RNA

Recursive Networked Data Structures
Low Level Object Support System

Gordon Cichon

Published by Gordon Cichon

1 Overview

RNA is a small and beautiful recursive data structure system. It is designed to share the advandtages of C and LISP data type system:

Some of its functionality is provided today by XML. RNA is far less bloated, however.

It has the speed and expressiveness of the static type system of languages like C that allows to express even comples data structres in human understandable manner. But it does not suffer from its rigidness that forbids specifying new types at run time nor does it cause tedious compile cycles.

On the other side it has the flexibilty of a dynamic type system like LISP that allows to express recursive data structures of complex problems. Its representation is almost as simple as that of LISP while being much more readable and nearly as storage efficient as C.

Additionally, it presents a solution to the data persistence problem by offering a means to store and retrieve complex the full featured data structures from and to files.

2 Design

The RNA structures are defined as items in Memory and ASCII files respectively. A RNA item consists of one machine word can be of one of seven possible flavors:

'`b`' Boolean. The machine word is interpreted as the boolean value false if it is zero, and true otherwise. In file form it is written as either 'true' or 'false'.

'`i`' Integer. The machine word is interpreted as signed integer. In file form it can be expressed in decimal, octal or hexadecimal. (F.e.: 0 -1 2 0xa5 074)

'`f`' Floating point. The machine word is interpreted as IEEE floating point number. (F.e.: 0.0 -1.0 27.3e-15)

'`p`' Generic Pointer. This flavor is restricted to use in memory. It can be used to store handles f.e. from the operating system or point to data structures that are not handled by the RNA system.

'`s`' String. The machine word contains a pointer to a zero terminated string. In a file it is represented as quoted string with double quotation marks. The usual C escape sequences are supported. (F.e.: "Hello, world!\n")

'`A`' The machine word contains a pointer to a Record structure. A Record contains a set of RNA items of a fixed length and variable types. It resembles a C language 'struct' in some way. (see below)

'`B`' The machine word contains a pointer to an Array structure. It is used to store a set of RNA items of variable length and a fixed type. It resembles a C language 'array'. (see below)

In the file representation, each RNA can be given a label by prefixing it with a name and a colon. (F.e.: "my_int: 1") The items named by these labels can be recalled later by writing its name. (F.e.: "my_int"). This is especially useful for recalling addresses of Record and Array structures to build up networked pointer structures.

2.1 Record

A record has a certain type. A pointer to a TypeDescriptor for this type is stored at the first word in the Record's memory image. TypeDescriptors are usually referenced by label when reading files. TypeDescriptors can also be generated by genclass. A TypeDescriptor has the following contents:

'`name`' the name of the type

'`length`' number of fixed items stored in each instance.

'`format`' a format string describing the types of the stored items. Every item is described by a single letter that is exactly the RNA flavor.

'`names`' optional names for the stored items. These can be used to access structure members by name.

'types' optional types for stored items. This entry is an array of pointers to Type-Descriptors. If the format string specifies an Record or Array type for a specific item, a more specific type constraint can be given here. These types are used by genclass.

F.e:

```
(Record "Address-Mail" 3 "sss" ["name" "nickname" "email"])
```

denotes a type named "Address-Mail" which consists of 3 components named "name" "nickname" and "email" repectively and that are all three strings.

```
(Record "AddressBook" 1 "B" ["contents"] types=[Address-Mail])
```

denotes a type named "AddressBook" which has a single component named "contents" that is a Array of Records. The element type of the array is to be "Address-Mail".

The second machine word of a Record stores a pointer to a linked list of attribute values. These list is explained later.

Starting with the third word, there follows a number of RNA words. The number is specified by the type's length, and the interpretation of the contents is also given by the type.

In the file representation, an Record is given by '(' typename {[label'=']item} ')'. F.e:

```
(Address-Mail "Gordon Cichon" "gordon" "Gordon@Cichon.de")
(Address-Mail email="Gordon@Cichon.de" name="Gordon Cichon" "gordon")
```

denote the same object.

In C, items of an Record RNA can be accessed by the '()' operator called either with the position integer or the item name. F.e:

```
TypeDescriptor * td = AddressMail::TypeInfo ();
Rna r = Record::Create (Memory::Main (), td,
                         "Gordon Cichon" "gordon"
                        "Gordon@Cichon.de");
char * s = r (1).readStr(); // "gordon"
char * t = r ("email").readStr(); // "Gordon@Cichon.de"
```

2.2 Array

While Record is designed to store a fixed number of items of a variable type, Array is designed to store a variable number of items of a fixed type.

The memory image contains first the number of items, a format character to designate the type of the stored items, and then the items itself.

In file form, the items are surrounded by brackets ('[' ']').

F.e.:

```
my_array: [1 2 3 4 5 6 7]
```

C Example:

```
RNA r = Array::Create (Memory::Main(), 2, STR, "foo", "bar");
char * s = r [0].readStr(), // "foo"
char * t = r [1].readStr(); // "bar"
```

2.3 Record Attributes

Additionally Record allows to store a named list of RNAs in an linked list called side chain.

File Example:

```
(Address-Mail name="Gordon Cichon" "gordon"
        "Gordon@Cichon.de"
        age=26 comment="I met him yesterday.")
```

C Example:

```
Rna r = Record::Create (Memory::Main(), td,
                        "Gordon Cichon", "gordon", "Gordon@Cichon.de");
r ["comment"] = "I met him yesterday.";
r ["age"] = 26;
char * s r ["comment"].readStr(); // "I met him yesterday.";
int i r ["age"].readInt(); // 26
```

3 Memory Management

The memory management functionality is encapsulated into the class of Memory.

A Space is an entity that allows memory to be allocated. Therefor a function Memory::alloc' is provided.

Memory in RNA is allways allocated in stacks. This allows to free a set of objects at once that were allocated after a certain time. This is organized with the checkpoint functionality.

Memory is currently implemented using GNU obstacks.

F.e:

```
Memory * a;
Memory * b;
Memory * c;

// now allocations are made from a
void * x = RNA:Create (a, ...);
// now allocations are made from b
void * y = RNA:Create (b, ...);
Memory::Checkpoint checkpoint = b.checkpoint ();
y = RNA:Create (b, ...);
y = RNA:Create (b, ...);
// now allocations are made from c
y = RNA:Create (c, ...);
// now allocations are made from b
y = RNA:Create (b, ...);
// now allocations are made from a
void * z = RNA:Create (a, ...);

// cleanup
b.free (checkpoint);
```

4 Reading and Writing Files

Operating system files are wrapped in the classes Stream and File that provide a means for error handling by line counting (in the latter case).

A context of label names and their memory bindings is maintained in a 'Context'. These contexts can be hierarchically stacked in order to provide namespace functionality.

A Context needs two extra Memories for maintaining its tables during parsing.

Example

```
// read a file (produces nice error messages
// with filename and line number)
File * file = new File ("filename.txt", "r");
Rna rna = Rna::Read (file, ARRAY); // Read a vector of Records

// write a stream (streams are not so nice as files
// but pipes do not have line numbers)
Stream stream (stdout);
Rna::Write (stream, ARRAY); // Write a vector of Records
```

5 Outlook

- Implement a graphical editor for these structures.
- Lets implement garbage collection. (lets do a new kind of space).
- Implement parsers for other languages into the given structure. F.e: HTML, LISP, VRML
- Implement a small subset of LISP as a little scripting language with features like AWK or perl but that are able to process hierarchical data structures adequately.

Philosophy

In the beginning, there was only RNA. This RNA was auto-catalythic, i. e. it was able to perform ceratin chemical reactions onto itself. A certain strand of RNA is able to cut a piece out of itself.

RNA gained to neat property to build a large tangle that is called ribosom today. This tangle has a whole through which another strand of RNA can run through in a way that other small RNA pieces can dock at groups of three RNA bases.

In a way that nobody can understand today, proteine got bound at these little pieces that docked at the strand running through the tangle - different types of proteine molecules for different sequences of bases on the RNA. Additionally these proteins got bound to each other forming a proteine strand resembling the sequence of bases on the RNA. These Proteins had the capability to catalyze other cemical reactions.

Then RNA learned to transcribe itself into DNA for persistent storage of its information.

Today, all genetic information is securely stored on DNA and nearly all catalytic activities are carries out by proteines – the role of RNA is nearly forgotten.

But remember: From tiny kernels, mighty systems grow!

Concept Index

Data Type Index

A

M

R

Function Index

M

R

Short Contents

Table of Contents