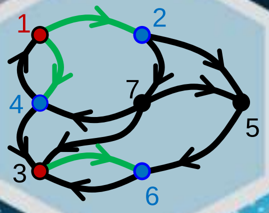
Carnegie Mellon University
Software Engineering Institute

GraphBLAS Updates

SC BoF: HPC Graph Toolkits and GraphBLAS Forum
20 November 2019

Scott McMillan
Email: smcmillan@sei.cmu.edu

Carnegie Mellon University
Software Engineering Institute



A Brief History…

Standards for Graph Algorithm Primitives

Tim Mattson (Intel Corporation), David Bader (Georgia Institute of Technology), Jon Berry (Sandia National Laboratory), Aydin Buluc (Lawrence Berkeley National Laboratory), Jack Dongarra (University of Tennessee), Christos Faloutsos (Carnegie Melon University), John Feo (Pacific Northwest National Laboratory), John Gilbert (University of California at Santa Barbara), Joseph Gonzalez (University of California at Berkeley), Bruce Hendrickson (Sandia National Laboratory), Jeremy Kepner (Massachusetts Institute of Technology), Charles Leiserson (Massachusetts Institute of Technology), Andrew Lumsdaine (Indiana University), David Padua (University of Illinois at Urbana-Champaign), Stephen Poole (Oak Ridge National Laboratory), Steve Reinhardt (Cray Corporation), Mike Stonebraker (Massachusetts Institute of Technology), Steve Wallach (Convey Corporation), Andrew Yoo (Lawrence Livermore National Laboratory)

- Sep. 2013: GraphBLAS "position paper" at IEEE HPEC
- Jun. 2015: GraphBLAS Forum "east coast/west coast" kickoff meeting
- Dec. 2015: Formation of the "GraphBLAS Signature Proposal Subcommittee" (C API Subcommittee)
- May 2017: GraphBLAS C API Specification v1.0 released ("provisional")
- Nov. 2017: SuiteSparse GraphBLAS v1.0 released
- May 2018: IBM GraphBLAS released, **"provisional" removed from C API spec. (v1.2.0)**
- Sep. 2019: GraphBLAS C API Specification v1.3.0 released

GraphBLAS Primitives

Operation	Description	Mathematical Description
mxm	Matrix multiplication (breadth-first traversal)	$\mathbf{C}\langle\neg\mathbf{M}, z\rangle = \mathbf{C} \odot (\mathbf{A}^T \oplus.\otimes \mathbf{B}^T)$
mxv, (vxm)		$\mathbf{c}\langle\neg\mathbf{m}, z\rangle = \mathbf{c} \odot (\mathbf{A}^T \oplus.\otimes \mathbf{b})$
eWiseMult	Element-wise ‘multiplication’ (graph intersection)	$\mathbf{C}\langle\neg\mathbf{M}, z\rangle = \mathbf{C} \odot (\mathbf{A}^T \otimes \mathbf{B}^T)$
eWiseAdd	Element-wise ‘addition’ (graph union)	$\mathbf{C}\langle\neg\mathbf{M}, z\rangle = \mathbf{C} \odot (\mathbf{A}^T \oplus \mathbf{B}^T)$
reduce (row/col)	Reduce along rows/cols (vertex degree)	$\mathbf{c}\langle\neg\mathbf{m}, z\rangle = \mathbf{c} \odot [\oplus_j \mathbf{A}^T(:,j)]$
apply	Apply unary function to each element (edge modification)	$\mathbf{C}\langle\neg\mathbf{M}, z\rangle = \mathbf{C} \odot f(\mathbf{A}^T)$
transpose	Swap rows and columns (reverse directed edges)	$\mathbf{C}\langle\neg\mathbf{M}, z\rangle = \mathbf{C} \odot \mathbf{A}^T$
extract	Extract a sub-matrix (sub-graph selection)	$\mathbf{C}\langle\neg\mathbf{M}, z\rangle = \mathbf{C} \odot \mathbf{A}^T(\mathbf{i},\mathbf{j})$
assign	Assign to a sub-matrix (sub-graph assignment)	$\mathbf{C}\langle\neg\mathbf{M}, z\rangle (\mathbf{i},\mathbf{j}) = \mathbf{C}(\mathbf{i},\mathbf{j}) \odot \mathbf{A}^T$
kronecker	Compute the Kronecker product of two matrices	$\mathbf{C}\langle\neg\mathbf{M}, z\rangle = \mathbf{C} \odot (\mathbf{A}^T \bigotimes \mathbf{B}^T)$
build (meth.)	Build a matrix from row, column, value tuples	$\mathbf{C} = \mathbb{S}^{mxn}(\mathbf{i},\mathbf{j},\mathbf{v},\odot)$
extractTuples (meth.)	Extract row, column, value tuples from a matrix	$(\mathbf{i},\mathbf{j},\mathbf{v}) = \mathbf{A}$

S. McMillan, et al., “Design and Implementation of the GraphBLAS Template Library (GBTL),” SIAM Annual Meeting (AN16), July 2016. Updated IEEE HPEC Conference, Sep 2017. Updated Sep 2019.

Notation: **i,j** – index arrays, **v** – scalar array, **m** – 1D mask, **other bold-lower** – vector (column), **M** – 2D mask, **other bold-caps** – matrix, **T** – transpose, ¬ - structural complement, z – clear output, ⊕ monoid/binary function, ⊕.⊗ semiring, **blue** – optional parameters, **red** – optional modifiers

GraphBLAS C API Spec. v1.3.0: Updates at a Glance

- Execution model updates (especially “completion”).
- New API methods and operations
- New predefined types
- Language Clarifications
- Miscellany

Language Regarding Completion Changed

The Concept of "Completion".

- A GraphBLAS Object is *"complete"* when it resides in memory and is "available" for another process or thread to access.
- This is required to construct "Happens Before" relations … which are the foundational concept for reasoning about concurrency

In GraphBLAS 1.0 to 1.2

- Completion of all involved GraphBLAS objects was required anytime a transparent object (e.g. nvals) was returned.
  - We combined the concepts of delivering a visible result and completion.
- This inhibits a number of key optimizations:
  - Example: triangle counting where we do matrix products and compute reductions on that matrix, but we never need the full matrix.

Solution for GraphBLAS 1.3

- Completion is a separate concept implemented by a call to GrB_wait(). This allows the obvious optimization for triangle count.
- We added a per-object GrB_wait() … e.g. GrB_wait(A)

New Operation: Kronecker matrix product

```
GrB_Info GrB_kronecker(GrB_Matrix             C,
                       const GrB_Matrix       Mask,
                       const GrB_BinaryOp     accum,
                       const GrB_BinaryOp     op, // or monoid/semiring
                       const GrB_Matrix       A,
                       const GrB_Matrix       B,
                       const GrB_Descriptor   desc);
```

GrB_kronecker computes the Kronecker product $\mathsf{C} = \mathsf{A} \otimes \mathsf{B}$ or, if an optional binary accumulation operator ($\odot$) is provided, $\mathsf{C} = \mathsf{C} \odot (\mathsf{A} \otimes \mathsf{B})$ (where matrices A and B can be optionally transposed). The Kronecker product is defined as follows:

$$\mathsf{C} = \mathsf{A} \otimes \mathsf{B} = \begin{bmatrix} A_{0,0} \otimes \mathsf{B} & A_{0,1} \otimes \mathsf{B} & \dots & A_{0,n_A-1} \otimes \mathsf{B} \\ A_{1,0} \otimes \mathsf{B} & A_{1,1} \otimes \mathsf{B} & \dots & A_{1,n_A-1} \otimes \mathsf{B} \\ \vdots & \vdots & \ddots & \vdots \\ A_{m_A-1,0} \otimes \mathsf{B} & A_{m_A-1,1} \otimes \mathsf{B} & \dots & A_{m_A-1,n_A-1} \otimes \mathsf{B} \end{bmatrix}$$

where $\mathsf{A} : \mathbb{S}^{m_A \times n_A}$, $\mathsf{B} : \mathbb{S}^{m_B \times n_B}$, and $\mathsf{C} : \mathbb{S}^{m_A m_B \times n_A n_B}$. More explicitly, the elements of the Kronecker product are defined as

$$\mathsf{C}(i_A m_B + i_B, j_A n_B + j_B) = A_{i_A, j_A} \otimes B_{i_B, j_B}$$

New variants of apply operation: binary op + scalar

- "Bind first" $C\langle\neg M,z\rangle \oplus= f_b(s, A)$ $w\langle\neg m,z\rangle \oplus= f_b(s, u)$

```
GrB_Info GrB_apply(GrB_Matrix             C,
                   const GrB_Matrix       Mask,
                   const GrB_BinaryOp     accum,
                   const GrB_BinaryOp     op,
                   <type>                 s,
                   const GrB_Matrix       A,
                   const GrB_Descriptor   desc);
```

```
GrB_Info GrB_apply(GrB_Vector             w,
                   const GrB_Vector       mask,
                   const GrB_BinaryOp     accum,
                   const GrB_BinaryOp     op,
                   <type>                 s,
                   const GrB_Vector       u,
                   const GrB_Descriptor   desc);
```

- "Bind second" $C\langle\neg M,z\rangle \oplus= f_b(A, s)$ $w\langle\neg m,z\rangle \oplus= f_b(u, s)$

```
GrB_Info GrB_apply(GrB_Matrix             C,
                   const GrB_Matrix       Mask,
                   const GrB_BinaryOp     accum,
                   const GrB_BinaryOp     op,
                   const GrB_Matrix       A,
                   <type>                 s,
                   const GrB_Descriptor   desc);
```

```
GrB_Info GrB_apply(GrB_Vector             w,
                   const GrB_Vector       mask,
                   const GrB_BinaryOp     accum,
                   const GrB_BinaryOp     op,
                   const GrB_Vector       u,
                   <type>                 s,
                   const GrB_Descriptor   desc);
```

New matrix and vector methods

- Resize matrices and vectors (larger and smaller):

```
GrB_Info GrB_Matrix_resize(GrB_Matrix  C,
                           GrB_Index   new_nrows,
                           GrB_Index   new_ncols);
GrB_Info GrB_Vector_resize(GrB_Vector  w,
                           GrB_Index   new_nsize);
```

- Remove single elements from matrices and vectors.

```
GrB_Info GrB_Matrix_removeElement(GrB_Matrix   C,
                                  GrB_Index    row_index,
                                  GrB_Index    col_index);
GrB_Info GrB_Vector_removeElement(GrB_Vector   w,
                                  GrB_Index    index);
```

Mask changes

- Added **`GrB_STRUCTURE`** (structure only)
  - Original mode (evaluate to true) is still the default
- Deprecating **`GrB_SCMP`** symbol for **`GrB_COMP`**
- Can compose **`GrB_COMP`** with **`GrB_STRUCTURE`** (complement of the structure only)

All 32 possible combinations of descriptor flags are now predefined:

Identifier	GrB_OUTP	GrB_MASK	GrB_INP0	GrB_INP1
GrB_NULL	–	–	–	–
GrB_DESC_T1	–	–	–	GrB_TRAN
GrB_DESC_T0	–	–	GrB_TRAN	–
GrB_DESC_T0T1	–	–	GrB_TRAN	GrB_TRAN
GrB_DESC_C	–	GrB_COMP	–	–
GrB_DESC_S	–	GrB_STRUCTURE	–	–
GrB_DESC_CT1	–	GrB_COMP	–	GrB_TRAN
GrB_DESC_ST1	–	GrB_STRUCTURE	–	GrB_TRAN
GrB_DESC_CT0	–	GrB_COMP	GrB_TRAN	–
GrB_DESC_ST0	–	GrB_STRUCTURE	GrB_TRAN	–
GrB_DESC_CT0T1	–	GrB_COMP	GrB_TRAN	GrB_TRAN
GrB_DESC_ST0T1	–	GrB_STRUCTURE	GrB_TRAN	GrB_TRAN
GrB_DESC_SC	–	GrB_STRUCTURE, GrB_COMP	–	–
GrB_DESC_SCT1	–	GrB_STRUCTURE, GrB_COMP	–	GrB_TRAN
GrB_DESC_SCT0	–	GrB_STRUCTURE, GrB_COMP	GrB_TRAN	–
GrB_DESC_SCT0T1	–	GrB_STRUCTURE, GrB_COMP	GrB_TRAN	GrB_TRAN
GrB_DESC_R	GrB_REPLACE	–	–	–
GrB_DESC_RT1	GrB_REPLACE	–	–	GrB_TRAN
GrB_DESC_RT0	GrB_REPLACE	–	GrB_TRAN	–
GrB_DESC_RT0T1	GrB_REPLACE	–	GrB_TRAN	GrB_TRAN
GrB_DESC_RC	GrB_REPLACE	GrB_COMP	–	–
GrB_DESC_RS	GrB_REPLACE	GrB_STRUCTURE	–	–
GrB_DESC_RCT1	GrB_REPLACE	GrB_COMP	–	GrB_TRAN
GrB_DESC_RST1	GrB_REPLACE	GrB_STRUCTURE	–	GrB_TRAN
GrB_DESC_RCT0	GrB_REPLACE	GrB_COMP	GrB_TRAN	–
GrB_DESC_RST0	GrB_REPLACE	GrB_STRUCTURE	GrB_TRAN	–
GrB_DESC_RCT0T1	GrB_REPLACE	GrB_COMP	GrB_TRAN	GrB_TRAN
GrB_DESC_RST0T1	GrB_REPLACE	GrB_STRUCTURE	GrB_TRAN	GrB_TRAN
GrB_DESC_RSC	GrB_REPLACE	GrB_STRUCTURE, GrB_COMP	–	–
GrB_DESC_RSCT1	GrB_REPLACE	GrB_STRUCTURE, GrB_COMP	–	GrB_TRAN
GrB_DESC_RSCT0	GrB_REPLACE	GrB_STRUCTURE, GrB_COMP	GrB_TRAN	–
GrB_DESC_RSCT0T1	GrB_REPLACE	GrB_STRUCTURE, GrB_COMP	GrB_TRAN	GrB_TRAN

New Predefined Operators and Monoids

- Unary Operators
  - **GrB_ABS_***T* – absolute value
  - **GrB_BNOT_***I* – bitwise comp. (integers only)

- Monoids (and their identities)
  - **GrB_PLUS_MONOID_***T* – (+, 0)
  - **GrB_TIMES_MONOID_***T* – (x, 1)
  - **GrB_MIN_MONOID_***T* – (min, uint/int_max or infinity)
  - **GrB_MAX_MONOID_***T* – (max, 0, int_min or -infinity)

  - **GrB_LOR_MONOID_BOOL** – logical OR
  - **GrB_LAND_MONOID_BOOL** – logical AND
  - **GrB_LXOR_MONOID_BOOL** – logical XOR
  - **GrB_LXNOR_MONOID_BOOL** – logical XNOR

- Binary Operators
  - **GrB_LXNOR** – logical XNOR (bool)
  - **GrB_BOR_***I* – bitwise OR (ints)
  - **GrB_BAND_***I* – bitwise AND (ints)
  - **GrB_BXOR_***I* – bitwise XOR (ints)
  - **GrB_BXNOR_***I* – bitwise XNOR (ints)

New Predefined Semirings

"True" semirings:

(additive identity ==

multiplicative annihilator)

GraphBLAS identifier	$(T \times T \rightarrow T)$	+ identity × annihilator	Description
GrB_PLUS_TIMES_SEMIRING_T	UINTx	0	arithmetic semiring
	INTx	0	
	FPx	0	
GrB_MIN_PLUS_SEMIRING_T	UINTx	UINTx_MAX	min-plus semiring
	INTx	INTx_MAX	
	FPx	INFINITY	
GrB_MAX_PLUS_SEMIRING_T	INTx	INTx_MIN	max-plus semiring
	FPx	-INFINITY	
GrB_MIN_TIMES_SEMIRING_T	UINTx	UINTx_MAX	min-times semiring
GrB_MIN_MAX_SEMIRING_T	UINTx	UINTx_MAX	min-max semiring
	INTx	INTx_MAX	
	FPx	INFINITY	
GrB_MAX_MIN_SEMIRING_T	UINTx	0	max-min semiring
	INTx	INTx_MIN	
	FPx	-INFINITY	
GrB_MAX_TIMES_SEMIRING_T	UINTx	0	max-times semiring
GrB_PLUS_MIN_SEMIRING_T	UINTx	0	plus-min semiring
GrB_LOR_LAND_SEMIRING_BOOL	BOOL	false	Logical semiring
GrB_LAND_LOR_SEMIRING_BOOL	BOOL	true	"and-or" semiring
GrB_LXOR_LAND_SEMIRING_BOOL	BOOL	false	same as NEQ_LAND
GrB_LXNOR_LOR_SEMIRING_BOOL	BOOL	true	same as EQ_LOR

New Predefined Semirings

"Useful" semirings

(no multiplicative annihilator)

GraphBLAS identifier	Domains, T $(T \times T \rightarrow T)$	+ identity	Description
GrB_MAX_PLUS_SEMIRING_T	UINTx	0	max-plus semiring
GrB_MIN_TIMES_SEMIRING_T	INTx	INTx_MAX	min-times semiring
	FPx	INFINITY	
GrB_MAX_TIMES_SEMIRING_T	INTx	INTx_MIN	max-times semiring
	FPx	-INFINITY	
GrB_PLUS_MIN_SEMIRING_T	INTx	0	plus-min semiring
	FPx	0	
GrB_MIN_FIRST_SEMIRING_T	UINTx	UINTx_MAX	min-select first semiring
	INTx	INTx_MAX	
	FPx	INFINITY	
GrB_MIN_SECOND_SEMIRING_T	UINTx	UINTx_MAX	min-select second semiring
	INTx	INTx_MAX	
	FPx	INFINITY	
GrB_MAX_FIRST_SEMIRING_T	UINTx	0	max-select first semiring
	INTx	INTx_MIN	
	FPx	-INFINITY	
GrB_MAX_SECOND_SEMIRING_T	UINTx	0	max-select second semiring
	INTx	INTx_MIN	
	FPx	-INFINITY	

Miscellany

- Added run-time getVersion() and compile-time version macros
- Updated all code examples
  - Update to use new capabilities where possible
  - Bug fix in non-batch BC code
  - Added parent-BFS example
- Clarifications:
  - Distributive law
  - init/finalize errors
  - Boolean/integer division
  - Aliasing in user-defined operators
  - Freeing predefined objects
  - Removed unnecessary language about annihilators and implied zeros

GraphBLAS Implementations

SuiteSparse library (Texas A&M): First fully conforming GraphBLAS C API release.

- http://faculty.cse.tamu.edu/davis/suitesparse.html

IBM-GraphBLAS: the second fully conforming C API release,

- https://github.com/IBM/ibmgraphblas

GBTL: GraphBLAS Template Library (CMU/SEI/PNNL/IU): GraphBLAS C++ API

- https://github.com/cmu-sei/gbtl

GraphBLAST: A C++ implementations for GraphBLAS for GPUs (UC Davis),

- https://github.com/gunrock/graphblast

Distributed GBTL: in progress (LLNL)

- See Roger Pearce

Python bindings:

- **pyGraphBLAS**: A Python Wrapper around SuiteSparse GraphBLAS
  - https://github.com/michelp/pygraphblas
- **pyGB**: A Python Wrapper around GBTL (UW/PNNL/CMU)
  - https://github.com/jessecoleman/gbtl-python-binding

pggraphblas: A PostgreSQL wrapper around SuiteSparse GraphBLAS

- https://github.com/michelp/pggraphblas

Matlab and Julia wrappers around SuiteSparse GraphBLAS

- https://aldenmath.com

Etc…

LAGraph: Curating a collection of high-level graph algorithms

- Launched May 2019 at GrAPL'19.
- Algorithms (current/pending):
  - BFS (simple, push/pull)
  - PageRank (multiple)
  - Triangle counting
  - K-truss (and All K-truss)
  - Clustering (label propagation)
  - Local Clustering Coefficient
  - SSSP (Bellman-Ford x 2, delta-stepping)
  - Connected components (LACC, FastSV)
  - Betweenness Centrality (single node, batch)
  - MIS (Luby's)
  - All GAP Benchmark Algorithms (under development).
- https://github.com/GraphBLAS/LAGraph (BSD 2 Clause).

LAGraph: A Community Effort to Collect Graph Algorithms Built on Top of the GraphBLAS

Tim Mattson[‡], Timothy A. Davis[°], Manoj Kumar[¶], Aydın Buluç[!], Scott McMillan[§], José Moreira[¶], Carl Yang[*,!]

[‡]*Intel Corporation* [!]*Computational Research Division, Lawrence Berkeley National Laboratory*
[°]*Texas A&M University* [¶]*IBM Corporation* [§]*Software Engineering Institute, Carnegie Mellon University*
[*]*Electrical and Computer Engineering Department, University of California, Davis*

GraphBLAS Forum Information

Steering Committee: David Bader, Aydın Buluç, John Gilbert, Jeremy Kepner, Tim Mattson, Henning Meyerhenke

API Subcommittee: Benjamin Brock, Aydın Buluç, Tim Mattson, Scott McMillan, Jose Moreira

Mailing list: Graphblas@lists.lbl.gov

- Join the Forum by joining the list (mailto:abuluc@lbl.gov)

Website: http://graphblas.org

- Link to the latest C API Specification
- Lists teams developing implementations
- Other useful resources including the “The Math Document”

Monthly teleconference:

- Second Friday of every month, 12pm Eastern Time
- Send email to Jeremy Kepner to receive the calendar invite.

Backups

GraphBLAS Signatures: mxm

$$C\langle \neg M, z\rangle = C \odot (A^T \oplus.\otimes B^T)$$

```
// GraphBLAS C API
GrB_Info GrB_mxm(GrB_Matrix                 C,
                 GrB_Matrix       const  M,
                 GrB_BinaryOp     const  accum,
                 GrB_Semiring     const  op,
                 GrB_Matrix       const  A,
                 GrB_Matrix       const  B,
                 GrB_Descriptor   const  desc);
```

```
// GBTL C++ API
namespace GraphBLAS
{
    template <typename CMatrixT,
              typename MaskT,
              typename AccumT,
              typename SemiringT,
              typename AMatrixT,
              typename BMatrixT>
    void mxm(CMatrixT             &C,
             MaskT          const &M,
             AccumT               accum,
             SemiringT            op,
             AMatrixT       const &A,
             BMatrixT       const &B,
             bool                 replace_flag);
}
```

GraphBLAS Signatures: mxm

$$\mathbf{C}\langle\neg\mathbf{M}, z\rangle = \mathbf{C} \odot (\mathbf{A}^{T} \oplus.\otimes \mathbf{B}^{T})$$

```
// GraphBLAS C API
GrB_Info GrB_mxm(GrB_Matrix                 C,
                 GrB_Matrix      const      M,
                 GrB_BinaryOp    const      accum,
                 GrB_Semiring    const      op,
                 GrB_Matrix      const      A,
                 GrB_Matrix      const      B,
                 GrB_Descriptor  const      desc);
```

- **C** stores the result
- **C** is also used as input if an optional accumulation operator ($\odot$) is specified.

GraphBLAS Signatures: mxm

$$\mathbf{C}\langle \neg \mathbf{M}, z\rangle = \mathbf{C} \odot (\mathbf{A}^{\mathrm{T}} \oplus.\otimes \mathbf{B}^{\mathrm{T}})$$

```
// GraphBLAS C API
GrB_Info GrB_mxm(GrB_Matrix                C,
                 GrB_Matrix      const     M,
                 GrB_BinaryOp    const     accum,
                 GrB_Semiring    const     op,
                 GrB_Matrix      const     A,
                 GrB_Matrix      const     B,
                 GrB_Descriptor  const     desc);
```

- Mask, **M**, is optional.
- If not specified (GrB_NULL), the entire **C** matrix is overwritten.

GraphBLAS Signatures: mxm

$$\mathbf{C}\langle\neg\mathbf{M}, z\rangle = \overbrace{\mathbf{C} \odot}\ (\mathbf{A}^{T} \oplus.\otimes\ \mathbf{B}^{T})$$

```
// GraphBLAS C API
GrB_Info GrB_mxm(GrB_Matrix              C,
                 GrB_Matrix       const  M,
                 GrB_BinaryOp     const  accum,
                 GrB_Semiring     const  op,
                 GrB_Matrix       const  A,
                 GrB_Matrix       const  B,
                 GrB_Descriptor   const  desc);
```

- The accumulation operator, $\odot$, is optional.
- If not specified (GrB_NULL), the **C** matrix is used as output only (i.e., does not appear on the right hand side).

GraphBLAS Signatures: mxm

$$C\langle \neg M, z\rangle = C \odot (A^{T} \underbrace{\oplus.\otimes} B^{T})$$

```
// GraphBLAS C API
GrB_Info GrB_mxm(GrB_Matrix              C,
                 GrB_Matrix      const   M,
                 GrB_BinaryOp    const   accum,
                 GrB_Semiring    const   op,
                 GrB_Matrix      const   A,
                 GrB_Matrix      const   B,
                 GrB_Descriptor  const   desc);
```

- The semiring used in the matrix multiply.
  - $\oplus$, a commutative monoid, replaces “plus”
  - $\otimes$, a binary operator, replaces “times”
- More on this is a minute…

GraphBLAS Signatures: mxm

$$\mathbf{C}\langle\neg\mathbf{M}, z\rangle = \mathbf{C} \odot (\mathbf{A}^{T} \oplus.\otimes \mathbf{B}^{T})$$

```
// GraphBLAS C API
GrB_Info GrB_mxm(GrB_Matrix             C,
                 GrB_Matrix      const  M,
                 GrB_BinaryOp    const  accum,
                 GrB_Semiring    const  op,
                 GrB_Matrix      const  A,
                 GrB_Matrix      const  B,
                 GrB_Descriptor  const  desc);
```

- Input matrices

GraphBLAS Signatures: mxm

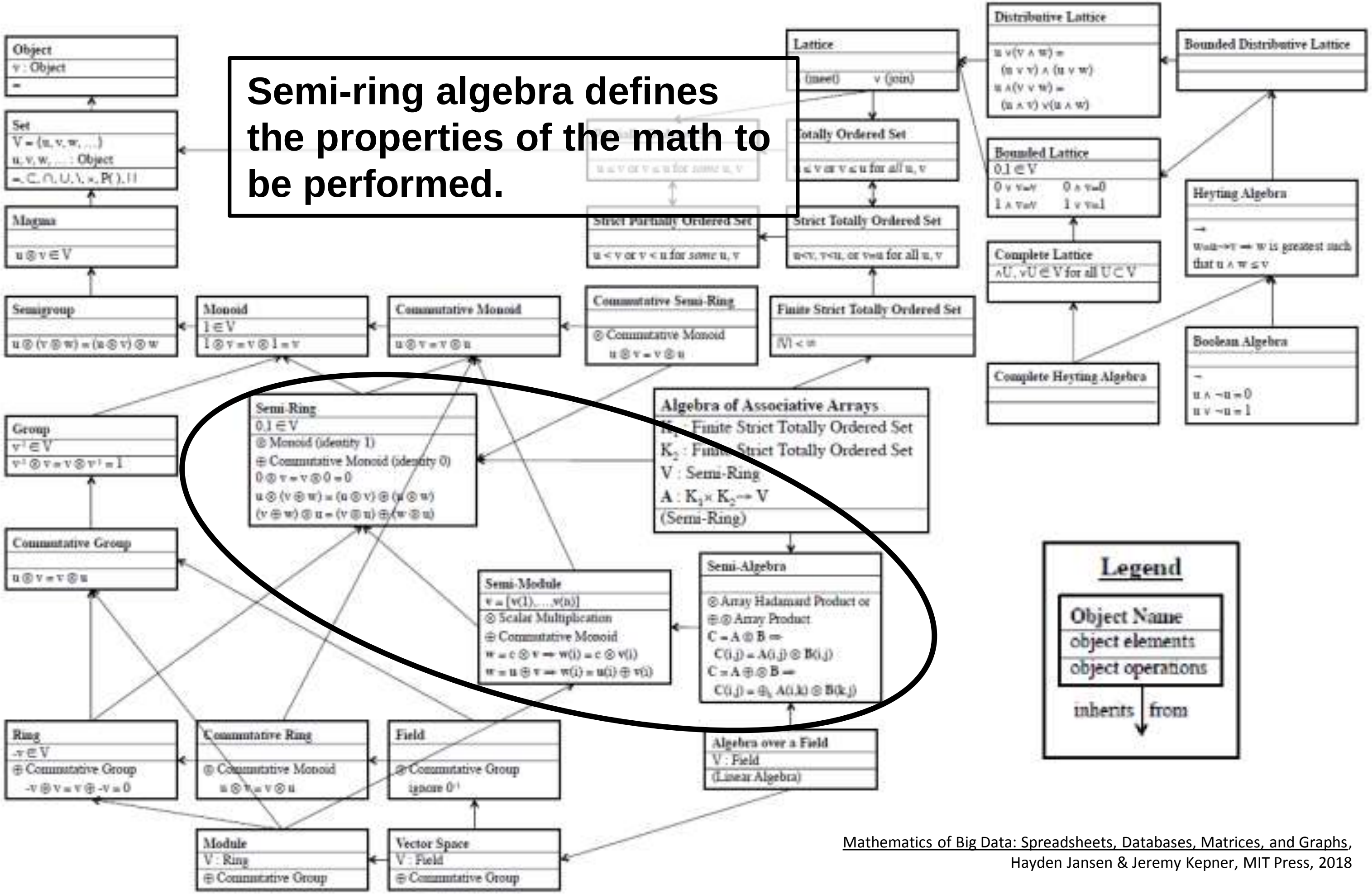
$$C\langle \neg M, z\rangle \;=\; C \odot (A^{T} \oplus.\otimes B^{T})$$

```
// GraphBLAS C API
GrB_Info GrB_mxm(GrB_Matrix                    C,
                 GrB_Matrix       const   M,
                 GrB_BinaryOp     const   accum,
                 GrB_Semiring     const   op,
                 GrB_Matrix       const   A,
                 GrB_Matrix       const   B,
                 GrB_Descriptor   const   desc);
```

- Optional Descriptor can specify any or all of the following:
  - Complement the mask, $\neg$
  - Clear the output matrix before writing final result, z
  - Transpose any input matrix, ^{T}

Algebra

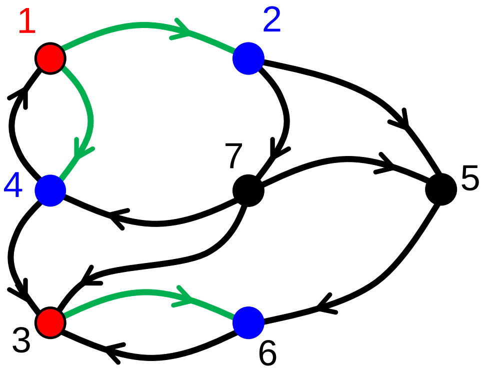
Semi-ring algebra defines the properties of the math to be performed.

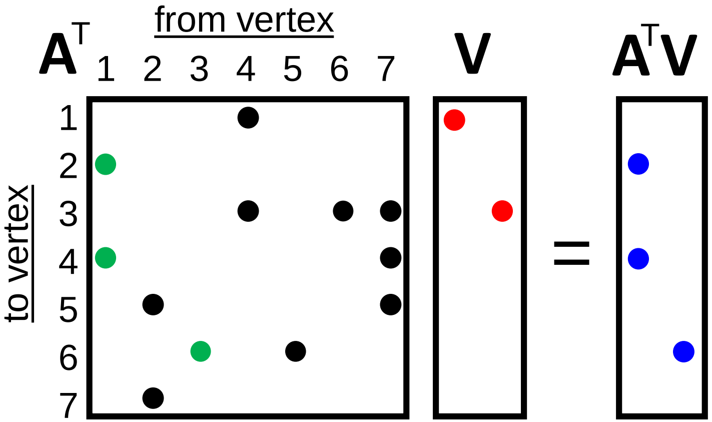


Mathematics of Big Data: Spreadsheets, Databases, Matrices, and Graphs, Hayden Jansen & Jeremy Kepner, MIT Press, 2018

Matrix Multiply using Semirings

$$\mathbf{C} = \mathbf{A} \oplus . \otimes \mathbf{B}$$





- The Semiring (⊕.⊗) determines how this computation is carried out.
- Consists of two Monoids (binary operator, identity)
  - ⊕, Commutative Monoid: e.g., (add, 0)
  - ⊗, Monoid: e.g., (multiply, 1)

where ⊕'s identity = ⊗'s annihilator

$$c_{i,j} = \sum_{l=1}^{k} a_{i,l} \times b_{l,j}$$

- In GraphBLAS:
  - ⊕, is a Commutative Monoid with identity (same as above)
  - ⊗, is a Binary Function (i.e., no identity required)
  - No enforcement that ⊕'s identity = ⊗'s annihilator, e.g. ⊕=(min, ∞) and ⊗='second'