Event Graphing and Step Inverses for CnC

Peter Elmers, Nick Vrvilo, Vivek Sarkar

The Event Graph

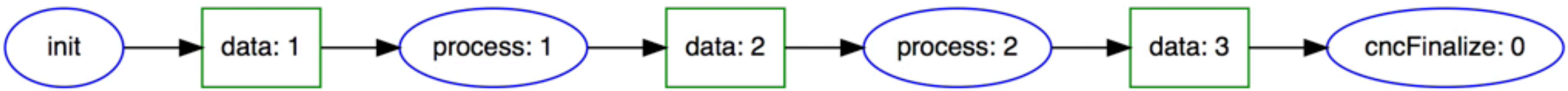
- Directed graph of execution flow of steps and items
- Acyclic! A cycle would be a deadlock
- Begins at an init step, ends at a finalize step

Simple Example

```
$context { int length; };
[ int data: i ];

( $initialize: () ) -> [ data : 1 ], ( process: 1 );

( process: x )
    <- [ data : x ]
    -> [ data : x + 1 ],
       ( process: x + 1 ) $when(x + 1 < #length);

( $finalize: () ) <- [ data: #length ];
```

How it looks

length = 3:



Motivation

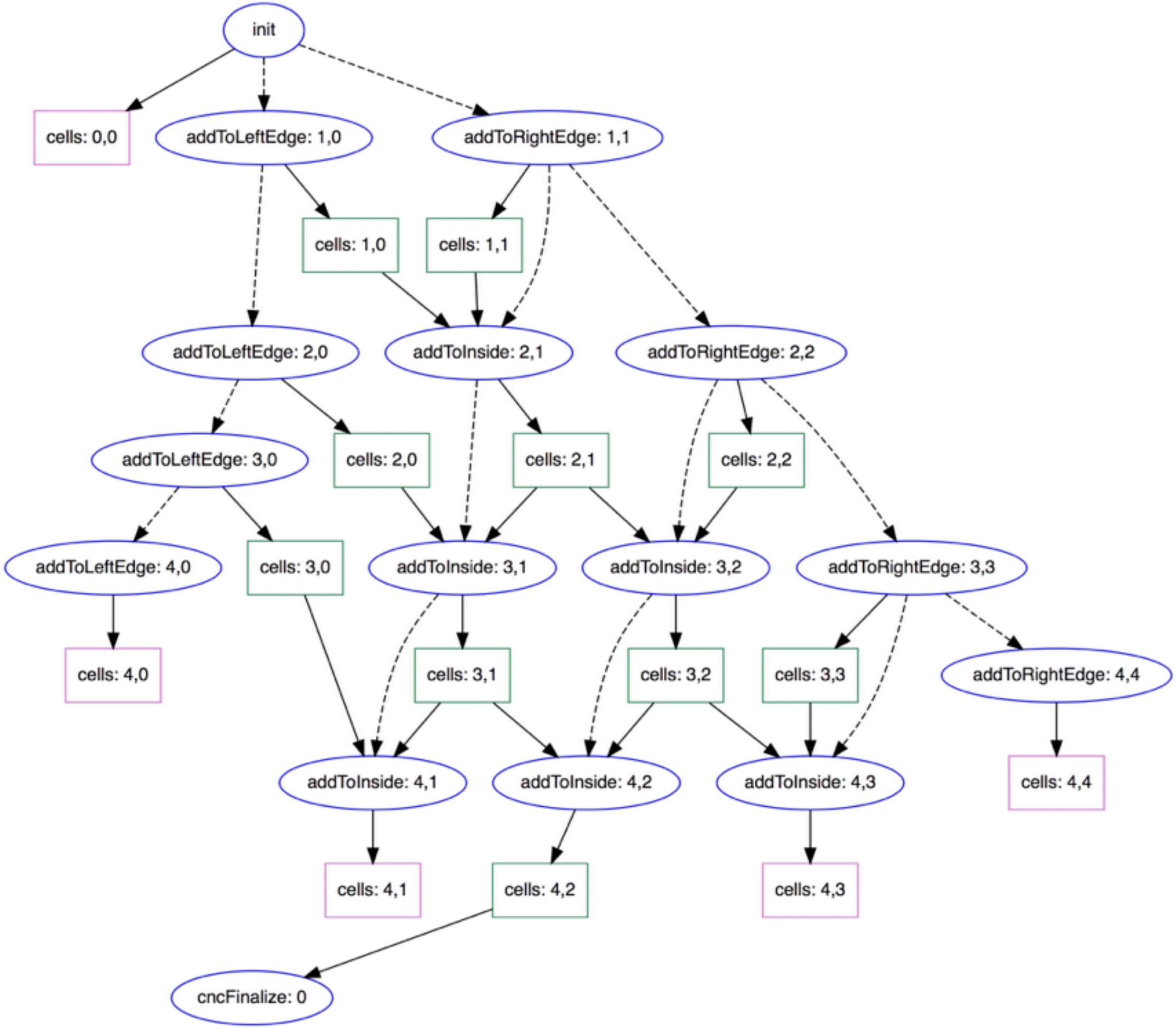
- Pretty pictures
- Making presentation slides

Motivation

- Debugging and understanding programs
- Confirm correctness of structural logic

Fancy example

```
( addToLeftEdge:     row,   col )
 -> [ out @ cells:   row,   col ], ( addToLeftEdge: row+1, col );


( addToRightEdge:     row,   col   )
 -> [ out @ cells:    row,   col   ],
    ( addToInside:    row+1, col   ), ( addToRightEdge: row+1, col+1 );


( addToInside:     row,   col   )
 <- [ a @ cells:   row-1, col-1 ], [ b @ cells:   row-1, col   ]
 -> [ out @ cells: row,   col   ],
    ( addToInside: row+1, col   );
```


init
cells: 0,0
addToLeftEdge: 1,0
addToRightEdge: 1,1
cells: 1,0
cells: 1,1
addToLeftEdge: 2,0
addToInside: 2,1
addToRightEdge: 2,2
addToLeftEdge: 3,0
cells: 2,0
cells: 2,1
cells: 2,2
addToLeftEdge: 4,0
cells: 3,0
addToInside: 3,1
addToInside: 3,2
addToRightEdge: 3,3
cells: 4,0
cells: 3,1
cells: 3,2
cells: 3,3
addToRightEdge: 4,4
addToInside: 4,1
addToInside: 4,2
addToInside: 4,3
cells: 4,4
cells: 4,1
cells: 4,2
cells: 4,3
cncFinalize: 0

Feature overview

- Single HTML file as output
- Highlight possible errors:
  - get without put
  - put without get
  - no path to finalize

Step inverse

- A step definition represents a map from step tags to output collection tags
  - Remark: this map is **one-to-one** if we do not output multiple items into a single collection
- So we can find an inverse.

What it entails

- Given a step with input tag space $X = (x_1, x_2 \ldots x_n)$,
- For each output collection with tags $T = (t_1, t_2, \ldots t_m)$ defined by mappings $t_i = f_i(x_1, x_2, \ldots, x_n)$ where $f_i: X \to \mathbb{Z}$,
  - For each tag x_k of the input tag space,
    - Find the mapping $T \to x_k$ (i.e. solve for x in terms of T)

Recall this one

```
$context { int length; };
[ int data: i ];

( $initialize: () ) -> [ data : 1 ], ( process: 1 );

( process: x )
    <- [ data : x ]
    -> [ data : x + 1 ],
       ( process: x + 1 ) $when(x + 1 < #length);

( $finalize: () ) <- [ data: #length ];
```

Simple Example

```
Step process:
{
'data': [{x: t1 - 1}],
'process': [{x: Piecewise(
        (t1 - 1, t1 < ctxlength), (nan, True))}]
}
```

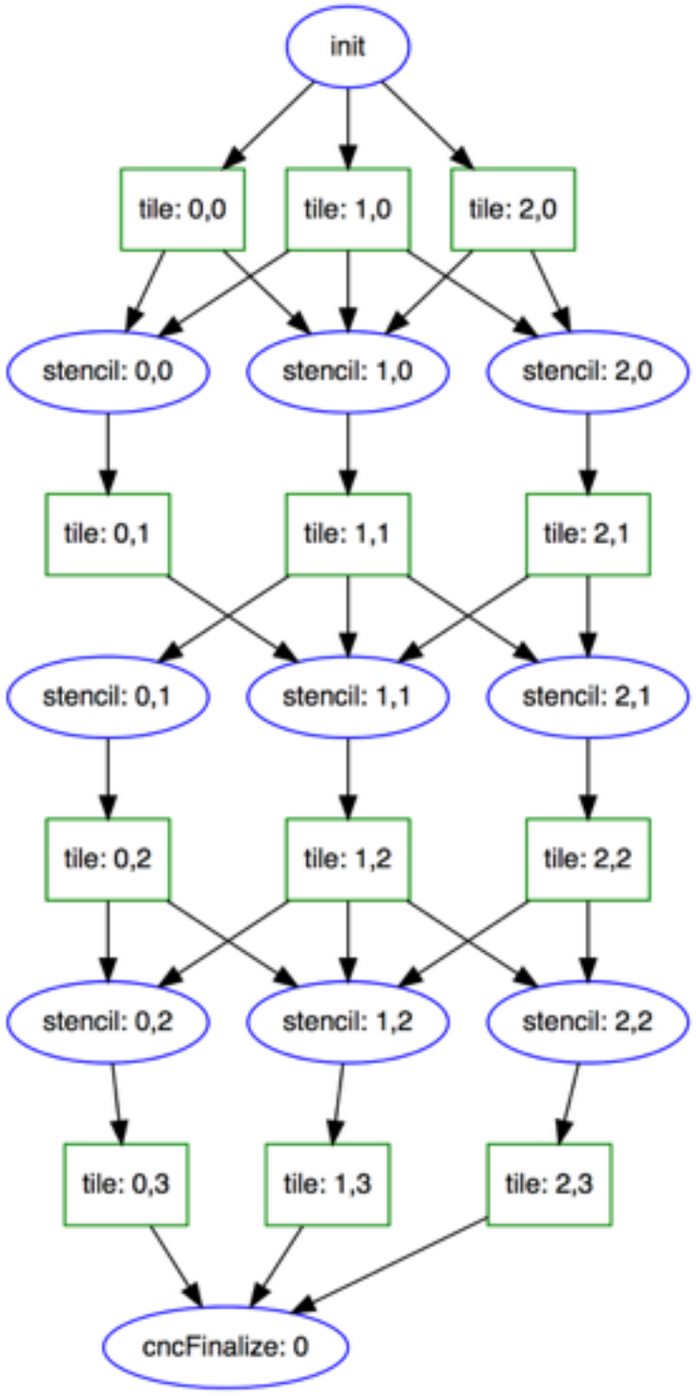
What's it for?

- Simplifies debugging
  - Can blame steps for specific prescribes/puts
  - Auto-blame by combining with event graph from an execution log

Stenciling

```
$context { int lastTile; int numIters; };

// tile at index i, iteration (timestep) t
[ float *tile: i, t ];

( $initialize: () )
 -> [ tile:    $rangeTo(#lastTile), 0 ],
    ( stencil: $rangeTo(#lastTile), 0 );

( stencil: i, t )
 <- [ prevT @ tile:  i, t ],
    [ prevL @ tile:  i-1, t ] $when(i > 0),
    [ prevR @ tile: i+1, t ] $when(i < #lastTile)
 -> [ nextT @ tile:  i,   t+1 ],
    ( stencil: i, t+1 ) $when(t+1 < #numIters);

( $finalize: () )
 <- [ tile: $rangeTo(#lastTile), #numIters ];
```

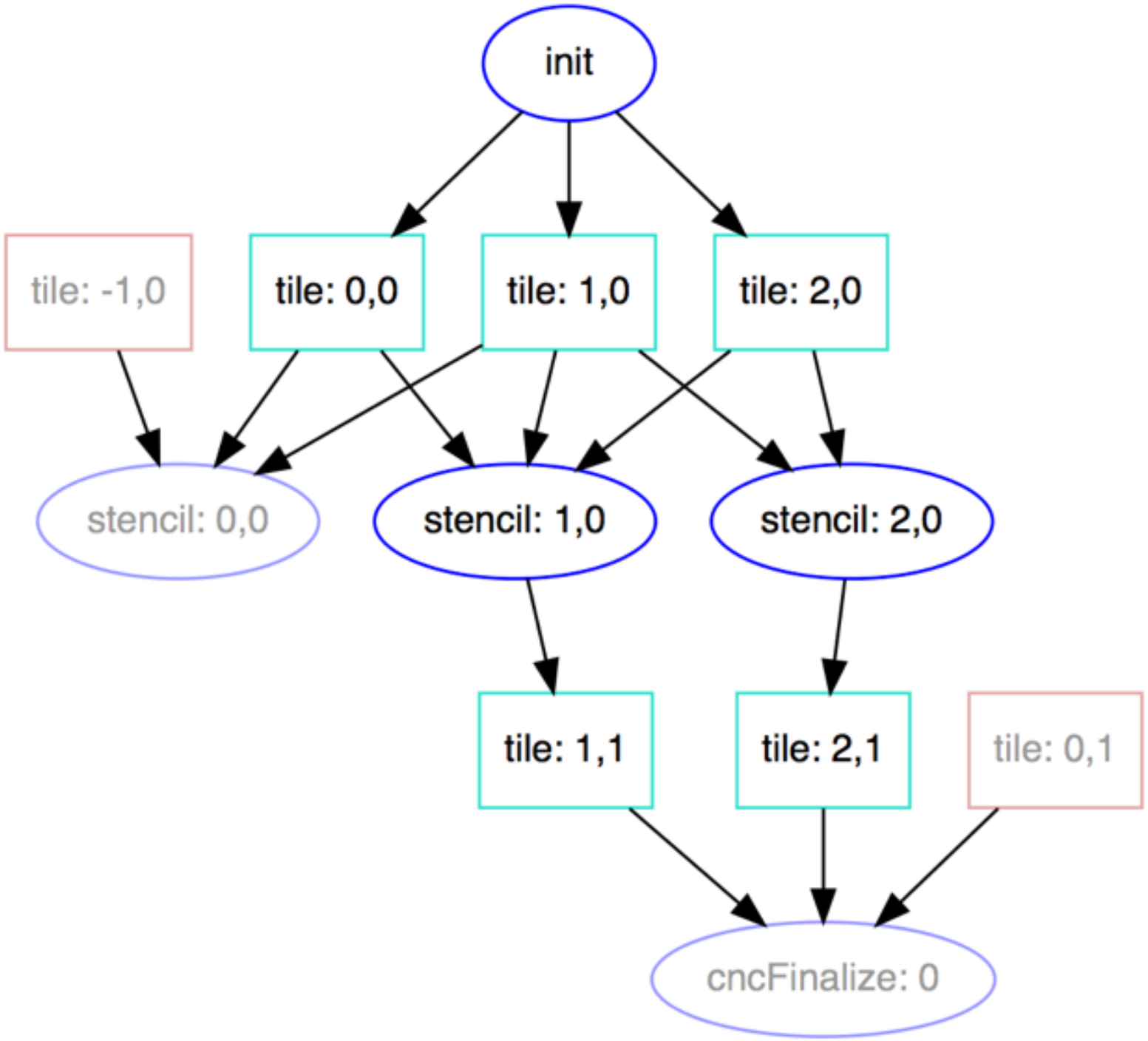
How it looks


init
tile: 0,0
tile: 1,0
tile: 2,0
stencil: 0,0
stencil: 1,0
stencil: 2,0
tile: 0,1
tile: 1,1
tile: 2,1
stencil: 0,1
stencil: 1,1
stencil: 2,1
tile: 0,2
tile: 1,2
tile: 2,2
stencil: 0,2
stencil: 1,2
stencil: 2,2
tile: 0,3
tile: 1,3
tile: 2,3
cncFinalize: 0

Uh-oh

```
( $initialize: () )
 -> [ tile:    $rangeTo(#lastTile), 0 ],
    ( stencil: $rangeTo(#lastTile), 0 );

( stencil: i, t )
 <- [ prevT @ tile:  i, t ],
    [ prevL @ tile:  i-1, t ] $when(i >= 0),
    [ prevR @ tile: i+1, t ] $when(i < #lastTile)
 -> [ nextT @ tile:  i,   t+1 ],
    ( stencil: i, t+1 ) $when(t+1 < #numIters);

( $finalize: () )
 <- [ tile: $rangeTo(#lastTile), #numIters ];
```

Look at the graph



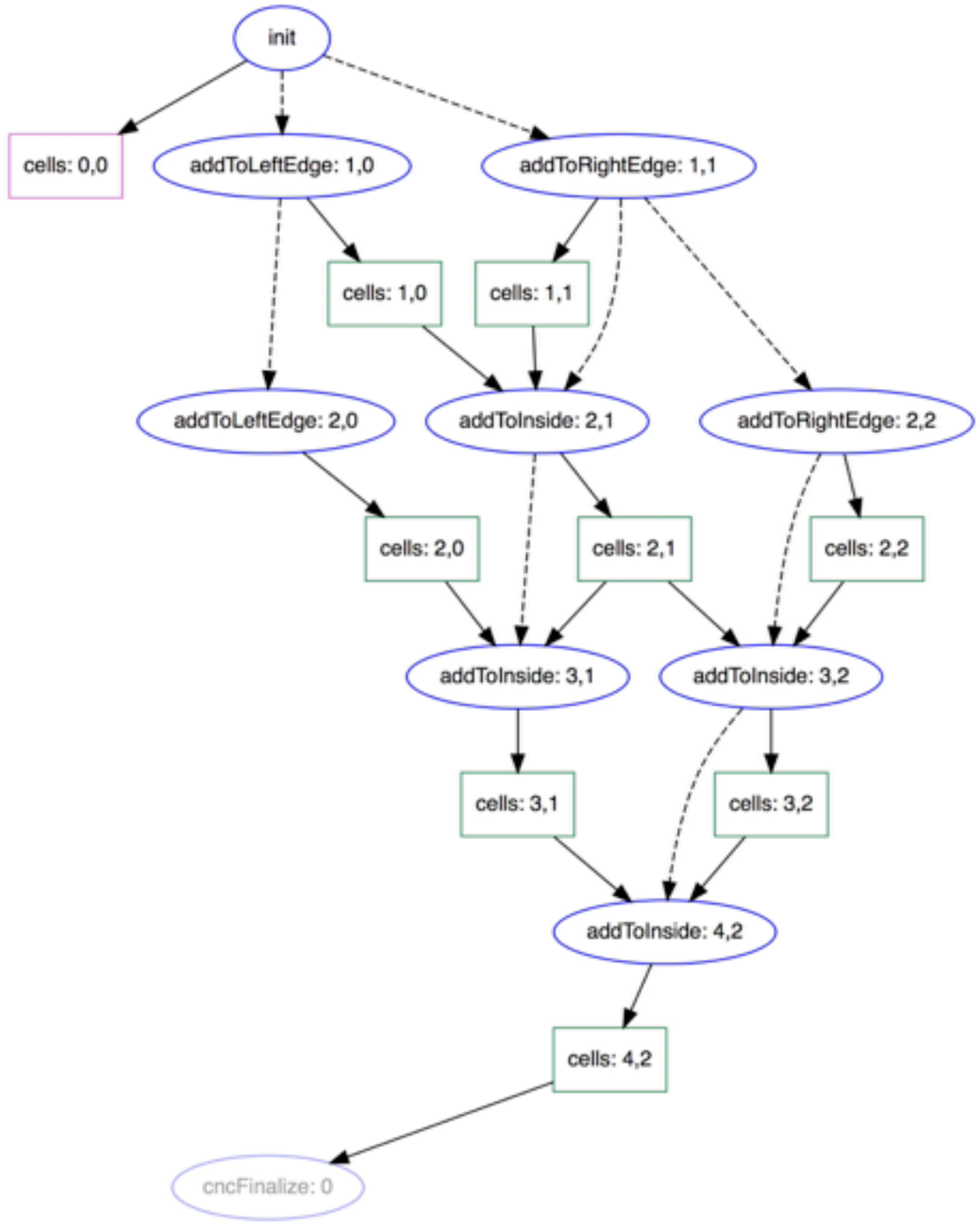
To the rescue

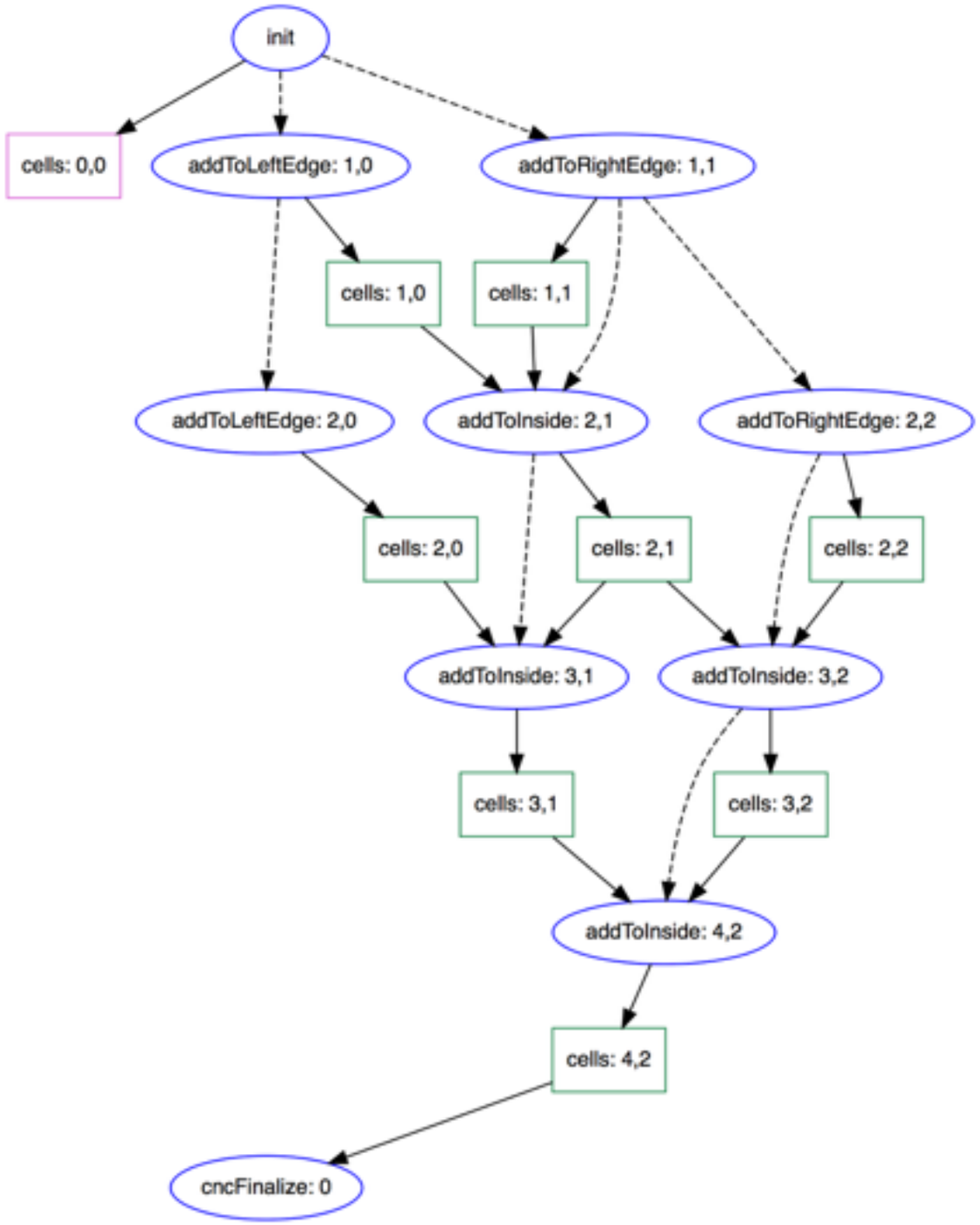
```
Performing blame on these nodes:
['tile@-1, 0’]

Blaming tile@-1, 0:
{'stencil': {'i': -1, 't': -1}}
```

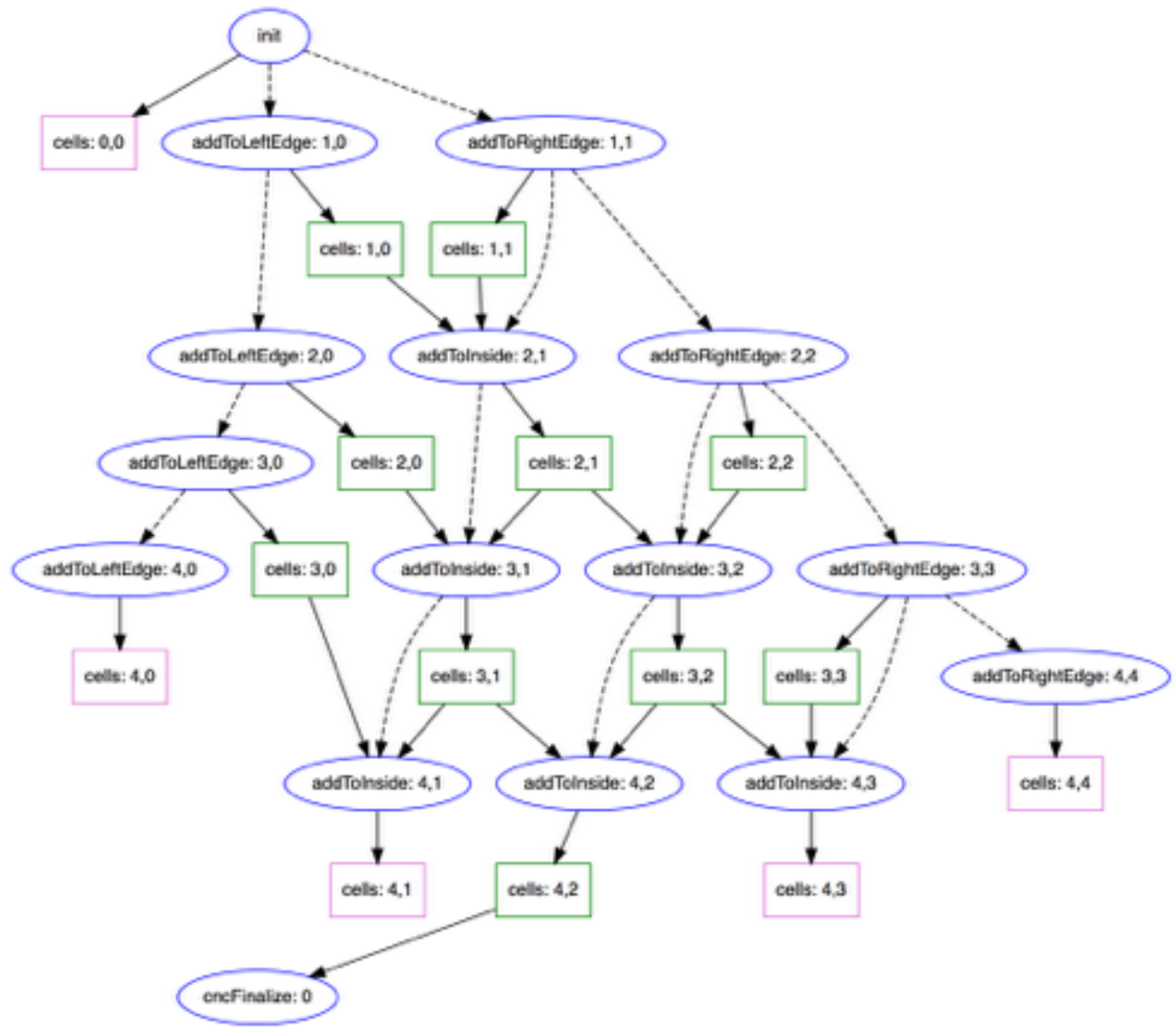
In the future…

- Demand-driven execution:
  - Specify the output, and solve backwards to the initial step
  - All steps visited should be prescribed

What that means




init
cells: 0,0
addToLeftEdge: 1,0
addToRightEdge: 1,1
cells: 1,0
cells: 1,1
addToLeftEdge: 2,0
addToInside: 2,1
addToRightEdge: 2,2
cells: 2,0
cells: 2,1
cells: 2,2
addToInside: 3,1
addToInside: 3,2
cells: 3,1
cells: 3,2
addToInside: 4,2
cells: 4,2
cncFinalize: 0


init
cells: 0,0
addToLeftEdge: 1,0
addToRightEdge: 1,1
cells: 1,0
cells: 1,1
addToLeftEdge: 2,0
addToInside: 2,1
addToRightEdge: 2,2
addToLeftEdge: 3,0
cells: 2,0
cells: 2,1
cells: 2,2
addToLeftEdge: 4,0
cells: 3,0
addToInside: 3,1
addToInside: 3,2
addToRightEdge: 3,3
cells: 4,0
cells: 3,1
cells: 3,2
cells: 3,3
addToRightEdge: 4,4
addToInside: 4,1
addToInside: 4,2
addToInside: 4,3
cells: 4,4
cells: 4,1
cells: 4,2
cells: 4,3
cncFinalize: 0

Wrap up

- Event graphs and step inverses
- Tool assisted debugging
- A peak at the future