Programming soft alife with SPLAT and ulam

#tutorial-HackSPLAT 16:00-17:30 EDT Monday, July 13, 2020 Virtual ALIFE2020 https://livingcomputation.org/edu/alife2020/

> Dave Ackley Living Computation Foundation

THANKS FOR COMING

Let's get started

Note the URL



https://living	computation.org/edu/alife2020/			
	Living Computation - Mozilla Firefox		- • ×	
Living Computation	× +			
\leftrightarrow > C $$	https://www.livingcomputation.org/edu/alife2020 ••••	>>	≡	
Tutorial re	sources			
Scripts and coc	le samples			
A single <u>zi</u>	p file with install information and code snippets.			
Slides				
TO COME				
Video				
TO COME				

– code samples are available there now, and slides and video will be soon

- 0) Organization: What are we doing here?
- Best-effort computing: MFM & ulam
 Space is the place: SPLAT
- 3) Biology and Beyond

- 0) Organization: What are we doing here?
- 1) Best-effort computing: MFM & ulam
- 2) Space is the place: SPLAT
- 3) Biology and Beyond

BIG PICTURE MOTIVATIONS + CONCRETE CODE EXAMPLES

- 0) Organization: What are we doing here?
- 1) Best-effort computing: MFM & ulam
- 2) Space is the place: SPLAT
- 3) Biology and Beyond



No seriously, who ARE you?

- You: Programmer of Deterministic Machines, Restless (at an Alife conference! Or you found this online in the future – what made you click?)
- Me: Recovering Deterministic Machine Programmer, restless
- This: (1) A conceptual reboot with (2) Lots of little code examples

PART O: ORGANIZATION WHO ARE YOU? WHO AM I? WHAT ARE WE DOING HERE?

CELLULAR AUTOMATA

WE'RE TALKING CELLULAR AUTOMATA OF A VERY DIFFERENT KIND – ASYNCHRONOUS, NON DETERMINISTIC, FAILURE-PRONE **PART O: ORGANIZATION**

WHO ARE YOU? WHO AM I? WHAT ARE WE DOING HERE? CHAOTIC GOOD CELLULAR AUTOMATA

PART O: THE GOAL FOR SOCIETY

Develop and deploy useful devices based on indefinitely scalable, besteffort computing

VS RECREATIONAL MATHEMATICS

VS SCIENCE / PHILOSOPHICAL

Focus on robustness in deployable SYSTEMS, rather than elegance in asymptopia

WHERE IS THE ALIFE HERE?

INDEFINITE SCALABILITY IMPLIES ALIFE

- **O)** Organization: What are we doing here?
- 1) Best-effort computing: MFM & ulam
- 2) Space is the place: SPLAT
- 3) Biology and Beyond

PART 1: MFM & ULAM

MFM: Movable Feast Machine computer architecture:

- Indefinitely-scalable (no internal size limits at all)
- Best-effort (no guaranteed hardware determinism) Ulam: An object-oriented procedural programming language for the MFM
- Very limited persistent state (just the CA grid itself)
- Stack and best-effort determinism during one event
- No pointers or general purpose RAM

PART 1: MFM & ULAM

HOW ABOUT AN EXAMPLE

Let's try to do Hello World



Here's a java version



Compile the program..

Run the program..



What could be simpler?

- But from our point of view, almost EVERYTHING about that example is wrong or even evil.
- Let's try to write Hello World in ulam, the first of our two languages for today.



So here's an ulam take on Hello World

Keywords are different – 'element' vs 'class' – and 'Void' is capitalized for some reason, but hey, we're restless deterministic programmers – this is not that terrifying.

We notice there isn't any 'main'.. But it looks like 'behave' is playing that role.



The ulam compiler is called 'ulam'..



But our first problem is it's way not standard.



But there's that zip file on the tutorial website ..

			emacs25@t	2sday - ·
M Filemode	Length	Date	Time	File
	443	 13-7u]-2020	02.48.18	README-ETRST tyt
drwxr-xr-x	0 0	13 Jul - 2020	00.40.10	setun/
	201	12 - Jul - 2020	07:40:50	setup/setup11-repos sh
	1310	13-Jul-2020	00:49:46	setup/README.txt
- rwxr-xr-x	249	13-Jul-2020	00:44:30	setup/setup10-packages-sudo.sh
- rwxr-xr-x	1240	12-Jul-2020	12:15:08	setup/RunMFZ
- rwxr - xr - x	117	13-Jul-2020	00:39:56	setup/setup12-evalthis.sh
drwxr-xr-x	0	12-Jul-2020	13:53:16	eg11/
- r w- rr -	145	12-Jul-2020	11:53:34	eg11/Foo.ulam
drwxr-xr-x	0	12-Jul-2020	14:25:48	eg12/
- rw-rr	212	12-Jul-2020	14:04:42	eg12/Bar.java
- rw-rr	184	12-Jul-2020	14:25:30	eg12/Bar.ulam
drwxr-xr-x	0	12-Jul-2020	13:49:18	eg10/
- rw-rr	108	12-Jul-2020	05:59:28	eg10/Foo.java
- rw-rr	97	12-Jul-2020	12:13:56	eg10/Foo.ulam
drwxr-xr-x	0	12-Jul-2020	15:17:10	eg15/
- rw-rr	401	12-Jul-2020	15:16:34	eg15/Bar.ulam
drwxr-xr-x	0	12-Jul-2020	14:29:20	eg14/
- rw-rr	194	12-Jul-2020	14:29:18	eg14/Bar.ulam
=:%%- ALIFE2020-	tutorial-f	files.zip Top	L3 (7in	-Archive Narrow)
Parsing archive	file	done.	(20)	

with READMEs and scripts to help set things up on a linux box –



At least on Ubuntu 18.04. But it hasn't got anything that weird in it. It's just packages and code

We can handle this



And it compiles.

So now what?

By default, ulam makes an 'a.mfz' file,



and the zip file comes with this weird RunMFZ script we can try:



Which generates a TON of output.

Where the heck is our 'Hello World'?



There it is.

Actually there it is over and over..

emacs@t2sday	
element Foo { Void behave () { ByteStreamLogger log; log.printf("Hello World\n"); } } }	
-: <mark>Foo.ulam<eg10></eg10></mark> All L4 (ulam Abbrev)	
20200713045153-3164 : 95AEPS [695D4347]MSG: s[17,25] t[1,0]: Hello World	
20200713045153-3165 : 95AEPS [695D4347]MSG: s[17,25] t[1,0]: Hello World	
20200713045153-3166 : 96AEPS [695D4347]MSG: s[17,25] t[1,0]: Hello World	
20200713045153-3167 : 98AEPS [695D4347]MSG: s[17,25] t[1,0]: Hello World	
20200713045153-3168 : 99AEPS [695D4347]MSG: s[17,25] t[1,0]: Hello World	
20200713045153-3169: 99AEPS [695D4347]MSG: s[17,25] t[1,0]: Hello World	
20200713045153-3170: 100AEPS [6965FE52]MSG: Saving to: /tmp/20200713045150/save/fina	
l-12-3100.mfs	
20200713045153-3171: 100AEPS [6965FE52]MSG: Sending exit requests to the tiles	
U:%*- <mark>*compilation*</mark> 96% L198 (Compilation sector [0])	
Compile command:/setup/RunMFZ a.mfz Fo -gui	

- If we stick '-gui' at the end of the RunMFZ command it'll bring up the mfms simulator interface (which is really the main way mfms is used.)
- This whole RunMFZ thing is kind of a hack to try to make ulam code look more the a 'normal' program.
- But it's really not. Ulam code is about writing spatiotemporally local state transitions, like 'any' cellular automata



And here's the mfms GUI

lt's... quirky.

The grey rectangles represent TILES, units of computation that can be connected together to form bigger and bigger machines.



Just for comparison, here's sixteen of the 'T2 Tiles' in real life.

(Officially speaking that red one there isn't special – a grid of T2 tiles has no 'head node' or 'master unit' or anything like that.

But the red one is the one I've been hacking on, which is useful to know.)



And here's that same grid running a simple program, from a recent timelapse video segment.



So the four tiles in this simulation are all empty – except for that white dot

Here we've typed 't' to bring up the Tool palette.



If we click left on the 'Atom View tool' (second from the right in the top row), we can then click left on the white dot..)



And see that it is indeed an atom of Foo, the 'element' that we wrote in our Foo.ulam class.



It's hard to see, but here we've typed the 'l' command, which bring up the logging window – that will show the same output we saw in the compilation window when we did RunMFZ without the '-gui' argument.



And if we click the 'Run' button, now we see our Hello World output.

Over and over and over, in the log window.

Let's fix that.

EG11 HELLO? WORLD



When a behave() method is called, the atom that it is called upon – called 'self' – is located at index 0 of a class called 'EventWindow'.

If our behave() method overwrites ew[0], self is modified or completely changed.

Doctor it hurts when I access self after rewriting ew[0].
EG11 HELLO? WORLD



So note we modify ew[0] at the very end of behave()!

Let's try it.

EG11 HELLO? WORLD

emacs@t2sday
//EG11
element Foo {
Void behave() {
Logger logger;
logger.log("Hello World");
Eventwindow ew;
ew[0] = Emply.instanceor;
-: Foo.ulam <eg11> All L7 (Ulam ADDrev)</eg11>
Mode: compliation; default-directory: /data/ackiey/PAR14/papers/ALIFE20/tutoriation
Compilation started at Sun Jul 12 13:53:16
ulam Foo.ulam:/setup/RunMFZ a.mfz Fo
SIGNED BY RECOGNIZED HANDLE: Dave Ackley (ee94-193-1cd4)
Skipping ulam source: Foo.ulam
Skipping ulam source: Empty.ulam
Skipping ulam source: C2D.ulam
Skipping ulam source: Random.ulam
U:%*- *compilation* Top L1 (Compilation:exds: [0])
Compilation finished

It builds okay, and runs.

EG11 HELLO? WORLD

emacs@t2sday
//EG11 element Foo {
Void behave() {
Logger logger;
logger.log("Hello World");
EventWindow ew;
}
-: <mark>Foo.ulam<eg11></eg11></mark> All L7 (ulam Abbrev)
20200712135323-239: 0AEPS [FD2FFBF2]MSG: Saving to: /tmp/20200712135323/autosave/10-
10.mfs
20200712135323-240: 0AEPS [FD274197]MSG: s[17,25] t[1,0]: Hello World
20200712135323-241: 0AEPS [FD2FFBF2]MSG: Saving to: /tmp/20200712135323/save/final-1
1-10.mts
20200/12135323-242: OAEPS [FD2FFBF2]MSG: Sending exit requests to the tiles
20200/12135324-243: 0AEPS [FD2FFBF2]MSG: Simulation driver exiting
Compilation finished at Sun Jul 12 13:53:24
U:%*- <mark>*compilation*</mark> Bot L66 (Compilation:exit [0])

And there's our single instance of 'Hello World'

EG11 GOODBYE WORLD

emacs@t2sday
//EG11
element Foo {
Void behave() {
Logger logger;
logger.log("Hello World");
EVENTWINDOW EW;
ew[0] = Empty.thstanceor;
-: FOO.ULAM <egii> ALLE/ (ULAM ADDIEV)</egii>
20200/12135323-239: 0AEPS [FD2FFBF2]MSG: Saving to: /tmp/20200/12135323/autosave/10-1
10,115 20200712125222 240, 04505 [50274107] MCC, $a[47, 25] \pm [4, 0], 10, 10, 10, 10, 10, 10, 10, 10, 10, 10$
20200712135323-240: 0AEPS [FD274197]MSG: S[17,25] t[1,0]: Hello World
20200/12135323-241: 0AEPS [FD2FFBF2]MSG: Saving to: /tmp/20200/12135323/save/final-1*
20200/12135323-242: 0AEPS [FD2FFBF2]MSG: Sending exit requests to the tiles
20200/12135324-243: 0AEPS [FD2FFBF2]MSG: Simulation driver exiting
Constitution (Collaboration Pull 42, 42, 52, 24
compliation rinished at Sun Jul 12 13:53:24
U:%*- *compilation* Bot L67 (Compilation:exit [0])
Mark set

Half-buried in log detritus, but there.



Let's try one more stereotypical first program. Let's add two and two.

Here's a java version. We use three int data members because, you'll see.



And it computes 4, the right answer



Compare the java code on top to the ulam code on the bottom

Similar, but different. Nothing too shocking.



Let's compile Bar and use RunMFZ a 'Ba' element



Uh oh

What did we do wrong?

emacs@t2sday
//EG12 element Bar {
Int x,y,z;
Void func () { x = 2; y = 2; z = x + y; }
ByteStreamLogger log;
<pre>func(); log_printf("Bac: %d\p"_z).</pre>
}
}
-: Bar.ulam <eg12> All L8 (ulam Abbrev)</eg12>
/files/eq12/" -*-
Compilation started at Sun Jul 12 14:22:21
ulam Bar.ulam:/setup/RunMF7 a.mfz Ba
Compile Bar.ulam, Ulam5StdlibControl.ulam, Res.ulam, Wall.ulam, DReg.ulam: ERROR: Cl
ose failed: 512
/data/ackley/PART4/papers/ALIFE20/tutorial/code/ULAM/bin//share/ulam/stdlib/ByteStr
eamString.ulam:35:1: ERROR: Trying to exceed allotted bit size (71) for element Bar w
ith 96 bits.
U:%^- *Compliation* Top L6 (Compliation:exit [1])

Exceed allotted bit size 71??

emacs@t2sday - • • •
//EG12
element Bar {
Int x,y,z;
Void func () { x = 2; y = 2; z = x + y; }
Void behave() {
ByteStreamLogger log;
log.printr("Bar: %d\n",Z);
-: Bar.ulam <eg12> All L8 (ulam Abbrev)</eg12>
/data/ackley/PARI4/papers/ALIFE20/tutorlal/code/ULAN/ptn//snare/uLaM/stdltD/Bytestr
eamString.ulam:35:1: ERROR: Trying to exceed allotted bit size (71) for element Bar wa
/Bac ulam:2:9: NOTE: Components of Bac are
/Bar ulam:3:3: NOTE: (32 of 96 bits at 0) Tot(32) x
/Bar.ulam:3:3: NOTE: (32 of 96 bits, at 32) Int(32) v.
/Bar.ulam:3:3: NOTE: (32 of 96 bits, at 64) Int(32) z.
<pre>/Bar.ulam:2:9: ERROR: CLASS (regular) 'Bar' SIZED 96 FAILED.</pre>
Unrecoverable Program Type Label FAILURE.
U:%*- *compilation* 28% L7 (Compilation:exit [1])
Mark set

Yeah. 71 bits.

Three 32 bit Ints takes 96 bits. An ulam object – an Atom – isn't that big.

emacs@t2sday	
//EG12	
element Bar {	
Int(3)_x,y,z;	
Void func () { x = 2; y = 2; z = x + y; }	
Void behave () {	
ByteStreamLogger log;	
<pre>func();</pre>	
log.printf("Bar: %d\n",z);	
_ }	
<u>}</u>	
-: Bar.ulam <eg12> All L3 (ulam Abbrev)</eg12>	
/data/ackley/PART4/papers/ALIFE20/tutorial/code/ULAM/bin//share/ulam/stdlib/ByteStr	
eamString.ulam:35:1: ERROR: Trying to exceed allotted bit size (71) for element Bar w	
ith 96 bits.	
<u>JBar.ulam:2:9:</u> NOTE: Components of Bar are	
<u>./Bar.ulam:3:3:</u> NOTE: (32 of 96 bits, at 0) Int(32) x.	
<u>./Bar.ulam:3:3:</u> NOTE: (32 of 96 bits, at 32) Int(32) y.	
./Bar.ulam:3:3: NOTE: (32 of 96 bits, at 64) Int(32) z.	
./Bar.ulam:2:9: ERROR: CLASS (regular) 'Bar' SIZED 96 FAILED.	
Unrecoverable Program Type Label FAILURE.	
U:%*- *compilation* 28% L/ (Compilation.exit [1])	
Wrote /data/acktey/PART4/papers/ALIFE20/tutortat/rttes/egiz/Bar.utam	

But ulam supports variable sized primitive types like ints.

Let's make them 3 bit ints. Three Int(3)'s will only burn 9 bits.



Now what is the compiler mad about??

Well, if you add two Int(3)s, the result could be an Int(4). But z is declared as an Int(3); it might not fit.



So we need to cast the result

Getting careful about primitive sizes and the results of primitive operations is part of the ulam way of life. We embrace it.

emacs@t2sday
//EG12
element Bar {
Int(3) x,y,z;
Void func() { $x = 2; y = 2; z = (Int(3)) (x + y);$ }
Void behave() {
ByteStreamLogger log;
func();
log.printf("Bar: %d\n",Z);
-: Bar.ulam <eg12> All L4 (ulam Abbrev)</eg12>
Screen resize: 1920 x 1052
20200/12142555-241: 0AEPS [41B6B9B2]MSG: Loaded configuration '/data/ackley/PART4/pa
pers/ALIFE20/tutorial/code/MFM/res/mts/start-5.0.5.mts
20200712142555-242: 0AEPS [41B6B9B2]MSG: Saving to: /tmp/20200712142555/autosave/10-
10.mts
20200712142556-243 : 3AEPS [41ADFE77]MSG: s[17,25] t[1,0]: Bar: 3
20200712142556-244 : 5AEPS [41ADFE77]MSG: s[17,25] t[1,0]: Bar: <u>3</u>
20200712142556-245 : 6AEPS [41ADFE77]MSG: s[17,25] t[1,0]: Bar: <u>3</u>
20200712142556-246: 6AEPS [41ADFE77]MSG: s[17,25] t[1,0]: Bar: 3
U:%*- *compilation* 47% L72 (Compilation:exit [0] Isearch)
I-search: Bar:

And then it compiles and runs.

And we see (once again over and over), that 2+2 = 3. Wait what?

The maximum value of an Int(3) is 3. Four doesn't fit.

- But wouldn't we expect the answer to be like negative something, then, due to overflow?
- Except ulam arithmetic doesn't overflow, it saturates. Because that's almost always less wrong.

EG13 2+2

emacs@t2sday //EG13 element Bar { Int(4) x,y,z; Void **func()** { x = 2; y = 2; z = (Int(4)) (x + y); } Void behave() { ByteStreamLogger log; func(); log.printf("Bar: %d\n",z); } All L4 (ulam Compiling Abbrev) Bar.ulam<eo13> Screen resize: 1920 x 1052 20200712142758-241: 0AEPS [B133FE52]MSG: Loaded configuration '/data/ackley/PART4/pa pers/ALIFE20/tutorial/code/MFM/res/mfs/start-5.0.5.mfs' 20200712142758-242: 0AEPS [B133FE52]MSG: Saving to: /tmp/20200712142758/autosave/10-10.mfs 20200712142758-243: 2AEPS [B12B4327]MSG: s[17,25] t[1,0]: Bar: 4 **20200712142758-244**: 2AEPS [B12B4327]MSG: s[17,25] t[1,0]: Bar: 4 **20200712142758-245**: 3AEPS [B12B4327]MSG: s[17,25] t[1,0]: Bar: 4 20200712142758-246: 3AEPS [B12B4327]MSG: s[17,25] t[1,0]: Bar: 4 (Compilation:exit [0]) U:%*- *compilation* 46% L72 Mark set

We could use Int(4), which goes from -8 to +7

EG14 2+2

emacs@t2sday '/EG14 element Bar { Unsigned(3) x,y,z; Void func() { x = 2; y = 2; z = (Unsigned(3)) (x + y); } Void behave() { ByteStreamLogger log; func(); log.printf("Bar: %d\n",z); } All L4 (ulam Compiling Abbrev) Bar.ulam<eo14> Screen resize: 1920 x 1052 20200712142927-241: 0AEPS [BAB12102]MSG: Loaded configuration '/data/ackley/PART4/pa pers/ALIFE20/tutorial/code/MFM/res/mfs/start-5.0.5.mfs' 20200712142927-242: 0AEPS [BAB12102]MSG: Saving to: /tmp/20200712142927/autosave/10-10.mfs 20200712142927-243: 2AEPS [BAA865C7]MSG: s[17,25] t[1,0]: Bar: 4 20200712142927-244: 3AEPS [BAA865C7]MSG: s[17,25] t[1,0]: Bar: 4 **20200712142927-245**: 3AEPS [BAA865C7]MSG: s[17,25] t[1,0]: Bar: 4 20200712142927-246: 4AEPS [BAA865C7]MSG: s[17,25] t[1,0]: Bar: 4 U:%*- *compilation* n* 48% L72 (Compilation:exit [0]) Mark set

And if we like we could use Unsigned(3), which covers 0..7

EG15 2+2 CLEANED UP



We could clean it up with a typedef

Typedefs are good.

And, here, we politely erase ourselves to simulate an old-fashioned terminating program

EG15 2+2 CLEANED UP	
emacs@t2sday	×
<pre>//EG15 /** Bar - a demo element symbol B color #a1fe20 symmetries all date Jul 2020 author Dave Ackley llicense public domain */ element Bar { typedef Unsigned(3) MyInt; MyInt x,y,z; Void func() { x = 2; y = 2; z = (MyInt) (x + y); } Void behave() { ByteStreamLogger log; func(); log.printf("Bar: %d\n",z); EventWindow ew; ew[0] = Empty.instanceof; } }</pre>	
U: <mark>Bar.ulam<eg15></eg15></mark> All L4 (ulam Abbrev)	

We can also add various kinds of metadata about our element

Those are backslashes in the structured comment, even though the italicizing almost stands them upright.

EG15 2+2 CLEANED UP	
emacs@t2sday	
<pre>//EG15 /** Bar - a demo element</pre>	
U: Bar.ulam <eg15> All L4 (ulam Abbrev) Compile command: ulam Bar.ulam;/setup/RunMFZ a.mfz B</eg15>	

Now that we declared Bar's atomic symbol is 'B' – not 'Ba' – that's what goes in the RunMFZ command

<pre>emacs@t2sday //EG15 /** Bar - a demo element \symbol B \color #a1fe20 \symmetries all \date Jul 2020 \author Dave Ackley \license public domain */ element Bar {</pre>
<pre>//EG15 /** Bar - a demo element symbol B color #a1fe20 symmetries all date Jul 2020 author Dave Ackley license public domain */ element Bar {</pre>
element Bar {
U: Bar.ulam <eg15> Top L3 (ulam Abbrev)</eg15>
Screen resize: 1920 x 1052
20200712151718-241: 0AEPS [E3A10462]MSG: Loaded configuration '/data/ackley/PART4/pa
20200712151718-242: 0AEPS [E3A10462]MSG: Saving to: /tmp/20200712151718/autosave/10- 10.mfs
20200712151719-243: 1AEPS [E39848F7]MSG: s[17.25] t[1.0]: Bar: 4
20200712151719-244: 1AEPS [E3A10462]MSG: Saving to: /tmp/20200712151718/save/final-1
1-11.mfs
20200712151719-245: 1AEPS [E3A10462]MSG: Sending exit requests to the tiles
20200712151710 246. 1AEDC [E2A10462]MSC. Cimulation driver evities
Mark set

And it all still works.

ULAM TAKE-AWAYS PART I

- Atoms ('objects') are instances of Elements ('classes')
 - An atom has only 71 bits of state
 - All atoms are the same size
- State transition code is per-element, not per atom
 - Void behave() is the entry point for a state transition
 - The behave()-ing atom is stored in the 'event window', at ew[0]
- The initial configuration is (generally) determined externally
 - Do not try and call the main. That's impossible. Instead, ...

EG16 SwapLine



This is the primordial SwapLine

- I'll be giving a lightning talk about some great grandchildren of SwapLine on Friday.
- But for now it's just about the EventWindow. What's ew[5]?

THE EVENT WIN	NDOW		
	New Tab - Mozilla Firefox	- ¤ ×	
🍅 New Tab	× +		
\leftrightarrow > C $$	Q robust.cs.unm.edu	→ ≫ ≡	
	Http:// robust.cs.unm.edu / — Visit	*	
	G robust.cs.unm.edu	_	
	This time, search with: G 🚯 W 🔅	¥	
	G Search the Web →		

I always go here

THE EVENT WINDOW				
start [Robust-fir	rst Computing Wiki] - Mozilla F	Firefox		- • ×
₿ start [Robust-first Computing × +				
← → C û 🛛 🖉 robust.c	s. unm.edu /doku.php	60% •••	>>	≡
		i	Login	
Robust-first Computing Wiki			Q	
Trace: • start		Table of Contents	start	
Trace: - start Quick links	38	Table of Contents + Quick links + News	start	
Trace: - start	$\begin{array}{c c c c c c c c c c c c c c c c c c c $	Table of Contents • Quick links • News • Components • Development	start	

To find the event window image



Showing the various indexing schemes that ulam code can use to examine its environment



ew[0] is always 'me'.

So ew[5] is like 'Northwest of me'

Similarly for ew[6]

EG16 SwapLine



- This code also uses capital Self, a predefined type meaning 'my own type' – so 'if (ew[5] is Self)' means if nw is also a SwapLine like I am
- Let's stop using RunMFZ and be 'official' by using mfzrun. We can specify an .mfz output file on the ulam command if we don't want to end up with 'a.mfz'. So let's use SwapLine.mfz, and run that..



Where's the SwapLine?

The single atom we were seeing before came from RunMFZ. Normally the world starts out empty, as here.



But we bring up the tool palette with 't'

And click left on 'SL' to select it

And use the pencil tool to create an SL in the grid.



And if we click Run on and off..



We see it heading east.

Why? Because its NW and SW are always empty, so it does the swap with east.



Until eventually



It runs out of universe

Note that ew.swap refuses to swap off the end of the universe! Sometimes that's what you want; sometimes not.



But suppose we clear the grid and then pencil in a whole line



Now the whole line heads east


But it's slope is never more than 45 degrees



Note how it tends to fall behind in the black areas

Why? Because those are 'intertile zones', where performing events requires coordination between multiple tiles.



And that slows things down in those areas.

And see what 'asynchronous cellular automata' really means! It's not just a random permutation, or uniform random sampling. There can be HIGH-ORDER spatial biases



Eventually the SwapLine starts to reach the end of the universe



And in this version, stops there.



- But note all those behave() methods are still being called! There is no actual halt or exit here!
- If another tile got connected in this black area here, for example...

ULAM/MFM DISCUSSION

- Isn't this fantastically inefficient? The 'program' keeps restarting!
 That is a principal source of its robustness.
- How can you write anything serious with only 71 bits of state?
 - View an atom less as a program and more as an ALU or 'mobile FPGA'
 - Many atoms will coordinate to perform complex and stateful tasks
- But how can atoms coordinate without a synchronous architecture?
 As demonstrated by the SwapLine, atoms can synchronize as needed.
- It's just too hard! Where's my IDE? Where's my autocomplete?
 - Indeed, living on the frontier is not for everybody.

OK, let's wrap up this section

Some Q&A-ish thoughts.



Big picture thought

Traditional CPU/RAM computing is all about complete flexibility about WHAT INSTRUCTION COMES NEXT.

Ulam has that too, but a lot less – just for one event. And even within an event, a single call on behave, there's a typical sequence of phases.



And underneath the hood, down in the MFM architecture, this is the event processing loop.

A sequence of steps.

Of course that's true of all sequential computer hardware – at some level it's got to be a looping sequence of steps, or else how is it implemented.

MFM and ulam carry that phasedcomputation-style up the stack.

OUTLINE

- **O)** Organization: What are we doing here?
- 1) Best-effort computing: MFM & ulam
- 2) Space is the place: SPLAT
- 3) Biology and Beyond

EG17 SwapLine Revisited emacs@t2sday = element SwapLine == Rules given @ isa SwapLine vote s isa SwapLine # Back man blocks advance s @ -> . s . # Otherwise advance @x -> x@ -:--- SwapLine.splat All L1 (Fundamental) Compile command: splattr SwapLine.splat SL.mfz; mfzrun SL.mfz

Spatial rules

Structure of a rule.

LHS.. check.. RHS..

EG17 SwapLine Revisited emacs@t2sday = element SwapLine == Rules given @ isa SwapLine vote s isa SwapLine # Back man blocks advance S @ -> . **ine.splat** Top L1 (Fundamental Compiling) -*- mode: compilation; default-directory: "/data/ackley/PART4/papers/ALIFE2<u>0/tutorial</u> /files/eg17/" -*-Compilation started at Mon Jul 13 13:28:31 splattr SwapLine.splat SL.mfz; mfzrun SL.mfz splattr: The SPLAT Language Translator Copyright (C) 2018 Dave Ackley Released under GPL3 U:%*- ***compilation*** Top L1 (Compilation:run Compiling)

It builds







But this one runs right off the edge of the universe







Easy to focus on the 2D rules, but...

OUTLINE

- **O)** Organization: What are we doing here?
- 1) Best-effort computing: MFM & ulam
- 2) Space is the place: SPLAT
- 3) Biology and Beyond

Discuss



CONCLUSIONS

THE PAST

• Traditional computer architecture: Good for safe, small tasks THE PRESENT

Computer security failures are the symptom

• Hardware determinism is the disease

THE FUTURE

• Living computation on robust-first architecture

THE LIVING COMPUTATION FOUNDATION

...advances a **unified view** of life and computing. By supporting science and engineering to explore that connection, and education and outreach to convey it, we seek to improve individual liberty and equal opportunity for the citizens of our emerging information technological societies.

https://livingcomputation.org/support.html

Brought to you by LCFN220