Proposal for a GraphBLAS C API

(*Working document from the* GraphBLAS *Signatures Subgroup*)

Aydin Buluc, Timothy Mattson, Scott McMillan, José Moreira, Carl Yang

GraphBLAS Signatures Subgroup

Generated on 2016/02/09 at 16:06:09 EDT

1 Introduction

This is a proposal for the C programming language binding of the GraphBLAS interface. We adopt C99 as the standard definition of the C programming language. Furthermore, we assume that the language has been extended with the `_Generic` construct from C11. After establishing some basic concepts, we proceed by describing the objects in GraphBLAS: spaces, vectors, matrices and descriptors. We then describe the various methods that operate on those objects. The appendix includes examples of GraphBLAS in C.

2 Basic concepts

2.1 Domains

GraphBLAS defines two kinds of collections: matrices and vectors. For any given collection, the elements of the collection belong to a *domain*, which is the set of valid values for the element. In GraphBLAS, domains correspond to the valid values for types from the host language (in our case, the C programming language). For any variable or object V in GraphBLAS we denote as $D(V)$ the domain of V. That is, the set of possible values that elements of V can take. The predefined types, and corresponding domains, of GraphBLAS are shown in Table **??**. The Boolean type is defined in `stdbool.h`, the integral types are defined in `stdint.h`, and the floating-point types are native to the language. GraphBLAS also supports user defined types. In that case, the domain is the set of valid values for a variable of that type.

2.2 Operations

In GraphBLAS, a *binary operation* is a function that maps two input values to one output value. A *unary operation* is a function that maps one input value to one output value. The value of the

Table 1: Predefined types and corresponding domains for GraphBLAS in C.

type	domain	GraphBLAS identifier
`bool`	{`false`, `true`}	GrB_BOOL
`int8_t`	$\mathbb{Z} \cap [-2^7, 2^7)$	GrB_INT8
`uint8_t`	$\mathbb{Z} \cap [0, 2^8)$	GrB_UINT8
`int16_t`	$\mathbb{Z} \cap [-2^{15}, 2^{15})$	GrB_INT16
`uint16_t`	$\mathbb{Z} \cap [0, 2^{16})$	GrB_UINT16
`int32_t`	$\mathbb{Z} \cap [-2^{31}, 2^{31})$	GrB_INT32
`uint32_t`	$\mathbb{Z} \cap [0, 2^{32})$	GrB_UINT32
`int64_t`	$\mathbb{Z} \cap [-2^{63}, 2^{63})$	GrB_INT64
`uint64_t`	$\mathbb{Z} \cap [0, 2^{64})$	GrB_UINT64
`float`	IEEE 754 binary32	GrB_FLOAT
`double`	IEEE 754 binary64	GrB_DOUBLE

output is uniquely determined by the value of the input(s). Binary functions are defined over two input domains and produce an output from a (possibly different) third domain. Unary functions are specified over one input domain and produce an output from a (possibly different) second domain. The predefined operations of GraphBLAS are listed in Table **??**.

3 Objects

3.1 Spaces

A GraphBLAS *space* $S = \langle D_1, D_2, D_3, \oplus, \otimes, \mathbf{0}[, \mathbf{1}]\rangle$ is defined by three domains D_1, D_2 and D_3, an additive operation $\oplus : D_3 \times D_3 \to D_3$, with corresponding identity $\mathbf{0} \in D_3$, and a multiplicative operation $\otimes : D_1 \times D_2 \to D_3$, with optional corresponding identity $\mathbf{1} \in D_1 \cap D_2$. If $\mathbf{1}$ is specified, then $D_1 \subseteq D_3$ and $D_2 \subseteq D_3$ must be satisfied. For a given GraphBLAS space $S = \langle D_1, D_2, D_3, \oplus, \otimes, \mathbf{0}, \mathbf{1}\rangle$ we define $D_1(S) = D_1$, $D_2(S) = D_2$, $D_3(S) = D_3$, $\oplus(S) = \oplus$, $\otimes(S) = \otimes$, $\mathbf{0}(S) = \mathbf{0}$ and $\mathbf{1}(S) = \mathbf{1}$. We note that, in the special case of $D_1 = D_2 = D_3$ and $\mathbf{1}$ defined, a GraphBLAS space reduces to the conventional *semiring* algebraic structure.

3.2 Vectors

A vector $\mathbf{v} = \langle D, N, \{(i, \mathbf{v}(i))\}\rangle$ is defined by a domain D, a size $N > 0$ and a set of tuples $(i, \mathbf{v}(i))$ where $0 \leq i < N$ and $\mathbf{v}(i) \in D$. A particular value of i can only appear at most once in $\mathbf{v}$. We define $n(\mathbf{v}) = N$ and $L(\mathbf{v}) = \{(i, \mathbf{v}(i))\}$. We also define the set $\mathbf{i}(\mathbf{v}) = \{i : (i, \mathbf{v}(i)) \in \mathbf{v}\}$, and $D(\mathbf{v}) = D$.

Table 2: Predefined operations for GraphBLAS in C. (Just a sample.)

kind	operation	domains	description
	GrB_NOP		no operation
unary	GrB_LNOT	bool → bool	logical inverse
binary	GrB_LAND	bool × bool → bool	logical AND
binary	GrB_LOR	bool × bool → bool	logical OR
binary	GrB_LXOR	bool × bool → bool	logical XOR
binary	GrB_PLUS	int64_t × int64_t → int64_t	signed integer addition
binary	GrB_PLUS	uint64_t × uint64_t → uint64_t	unsigned integer addition
binary	GrB_PLUS	double × double → double	floating-point addition
binary	GrB_MINUS	int64_t × int64_t → int64_t	signed integer subtraction
binary	GrB_MINUS	uint64_t × uint64_t → uint64_t	unsigned integer subtraction
binary	GrB_MINUS	double × double → double	floating-point subtraction
binary	GrB_TIMES	int64_t × int64_t → int64_t	signed integer multiplication
binary	GrB_TIMES	uint64_t × uint64_t → uint64_t	unsigned integer multiplication
binary	GrB_TIMES	double × double → double	floating-point multiplication
binary	GrB_DIV	int64_t × int64_t → int64_t	signed integer division
binary	GrB_DIV	uint64_t × uint64_t → uint64_t	unsigned integer division
binary	GrB_DIV	double × double → double	floating-point division
binary	GrB_EQ	int64_t × int64_t → bool	signed integer equal
binary	GrB_EQ	uint64_t × uint64_t → bool	unsigned integer equal
binary	GrB_EQ	double × double → bool	floating-point equal
binary	GrB_NE	int64_t × int64_t → bool	signed integer not equal
binary	GrB_NE	uint64_t × uint64_t → bool	unsigned integer not equal
binary	GrB_NE	double × double → bool	floating-point not equal
binary	GrB_GT	int64_t × int64_t → bool	signed integer greater than
binary	GrB_GT	uint64_t × uint64_t → bool	unsigned integer greater than
binary	GrB_GT	double × double → bool	floating-point greater than
binary	GrB_LT	int64_t × int64_t → bool	signed integer less than
binary	GrB_LT	uint64_t × uint64_t → bool	unsigned integer less than
binary	GrB_LT	double × double → bool	floating-point less than
binary	GrB_GE	int64_t × int64_t → bool	signed integer greater than or equal
binary	GrB_GE	uint64_t × uint64_t → bool	unsigned integer greater than or equal
binary	GrB_GE	double × double → bool	floating-point greater than or equal
binary	GrB_LE	int64_t × int64_t → bool	signed integer less than or equal
binary	GrB_LE	uint64_t × uint64_t → bool	unsigned integer less than or equal
binary	GrB_LE	double × double → bool	floating-point less than or equal

3.3 Matrices

A matrix $\mathbf{A} = \langle D, M, N, \{(i, j, \mathbf{A}(i, j))\}\rangle$ is defined by a domain D. its number of rows $M > 0$, its number of columns $N > 0$ and a set of tuples $(i, j, \mathbf{A}(i, j))$ where $0 \leq i < M$, $0 \leq j < N$, and $\mathbf{A}(i, j)i \in D$. A particular pair of values i, j can only appear at most once in $\mathbf{A}$. We define $n(\mathbf{A}) = N$, $m(\mathbf{A}) = M$ and $L(\mathbf{A}) = \{(i, j, \mathbf{A}(i, j))\}$. We also define the sets $\mathbf{i}(\mathbf{A}) = \{i : \exists(i, j, \mathbf{A}(i, j)) \in \mathbf{A}\}$ and $\mathbf{j}(\mathbf{A}) = \{j : \exists(i, j, \mathbf{A}(i, j)) \in \mathbf{A}\}$. (These are the sets of nonempty rows and columns of $\mathbf{A}$, respectively.) Finally, $D(\mathbf{A}) = D$.

If $\mathbf{A}$ is a matrix and $0 \leq j < N$, then $\mathbf{A}(:, j) = \langle D, M, \{(i, \mathbf{A}(i, j)) : (i, j, \mathbf{A}(i, j)) \in L(\mathbf{A})\}\rangle$ is a vector called the j-th *column* of $\mathbf{A}$. Correspondingly, if $\mathbf{A}$ is a matrix and $0 \leq i < M$, then $\mathbf{A}(i, :) = \langle D, N, \{(j, \mathbf{A}(i, j)) : (i, j, \mathbf{A}(i, j)) \in L(\mathbf{A})\}\rangle$ is a vector called the i-th *row* of $\mathbf{A}$.

3.4 Descriptors

Descriptors are used as input parameters in various GraphBLAS methods to provide more details of the operation to be performed by those methods. In particular, descriptors specify how the other input parameters should be processed before the main operation of a method is performed. For example, a descriptor may specify that a particular input matrix needs to be transposed or that a mask needs to be inverted before using it in the operation. Some methods may also allow additional processing of the result before generating the final output parameter.

For the purpose of constructing descriptors, the parameters of a method are identified by specific names. The output parameter (typically the first parameter in a GraphBLAS method) is OUTP. The input parameters are named ARG0, ARG1, ARG2 and so on from the first input parameter to the last. The mask (typically the next to last parameter in a method) is named MASK. Finally, the descriptor (typically the last parameter in a method) is not named, since GraphBLAS does not support modifications of descriptors themselves.

4 Methods

4.1 Vector-matrix multiply (vxm)

Multiplies a vector by a matrix within a space. The result is a vector.

C99 Syntax

```
#include "GraphBLAS.h"
GrB_info GrB_vxm(GrB_Vector *u, const GrB_Space s, const GrB_Vector v,
                 const GrB_Matrix A, const GrB_Vector m, const GrB_Descriptor d)
```

Input Parameters

- s (ARG0) Space used in the vector-matrix multiply.
- v (ARG1) Vector to be multiplied.
- A (ARG2) Matrix to be multiplied.
- m (MASK) Operation mask. The mask specifies which elements of the result vector are to be computed. If no mask is necessary (i.e., compute all elements of result vector), GrB_NULL can be used.
- d Operation descriptor. The descriptor is used to specify details of the operation, such as transpose the matrix or not, invert the mask or not (see below). If a *default* descriptor is desired, GrB_NULL can be used.

Output Parameter

- u (OUTP) Address of result vector.

Return Value

GrB_SUCCESS	operation completed successfully
GrB_PANIC	unknown internal error
GrB_OUTOFMEM	not enough memory available for operation
GrB_MISMATCH	mismatch among vectors, matrix and/or space

Description

Vectors $\mathbf{v}, \mathbf{m}$ and matrix $\mathbf{A}$ are computed from input parameters v, m and A, respectively, as specified by descriptor d. (See below for the properties of a descriptor. In the simplest form, these are just copies, but additional preprocessing, including casting, can be specified.) $D(\mathbf{v}) \equiv D_1(\mathsf{s})$ and $D(\mathbf{A}) \equiv D_2(\mathsf{s})$. If m is GrB_NULL then $\mathbf{m}$ is a Boolean vector of size $n(\mathbf{A})$ and with all elements set to true.

A consistency check is performed to verify that $n(\mathbf{v}) = m(\mathbf{A})$ and $n(\mathbf{m}) = n(\mathbf{A})$. If a consistency check fails, the operation is aborted and the method returns GrB_MISMATCH.

A new vector $\mathbf{u} = \langle D_3(\mathsf{s}), n(\mathbf{A}), L(\mathbf{u}) = \{(i, \mathbf{u}(i)) : \mathbf{m}(i) = \mathsf{true}\}\rangle$ is created. The value of each of its elements is computed by $\mathbf{u}(i) = \bigoplus_{j \in \mathbf{i}(\mathbf{v}) \cap \mathbf{i}(\mathbf{A}(:,i))} (\mathbf{v}(j) \otimes \mathbf{A}(j,i))$, where $\oplus$ and $\otimes$ are the additive and multiplicative operations of space s, respectively. If $\mathbf{i}(\mathbf{v}) \cap \mathbf{i}(\mathbf{A}(:,i)) = \emptyset$ then the pair $(i, \mathbf{u}(i))$ is not included in $L(\mathbf{u})$.

Finally, output parameter u is computed from vector $\mathbf{u}$ as specified by descriptor d. (Again, in the simplest case this is just a copy, but additional postprocessing, including casting and accumulation of result values, can be specified.) A consistency check is performed to verify that $n(\mathsf{u}) = n(\mathbf{u})$. If the consistency check fails, the operation is aborted and the method return GrB_MISMATCH.

4.2 Create new descriptor (Descriptor_new)

Creates a new (empty) descriptor.

C99 Syntax

```
#include "GraphBLAS.h"
GrB_info GrB_Descriptor_new(GrB_Descriptor *d)
```

Output Parameters

d Identifier of new descriptor created.

Return Value

GrB_SUCCESS	operation completed successfully
GrB_PANIC	unknown internal error
GrB_OUTOFMEM	not enough memory available for operation
GrB_MISMATCH	mismatch between field and new value

Description

Returns in d the identifier of a newly created empty descriptor. A newly created descriptor can be populated with calls to Descriptor_add.

4.3 Add content to descriptor (Descriptor_add)

Adds additional content (details of an operation) to an existing descriptor.

C99 Syntax

```
#include "GraphBLAS.h"
GrB_info GrB_Descriptor_add(GrB_Descriptor d,GrB_Field f,GrB_Value v)
```

Input Parameters

d The descriptor being modified by this method.

f The field of the descriptor being modified.

v New value for the field being modified.

Return Value

GrB_SUCCESS	operation completed successfully
GrB_PANIC	unknown internal error
GrB_OUTOFMEM	not enough memory available for operation
GrB_MISMATCH	mismatch between field and new value

Description

The fields of a descriptor include: GrB_OUTP for the output parameter (result) of a method; GrB_MASK for the mask argument to a method; GrB_ARG0 through GrB_ARG9 for the input parameters (from first to last) of a method.

Valid values for a field of a descriptor are as follows:

GrB_NOP	no operation to be performed for the corresponding parameter
GrB_LNOT	compute the logical inverse of the corresponding parameter
GrB_TRAN	compute the transpose of the corresponding parameter (for matrices)
GrB_ACC	accumulate result of operation to current values in destination (for output parameter)
GrB_CAST	cast values from input parameters to input domains of operation or from output domain of operation to output parameter. (Otherwise, incorrect domain will cause a run-time error.)

It is possible to specify a combination of values for a field. For example, if a matrix is to be both transposed and logically inverted (element by element), one would use the field value GrB_TRAN | GrB_LNOT.

4.4 Create new space (Space_new)

Creates a new space with specified domain, operations and identities.

C99 Syntax

```
#include "GraphBLAS.h"
GrB_info GrB_Space_new(GrB_Space *s,GrB_type t1, GrB_type t2, GrB_type t3,
                       GrB_operation a,GrB_operation m, t3 z[, t3 o]))
```

Input Parameters

t1 The type defining the first domain of the space being created. Should be one of the predefined GraphBLAS types in Table **??**, or a user created type.

t2 The type defining the second domain of the space being created. Should be one of the predefined GraphBLAS types in Table **??**, or a user created type.

t3 The type defining the third domain of the space being created. Should be one of the predefined GraphBLAS types in Table **??**, or a user created type.

a The additive operation of the space.

m The multiplicative operation of the space.

z The additive identity of the space.

o The multiplicative identify of the space.

Output Parameter

s Identifier of the newly created space.

Return Value

GrB_SUCCESS	operation completed successfully
GrB_PANIC	unknown internal error
GrB_OUTOFMEM	not enough memory available for operation

Description

Creates a new space $S = \langle D(\mathsf{t}), \mathsf{a}, \mathsf{m}, \mathsf{z}, \mathsf{o} \rangle$ and returns its identifier in s.

4.5 Create new vector (Vector_new)

Creates a new vector with specified domain and size.

C99 Syntax

```
#include "GraphBLAS.h"
GrB_info GrB_Vector_new(GrB_Vector *v,GrB_type t,GrB_index n)
```

Input Parameters

t The type defining the domain of the vector being created. Should be one of the predefined GraphBLAS types in Table **??**, or a user created type.

n The size of the vector being created.

Output Parameter

v Identifier of the newly created vector.

Return Value

GrB_SUCCESS	operation completed successfully
GrB_PANIC	unknown internal error
GrB_OUTOFMEM	not enough memory available for operation

Description

Creates a new vector $\mathbf{v}$ of domain $D(\mathsf{t})$, size n, and empty $L(\mathbf{v})$. It return in v this vector $\mathbf{v}$.

4.6 Number of rows in a matrix (Matrix_nrows)

Retrieve the number of rows in a matrix.

C99 Syntax

```
#include "GraphBLAS.h"
GrB_info GrB_Matrix_nrows(GrB_index *m,GrB_Matrix A)
```

Input Parameters

A Matrix being queried.

Output Parameters

m The number of rows in the matrix.

Return Value

GrB_SUCCESS	operation completed successfully
GrB_PANIC	unknown internal error
GrB_NOMATRIX	matrix does not exist

Description

Return in m the number of rows (parameter M in Section **??**) in matrix A.

4.7 Assign values to the elements of an object (assign)

4.7.1 Flat variant

Set all the elements of a vector to a given value.

```
#include "GraphBLAS.h"
GrB_info GrB_assign(GrB_Vector *v,scalar s[,GrB_Vector m])
```

Input Parameters

- v Vector to be assigned.
- s Scalar value for the elements.
- m (Optional) mask for assignment.

Return Value

GrB_SUCCESS	operation completed successfully
GrB_PANIC	unknown internal error
GrB_NOVECTOR	vector does not exist
GrB_MISMATCH	mismatch between vector domain and scalar type

4.7.2 Indexed variant

Set some of the elements of a vector to a given value.

C99 Syntax

```
#include "GraphBLAS.h"
GrB_info GrB_assign(GrB_Vector *v,scalar s,GrB_index i)
```

Input Parameters

- v Vector to be assigned.
- s Scalar value for the elements.
- i Index of element to be assigned

Return Value

GrB_SUCCESS	operation completed successfully
GrB_PANIC	unknown internal error
GrB_NOVECTOR	vector does not exist
GrB_MISMATCH	mismatch between vector domain and scalar type

4.8 Perform of a reduction across the elements of an object (reduce)

Computes the reduction of the values of the elements of a vector or matrix.

4.8.1 Vector variant

C99 Syntax

```
#include "GraphBLAS.h"
GrB_info GrB_reduce(scalar *s,GrB_Vector v,GrB_operations f)
```

Input Parameters

v Vector to be reduced.

f Operation to be applied in the reduction.

Output Parameters

s Initial and final value of the reduction.

Return Value

GrB_SUCCESS	operation completed successfully
GrB_PANIC	unknown internal error
GrB_NOVECTOR	vector does not exist
GrB_MISMATCH	mismatch between vector domain and scalar type

4.9 Destroy object (free)

Destroys a previously created GraphBLAS object.

C99 Syntax

```
#include "GraphBLAS.h"
GrB_info GrB_free(GrB_Object o)
```

Input Parameter

`o` GraphBLAS object to be destroyed. Can be a matrix, vector or descriptor.

Return Value

GrB_SUCCESS	operation completed successfully
GrB_PANIC	unknown internal error
GrB_NOOBJECT	object does not exist

A Example breadth first search with GraphBLAS

```
#include <stdlib.h>
#include <stdio.h>
#include <stdint.h>
#include <stdbool.h>
#include "GraphBLAS.h"

GrB_info BFS(GrB_Vector *v, GrB_Matrix A, GrB_index s)
/*
 * Given a boolean n x n adjacency matrix A and a source vertex s, performs a BFS traversal
 * of the graph and sets v[i] to the level in which vertex i is visited (v[s] == 1).
 * If i is not reacheable from s, then v[i] = 0. (Vector v should be empty on input.)
 */
{
  GrB_index n;
  GrB_Matrix_nrows(&n,A);                        // n = # of rows of A

  GrB_Vector_new(v,GrB_INT32,n);                 // Vector<int32_t> v(n)
  GrB_assign(v,0);                               // v = 0

  GrB_Vector q;                                  // vertices visited in each level
  GrB_Vector_new(&q,GrB_BOOL,n);                 // Vector<bool> q(n)
  GrB_assign(&q,false);
  GrB_assign(&q,true,s);                         // q[s] = true, false everywhere else

  GrB_Space Boolean;                             // Boolean space <bool,bool,bool,||,&&,false,true>
  GrB_Space_new(&Boolean,GrB_BOOL,GrB_BOOL,GrB_BOOL,GrB_LOR,GrB_LAND,false,true);

  GrB_Descriptor desc;                           // Descriptor for vxm
  GrB_Descriptor_new(&desc);
  GrB_Descriptor_add(desc,GrB_ARG1,GrB_NOP);     // no operation on the vector
  GrB_Descriptor_add(desc,GrB_ARG2,GrB_NOP);     // no operation on the matrix
  GrB_Descriptor_add(desc,GrB_MASK,GrB_LNOT);    // invert the mask

  /*
   * BFS traversal and label the vertices.
   */
  int32_t d = 1;                                 // d = level in BFS traversal
  bool succ = false;                             // succ == true when some successor found
  do {
    GrB_assign(v,d,q);                           // v[q] = d
    GrB_vxm(&q,Boolean,q,A,*v,desc);             // q[!v] = q ||.&& A ; finds all the unvisited
                                                 // successors from current q
    GrB_reduce(&succ,q,GrB_LOR);                 // succ = ||(q)
    d++;                                         // next level
  } while (succ);                                // if there is no successor in q, we are done.

  GrB_free(q);                                   // q vector no longer needed
  GrB_free(Boolean);                             // Boolean semiring no longer needed
  GrB_free(desc);                                // descriptor no longer needed

  return GrB_SUCCESS;
}
```