Grammar-based Specification and Parsing for Binary File Formats

William Underwood

Georgia Tech Research Institute

Atlanta, Georgia, USA

7th International Digital Curation Conference 2011

Bristol, UK

December 5-7, 2011

Motivation: Digital Curation Tools

Digital Curators need automated tools for:

- **Identification of file formats**
- **Validation of file formats, with pertinent error messages**
- **Extraction of metadata**
- **Viewing/playing/reading file formats**
- **Conversion of legacy formats to current/standard formats**

Motivation: Digital Curation Tools

- **Identification: DROID/PRONOM; File/Magic**
- **Validation: Harvard's JSTOR/Harvard Object Validation Environment (JHOVE), UCDL (JHOVE2)**
- **Metadata Extractor: National Library of NZ Metadata Extractor; GNU libextractor**
- **Viewers/Players: NASAView, QuickView Plus, IrfanView, XNView, KeyView, Columbus viewer**
- **Conversion: XML Electronic Normalization of Archives (Xena), OpenOffice.org's Format Converter, Alchemy**

Motivation: Digital Curation Tools

- Almost all of these tools, especially those for binary file formats, are manually programmed from file format specifications.
- The tools become obsolete when the hardware/software platform on which the programs run becomes obsolete. The tools either need to be reprogrammed, or become unavailable.
- Need for a sustainable, more cost effective, digital preservation strategy for binary file formats.

Observation: Specification of Binary File Formats

File Layouts

0ffset	Description
0	"RIFF" 4-byte tag
4	size of data chunk starting at offset 8
8	"WEBP" the form-type signature
12	"VP8 " 4-bytes tag, describing the raw video format used
16	size of the raw VP8 image data chunk, starting at offset 20
20	the VP8 image data

C Data Structures

```
typedef struct {
        UWORD w, h;
        WORD  x, y;
        UBYTE nPlanes;
        Masking masking;
        Compression compression;
        UBYTE pad1;
        UWORD transparentColor;
        UBYTE xAspect, yAspect;
        WORD  pageWidth, pageHeight;
        } BitMapHeader;
```

Observation: Textual File Formats are Specified with Grammars

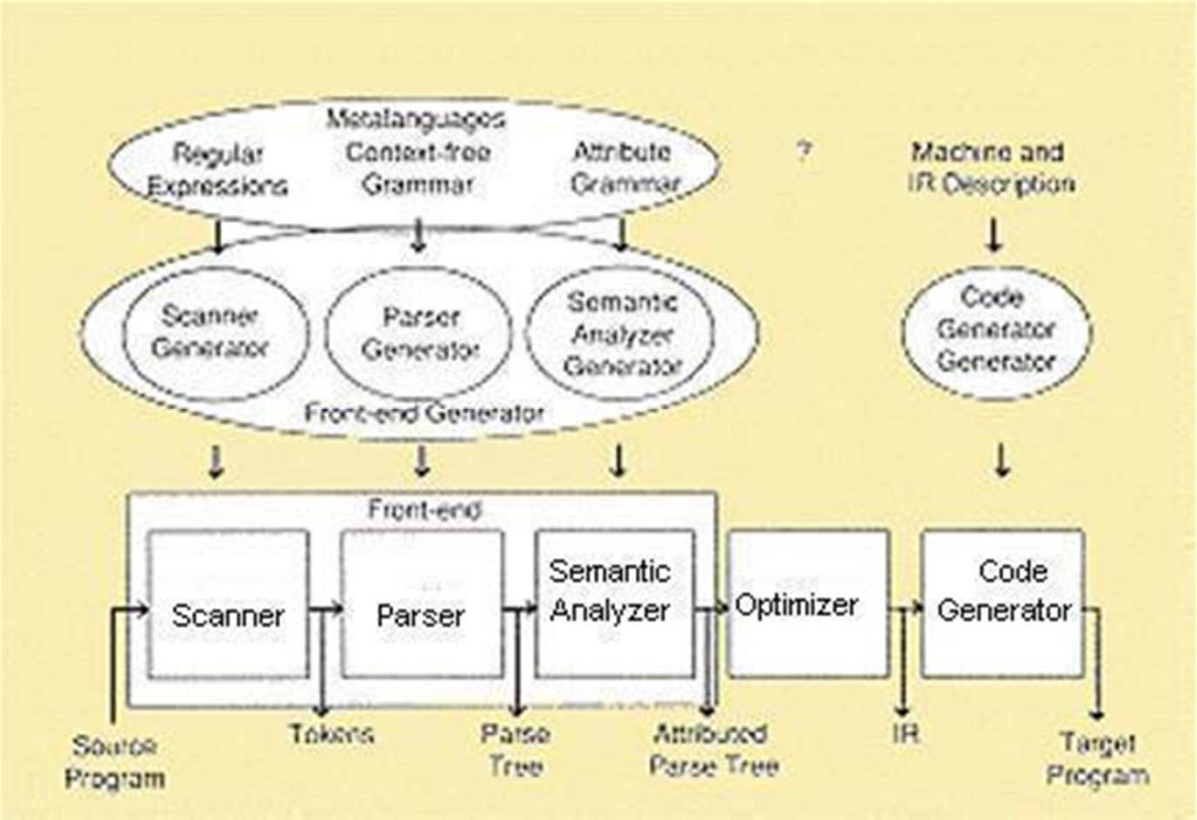
Simple Grammar for LISP Programming Language Syntax

```
expression ::= atom | list
atom ::= number | symbol
number ::= [+-]?['0'-'9']+
symbol ::= ['A'-'Z''a'-'z'].*
list ::= '(' expression* ')'
```

Scalable Vector Graphics Description of a 2D Image

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
"http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg width="100" height="60"
     xmlns="http://www.w3.org/2000/svg">
  <rect fill="green" x="15" y="15" width="70" height="30"/>
  <rect fill="red" x="20" y="20" width="60" height="20"/>
</svg>
```

Observation: Compiler-Compiler Technology



Research Questions

- Is it possible to extend the concept of context-free grammars from textual languages to binary file formats?
- Is it possible to specify binary file formats using these extended context-free binary file grammars?
- Is it possible to create parsers from these grammars for validating file formats?
- Is it possible to develop a parser generator that takes a binary file grammar for a binary file format and generates a parser that can validate the file format?

A *context-free binary array grammar* G is a quintuple <N, D, Σ, S, P> where:

N is a finite set of non-terminal symbols,

D is a set of data types,

Σ is a finite set of binary values of data types D called terminals,

$S \in N$ is the start symbol,

P is a set of production rules of the form $N \rightarrow \{N \cup \Sigma\}^*$

Limitations of Context-free Grammars

- Context-free grammars cannot represent context-sensitive aspects of programming languages.
- They also cannot represent the semantics of programming languages, i.e., the actual computation or an interpretation in assembly language or machine language.
- Binary file formats have context-free and context-sensitive features.

Donald Knuth [1968] proposed an extension of CFGs to address the context-sensitivity and semantics of programming languages.

An *attribute grammar* AG is a triple <G, A, AR>, where:

G is a context-free grammar for the language,

A associates each grammar symbol $X \in (N \cup \Sigma)$ with a set of attributes, and

AR associates each production $R \in P$ with a set of attribute computation rules or conditions.

Kinds (Families) of Binary File Formats Based on File Structure

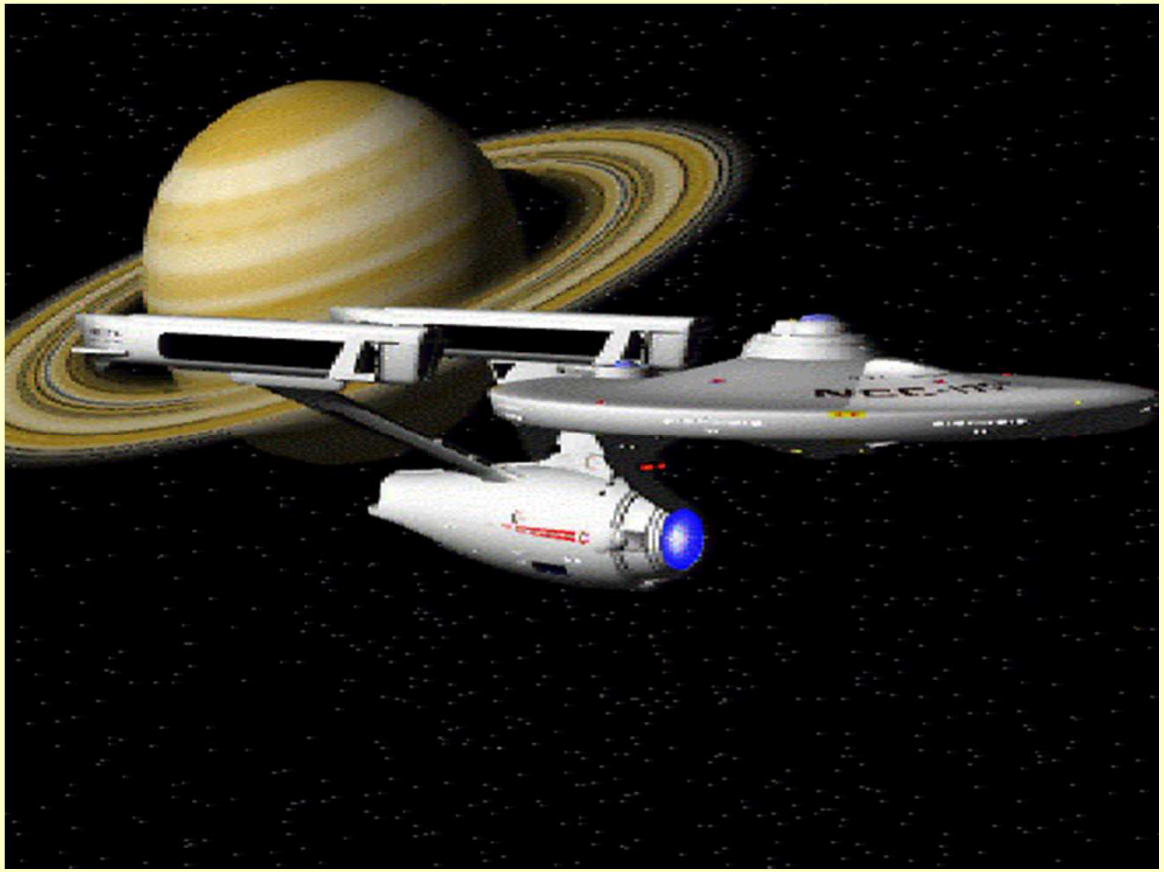
- Chunk-based
- Directory-based
- Executable
- Header-Body
- Others

Chunk-based Binary File Formats

- Interchange File Format (IFF)
- Electronic Arts & Commodore-Amiga
- A chunk consists of a chunk-id, a chunk-size and chunk-data.
- Chunk data can contain image, audio or text data. It can also contain sub-chunks and metadata.
- Sub-chunks can contain sub-sub-chunks

Chunk-based File Format Family

- Apple Audio Interchange File Format (AIFF)
- Resource Interchange File Format (RIFF) – WAV, AVI, ANI, Riff MIDIfile, Device-Independent Bitmap, Webp
- JPEG
- Advanced Systems Format – WMA, WMV
- Portable Network Graphics -- PNG, MNG, JNG
- Binary Interchange File Format (Microsoft Excel)
- 3D Studio – 3ds
- Autodesk Animator Pro – fli, flc, pic
- CorelDRAW Vector Graphics-cdw
- Apple QuickTime – mov, qt
- Structured Data Exchange Format (SDXT), and many more



Starship Enterprise Bitmap in ILBM IFF Binary File Format

Bytes 0-511 of the ILBM Binary File

```
0000:   46 4F 52 4D 00 00 C5 18 - 49 4C 42 4D 42 4D 48 44   FORM....ILBMBMHD
0010:   00 00 00 14 01 60 01 B8 - 00 00 00 00 06 00 01 80   .....`..........
0020:   00 00 0C 07 01 60 01 B8 - 41 4E 4E 4F 00 00 00 48   .....`..ANNO...H
0030:   24 56 45 52 3A 20 57 72 - 69 74 74 65 6E 20 62 79   $VER: Written by
0040:   20 41 53 44 47 27 73 20 - 41 72 74 20 44 65 70 61    ASDG's Art Depa
0050:   72 74 6D 65 6E 74 20 50 - 72 6F 66 65 73 73 69 6F   rtment Professio
0060:   6E 61 6C 20 49 46 46 33 - 2E 30 2E 34 20 28 31 39   nal IFF3.0.4 (19
0070:   2E 30 32 2E 39 35 29 00 - 43 4D 41 50 00 00 00 30   .02.95).CMAP...0
0080:   11 11 11 44 33 22 44 44 - 44 77 55 22 66 66 55 99   ...D3"DDDwU"ffU.
0090:   66 22 88 77 55 77 77 77 - AA 88 55 22 22 EE BB 99   f".wUwww..U""...
00A0:   44 99 99 AA BB AA 77 BB - BB AA DD CC AA EE EE EE   D.....w.........
00B0:   43 41 4D 47 00 00 00 04 - 00 00 08 04 44 50 49 20   CAMG........DPI
00C0:   00 00 00 04 00 96 01 01 - 42 4F 44 59 00 00 C4 50   ........BODY...P
00D0:   D5 00 F8 00 00 08 FD 00 - 00 20 FE 00 00 80 FE 00   ......... ......
00E0:   03 81 00 80 10 FA 00 02 - 01 00 08 F9 00 D5 00 D5   ................
00F0:   00 00 BF F9 FF 00 F2 FD - FF 00 CB FE FF 00 2F FE   ................/.
0100:   FF 03 2E 5F 2F E5 FA FF - 02 FE 5F F2 F9 FF 00 DF   ..._/....._.....
0110:   F9 FF 01 F3 7F FE FF 00 - CD FE FF 00 37 FE FF 03   ............7...
0120:   36 6F 37 E6 FA FF 03 FE - 6F F3 7F FA FF D5 00 FC   6o7.....o.......
0130:   00 00 20 FE 00 00 02 F9 - 00 02 20 00 01 FA 00 00   .. ....... .....
0140:   04 FC 00 00 20 FB 00 02 - 20 00 00 D5 00 D5 00 00   .... ... .......
0150:   BF FD FF 00 CB FE FF 01 - FC BF FA FF 03 CB FF FE   ................
0160:   5F FB FF 01 F9 7F FD FF - 00 CB FB FF 02 CB FF FF   _...............
0170:   00 DF FD FF 00 CD FE FF - 01 FC DF FA FF 03 CD FF   ................
0180:   FE 6F FB FF 01 F9 BF FD - FF 00 CD FB FF 02 CD FF   .o..............
0190:   FF D8 00 02 20 00 00 F8 - 00 00 08 FC 00 00 80 E5   .... ...........
01A0:   00 D5 00 D5 00 00 BF F9 - FF 00 F2 FC FF 00 2F E8   ................/.
01B0:   FF 02 CB FF FF 00 DF F9 - FF 01 F3 7F FD FF 00 37   ...............7
01C0:   E8 FF 02 CD FF FF D5 00 - D5 00 D5 00 D5 00 00 BF   ................
01D0:   D6 FF 00 DF D6 FF D5 00 - 04 00 40 00 00 08 FD 00   ..........@.....
01E0:   06 10 00 00 10 00 00 04 - F5 00 00 10 F8 00 00 04   ................
01F0:   FC 00 D5 00 D5 00 04 BF - 97 FF FF F2 FD FF 07 E5   ................
```

Binary Array Attribute Grammar for Interleaved Bitmap File Format

```
<ILBM> → "FORM" <cksize UINT32> "ILBM"
            <propertyChunk> {propertyChunk.foundBMHD==true}?
            <dataChunk><BODY>
<propertyChunk> → (<BMHD>|<CMAP>|<CAMG>)+
<BMHD> → "BMHD" <cksize UINT32>{cksize.value==20}?
            <BitmapHeader>{foundBMHD=true}
<BitMapheader> →  <width UINT32>
                  <height UINT16>
                  <xposition INT16>
                  <yposition  INT16>
                  <nplanes  BYTE>
                  <masking  BYTE>
                  <compression BYTE>
                  <reserved BYTE>
                  <transparentcolor  UINT16>
                  <xaspect BYTE>
                  <yaspect BYTE>
                  <pagewidth INT16>
                  <pageheight INT16>
<CMAP> -→"CMAP" <cksize UINT32> {cksize.value mod 3==0}? {n=cksize.val/3}
            <color>[n]
<color> → <red BYTE> <green BYTE> <blue BYTE>
<CAMG> → "CAMG" <cksize UINT32> {cksize.value==4}?
            <viewmode type=INT32>
<dataChunk> → <CRNG><CCRT>
<CRNG> → "CRNG" <cksize UINT32> <CRange>
<CCRT> →  "CCRT" <cksize UINT32> <cycleinfo>
<CRange> → <pad1, WORD> {pad1.value==0}?
            <rate, WORD> <active, WORD> <low, UBTYE> <high, UBYTE>
<cycleinfo> → <direction, WORD><start, UBYTE><end, UBYTE>
            <seconds, INT32><microseconds, INT32><pad, WORD>{pad.value==0}?
<BODY> → "BODY" <cksize UINT32> <data BYTE>[cksize.value]
```

Recursive Descent Parsers for Binary Array AttributeGrammars

Top-level Grammar Rule

<ILBM> → “FORM”

<cksize UINT32>

“ILBM”

<propertyChunk>

{propertyChunk.foundBMHD == true}?

<dataChunk>

<BODY>

Top-level Parser Function

```
private void ilbm()
{
      try
      {
            if(nextTerminal(STRING, 4).equals("FORM"))
            {
                  Long ckSize = (Long)nextTerminal(INT32).value();

                  if(nextTerminal(STRING, 4).equals("ILBM"))
                  {
                      if(propertyChunks().foundBMHD())
                      {
                        dataChunk();
                        body();
                      }
                      else
                      {
                        throw new Exception("BMHD not found");
                      }
                  }
                  else
                  {
                        throw new Exception("ILBM not found");
                  }
            }
            else
            {
                  throw new Exception("FORM not found");
            }
      }
      catch (Exception e)
      {
              e.printStackTrace();
      }
}
```

Parse Tree for ILBM File

<ILBM> : Start Symbol of the Grammar

File Signature

- **'FORM' at offset 0**
- **'ILBM' at offset 7**

<ILBM> chunk size – unsigned 32-bit integer with decimal value 50,456

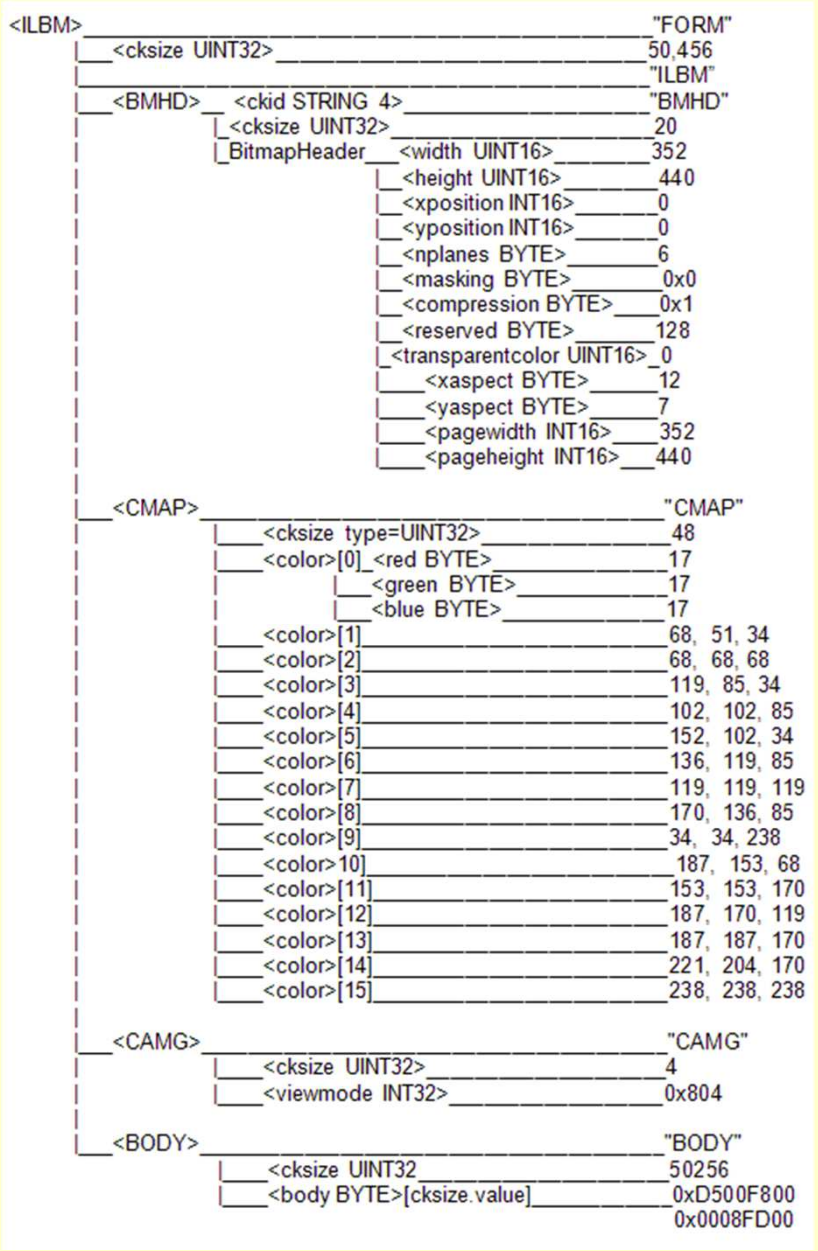
<BMHD> chunk id = 'BMHD

<BMHD> Chunk size = 20

Data chunk BitmapHeader has metadata about the Bitmap

Color pallet is stored in the <CMAP> chunk

Bitmap is stored in the <BODY> chunk.



Generating Parsers for Binary Formats

- Goal is a parser generator for binary file formats.
- ANTLR (Another Tool for language Recognition) is a parser generator
  - Input: Attribute LL(k) grammar for a string language
  - Output: Source code (e.g. Java for a recognizer of that language)
  - Wrote functions for each data type that converts character (byte) tokens to binary data types.
- Uses ANTLR to demonstrate the capability of generating parsers from file format grammars

Related Research: File Description Languages

- ASN.1 (Abstract Syntax Notation One)
- EAST/DEDSL (Extended ADA Subset / Data Entity Dictionary Specification Language)
- DATASCRIPT
- PADS/C (Processing Ad hoc Data Sources)
- DFDL (Data Format Description Language)

File Description Languages vis-a-vis Attribute Array Grammars

- Each of the data description languages described can be used to define data types and file structures of binary files.
- The binary file format grammar described in this paper most closely resembles ASN.1 and DFDL.
- The binary file grammar is the only file description language based on formal grammars that is used for creating recognizers for file formats.

Results

- It is possible to extend context-free grammars for textual languages to the specification of some chunk-based and directory-based binary file formats.
- It is possible to create parsers from these grammars for validating thse binary file formats.
- ANTLR, a parser generator for LL(k) grammars, has been successfully used to generate parsers for two chunk-based file formats and two directory-based file formats.

Further Research Questions

- Can we construct binary array attribute grammars for executable and header-body binary format families?
- Can semantics be incorporated into the grammars for binary file formats to enable the generation of viewers/players and file format converters?
- Can we construct a parser generator (a Compiler-Compiler) for binary array attribute grammars?

Further Research Questions

- Can grammar-based specifications for binary file formats be as intelligible as those that are not grammar based?
- Explicitly, how do binary array attribute grammars compare with other file description languages?
- How do we establish the correctness of the programs for validation, metadata extraction, rendering and converting file formats that are generated from attribute grammar specifications and a compiler/compiler?

Acknowledgement

This research was sponsored by the Army Research Laboratory (ARL) and the Applied Research Division of the National Archives and Records Administration (NARA) and was accomplished under Cooperative Agreement Number W911NF-10-2-0030.

The author is grateful to the reviewers who through their questions and comments improved the paper and the presentation.