrary sets of metadata to be associated with ranges. This extension is called Attributed Range Algebra, or ARA.

ARA provides a common infrastructure for modeling and manipulating arbitrary data streams, and in particular, arbitrary markup languages. By recognizing that syntax is typically not what an application processes, but rather the structures encoded by the syntax, attributed range algebra provides a means for writing applications that are independent of syntax, thereby allowing arbitrary syntax's to be used to represent the same logical structures.

In addition, attributed range algebra provides a clean means for extracting arbitrary structures from an arbitrary set of identified ranges in a data stream, allowing data to be interpreted as seen fit by the consumer.

Future Work

While Attributed Range Algebra is a significant improvement over Core Range Algebra, there is still much work to do. In this paper, it has been shown that arbitrary syntax's can result in identical high-level structures, but what has not been shown is a basis for handling arbitrary sequences to create layers of sequences of ranges. In a future paper, a formalism known as a Range Constructor will be presented that covers this aspect of Attributed Range Algebra.

In addition to the Range Constructor formalism, a future paper will present RAQUEL, the Range Algebra QUEry Language. That paper will present a query language built around Attributed Range Algebra allowing querying, extraction, and transformation of data in arbitrary, and potentially mixed syntax's.

Further papers will show how forest regular expressions can be used to validate arbitrary syntax's, and how node-centric data models, such as those typically associated with XML, can be mapped into Attributed Range Algebra.

5. Bibliography

durasuE2002: Durasu, Patrick, , 2002
nelsonEmbed: Nelson, Theodor Holm, Embedded Markup Considered Harmful, Fall 1997
nicolE2002: Nicol, Gavin T., Core Range Algebra, 2002