GRAPH-MAP ISOMORPHISM

William Bricken
October 1996

Graph=Map?

Consider a set-theoretic vocabulary rather than a diagrammatic one. [I rarely do this.] So a "graph" is two sets, V and E. E is structured to contain pairs from V.

Initial question: why are two sets and the structure of set E desirable/necessary for a model?

A "map" is two sets, T and B. B is structured to contain pairs from T. Composition of maps is something we haven't discussed, I'll it skip for now, but it is not as straightforward as graphs.

We change the connectivity of a graph by removing a member from E. When we change a map, we also remove a member from B, but this also changes the composition of T. So what I see as *the breakdown in graph-map isomorphism* is the difference in the effect of a connectivity change on the respective structures V and T.

The map change is that two members of T, say Tm and Tn, meld into one member Tmn, which is what we mean by spatial composition. The B set is also modified by renaming references to both Tm and Tn as Tmn. The graph change does not give melding, since each of the vertices may have other edge-references which are not remapped, so neither V nor remaining members of E change.

However(!) I've convinced myself that graph=map for all changes which maintain Boolean equivalence classes, so I'm ready to say that they are the same for our purposes.

The bipartite graph model has a partition of V into two classes, say Vd and Vs, and additional structure on E in that all member pairs have one reference to a member of Vd and one to Vs. When a boundary is removed, two members of E are deleted, and one member of Vd. No renaming in E occurs.

Finally, in the Dedge model, we have sets S and E, as in a graph. And pins or something, since I don't know how to express spatial composition. [Recent correspondence has solved this, we have partitioned S into two sets, named and unnamed spaces.]

One thing that we haven't discussed is the role of equivalence sets of graphs. In what way can a graph express an algebra? I tend to focus on transformations of graphs, since this is what area/time tradeoff maps to.

There is also the representation question, what is the most efficient form of a graph with a particular behavior. You have a strong dataflow orientation: we have the graph we want, lets pump evaluations through it. I see even evaluation as transformation rather than transmission.

Open question: are these differences only in implementation, or do they map onto something fundamental? I suspect a mixture.

So to explore, I'll try to express the foundational structures of boundary math in each. Note that I'm using caps for arbitrary structures, facing composition/hierarchy directly, and considering ground variables as a simple case. POV is added as • on the outside when appropriate for expressability [I trust this is not controversial, it is a point to consider carefully.]

Bottom line on the exploration is that graph=map, but type of graph is still an open question.

Representational Issues

And while at it, I have included the elementary examples of the representational problems of boundary logic (and any computation in general), which are specifically:

1. shared vs. confounded space: check coalesce for eg, how to switch POV for compound expressions.

2. coalesce: when we say A=A, we pose the matching problem, how to match the components of each A, when A is compound. Jeff and I have excellent graph algorithms for this. Matching is particularly difficult when the two As are not in a canonical form.

3. sided pervasion: reference, pervasion, and subsumption are all the same rule, they are all co-derivable. POV plays a role in clarifying this.

4. Robbins: resolution is sufficient as a single axiom when void substitution is not allowed, but Robbins is not. A fundamental unsolved foundational problem.

5. distribution: the only rule which does not insist upon a direction. All NP proofs require using distribution in both directions at the right time. This is the *only* source of computational intractability.

Of the above problems, #2 and #5 define what is meant by non-trivial computation. #1 and #4 define non-trivial foundational issues. #3 is purely representational.

==========

PARENS

<void>	
distinction	( )
outside	( ) ( )
inside	(( ))
named structure	A
replicated	A A
uniqueness	A•A
shared space	A B
confounded space	A•B
distinguished	(A) B
joined	((A)(B))
cross	(( )) =
call	( ) ( ) = ( )
dominion	A ( ) = ( )
involution	((A)) = A
pervasion	A (A B) = A (B)
reference	A A = A
coalesce	(A) (A) = (A)
subsumption	(A) (A B) = (A)
resolution	((A B)(A (B))) = A
Robbins	(A B)(A (B)) = (A)
distribution	((A B)(A C)) = A ((B)(C))
pivot	((A B) ((A)(C))) = (A (B)) ((A) C)

==========

```
=================================================================
DNET  GRAPH           V-set              E-set

<void>             •                   <empty>
distinction        • d1               {• d1}
outside            • d1 d2            {• d1}{• d2}
inside             • d1 d2            {• d1}{d1 d2}

named structure    • A                {• A}
replicated         • A                {• A}{• A}
uniqueness         • A                {• A}{• A}
shared space       • A B              {• A}{• B}
confounded space   • A B              {• A}{• B}
distinguished      • d1 A B           {• d1}{d1 A}{• B}
joined             • d1 d2 d3 A B     {• d1}{d1 d2}{d1 d3}{d2 A}{d3 B}

cross              • d1 d2            {• d1}{d1 d2}
      =            •                   <empty>

call               • d1 d2            {• d1}{• d2}
      =            • d1               {• d1}

dominion           • A d1             {• d1}{• A}
      =            •   d1             {• d1}

involution         • d1 d2 A          {• d1}{d1 d2}{d2 A}
      =            •       A                      { • A}

pervasion          • d1 A B           {• A}{• d1}{d1 B}{d1 A}
      =            • d1 A B           {• A}{• d1}{d1 B}

reference          • A                {• A}{• A}
      =            • A                {• A}

coalesce           • d1 d2 A          {• d1}{d1 A}{• d2}{d2 A}
      =            • d1    A          {• d1}{d1 A}{d1 A}
      =            • d1    A          {• d1}{d1 A}

subsumption        • d1 d2 A B        {• d1}{d1 A}{• d2}{d2 A}{d2 B}
      =            • d1    A          {• d1}{d1 A}

resolution         • d1 d2 d3 d4 A B
                           {• d1}{d1 d2}{d1 d3}{d2 A}{d2 B}{d3 A}{d3 d4}{d4 B}
      =            • A                {• A}
```

DNET GRAPH	*V-set*	*E-set* *[continued]*
Robbins	• d1 d2 d3 A B	{• d1}{d1 A}{d1 B}{• d2}{d2 A}{d2 d3}{d3 B}
=	• d1 A	{• d1}{d1 A}
distribution	• d1 d2 d3 A B C	{• d1}{d1 d2}{d1 d3}{d2 B}{d3 C}{d2 A}{d3 A}
=	• d1 d2 d3 A B C	{• d1}{d1 d2}{d1 d3}{d2 B}{d3 C}{ • A}
pivot	• d1 d2 d3 d4 d5 A B C {• d1}{d1 d2}{d2 A}{d2 B}{d1 d3}{d3 d4}{d3 d5}{d4 A}{d5 C}	
=	• d2 d3 d4 d6 A B C {• d2}{d2 A}{d2 d6}{d6 B}{• d3}{d3 d4}{d4 A}{d3 C}	

==========

Notes

1. Replicated/uniqueness and shared/confounded space are not different in this form.

2. Coalesce has an intermediate form, emphasizing that the distinction itself is replicated. Distinctions can be seen as replicated as well in subsumption, resolution, Robbins and pivot.

3. Replicated distinctions are named in pivot. Pivot requires only one variable at odd and even depth, so has four distinct forms. Similarly distribution has four forms, for eg: ((A (B))(A (C))) = A (B C)

Here's the rough workup for the new suggested notation:

```
=========================================================================
DEDGE  GRAPH

<void>                                      •
distinction          ( )                    •-|
outside              ( ) ( )                |-• •-|
inside               ( ( ) )                •-|-*-|

named structure        A                    A•
replicated             A A                  A• •A
uniqueness             AA                   A•A = A•
shared space           A B                  A• •B
confounded space       AB                   A•B
distinguished         (A) B                 A-|-B•
joined               ((A)(B))               •-|-*-|-A
                                                \-|-B

cross                ( ( ) ) =              •-|-*-|  =  •

call                 ( ) ( ) = ( )          |-• •-| =~ |-•-| ~= •-|

dominion              A  ( ) = ( )          A• •-| =~ A•-| ~= •-|

involution           ( (A) ) =  A           •-|-*-|-A = •A

pervasion            A (A B) = A (B)        A• •-|-AB =~ A•-|-B ~= •A-|-B
                                                          \--/

reference             A   A  = A            A• •A =~ A•A ~= A•

coalesce             (A) (A) = (A)          A-|-• •-|-A =~ •-|-A ~= •-|-A
                                                        \-/

subsumption        (A) (A B) = (A)          A-|-• •-|-AB =~ A-|-•-|-B ~= A-|-•
                                                          \-----/

resolution  ((A B)(A (B))) =  A             •-|-*-|-AB
                                                \-|-A-|-B

Robbins       (A B)(A (B))  = (A)
distribution  ((A B)(A C)) =  A ((B)(C))          [later]
pivot      ((A B) ((A)(C))) = (A (B)) ((A) C)

=========================================================================
```

Notes

1. I've used

=~ ... ~=

to mean the imaginary/transitional state that never really exists. It also shows that the process of unifying POV creates (and is sufficient to create) the change (!). A very nice distinction then arises between POV transforms associated with spatial phenomena (call, dominion, reference) and topological transforms associated with distinction phenomena (cross, involution). Pervasion/subsumption are the same and are both spatial and topological, which I have known for a while and have always had difficulty communicating.

2. May be some clerical errors.

3. • is the name of the top level space

4. Adjacency means confounded in space

5. Pervasion on is unclear, see last paragraph below.

6. Resolution:

(A (B)) =?= •-|-A-|-B

I'm still not comfortable with coalesce, aka complex reference. This seems tied into the equivalence class issue, so:

Boundary logic as an interpretation of boundary math. Integers and sets are other interpretations of the abstract structure. Interpretation does not become important until we begin to establish equivalence classes. So, for example,

A A = A

is an interpretation which implements idempotency and thus Booleanism.

A A = 2A

is an interpretation which implements integers. Yes, we can use propositional logic as a primary interpretation (built-in as crossing and calling axioms), then interpret integers, etc. as logic. But this is a choice.

My questions start up at multiple reference (no surprise). When we say A=A, that is trivial only when A is a ground object. When it is a complex form,

we must invest computational effort to establish either the identicality of each side of the equation (A=A), or equivalently the truth of the equation (A=B is true).

So I'm calling this the equivalence class issue. I think hierarchy is also involved in the sense that once an encapsulation is constructed, we loose ability to know what is inside. In any event, this issue is at the heart of most of my practical work in logic synthesis. It comes down to

1) how can you identify identical structures (the match issue), and

2) how can you identify redundant structures (the optimization issue).

I was very comfortable with (1) until we began thinking about confounded spaces.

Expressions with grounds can easily require work to match. The theorems above are good examples, here are some elementary tricky others.

Redundant structure:

(a (b c)) (a (b)) (a (c)) = (a (b c))

Classical two-variable tautology:

((a b) ((a)(b))) ((a (b)) (b (a))) = ( )

Three-variable variation tautology:

((a b) ((a)(c))) ((a (b)) (c (a))) = ( )

The problem is that it takes search to figure out the right thing to do with these. But the bigger problem is that they are true also of the algebra, in which each letter can be an arbitrary configuration. So

1) you can quickly construct large tangled structures which carry no functionality or which present hazards etc. in circuits, and

2) in algebraic transformation, you can no longer resort to case analysis. Also note that these equations are true only when we interpret boundary math in a particular domain, Boolean logic.

What this all means is that a good representation of the foundations is not sufficient, we also need a good transformational structure built into the representation. But even without pragmatics, I suspect that tangled reference interacts at basic levels with the notion of confounded space. The simplest example is

(a)(a) a-|-• •-|-a

where a is a ground. When we join POV, as in

 a-|-•-|-a

we still cannot know what is on the other side of the distinction without exerting effort. I suppose

a• •a ==> a•a

might present the same problem, but here at least you can argue that there is only one a and it is not confounded with itself. But in the case of (a), the confounding is *at a distance*, not in the space of your perspective. And I don't think you can go to each space simultaneously. So you either have to

1) construct memory and visit both spaces over time, or

2) have the unique a signal the apparently duplicate D, or

3) use only one D so that the duplication is more apparent, or

4) have a global perspective and do matching, or ...

Anyway, I'm not at all sure about how to represent pervasion,

a (a b) = a (b)

Is the "a b" structure confounded as in

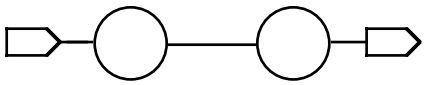
a (a b)

or not? If it is, how do you disambiguate the a from the b when in fact it is there only as an artifact?

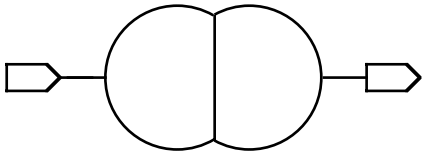
GRAPH ISOMORPHISM UNDER VOID TRANSFORMATIONS

Herein "old" refers to the conventional way of building graphs. "New" refers to the boundary techniques being explored. The following figures illustrate our choices (ASCII is used for the rest of the paper)

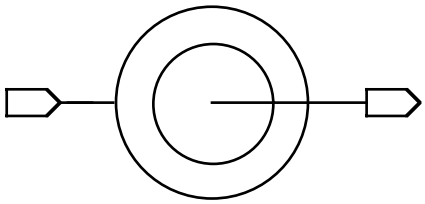
Simple GRAPH diagram with abstract homogeneous functional units



Simple territory-based MAP diagram



Simple containment-based PATH diagram



Graph isomorphism is a deeper question, as is the issue of differentiating between Join and Share. Graphs and maps are Old-isomorphic and New-isomorphic, but not isomorphic between Old and New (!). It's the thing about representations hiding some aspects of meaning and highlighting others. From a set theoretic Old-description model, you do not have the tools to say how they are different, but erasing an Old-edge is not the same as erasing a border or a path.

Consider the instruction "Erase an edge/border/path"

Graph:	( )---( )	=	( )	( )	or	---	=
Map:	(|)	=	( )		or	|	=
Path:	----(-)	=	( )		or	-----	=

Consider: "Erase a vertex/territory/location"

Graph:	( )---( )	=	( )	or	( )---	=
Map:	(|)	=	|)	or	(	=
Path:	----(-)	=	---- -	or	( )	=

These differences are in the pure structures, not in any interpretations. Let me try Cross and Call from nine notations (all have duals). (The O- means oriented, your POV is explicit. ^/> in rooms is a door in a wall.)

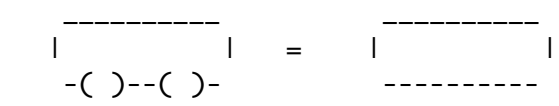
CROSS

Parens:	(( ))	=	
O-graph:	.---( )---( )	=	.
O-D-edge:	.-D-( )-D-( )	=	.
Map:	(|)	=	
O-map:	(. | |)	=	(.)
Rooms:	(> >)	=	( )
Path:	--(---)--	=	-- ( )

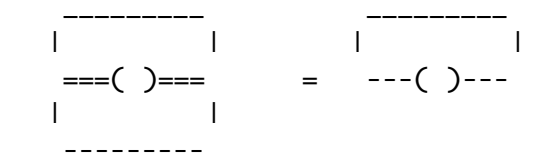
CALL

Parens:	( ) ( )	=	( )
O-graph:	( )---.---( )	=	.---( )
O-D-edge:	( )-D-.-D-( )	=	.-D-( )
Map:	( ) ( )	=	( )
O-map:	(| . |)	=	(| .)
Rooms:	(^ | ^)	=	(^)
Path:	===(=)	=	---(-)

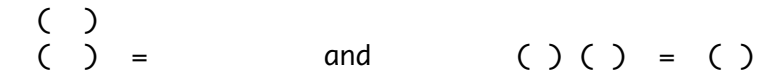
Closed-cycle-graphs



and



It is even possible to differentiate the axioms horizontally and vertically, as in Rock-wall:



[Incidentally, Tetris uses these rules with 90-degree rotation.]

Now, I might not be doing justice to D-edges by forcing them to be O-, but the essential phenomena shows up in all notations:

We need to distinguish between combination-in-space (no shared boundary/border) and combination-in-bound (shared boundary/border). This can be achieved by making O- explicit, in particular with the O-map, when we get c-in-space by placing our POV in between, and c-in-bound by not placing POV in between.