Toward the Discovery and Extraction of Money Laundering Evidence from Arbitrary Data Formats using Combinatory Reductions

Alonza Mumford, Duminda Wijesekera

George Mason University

amumford@gmu.edu, dwijesek@gmu.edu

November 19, 2014
Overview

1. Problem, Background & Architecture
2. Motivating Example
3. Descriptive Data Modeling with DFDL
4. Ontological Modeling with XLink
5. Compiler Specification in CRSX
6. Future Work
—The "wild, wild west analogy" is applicable to the variety of non-standardized, structured and unstructured data that exist in public and private domains

—This phenomena disrupts the opportunity to "get to" this heterogeneous raw data in a generic manner, represent the connections between this data that is distributed, and make new discovers based partially on those connections.

—The manner in which we express these connections must contextualize specific data with a conceptual understanding of a problem, investigation or inquiry without having to change or alter the data itself
Background (Challenges)

Challenge 1
To provide a mechanism that can be used to describe and access any number of data formats.

Challenge 2
To provide a lightweight, metadata-based discovery and extraction of arbitrary data fragments from raw data stores not co-located with each other. The intent is to avoid some of the system development and maintenance costs associated with major data conversion, and database storage and indexing.

Challenge 3
To provide a mechanism that incorporates ontological engineering for describing the meaning of data fragments (i.e., parts of data) within the context of a particular domain of understanding, inquiry or investigation.
Motivating Example: Analysis of a Money Laundering Scheme

**Step 1: Placement Stage**
A number of policies entered into by the same insurer (i.e. a person or company that underwrites an insurance risk) for small amounts and then canceled early at the same time [1, 2].

**Step 2: Layering Stage**
Requests for return premiums in currencies different to the original premium or requests for return premiums to an account different from the original account [1, 2].

**Step 3: Integration Stage**
Return premium being credited in different currency or different account, which represents launder money [1, 2].
Example (part 1): Inspection of a Policy Account Record

(a.1) Input "policy account record" data:

1. PLCYACC/741032−1071/
2. DATE/2013−09−28/
3. PLCYHLD/Allegier, Cox & Associates, Inc./
4. INSUR/ALI Corp./
5. PAYER/Grupo Palermo S.A./
6. PAYCUR/Peso (ARG)/
7. PRMAMT/42004.98/

(a.2) DFDL generated XML model:

```
<policyAccountRecord>
  <policyAccountIdentifier>741032−1071</policyAccountIdentifier>
  <policyStartDate>2013−09−28</policyStartDate>
  <policyHolder>Allegier, Cox & Associates Inc.</policyHolder>
  <policyInsurer>ALI Corp.</policyInsurer>
  <payerName>Grupo Palermo S.A.</payerName>
  <payerCurrency>Peso (Argentine)</payerCurrency>
  <premiumAmount>42004.98</premiumAmount>
</policyAccountRecord>
```
Example (part 2): Inspection of a Credit Request Record

1 (b.1) Input "credit request record" data:

- CRDREQ%]741032−1071%
- CRDATE%]2013−11−02%
- PAYEE%]Allegier, Cox & Associates, Inc.%
- PAYCUR%]USD%
- CRDAMT%]5000.00%

2 (b.2) DFDL generated XML model:

```xml
<creditRequestRecord>
  <creditRequestId>741032−1071</creditRequestId>
  <creditRequestDate>2013−11−02</creditRequestDate>
  <payeeName>Allegier, Cox & Associates, Inc.</payeeName>
  <payeeCurrency>USD</payeeCurrency>
  <creditAmount>5000.00</creditAmount>
</creditRequestRecord>
```
—Data Format Description Language [3] is not a data-format... it’s a way to describe any data format (i.e., dense binary and text)
—Prescriptive standards (e.g. ASN.1, JSON) provides a structure for data as certain people think it should be in lieu of the structure of data as it actually is
—DFDL’s syntax uses a subset of XML Schema with DFDL annotations to represent the physical properties of data
—As of January 2011, DFDL v1 is a "Recommendation" standard of the Open Grid Forum (OGF) designed to "enable efficient representation of data... essential to high performance data exchange in grid computing and... distributed computing."
### Logical Datatypes & Constraints

<table>
<thead>
<tr>
<th>Logical Datatypes &amp; Constraints</th>
<th>Context Free Grammar (CFG) example</th>
<th>Higher-Order Abstract Syntax (HOAS) example</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Structures</strong> (xs:complexType)</td>
<td>2 <code>&lt;dcl_xs&gt; ::= &quot;&lt;&quot; XS_COMPONENT </code>&lt;stmt&gt;* &quot;&lt;/&quot; XS_COMPONENT &quot;&gt;&quot; 11 XS_COMPONENT ::= &quot;xs:complexType&quot;</td>
<td>XsComponent[ComplexType]</td>
</tr>
<tr>
<td><strong>Atomic data values</strong> (xs:simpleType)</td>
<td>11 XS_COMPONENT ::= &quot;xs:simpleType&quot;</td>
<td>XsComponent[SimpleType]</td>
</tr>
<tr>
<td><strong>Ordering</strong> (xs:sequence or xs:choice)</td>
<td>11 XS_COMPONENT ::= &quot;xs:sequence&quot;</td>
<td>XsComponent[Sequence], XsComponent[Choice]</td>
</tr>
<tr>
<td><strong>Occurences</strong> (xs:minOccurs or xs:maxOccurs)</td>
<td>5 <code>&lt;stmt&gt; ::= XS_ATTRIBUTE &quot;;&quot; </code>&lt;xs_attribute_value&gt;</td>
<td>ComponentAttribute[MinOccurs], ComponentAttribute[MaxOccurs]</td>
</tr>
<tr>
<td></td>
<td>16 XS_ATTRIBUTE ::= &quot;xs:minOccurs&quot;</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>XsComponent[Sequence], XsComponent[Choice]</td>
</tr>
<tr>
<td></td>
<td>8 <code>&lt;xs_attribute_value&gt; ::= </code>&lt;xs_number&gt;`</td>
<td></td>
</tr>
</tbody>
</table>

---
## Modeling of Physical Representations

<table>
<thead>
<tr>
<th>DFDL Physical Representation Properties</th>
<th>Context Free Grammar (CFG) example</th>
<th>Higher-Order Abstract Syntax (HOAS) example</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Physical types</strong> (dfdl:representation)</td>
<td>5 ( \text{stmt} ) ::= DFDL_ATTRIBUTE &quot;=&quot; &lt;dfdl_attribute_value&gt;</td>
<td>FormatProperty[Represenation]</td>
</tr>
<tr>
<td></td>
<td>18 DFDL_ATTRIBUTE ::= &quot;representation&quot;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>7 &lt;dfdl_attribute_value&gt; ::= &lt;dfdl_enum_number&gt;</td>
<td></td>
</tr>
<tr>
<td><strong>Delimiters</strong> (dfdl:initiator, dfdl:separator, dfdl:terminator)</td>
<td>18 DFDL_ATTRIBUTE ::= &quot;initiator&quot;</td>
<td>FormatProperty[Initiator], FormatProperty[Separator], FormatProperty[ Terminator]</td>
</tr>
<tr>
<td></td>
<td>7 &lt;dfdl_attribute_value&gt; ::= &lt;dfdl_string_value&gt; ( \text{</td>
<td>} ) &lt;reg_exp_value&gt;</td>
</tr>
<tr>
<td><strong>Extraction of elements</strong> (dfdl:lengthKind)</td>
<td>18 DFDL_ATTRIBUTE ::= &quot;lengthKind&quot;</td>
<td>FormatProperty[LengthKind]</td>
</tr>
<tr>
<td></td>
<td>7 &lt;dfdl_attribute_value&gt; ::= &lt;dfdl_enum_value&gt;</td>
<td></td>
</tr>
<tr>
<td><strong>Points of uncertainty</strong> (dfdl:discriminator)</td>
<td>3 &lt;dcl_dfdl&gt; ::= &quot;&quot;&lt;&quot; DFDL_ADMIN &lt;stmt&gt;* &quot;&quot;&gt;&quot; &lt;dcl_dfdl&gt; &quot;=&quot;/&quot; DFDL_ADMIN &quot;&quot;&gt;&quot;</td>
<td>DfdlValidation[Discriminator]</td>
</tr>
<tr>
<td></td>
<td>15 DFDL_ADMIN ::= &quot;dfdl:discriminator&quot;</td>
<td></td>
</tr>
<tr>
<td><strong>Detecting occurrences</strong> (dfdl:occursCount)</td>
<td>18 DFDL_ATTRIBUTE ::= &quot;occursCount&quot;</td>
<td>FormatProperty[OccursCount]</td>
</tr>
<tr>
<td></td>
<td>7 &lt;dfdl_attribute_value&gt; ::= &lt;non_neg_int_value&gt; ( \text{</td>
<td>} ) &lt;dfdl_exp_value&gt;</td>
</tr>
</tbody>
</table>
Example (part 3): Policy Account Record Schema

```xml
<xs:element name="policyAccountRecord" minOccurs="0" maxOccurs="unbounded" dfdl:lengthKind="implicit">
  <xs:complexType>
    <xs:sequence dfdl:sequenceKind="ordered">
      <annotation>
        <xs:appinfo source="http://www.ogf.org/dfdl/v1.0">
          <dfdl:element_representation="text" encoding="ascii" lengthKind="delimited" sequenceKind="ordered" initiator="//" separator="/" separatorPosition="infix" separatorPolicy="required"/>
        </xs:appinfo>
      </annotation>
      <xs:element name="policyAccountId" type="xs:string" dfdl:lengthKind="explicit" dfdl:length="20"/>
      <xs:element name="policyStartDate" type="xs:string" dfdl:lengthKind="explicit" dfdl:length="20"/>
      <xs:element name="policyHolder" type="xs:string" dfdl:lengthKind="explicit" dfdl:length="20"/>
      <xs:element name="policyInsurer" type="xs:string" dfdl:lengthKind="implicit"/>
      <xs:element name="payerName" type="xs:string" dfdl:lengthKind="implicit"/>
      <xs:element name="payerCurrency" type="xs:string" dfdl:lengthKind="implicit"/>
      <xs:element name="premiumAmount" type="xs:string" dfdl:lengthKind="implicit"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```
Use XLink-based ontological engineering and analytical reasoning to define the concepts relevant to the money laundering domain [4, 5]:

— A concept definition conveys the name of an evidentiary fact and its value data type

— A conceptual ontology of the "anti-money laundering" domain is given to show the kinds of classes and properties used in the domain

— Classes are identified by nodes and properties are identified by directed paths or arcs

— The conceptual labels associated with properties represent composition (i.e., hasPart) and equivalence (i.e., sameAs) relations between classes.
## Ontological Modeling with XLink

<table>
<thead>
<tr>
<th>Concept</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Classes</td>
<td>propertyAccountRecord, cancelRequestRecord and creditRequestRecord (ref: figs. 5 and 8)</td>
</tr>
<tr>
<td>Instances</td>
<td>An instance of a propertyAccountRecord is one bearing &quot;741032-1071&quot; as the policyAccountIdentifier (ref: fig 2, a.1).</td>
</tr>
<tr>
<td>Relations:</td>
<td>The three properties, policyAccountIdentifier, cancelRequestIdentifier, and creditRequestIdentifier are equivalent (sameAs) (ref. figs. 8 and 9).</td>
</tr>
<tr>
<td></td>
<td>hasPart, sameAs</td>
</tr>
<tr>
<td>Properties</td>
<td>policyAccountIdentifier, policyStartDate, policyHolder, policyInsurer are properties of a policyAccountRecord (ref: figs. 5, 8 and 9).</td>
</tr>
<tr>
<td>Values</td>
<td>&quot;USD&quot; and &quot;5000.00&quot; are the values of payeeCurrency and premiumAmount respectively for a particular instance of a creditRequestRecord (ref: fig. 2, b.2).</td>
</tr>
<tr>
<td>Rules</td>
<td>The three properties, policyAccountIdentifier, cancelRequestIdentifier, and creditRequestIdentifier are equivalent (sameAs) if they evaluate to the same value, for example, &quot;741032-1071&quot;.</td>
</tr>
</tbody>
</table>
## XLink-XPointer-based Concept Modeling

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>xlink:type</td>
<td>extended</td>
<td>Parent element, which defines a complex link in which multiple links can be combined based on other attributes.</td>
</tr>
<tr>
<td>xlink:type</td>
<td>resource</td>
<td>Child element of extended-Type element, which provides a local resource to participate in the link.</td>
</tr>
<tr>
<td>xlink:type</td>
<td>locator</td>
<td>Child element of extended-Type element, which specifies the location of a remote resource participating in the link.</td>
</tr>
<tr>
<td>xlink:type</td>
<td>arc</td>
<td>Child element of extended-Type element, which define traversal rules between the link’s participating resources.</td>
</tr>
<tr>
<td>xlink:label</td>
<td>&lt;string&gt;</td>
<td>Traversal attribute of extended-, resource-Type elements, which provides a reference (of itself) to arc-Type in creating a traversal arc.</td>
</tr>
<tr>
<td>xlink:from, xlink:to</td>
<td>&lt;string&gt;</td>
<td>Traversal attributes of arc-Type element, which define the source and target resources of the arc link.</td>
</tr>
</tbody>
</table>
### XLink-XPointer-based Concept Modeling

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>xlink:href</td>
<td>URL</td>
<td>The linked URL</td>
</tr>
<tr>
<td>xlink:role</td>
<td>&lt;string&gt;</td>
<td>Semantic attribute of extended-, resource-Type elements, which indicates a property of the resource in a computer readable-form.</td>
</tr>
<tr>
<td>xlink:arcrole</td>
<td>&lt;string&gt;</td>
<td>Semantic attribute of arc-Type element, which coincides with the [RDF] notion of a property, where &quot;the role can be interpreted as stating that &quot;starting-resource HAS arc-role ending-resource.&quot;</td>
</tr>
<tr>
<td>#xpointer</td>
<td></td>
<td>Creates XPointer fragment links with syntax: #xpointer(id(&quot;&quot;))</td>
</tr>
</tbody>
</table>
Example (part 1): Policy Account Schema with XLink-XPointer Syntax

```
1  ...  
2  <xs:element name="policyAccountRecord" minOccurs="0"  
         maxOccurs="unbounded" dfdl:lengthKind="implicit"  
3  ...  
4  <xs:element name="policyAccountIdentifier"  
         type="xs:string" dfdl:lengthKind="explicit" dfdl:length="20" xlink:label=  
5  ...  
```
Example (part 2): Linkbase loads on extraction

```xml
<linkbase xmlns:xlink="http://www.w3.org/1999/xlink"
    xlink:linkbase="http://www.w3.org/1999/xlink/properties/linkbase">
    <link xlink:type="extended" xlink:title="moneyLaunderLinkbase">
        <!-- Linkbase loads on extraction request. -->
        <basesloaded>
            <startsrc xlink:label="filter_spec" xlink:href="/local/filter_spec.xml#params" />
            <linkbase xlink:label="linkbase" xlink:href="/local/linkbase.xml" />
            <load xlink:from="filter_spec" xlink:to="linkbase"
                  actuate="onRequest" />
        </basesloaded>
    </link>
</linkbase>
```
Example (part 3): Linkbase arcs

1. ...

3. <!-- Arcs between linkbase and DFDL-data stores. -->
   <invokeStoreArc xlink:type="arc" xlink:arcrole="linkbase"
               xlink:from="linkbase" xlink:from="PolicyAccountRecord"/>

4. <invokeStoreArc xlink:type="arc" xlink:arcrole="linkbase"
               xlink:from="linkbase" xlink:from="CancelRequestRecord"/>

5. <invokeStoreArc xlink:type="arc" xlink:arcrole="linkbase"
               xlink:from="linkbase" xlink:from="RefundRequestRecord"/>

6. ...

7. ...

Alonza Mumford, Duminda Wijesekera (GMU)
Example (part 4): Linkbase locators

```
1 ...  
2  
3 <!-- Locator elements. -->  
6 <loc xlink:type="locator" xlink:label="PolicyStartDate" xlink:href="http://tmp1.linux.org/policyAccountSchema.dfdl#xpointer(/////policyStartDate[@xs:date=value])"/>
7 ...  
```
Example (part 5): Linkbase relationship between insurance records

```
1 ... 
2
3 <!-- Relationship between policy account, cancel request and refund request identifiers. -->
4 <invokeldArc xlink:type="arc" xlink:arcrole="owl:sameAs" xlink:from="PolicyAccountIdentifier" xlink:to="CancelRequestIdentifier"/>
5 <invokeldArc xlink:type="arc" xlink:arcrole="owl:sameAs" xlink:from="PolicyAccountIdentifier" xlink:to="RefundRequestIdentifier"/>
6 <invokeldArc xlink:type="arc" xlink:arcrole="owl:sameAs" xlink:from="CancelRequestIdentifier" xlink:to="PolicyAccountIdentifier"/>
7
8 ... 
```
Example (part 6): Linkbase relationship between parts

1. ... 
2. <!-- Relationship between policy account record and its parts. -->
3. <invokeParArc xlink:type="arc" xlink:arcrole="owl:hasPart" xlink:from="PolicyAccountRecord" xlink:to= "PolicyAccountIdentifier" />
4. <invokeParArc xlink:type="arc" xlink:arcrole="owl:hasPart" xlink:from="PolicyAccountRecord" xlink:to= "PolicyStartDate" />
5. <invokeParArc xlink:type="arc" xlink:arcrole="owl:hasPart" xlink:from="PolicyAccountRecord" xlink:to= "PolicyHolder" />
6. <invokeParArc xlink:type="arc" xlink:arcrole="owl:hasPart" xlink:from="PolicyAccountRecord" xlink:to= "PolicyInsurer" />
7. ...
— For syntactic analysis, construct a DFDL grammar as a set of recursive definitions. This effort entails creating a context free grammar (CFG) production rule for each code fragment in the DFDL schema language.

— For semantic analysis:
  — Devise the desired higher-order abstract syntax (HOAS) for DFDL. This also entails creating a HOAS code fragment for each CFG production rule.
  — Devise CRS transformation rules to address a multitude of issues such as DFDL scoping semantics. DFDL addresses scoping semantics through transformation rules by exploiting lambda lifting and dropping
What is CRSX?

—Rewriting is theory of stepwise or discrete transformations of objects
—Combinatory Reduction System (CRS) extends the format of first-order rewriting (TRS) with an operator for abstraction over variables
—A legal TRS cannot express and manipulate terms with bound variables,
  e.g. untyped \(\lambda\)-calculus with a beta reduction,
  \[
  (\lambda x. M)N \rightarrow M[x:=N]
  \]
  e.g. a map function that applies a function to all elements of a list,
  
  \[
  \text{map}(f,\text{nil}) \rightarrow \text{nil} \\
  \text{map}(f, \text{cons}(h,t)) \rightarrow \text{cons}(f(h),\text{map}(f,t))
  \]
—Kristoffer Rose contributes CRSX, which implements Klop’s CRS with extensions to support the writing of compilers
—Consider example of the rewriting rule system,

\+[3,1]  
\text{PlusS: } \+[S[#1],#2] \rightarrow S[+[#1,#2]];

—Consider the meaning of the previous syntax [6]:

—Consider the form: \text{name[options]:pattern} \rightarrow \text{contraction},

where \text{name} should be a constructor and the pattern and contraction should be terms

—Capitalized words (e.g. Cons and Nil), numbers, and most other symbols are \text{constructors}, which take an optional ordered or positional \text{parameter} list in immediately following \text{[ ]s}

—Each parameter is itself a \text{term}, and called a \text{subterm}

—Names containing \# are special pattern variables, or \text{meta-variables}, that are used to match arbitrary subterms at the indicated position
Example 2 —CRSX Rewriting Rule System

—Consider example of the rewriting rule system,

Let[E₁, x.E₂]
Let[Copy[#1]]: Let[#₁, x.#₂[x]] → #₂[#₁]

—Consider the meaning of the previous syntax [6]:

—Uncapitalized words (e.g. x and foo) denote variables
—Variables are useful for defining binding constructs. It allows explicit scoping consider,

  e.g. E ::= let x := E₁ in E₂

—Let[#₁, x.#₂[x]], where #₁ is a meta-variable pattern and x.#₂[x] is a meta-application pattern; the constructor must be Let; there are exactly two subterms; the second subterm under the binder can be anything wherein the variable matched by x may occur, and is matched by #<sub>2</sub> where we keep track of all the actual occurrences of the bound variable
Example (part 1): Syntax to CFG Production Rule for DFDL

```xml
1 <xs:annotation>
2    <dfdl:defineEscapeScheme name= "myEscapeScheme">
3        ...
4    </dfdl:escapeScheme escapeCharacter= "/" />
5       ...
6    <dfdl:element representation= "text" escapeSchemeRef= "myEscapeScheme"
7        ...
8 </xs:annotation>
```

Figure: Consider a small fragment of the DFDL Language that allows us to construct a data description of an escape character, which is used to invoke an alternative interpretation on subsequent characters in a character sequence.
## Modeling of Logical Datatypes & Constraints

<table>
<thead>
<tr>
<th>Logical Datatypes &amp; Constraints</th>
<th>Context Free Grammar (CFG) example</th>
<th>Higher-Order Abstract Syntax (HOAS) example</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Structures</strong> (xs:complexType)</td>
<td>2 (&lt;\text{dcl}_\text{xs}&gt; ::= &quot;\text{&quot; XS_COMPONENT}&lt;\text{stmt}&gt;* &quot;\text{&quot; XS_COMPONENT &quot;}&quot;&gt;)&quot;</td>
<td>XsComponent[ComplexType]</td>
</tr>
<tr>
<td></td>
<td>11 XS_COMPONENT ::= &quot;xs:complexType&quot;</td>
<td></td>
</tr>
<tr>
<td><strong>Atomic data values</strong> (xs:simpleType)</td>
<td>11 XS_COMPONENT ::= &quot;xs:simpleType&quot;</td>
<td>XsComponent[SimpleType]</td>
</tr>
<tr>
<td><strong>Ordering</strong> (xs:sequence or xs:choice)</td>
<td>11 XS_COMPONENT ::= &quot;xs:sequence&quot;</td>
<td>XsComponent[Sequence], XsComponent[Choice]</td>
</tr>
<tr>
<td></td>
<td>11 XS_COMPONENT ::= &quot;xs:choice&quot;</td>
<td></td>
</tr>
</tbody>
</table>
| **Occurences** (xs:minOccurs or xs:maxOccurs) | 5 \(<\text{stmt}> ::= XS\_ATTRIBUTE "="
<xs\_attribute\_value>\) | ComponentAttribute[MinOccurs], ComponentAttribute[MaxOccurs] |
|                                  | 16 XS\_ATTRIBUTE ::= "xs:minOccurs" |          |
|                                  |  | 16 XS\_ATTRIBUTE ::= "xs:maxOccurs" |
|                                  | 8 \(<xs\_attribute\_value> ::= <xs\_number>\) |          |
Example (part 2): Syntax to CFG Production Rule for DFDL

```
1 2 <dcl_xs> ::= "<" XS_ADMIN <stmt>* ">" <dcl_xs>* "</" XS_ADMIN ">" | "<" XS_COMPONENT <stmt>* ">" <dcl_xs>* "</" XS_COMPONENT ">
2
3 3 <dcl_dfdl> ::= "<" DFDL_ADMIN <stmt>* ">" <dcl_dfdl> "</" DFDL_ADMIN "">" | "<" DFDL_COMPONENT <stmt>* ">" <dcl_dfdl> "</" DFDL_COMPONENT "">" | ... 
4
5 5 <stmt> ::= "escapeSchemeRef" = <NCName_value> | "name" = <QName> | ... 
6
7 15 DFDL_ADMIN ::= "dfdl:defineEscapeScheme" | ... 
```

**Figure:** The following production rules specify how these data descriptions are formed:
Example (part 3): CFG Production Rule to HOAS for DFDL

```
TERM ::= (  
  Let [ VALUE, TYPE, x::VALUE . TERM ];
  Lam [ VALUE, TYPE, x::VALUE . TERM ];
  Context [ ];
  Element [ KIND, $ List [ ATTRIBUTE], $ List [DFDLPROPERTY],
          $ List [XLP_ATTRIBUTE], TERM ];
  Pair [ TERM, TERM ];
  Nil;
  T;
  T−Attribute
  T−BuildSchema
  T−BuildElement
  XML−Visit [ XLink−XPointer ]
);
```

Figure: Consider our top level terms for the DFDL CRSX system after normalization. Consider our top level term for representing explicit scoping. The terms are written in the form of a higher-order abstract syntax.
Example (part 4): Using rewrite rules to define explicit scoping of attributes

Figure: The following rewrite rule match "XML-Attribute" constructor with parameters and appends the entry to the the local environment:

```plaintext
1 XsComponent-Attribute [Copy[#QName]]
2 :
3 {#Env; #QName: ComponentAttribute[#kind]}
4 XML-Attribute [ #prefix, #QName, #Value, ok.#Continuation [ ok ]]
5 ->
6 {#Env}
7 Let[ #Value, a.{#Env} AddXsAssoc[#prefix, #QName, a, ok.#Continuation [ ok ]]
8 ;
```
Example (part 5): Using rewrite rules to define nested scopes of elements

1
2 <dfdl:discriminator ...> ... </dfdl:discriminator>

Figure: Consider the block scope a DFDL element
Example (part 6): Using rewrite rules to define nested scopes of elements

```
1 DfdIValidation-Start [Copy[#QName]]
2 :
3 {#Env; #QName: DfdIValidation[#kind]}
4 XML-Start[#prefix, #QName, ok prefix.#Continuation[ok, prefix]]
5 -->
6 {#Env; "T-BuildElement": Save-BuildElement[#QName]} #
    Continuation[OK, #QName]
7 ;
8 ...
```

Figure: The following rewrite rules "mark" the start-of block in the environment:
Example (part 7): Using rewrite rules to define nested scopes of elements

1. DfdlValidation–End
2. :
3. {#Env}
4. XML–End[ #prefix, ok.#Continuation[ok]]
5. -->
6. {#Env}
7. T–BuildSchema[OK, QName, DfdlValidation[#kind], #content, ok.#[ok]]

Figure: The following rewrite rules "mark" the end-of block in the environment:
Example (part 8): Using rewrite rules to fold and unfold the DFDL program

```xml
1 1 <dcl_schema> ::= "<" XS_SCHEMA <stmt>* ">" <dcl_xs>*
                          "</" XS_SCHEMA">" $
2
3 3 <dcl_xs> ::= "<" DFDL_COMPONENT <stmt>* ">" <dcl_xs>
                      "</" DFDL_COMPONENT ">"
```

**Figure:** Consider the recursive definition of Schema element:
Example (part 9): Using rewrite rules to fold and unfold the DFDL program

```plaintext
1 Schema-END
2 : {#Env}
3 XML-END[ #prefix, ok.#Continuation[ok]]
4 -->
5 {#Env}
6 T-BuildSchema[OK, #QName, Schema, #content, ok.#[ok]]
7 ... 
8 -->[Copy[#QName]]
9 :
10 {#Env}
11 T-BuildSchema[OK, #QName, Schema, #content, ok.#[ok]]
12 -->
13 Element[#QName, Schema, ${[#Env}Get,"ATTRIBUTE",()], [ ${[#Env}Get,"DFDLPROPERTY",()], ${[#Env}Get,"CONTENT",()] ]
14 ;
```
Future Work

The future work includes:

— Specifying the transformation and evaluation of the DFDL/XLink/HOF specification into parser combinator form
— Investigating the operational semantics of the higher-order function (HOF) and linking abstractions in order to optimize distributed data extraction
— Generating comparative performance metrics
References


O. D. WG, S. M. Hanson, and A. W. Powell, “Data format description language (dfdl) v1. 0 specification.”

S. DeRose, E. Maler, D. Orchard, and N. Walsh, “Xml linking language (xlink) version 1.1, w3c recommendation 06 may 2010,” 2010.


The End