

Constraint Logic Propagation Conflict Spreadsheets

William Taysom

2015

...an unruly name for an unruly topic.

inspiration.clp - /Users/wtaysom/Desktop/working-clp - Atom

inspiration.clp • example.clp • review.clp

1 Inspiration
2

Commands and Keystrokes:

editor:newline

Alt+N S P I R A T I O N

inspiration.clp* 2:1 UTF-8 Constraint Logic Prop...

Good afternoon. Today is the future. Let me share a programming concept inspired by the confluence of three things:

Jason Eisner's Dyna programming language, a sort of Prolog with aggregation operators,

inspiration.clpcts - /Users/wtaysom/Desktop/working-clpcts - Atom

inspiration.clpcts example.clpcts review.clpcts

1 Inspiration
2 Jason Eisner Dyna Prolog with Aggregation
3 Alexey Radul Propagation Networks Information Cells
4

Commands and Keystrokes:

editor:newline ↵
editor:indent ⌘I

editor:indent ⌘I
editor:indent ⌘P

inspiration.clpcts* 4:2 UTF-8 Constraint Logic Prop...

The screenshot shows a Mac OS X desktop environment with a dark theme. An Atom code editor window is open, displaying three tabs: 'inspiration.clpcts', 'example.clpcts', and 'review.clpcts'. The 'inspiration.clpcts' tab is active. The code in the editor is as follows:

```
1 Inspiration
2 Jason Eisner Dyna Prolog with Aggregation
3 Alexey Radul Propagation Networks Information Cells
4
```

A floating command palette titled 'Commands and Keystrokes' is visible, listing keyboard shortcuts for 'editor:newline' (represented by a return arrow icon) and 'editor:indent' (represented by a right arrow icon). Below the palette, the status bar shows the file name 'inspiration.clpcts*', line number '4:2', encoding 'UTF-8', and file type 'Constraint Logic Prop...'. The status bar also includes icons for file operations like save and close.

Alexey Radul's Propagation Networks in which reactive cells store information about values rather than just values,

A screenshot of the Atom code editor interface. At the top, there are three tabs: "inspiration.clpcts", "example.clpcts", and "review.clpcts". The "inspiration.clpcts" tab is active. Below the tabs, the main editor area displays the following text:

```
1 Inspiration
2 Jason Eisner Dyna          Prolog with Aggregation
3 Alexey Radul Propagation Networks Information Cells
4 Analysts      Excel          Beyond SUMPRODUCT Pivot Tables
```

At the bottom of the editor area, there is a "Commands and Keystrokes" palette with two sections:

- "editor:indent" with keybindings: ⌘B, E, Y, O, N, D, ⌘S, ⌘U, ⌘M, ⌘P, ⌘I, ⌘E, X, C, E, L.
- "editor:outdent" with keybinding: ⌘I.

The status bar at the bottom shows "inspiration.clpcts* 4:66" on the left and "UTF-8 Constraint Logic Prop..." on the right.

and, most especially, work with analysts using Excel in realtime during multibillion dollar spectrum auctions. Imagine a secure, windowless room in a national capital, locked in with consultants, economists, analysts. No external devices, no internet access except for the auction software. The job is to help them make sense of what they're seeing, to live code beyond the SUMPRODUCTs and Pivot Tables at which Excel excels.

A screenshot of the Atom code editor interface. The title bar shows three tabs: "inspiration.clpcts", "example.clpcts", and "review.clpcts". The "inspiration.clpcts" tab is active. The main content area displays a list of items:

- 1 Inspiration
- 2 Jason Eisner Dyna Prolog with Aggregation
- 3 Alexey Radul Propagation Networks Information Cells
- 4 Analysts Excel Beyond SUMPRODUCT Pivot Tables
- 5 Awe Dismay Confounding

Below this list, a "Commands and Keystrokes" section is visible, containing a grid of keybinding cards:

editor:indent	↑C	O	N	F	O	U	N	D	A	T	I	O	N
	→I												
editor:indent	↑D	I	S	M	A	Y							
	→I												

The status bar at the bottom shows "inspiration.clpcts* 5:27" and "UTF-8 Constraint Logic Prop...".

On these excursions, I'm awed by how efficiently analysts munge their data, dismayed by the hacks they go through to maintain their spreadsheets, and confounded that despite all my programmer prowess, they come up with their numbers faster. All the same, after a week, I have scripts to tabulate interesting figures, and I spend most of the next month stuck in the room ruminating over what it all means. This idea is the outcome of my most recent engagement: what can we do to enhance the data-centric reactivity of spreadsheets?

A screenshot of the Atom code editor interface. The title bar shows three tabs: 'inspiration.clpcts' (active), 'example.clpcts', and 'review.clpcts'. The main editor area displays the following text:

```
1 Inspiration
2 Jason Eisner Dyna          Prolog with Aggregation
3 Alexey Radul Propagation Networks Information Cells
4 Analysts Excel             Beyond SUMPRODUCT Pivot Tables
5 Awe Dismay Confounding

6
7 Bridge
8 Table Relation
9 Formula Rule
10 Pivot Aggregator
11 What-If Parameter|
```

The bottom right corner of the editor area has a small preview window showing a graphical representation of data.

At the bottom of the screen, a command palette is open with the following sections:

- Commands and Keystrokes:
 - ⌘P A R A M E T E R
 - editor:indent ↵
 - editor:newline ⌘W H A T - ⌘I F
- inspiration.clpcts* 11:19
- UTF-8 Constraint Logic Prop...

Constraint logic programming provides a natural bridge. Tables correspond to relations; formulas to rules; pivot tables to aggregators; what-if analysis to parameterization. That's all cute, but then I had an idea worth sharing:

*Treat Prolog
facts and relations
as
nodes and edges
in a
propagation network.*

The screenshot shows the Atom text editor interface. The title bar reads "inspiration.clpcs - /Users/wtayson/Desktop/working-clpcs - Atom". There are three tabs open: "inspiration.clpcs" (selected), "example.clpcs", and "review.clpcs". The "inspiration.clpcs" tab contains the following text:

```
1 Inspiration
2 Jason Eisner Dyna          Prolog with Aggregation
3 Alexey Radul Propagation Networks Information Cells
4 Analysts      Excel        Beyond SUMPRODUCT Pivot Tables
5 Awe Dismay Confounding

6
7 Bridge
8 Table Relation
9 Formula Rule
10 Pivot Aggregator
11 What-If Parameter
```

Below the code, there is a "Commands and Keystrokes" section with two rows of key mappings:

editor:indent	⌃P	A	R	A	M	E	T	E	R
	→								
editor:newline	⌃W	H	A	T	-	⌃I	F		
	↩								

At the bottom of the editor window, it says "inspiration.clpcs* 11:19" and "UTF-8 Constraint Logic Prop...".

What happens when we treat prolog facts and relations as nodes and edges in a propagation network? By way of example, let's see how this might play out.

```
inspiration.clpcs - /Users/wtayson/Desktop/working-clp - Atom
inspiration.clpcs example.clpcs review.clpcs

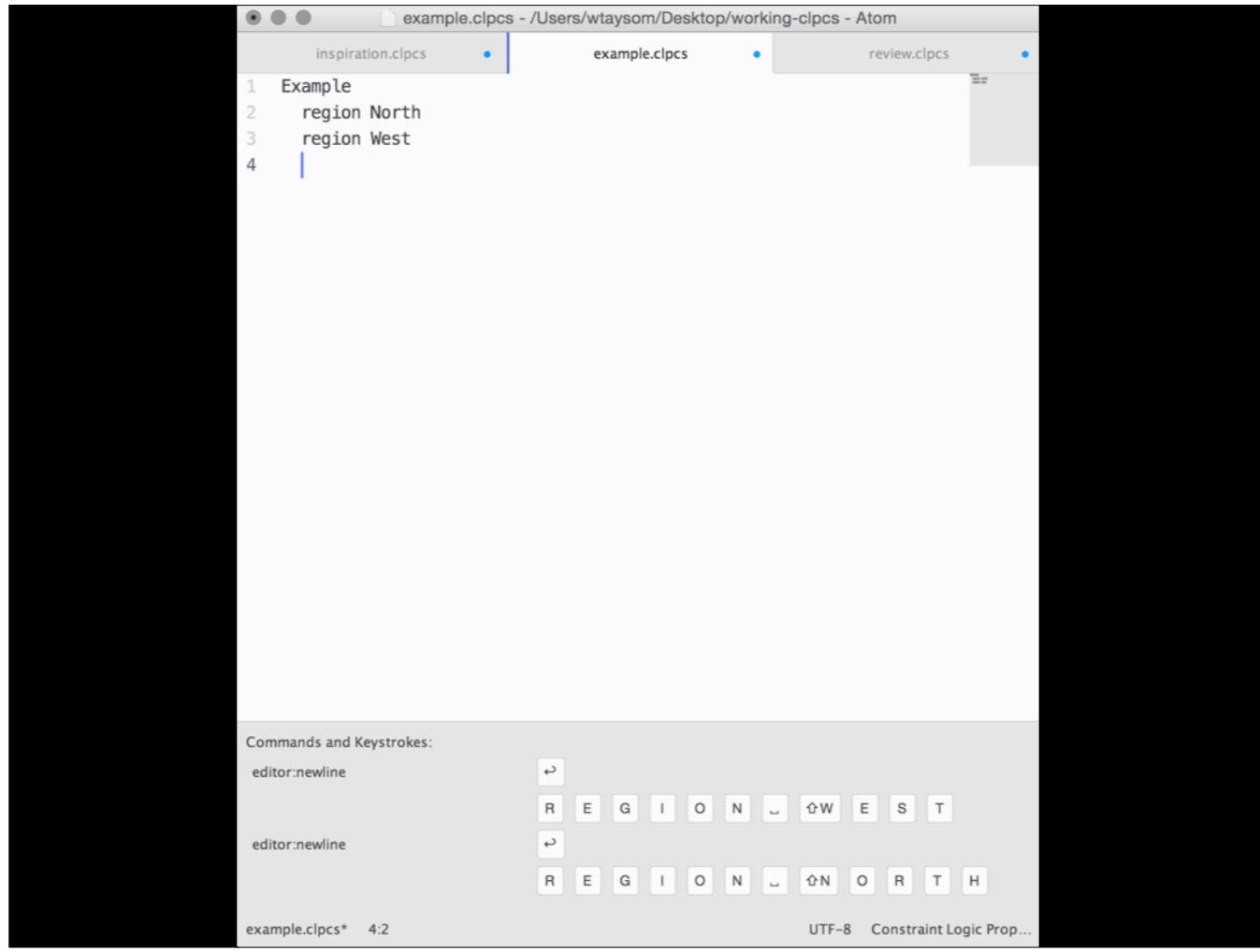
1 Inspiration
2 Jason Eisner Dyna Prolog with Aggregation
3 Alexey Radul Propagation Networks Information Cells
4 Analysts Excel Beyond SUMPRODUCT Pivot Tables
5 Awe Dismay Confounding
6
7 Bridge
8 Table Relation
9 Formula Rule
10 Pivot Aggregator
11 What-If Parameter

Commands and Keystrokes:
editor:indent ⌘P A R A M E T E R
editor:newline ↵
inspiration.clpcs* 11:19 UTF-8 Constraint Logic Prop...
```

Act I

Relation

In the auction room, we often have measurements of an aggregate phenomena from which we want to infer the contribution of individual factors. For illustration, suppose we can measure all traffic (of some sort) into a region, but we would really like to find how much comes from one specific place. Certainly, the amount has to be between all and none, and it turns out that provided information about traffic into and out of other regions, we can make inferences.



```
example.clp - /Users/wtayson/Desktop/working-clp - Atom
inspiration.clpcs • example.clpcs • review.clpcs
1 Example
2 region North
3 region West
4
```

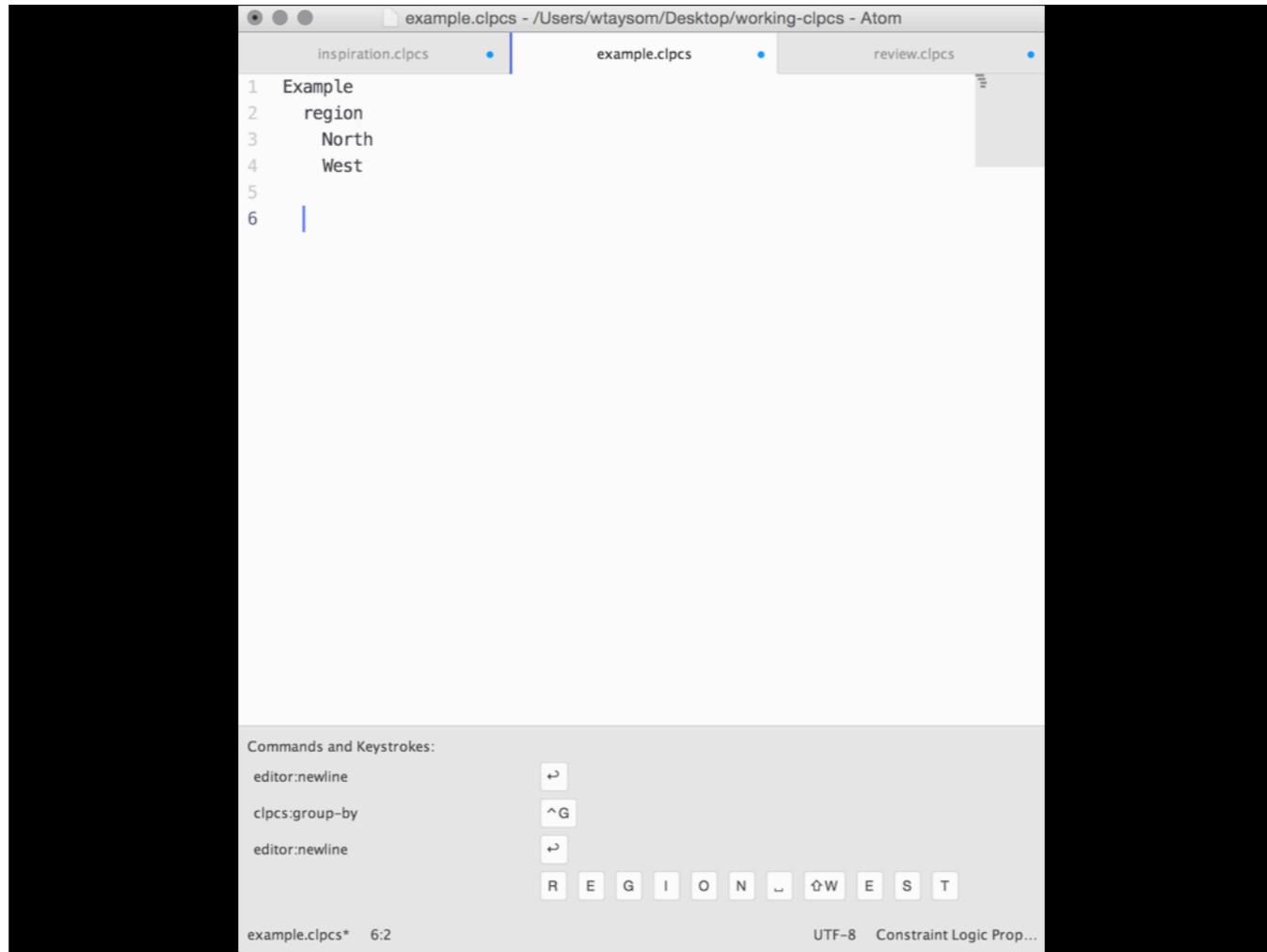
Commands and Keystrokes:

```
editor:newline ↵
R E G I O N ↵ ↵ W E S T
```

```
editor:newline ↵
R E G I O N ↵ ↵ N O R T H
```

example.clp* 4:2 UTF-8 Constraint Logic Prop...

Suppose we live on a big island divided into regions. We have the North and the West. Two facts. Basically Prolog minus parenthesis and case insensitive.



```
example.clpcs - /Users/wtayson/Desktop/working-clpcs - Atom
inspiration.clpcs • example.clpcs • review.clpcs
1 Example
2 region
3 North
4 West
5
6
```

Commands and Keystrokes:

- editor:newline ↵
- clpcs:group-by ^G
- editor:newline ↵

R E G I O N U ⌘W E S T

example.clpcs* 6:2 UTF-8 Constraint Logic Prop...

Let's look at the facts differently. Type `ctrl-g` to "group by" region. It's a table, just like the Inspiration, Bridge, and Example tables we've already seen. I typed notes for the introduction so you could see that this tool starts with structured text like Emacs Org mode. The grouped region block is basically the same as prefixing each line with `region`.

Let's see another relation.

The screenshot shows a Mac OS X desktop environment with the Atom code editor open. The window title is "example.clpcs - /Users/wtayson/Desktop/working-clpcs - Atom". There are three tabs visible: "inspiration.clpcs", "example.clpcs" (which is the active tab), and "review.clpcs". The content of the "example.clpcs" tab is as follows:

```
1 Example
2   region
3     North
4     West
5
6   into  North = 365
7   into  West  = 404
8   out of North = 385
9   out of West  = 415
10
```

Below the code editor, a "Commands and Keystrokes" palette is displayed, listing the following commands:

Command	Keystroke
core:paste	⌘V
editor:newline	↵
clpics:group-by	⌃G
editor:newline	↵

The status bar at the bottom of the editor shows "example.clpcs* 10:2" on the left and "UTF-8 Constraint Logic Prop..." on the right.

Traffic data into and out of regions. These facts have associated values. Notice from the spacing that this data is tabbed into columns. Now let's pivot the table.

example.clp - /Users/wtayson/Desktop/working-clp - Atom

inspiration.clpcs • example.clpcs • review.clpcs

```
1 Example
2   region
3     North
4     West
5
6   into
7     North = 365
8     West  = 404
9   out of
10    North = 385
11    West   = 415
12
```

Commands and Keystrokes:

clp:group-by	^G
core:paste	⌘V
editor:newline	↵
clp:group-by	^G

example.clp* 12:2 UTF-8 Constraint Logic Prop...

Group by the direction of traffic.

The screenshot shows the Atom text editor with three tabs open: 'inspiration.clp', 'example.clp', and 'review.clp'. The 'example.clp' tab is active, displaying the following CLP-CS code:

```
1 Example
2 region
3 North
4 West
5
6     North West
7 into 365 404
8 out of 385 415
9
```

A tooltip is displayed over the 'group-by' command in the code, listing its keybindings: `clp:group-by` (`^G`) and `clp:group-by` (`^G`). The tooltip also includes other commands: `core:paste` (`⌘V`), `editor:newline` (`↵`), and file information: `example.clp* 9:2`, `UTF-8 Constraint Logic Prop...`.

Group by region. We have a table with column and row headers. The data is the same. The group-by command changes the view: both its appearance and the commands for interacting with it. I present the interface in a text editor not because text is best (has some advantages) rather as programmers we are all used to it. I chose the Atom editor because it's fairly easy to instrument. A greater man would, no doubt, use Emacs. And some of you have built your own.

example.clp - /Users/wtayson/Desktop/working-clp - Atom

inspiration.clpcs • example.clpcs • review.clpcs

```
1 Example
2   region
3     North
4     West
5
6 traffic
7   North West
8   into 365 404
9   out of 385 415
10
```

Commands and Keystrokes:

- clp:complete-rename ↵
- clp:indent-and-name-table ⌘J
- clp:group-by ⌘G

example.clp* 10:2 UTF-8 Constraint Logic Prop...

Name the table. Indenting in this system creates a new scope. By naming the table, we add a prefix to all the relations represented in the table.

```
example.clpcs - /Users/wtayson/Desktop/working-clpics - Atom
inspiration.clpcs • example.clpcs • review.clpcs
1 Example
2   region
3     North
4     West
5
6   traffic
7     North West
8     into 365 404
9     out of 385 415
10
11 ? traffic into West
12 traffic into West = 404
```

Commands and Keystrokes:

- clpcs:answer
- editor:newline
- clpcs:complete-rename

example.clpcs* 11:21 UTF-8 Constraint Logic Prop...

Let's ask some questions about traffic. What do we know about traffic into the West? The traffic equals 404.

The screenshot shows the Atom code editor with three tabs open: 'inspiration.clp:cs', 'example.clp:cs' (which is the active tab), and 'review.clp:cs'. The 'example.clp:cs' tab contains the following code:

```
1 Example
2 region
3 North
4 West
5
6 traffic
7     North West
8     into 365 404
9     out of 385 415
10
11 include traffic
12
13 ? into West
14 into West = 404
15
```

The line 'into West = 404' is highlighted with a light purple background. At the bottom of the editor, there is a 'Commands and Keystrokes' panel with several command buttons and a status bar indicating 'example.clp:cs* 15:2' and 'UTF-8 Constraint Logic Prop...'. The status bar also shows a small icon of a person.

To avoid the prefix, we can include traffic facts as Example facts.

The screenshot shows a window titled "example.clpcs - /Users/wtayson/Desktop/working-clpcs - Atom". The editor has three tabs: "inspiration.clpcs", "example.clpcs", and "review.clpcs". The "example.clpcs" tab is active and contains the following CLP code:

```
1 Example
2   region
3     North
4     West
5
6 traffic
7   North West
8   into 365 404
9   out of 385 415
10
11 include traffic
12
13 ? into West
14   into West = 404
15
16 into West = 435
17
18 ? into West
19   into West = 404
20   into West = 435
```

The code uses indentation and multiple assignment operators (=) to assign values to variables. Lines 14 and 19 both assign the value 404 to the variable 'into West'. Lines 16 and 20 both assign the value 435 to the same variable. This results in a conflict, as indicated by the error message in the status bar.

Commands and Keystrokes:

- clpcs:answer
- editor:newline
- editor:newline

example.clpcs* 18:13 UTF-8 Constraint Logic Prop...

If we "change" the value, what then? We get a conflict: two values that are at odds. In the future, the equals sign is never used for assignment. Equals is an assertion that two expressions have the same value. Here we have assigned `into West` to multiple values: both 404 and 435.

The screenshot shows a window titled "example.clpcs - /Users/wtayson/Desktop/working-clpcs - Atom" with three tabs: "inspiration.clpcs", "example.clpcs", and "review.clpcs". The "example.clpcs" tab is active, displaying the following CLP(CS) code:

```
1 Example
2   region
3     North
4     West
5
6 traffic
7   North West
8   into 365 404
9   out of 385 415
10
11 include traffic
12
13 ? into West
14   into West = 404
15
16 ! into West
17   into West = 435
18
19 ? into West
20   into West = 435
```

The code defines regions North and West, and a traffic fact with values 365, 404, 385, and 415. It includes the traffic fact and then overrides the value for 'into West' to 404, then 435, and finally 435 again.

Below the code editor, there is a "Commands and Keystrokes" section with two rows of buttons:

editor:newline	↑	! ↵	I ↵	N ↵	T ↵	O ↵	⌫	⌃W ↵	E ↵	S ↵	T ↵	
clpcs:answer	↑	? ↵	⌫	I ↵	N ↵	T ↵	O ↵	⌫	⌃W ↵	E ↵	S ↵	T ↵

At the bottom left, it says "example.clpcs* 17:2". At the bottom right, it says "UTF-8 Constraint Logic Prop...".

To properly override a value, first we should forget what we already asserted. To forget suggests time and state. In order to permit imperative refinement of a model while retaining reactive update, facts collect from top to bottom. Each assertion is a function from one set of facts to another. A change to one fact may change the answers to all the questions below it. In contrast reevaluating an expression in a Mathematica notebook does not affect related expressions, so the text can get out of sync with the heap of definitions. Ask non-programmer Mathematica-ticians about their workarounds sometime.

```
example.clp.cs - /Users/wtayson/Desktop/working-clp.cs - Atom
inspiration.clp.cs • example.clp.cs • review.clp.cs
1 Example
2   region
3     North
4     West
5
6   traffic
7     North West
8     into 365 404
9     out of 385 415
10
11 include traffic
12
13 ? into West
14   into West = 404
15
16 : into West = 435
17
18 ? into West
19   into West = 435
```

Commands and Keystrokes:

- clp.cs:reduce ⌘R
- editor:newline ↵
- clp.cs:answer ⌘P

example.clp.cs* 16:19 UTF-8 Constraint Logic Prop...

To retract and assert together is to revise. The reduce command merges facts. It normalizes their total effect on the model. Call it function composition.

```
example.clpcs - /Users/wtayson/Desktop/working-clpcs - Atom
inspiration.clpcs • example.clpcs • review.clpcs
1 Example
2 region
3 North
4 West
5
6 traffic
7     North West
8     into 365 435
9     out of 385 415
10 include traffic
11
12 ? into West
13     into West = 435
```

Commands and Keystrokes:

- clpcs:reduce ⌘R
- clpcs:reduce ⌘R
- editor:newline ↵

I ⌘I N ⌘N T ⌘T O ⌘O ⌘W E ⌘E S ⌘S ⌘T

example.clpcs* 6:2 (84) UTF-8 Constraint Logic Prop...

Let's reduce again to revise the original. We update `into West` by patching in-place.

The screenshot shows a window titled "example.clpcs - /Users/wtayson/Desktop/working-clpcs - Atom" with three tabs: "inspiration.clpcs", "example.clpcs", and "review.clpcs". The "example.clpcs" tab is active, displaying the following CLP(CS) code:

```
1 Example
2   region
3     North
4     West
5
6   traffic
7     North West
8     into 365 435
9     out of 385 415
10
11 include traffic
12
13 ? _ West
14   region West
15   into West = 435
16   out of West = 415
```

The code uses the `region` predicate to define regions North and West, and the `traffic` predicate to define traffic volumes between them. The query `? _ West` is shown, with the response `region West` and its associated facts `into West = 435` and `out of West = 415` highlighted.

At the bottom of the editor, there is a "Commands and Keystrokes" panel with the following entries:

- clpcs:answer
- clpcs:reduce
- clpcs:reduce

Below the editor, the status bar shows "example.clpcs* 13:10" and "UTF-8 Constraint Logic Prop...".

What do we know about the West overall? A blank in a question is answered by every unifying substitution. The West is the object of three predicates: `region`, `into`, and `out of`.

The screenshot shows the Atom code editor with three tabs open: 'inspiration.clp', 'example.clp', and 'review.clp'. The 'example.clp' tab is active and contains the following CLP(PS) code:

```
1 Example
2   region
3     North
4     West
5
6 traffic
7   North West
8   into 365 435
9   out of 385 415
10
11 include traffic
12
13 ? _-
14   region North
15   region West
16   into North = 365
17   into West = 435
18   out of North = 385
19   out of West = 415
```

The code editor interface includes a status bar at the bottom with the file name 'example.clp', line count '13:7', encoding 'UTF-8', and a tooltip 'Constraint Logic Prop...'. A 'Commands and Keystrokes' section is also visible.

And if we add another blank? We get all of our facts because they all happen to be predicate object pairs.

The screenshot shows the Atom text editor with three tabs open: 'inspiration.clp', 'example.clp', and 'review.clp'. The 'example.clp' tab is active and contains the following CLP(PS) code:

```
1 Example
2   region
3     North
4     West
5
6 traffic
7   North West
8   into 365 435
9   out of 385 415
10
11 include traffic
12
13 ? :direction
14 (X) undefined predicate direction_
```

The line '13 ? :direction' is highlighted with a red rectangle, and the message '(X) undefined predicate direction_' is displayed in a red box below it. At the bottom of the editor window, there is a 'Commands and Keystrokes' panel with several key mappings:

- clp:answer: ↵
- : D I R E C T I O N
- core:backspace: ⌫
- core:move-left: ←

The status bar at the bottom shows 'example.clp* 13:16' and 'UTF-8 Constraint Logic Prop...'. A small icon for 'Atom' is visible in the top right corner.

Suppose we just want the facts involving a direction. `:direction` unifies with everything that satisfies the direction predicate; however, `direction` is undefined.

The screenshot shows a window titled "example.clp" with three tabs: "inspiration.clp", "example.clp", and "review.clp". The "example.clp" tab is active, displaying the following Prolog-like code:

```
1 Example
2   region
3     North
4     West
5
6 traffic
7   North West
8   into 365 435
9   out of 385 415
10
11 include traffic
12
13 direction
14   region
15   into
16   out of
17
18 ? :direction _
19   region North
20   region West
21   into North = 365
22   into West = 435
23   out of North = 385
24   out of West = 415
```

The code editor has syntax highlighting and a code completion feature. A tooltip for the predicate "? :direction _" lists possible completions: "region North", "region West", "into North = 365", "into West = 435", "out of North = 385", and "out of West = 415".

At the bottom of the editor, there is a "Commands and Keystrokes" section with the following entries:

- clp:quick-fix
- clp:answer
- core:backspace

Below the commands, the status bar shows "example.clp* 18:16" and "UTF-8 Constraint Logic Prop...".

Hit enter again for a quick fix. The system can guess what might be a direction from the predicates that already match an unrestricted blank. No magic here, just requires some context. The trick is having a system where context is readily available.

The screenshot shows a window titled "example.clpcs - /Users/wtayson/Desktop/working-clpcs - Atom". The editor has three tabs: "inspiration.clpcs", "example.clpcs" (which is active), and "review.clpcs". The "example.clpcs" tab contains the following CLP code:

```
1 Example
2   region
3     North
4     West
5
6 traffic
7   North West
8   into 365 435
9   out of 385 415
10
11 include traffic
12
13 direction
14   into
15   out of
16
17 ? :direction _
18   into North = 365
19   into West = 435
20   out of North = 385
21   out of West = 415
```

A light blue selection highlights the last four lines of the code (lines 18-21). Below the code editor, a "Commands and Keystrokes" panel lists:

- core:backspace
- clpcs:quick-fix
- clpcs:answer

Below the commands, there is a row of icons and a status bar:

- Icons: a square with a diagonal line, a double arrow, a double arrow, and a double arrow.
- Status bar: "example.clpcs* 14:1" on the left and "UTF-8 Constraint Logic Prop..." on the right.

Restrict `direction` properly, and we reproduce the traffic table.

The screenshot shows a window titled "example.clpcs - /Users/wtayson/Desktop/working-clpcs - Atom". The editor has three tabs: "inspiration.clpcs", "example.clpcs", and "review.clpcs". The "example.clpcs" tab is active and contains the following code:

```
1 Example
2   region
3     North
4     West
5
6   traffic
7     direction North West
8     into 365 435
9     out of 385 415
10
11 include traffic
12
13 ? :direction _
14   into North = 365
15   into West = 435
16   out of North = 385
17   out of West = 415
```

A light blue selection highlights the question mark and the "direction" atom. Below the code, there is a "Commands and Keystrokes" section with the following entries:

clpcs:reduce	⌘R
core:backspace	⌫
clpcs:quick-fix	⟳
clpcs:answer	⟳

At the bottom of the editor, it says "example.clpcs* 11:17" and "UTF-8 Constraint Logic Prop...".

Since directions label a column in the traffic table, we can combine them.

The screenshot shows a window titled "example.clpcs - /Users/wtayson/Desktop/working-clpcs - Atom". The editor has three tabs: "inspiration.clpcs", "example.clpcs", and "review.clpcs". The "example.clpcs" tab is active and contains the following CLP CS code:

```
1 Example
2 traffic
3     region
4         direction North West
5         into    365  435
6         out of   385  415
7
8     include traffic
9
10 ? :direction _
11     into  North = 365
12     into  West   = 435
13     out of North = 385
14     out of West  = 415
```

The code editor interface includes a status bar at the bottom with the following information:

- Commands and Keystrokes:
 - clpcs:reduce ⌘R
 - clpcs:reduce ⌘R
 - core:backspace ⌫
 - clpcs:quick-fix ⌘P
- example.clpcs* 7:1
- UTF-8 Constraint Logic Prop...

Likewise, since regions label a row, let's merge again. We now have three relations embedded in one table: region, direction, and how their product maps onto values.

```
example.clpcs - /Users/wtayson/Desktop/working-clpics - Atom
inspiration.clpcs • example.clpcs • review.clpcs
1 Example
2 traffic
3 region
4 direction North West Hilo
5 into 365 435 500
6 out of 385 415
7
8 include traffic
9
10 ? :direction _
11 into North = 365
12 into West = 435
13 into Hilo = 500
14 out of North = 385
15 out of West = 415
16 out of Hilo

Commands and Keystrokes:
clpics:edit-next-row ↵
5 0 0
clpics:edit-next-row ↵
◊H I L O
example.clpics* 6:24 UTF-8 Constraint Logic Prop...
```

Now when we modify our traffic table, we assert several facts at once. Notice that new Hilo facts appear as an answer to our direction question.

```
example.clpcs - /Users/wtayson/Desktop/working-clpics - Atom
inspiration.clpcs • example.clpcs • review.clpcs
1 Example
2 traffic
3     region
4         direction North West Hilo
5             into    365  435  500
6             out of   385  415
7
8     include traffic
9
10 ? :direction Hilo
11     into  Hilo = 500
12     out of Hilo
```

Commands and Keystrokes:

- clpcls:answer
- core:backspace
- clpcls:edit-next-row

example.clpcs* 10:19 UTF-8 Constraint Logic Prop...

Consider Hilo specifically. We know that `out of Hilo` is a fact, but we don't know its value since we left a cell blank.

```
example.clpcs - /Users/wtayson/Desktop/working-clpcs - Atom
inspiration.clpcs • example.clpcs • review.clpcs
1 Example
2 traffic
3     region
4         direction North West Hilo
5             into    365  435  500
6             out of   385  415 =
7
8     include traffic
9
10 ? :direction Hilo
11     into  Hilo = 500
12     out of Hilo
```

Commands and Keystrokes:

- clpcs:answer
- core:backspace
- clpcs:edit-next-row

example.clpcs* 6:25 UTF-8 Constraint Logic Prop...

To fix the value, let's use a formula. Equals.

example.clpcs - /Users/wtayson/Desktop/working-clpcs - Atom

inspiration.clpcs example.clpcs review.clpcs

```
1 Example
2 traffic
3         region
4 direction North West Hilo
5 into    365   435  500
6 out of  385   415 =
7
8 include traffic
9
10 ? :direction Hilo
11     into  Hilo = 500
12     out of Hilo
```

Commands and Keystrokes:

clpcs:answer

core:backspace

clpcs:edit-next-row

example.clpcs* 6:25

Click.

example.clpcs - /Users/wtayson/Desktop/working-clpcs - Atom

inspiration.clpcs example.clpcs review.clpcs

```
1 Example
2 traffic
3     region
4     direction North West Hilo
5     into    365  435  500
6     out of   385  415 = into Hilo
7
8 include traffic
9
10 ? :direction Hilo
11     into  Hilo = 500
12     out of Hilo
```

Commands and Keystrokes:

clpcs:answer

core:backspace

clpcs:edit-next-row

example.clpcs* 6:35

UTF-8 Constraint Logic Prop...

Return. Just like a spreadsheet.

```
example.clpcs - /Users/wtayson/Desktop/working-clpacs - Atom
inspiration.clpcs • example.clpcs • review.clpcs
1 Example
2 traffic
3     region
4         direction North West Hilo
5             into    365  435  500
6             out of   385  415  500
7
8     include traffic
9
10 ? :direction Hilo
11     into  Hilo = 500
12     into  Hilo = out of Hilo
13     out of Hilo = into  Hilo
14     out of Hilo = 500
```

Commands and Keystrokes:

- clpacs:answer ↵
- clpacs:answer ↵
- core:backspace ⌘H I L O

example.clpcs* 6:27 UTF-8 Constraint Logic Prop...

Return. Just like a spreadsheet.

The screenshot shows a window titled "example.clp.cs - /Users/wtayson/Desktop/working-clp.cs - Atom". The code editor has three tabs: "inspiration.clp.cs", "example.clp.cs", and "review.clp.cs". The "example.clp.cs" tab is active and contains the following code:

```
1 Example
2 traffic
3     region
4         direction North West Hilo
5         into    365  435  500
6         out of   385  415  500
7
8     include traffic
9
10 ? :direction Hilo
11     into Hilo = 500
12     into Hilo = out of Hilo
13     out of Hilo = into Hilo
14     out of Hilo = 500
15     into Hilo = 500
16     out of Hilo = into Hilo
17
```

A portion of the code from line 10 to line 16 is highlighted with a pink background. Below the code editor, there is a "Commands and Keystrokes" section with the following entries:

- clp.cs:elaborate
- clp.cs:answer
- clp.cs:answer

At the bottom of the editor, there are several small icons: a left arrow, a right arrow, a double left arrow, a double right arrow, and a square. The status bar at the bottom shows "example.clp.cs* 17:1" on the left and "UTF-8 Constraint Logic Prop..." on the right.

From the formula, the system infers the value of `out of Hilo`. `out of Hilo` equals `into Hilo`. `into Hilo` equals 500. A two link equality chain.

```
example.clpcs - /Users/wtayson/Desktop/working-clpics - Atom
inspiration.clpcs • example.clpcs • review.clpcs
1 Example
2 traffic
3     region
4         direction North West Hilo
5         into    365  435  500
6         out of   385  415  500
7
8 include traffic
9
10 ?= :direction Hilo
11     500 = into Hilo
12     500 = out of Hilo
```

Commands and Keystrokes:

- clpics:answer ↵
- clpics:elaborate ↵
- clpics:answer ↵

example.clpcs* 10:20 UTF-8 Constraint Logic Prop...

If we want to focus on values instead of facts, we ask a different sort of question.

example.clpcs - /Users/wtayson/Desktop/working-clpcs - Atom

inspiration.clpcs example.clpcs review.clpcs

```
1 Example
2 traffic
3     region
4         direction North West Hilo
5         into    365  435  500
6         out of   385  415  500
7
8 include traffic
9
10 ?= out of Hilo
11 500
```

Commands and Keystrokes:

clpcs:answer ↵ O U T ← O F

clpcs:answer ↵ =

example.clpcs* 10:16 UTF-8 Constraint Logic Prop...

When the question is grounded, we find familiar functional evaluation.

```
example.clp - /Users/wtayson/Desktop/working-clp - Atom
inspiration.clpcs example.clpcs review.clpc
1 Example
2 traffic
3 region
4 direction North West Hilo
5 into 365 435 500
6 out of 385 415 500
7
8 include traffic
9
10 ?= out of Hilo
11 500
```

Commands and Keystrokes:

clp:answer	↶
	O U T ← O F
clp:answer	↶
	=

example.clp* 10:16 UTF-8 Constraint Logic Prop...

Act II

Propagation

Evaluation is old school; in the future programmers upgrade to inference.

The screenshot shows the Atom code editor with three tabs: 'inspiration.clpcs', 'example.clpcs' (which is active), and 'review.clpcs'. The 'example.clpcs' tab contains the following CLP CS code:

```
1 Example
2 traffic
3     region
4         direction North West Hilo
5             into    365  435  500
6             out of   385  415  500
7
8 include traffic
9
10 from :region to :region ≥ 0
11 ?
12     from North to North ≥ 0
13     from North to West  ≥ 0
14     from North to Hilo   ≥ 0
15     from West   to North ≥ 0
16     from West   to West  ≥ 0
17     from West   to Hilo   ≥ 0
18     from Hilo    to North ≥ 0
19     from Hilo    to West  ≥ 0
20     from Hilo    to Hilo   ≥ 0
```

The code editor interface includes a status bar at the bottom with 'Commands and Keystrokes' for 'clpcs:answer' and 'editor:newline', and a file status 'example.clpcs* 11:3'. The bottom right corner shows 'UTF-8 Constraint Logic Prop...'. A small sidebar on the right displays a tree structure.

From our data about traffic into and out of regions, we can learn about the traffic going directly from one region to another. Let's see how this could work. The question mark refers to the previous relation. Notice that `from_to_` is defined for every pair of regions. Rather than declaring a type restriction, the colon syntax populates a relation with all values matching a predicate. Too many in this case: traffic only makes sense between regions.

```
example.clp.cs - /Users/wtayson/Desktop/working-clp.cs - Atom
inspiration.clp.cs • example.clp.cs • review.clp.cs
1 Example
2 traffic
3     region
4         direction North West Hilo
5         into    365 435 500
6         out of   385 415 500
7
8 include traffic
9
10 from _r:region to _s:region >= 0 :- _r /~ _s
11 ?
12     from North to West >= 0
13     from North to Hilo >= 0
14     from West to North >= 0
15     from West to Hilo >= 0
16     from Hilo to North >= 0
17     from Hilo to West >= 0

Commands and Keystrokes:
clp.cs:answer
editor:move-to-end-of-screen-line
example.clp.cs* 10:45
UTF-8 Constraint Logic Prop...
```

To correctly define the `from_to` relation, we should stipulate that the two regions be different. (Instead of case, I use underscores to mark variables.) At last we see Prolog's famous turnstile notation. The slash-tilde says that the two regions `_r` and `_s` should not unify. So read the whole expression formally as "the traffic from one region to another both exists and is bounded below by zero provided the two regions are distinct."

The screenshot shows a window titled "example.clpcs - /Users/wtayson/Desktop/working-clpcs - Atom". The editor has three tabs: "inspiration.clpcs", "example.clpcs", and "review.clpcs". The "example.clpcs" tab is active and contains the following CLP CS code:

```
1 Example
2 traffic
3     region
4         direction North West Hilo
5         into    365  435  500
6         out of   385  415  500
7
8 include traffic
9
10 from _r:region to :region/~/_r >= 0
11 ?
12     from North to West >= 0
13     from North to Hilo >= 0
14     from West to North >= 0
15     from West to Hilo >= 0
16     from Hilo to North >= 0
17     from Hilo to West >= 0
```

The code editor interface includes a status bar at the bottom with "example.clpcs* 10:2 (33)" and "UTF-8 Constraint Logic Prop...". A "Commands and Keystrokes" panel is also visible.

As we did with the grouped and transposed traffic table, we can add syntactic sugar. I feel this makes for a more natural read, "The traffic from one region to another distinct region is bounded below by zero."

The screenshot shows a window titled "example.clpcs - /Users/wtayson/Desktop/working-clpcs - Atom". The editor has three tabs: "inspiration.clpcs", "example.clpcs", and "review.clpcs". The "example.clpcs" tab is active, displaying the following CLP(CS) code:

```
1 Example
2 traffic
3     region
4         direction North West Hilo
5         into    365 435 500
6         out of   385 415 500
7
8 include traffic
9
10 from _r to _s ≥ 0 :- region _r, region _s, _r /~ _s
11 ?
12 from North to West ≥ 0
13 from North to Hilo ≥ 0
14 from West to North ≥ 0
15 from West to Hilo ≥ 0
16 from Hilo to North ≥ 0
17 from Hilo to West ≥ 0
```

The line "from _r to _s ≥ 0 :- region _r, region _s, _r /~ _s" is highlighted in orange. Lines 12 through 17 are highlighted in light blue.

Below the code editor, there is a "Commands and Keystrokes" section with the following entries:

clpcs:elaborate	^E
clpcs:elaborate	^E
clpcs:abbreviate	^R
editor:select-to-first-character-of-line	⇧ ⌘ ←

At the bottom left, it says "example.clpcs* 10:2 (51)". At the bottom right, it says "UTF-8 Constraint Logic Prop...".

Sifting out syntactic sugar reveals that the colon syntax is shorthand for a predicate in the body.

The screenshot shows the Atom code editor interface with three tabs open: 'inspiration.clpcs', 'example.clpcs' (which is the active tab), and 'review.clpcs'. The 'example.clpcs' tab contains the following CLP CS code:

```
1 Example
2 traffic
3     region
4         direction North West Hilo
5             into    365  435  500
6             out of   385  415  500
7
8 include traffic
9
10 from _r:region to :region/~/_r ≥ 0
11 ?
12     from North to West ≥ 0
13     from North to Hilo ≥ 0
14     from West to North ≥ 0
15     from West to Hilo ≥ 0
16     from Hilo to North ≥ 0
17     from Hilo to West ≥ 0
```

The code editor has syntax highlighting and code completion. A tooltip for the 'from' keyword is visible, showing its definition: 'from _r:region to :region/~/_r ≥ 0'. Below the editor, a 'Commands and Keystrokes' panel lists key bindings for abbreviate and elaborate commands.

Command	Keystroke
clpcs:abbreviate	^R
clpcs:abbreviate	^R
clpcs:elaborate	^E
clpcs:elaborate	^E

At the bottom of the editor window, status bars show 'example.clpcs*' and '10:2 (33)' on the left, and 'UTF-8 Constraint Logic Prop...' on the right.

Surface syntax is a view rotated one way or another. To make these syntax transformations bidirectional, one should retain representation metadata such as the variable name `_s` and the order of arguments to the does-not-unify-with operator.

example.clpcs - /Users/wtayson/Desktop/working-clpcs - Atom

inspiration.clpcs example.clpcs review.clpcs

```
1 Example
2 traffic
3     region
4         direction North West Hilo
5         into    365  435  500
6         out of   385  415  500
7
8     include traffic
9
10 from _r:region to :region/~/r ≥ 0
11
12 into _r += from _ to _r
13
14
```

Commands and Keystrokes:

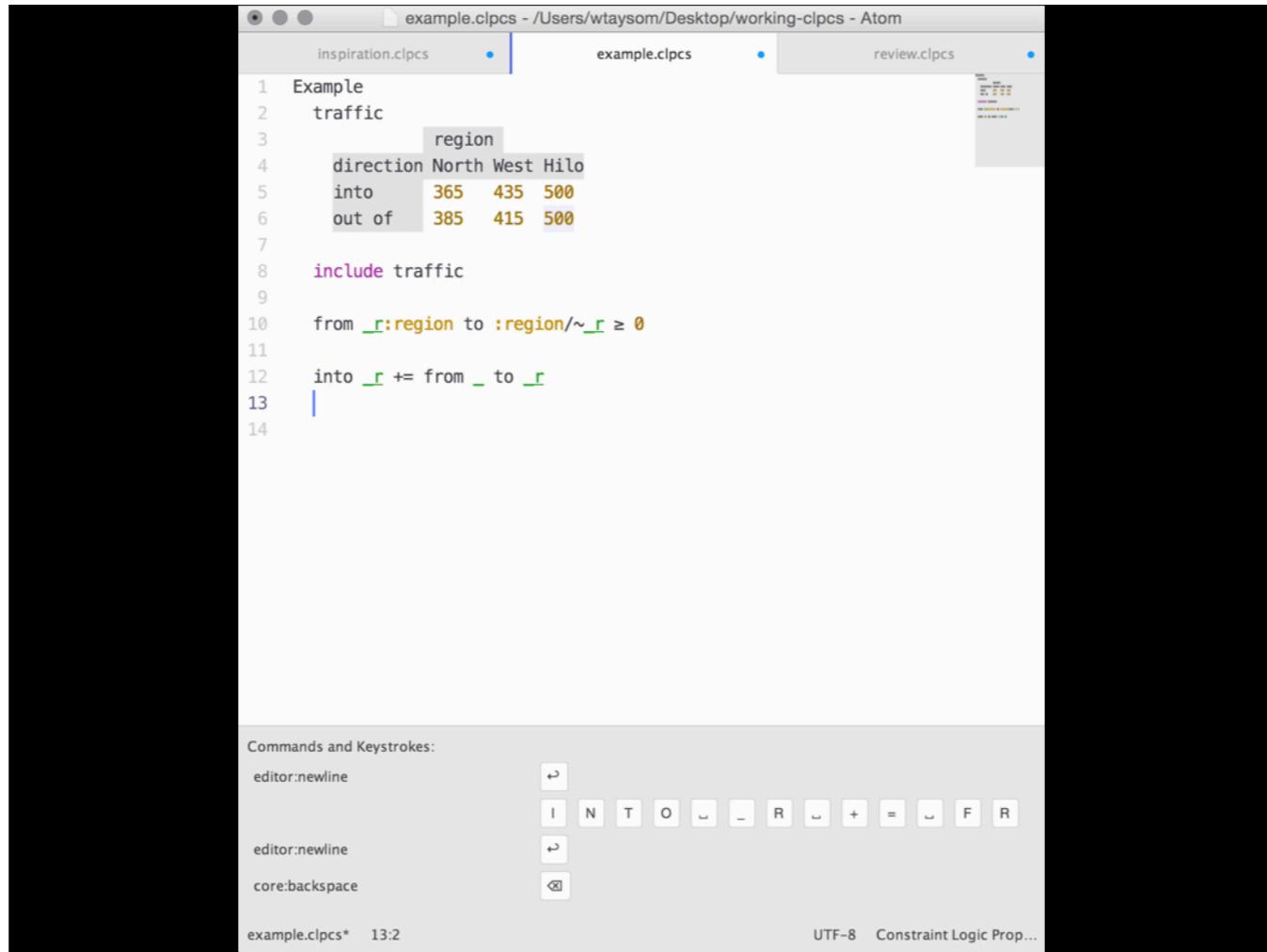
editor:newline ↵

editor:newline ↵

core:backspace ⌫

example.clpcs* 13:2

UTF-8 Constraint Logic Prop...



To continue our analysis, associate the `from_to_` relation with our existing traffic info. The traffic into a region is the sum of all traffic from somewhere into the region.

example.clpcs - /Users/wtayson/Desktop/working-clpcs - Atom

inspiration.clpcs example.clpcs review.clpcs

```
1 Example
2 traffic
3     region
4         direction North West Hilo
5         into    365  435  500
6         out of   385  415  500
7
8     include traffic
9
10    from _r:region to :region/~/r ≥ 0
11
12    into _r += from _ to _r
13    out of _r += from _r to _
14
15
```

Commands and Keystrokes:

editor:newline ↵
O U T ← O F ← ← R ← + = ←

editor:newline ↵
I N T O ← ← R ← + = ← F R

example.clpcs* 14:2 UTF-8 Constraint Logic Prop...

The traffic out of a region is the sum of all traffic from the region to somewhere else.

The screenshot shows the Atom code editor with three tabs open: 'inspiration.clpcs', 'example.clpcs' (which is the active tab), and 'review.clpcs'. The 'example.clpcs' tab contains the following CLP-CS code:

```
1 Example
2 traffic
3     region
4         direction North West Hilo
5         into    365  435  500
6         out of   385  415  500
7
8     include traffic
9
10    from _r:region to :region/~/r ≥ 0
11
12    into _r += from _ to _r
13    out of _r += from _r to _
14
15    ? into West
16        into West = 435
17        into West += from North to West
18        into West += from Hilo to West
19
```

The line 'into West = 435' is highlighted with a light purple background. At the bottom of the editor, there is a 'Commands and Keystrokes' panel with several command buttons and a status bar indicating 'example.clpcs*' and '15:13'.

In particular, traffic into the West is the sum of the traffic from the North plus traffic from Hilo.

```

example.clp - /Users/wtayson/Desktop/working-clp - Atom
inspiration.clp   example.clp   review.clp
ample
traffic
    region
direction North West Hilo
into 365 435 500
out of 385 415 500

include traffic

from _r:region to :region/~/r >= 0

into r += from _ to r
out of r += from r to _

? into West
into West = 435
into West += from North to West
into West += from Hilo to West

```

Is and Keystrokes:

wer	?	l	N	T	O	~	W	E	S	T
wline	?									
wline	?									

clp* 15:13 UTF-8 Constraint Logic Prop...

Aggregators

fold over all

unifying substitutions.

Aggregators fold over all unifying substitutions. They remove the need to explicitly construct a set of items to sum over. Moreover, each line actually defines three sums: one for each region. Three regions in each of two lines makes for six linear equations, six unknowns. A linear dependency ensures that we won't get a unique solution. This makes asking about bounds worthwhile.

The screenshot shows the Atom code editor with three tabs: 'inspiration.clpcs', 'example.clpcs' (which is the active tab), and 'review.clpcs'. The 'example.clpcs' tab contains the following CLP CS code:

```
1 Example
2 traffic
3     region
4         direction North West Hilo
5             into 365 435 500
6             out of 385 415 500
7
8     include traffic
9
10    from _r:region to :region/~/_r ≥ 0
11
12    into _r += from _ to _r
13    out of _r += from _r to _
14
15    ? into West
16        into West = 435
17        into West += from North to West
18        into West += from Hilo to West
19
20    ? from Hilo to West
21        from Hilo to West ≥ 0
22        into West += from Hilo to West
23        out of Hilo += from Hilo to West
```

Below the code editor, there is a 'Commands and Keystrokes' section with the following entries:

clpcs:infer-interactively	⇧ ↵
core:paste	⌘ V
editor:newline	? ↵

At the bottom of the editor window, it says 'example.clpcs* 21:25 (21)' and 'UTF-8 Constraint Logic Prop...'. The status bar also shows 'Atom'.

Let's take a look at traffic from Hilo to the West. With shift-enter, we interactively draw inferences.

The screenshot shows a window titled "example.clpcs - /Users/wtayson/Desktop/working-clpcs - Atom". The code editor displays a CLP CS script named "example.clpcs". The code defines a "traffic" relation with "region" atoms and constraints for "direction", "into", and "out of" traffic counts. It includes an "include traffic" command and several facts about traffic flow between "North", "West", and "Hilo" regions. A specific fact at line 16, "into West = 435", is highlighted with a light blue background. Another fact at line 23, "into West = 435", is highlighted with a pink background. The bottom of the editor shows a toolbar with commands like "elaborate", "move-to-child", "move-to-next-sibling", and "infer-interactively".

```
1 example
2 traffic
3     region
4         direction North West Hilo
5         into    365  435  500
6         out of   385  415  500
7
8     include traffic
9
10    from _r:region to :region/~/_r ≥ 0
11
12    into   _r += from _ to _r
13    out of _r += from _r to _
14
15    ? into West
16        into West = 435
17        into West += from North to West
18        into West += from Hilo to West
19
20    ? from Hilo to West
21        from Hilo to West ≥ 0
22        into West += from Hilo to West
23        into West = 435
24        into West += from North to West
25        out of Hilo += from Hilo to West
```

Expand 'into West' to see other facts about it.

example.clpcs - /Users/wtayson/Desktop/working-clpcs - Atom

inspiration.clpcs example.clpcs review.clpcs

```
1 Example
2 traffic
3     region
4         direction North West Hilo
5         into    365  435  500
6         out of   385  415  500
7
8     include traffic
9
10    from _:r:region to :region/~/_r ≥ 0
11
12    into _r += from _ to _r
13    out of _r += from _r to _
14
15    ? into West
16        into West = 435
17        into West += from North to West
18        into West += from Hilo to West
19
20    ? from Hilo to West
21        from Hilo to West ≥ 0
22        435 = from Hilo to West + from North to West
23        out of Hilo += from Hilo to West
```

Commands and Keystrokes:

clpcs:reduce	↶
clpcs:select-next-sibling	↑↓
clpcs:elaborate	↶
clpcs:move-to-child	→

example.clpcs* 22:47 (44) UTF-8 Constraint Logic Prop...

Reduce them into one equation.

```
1 example
2 traffic
3     region
4         direction North West Hilo
5         into    365  435  500
6         out of   385  415  500
7
8     include traffic
9
10    from _r:region to :region/~/_r >= 0
11
12    into _r += from _ to _r
13    out of _r += from _r to _
14
15    ? into West
16        into West = 435
17        into West += from North to West
18        into West += from Hilo to West
19
20    ? from Hilo to West
21        from Hilo to West >= 0
22        435 = from Hilo to West + from North to West
23        from North to West >= 0
24        out of North += from North to West
25        out of Hilo += from Hilo to West
26
27 Commands and Keystrokes:
28 clpbc:elaborate
29 clpbc:move-to-next-sibling
30 clpbc:move-to-child
31 clpbc:reduce
32
33 example.clpbc* 23:26 (22)
34 UTF-8 Constraint Logic Prop...
```

Expand to see more facts. We have arrived at the propagation part of our tour. Remember grounded terms have associated values. Each term is like a slot or cell where you can stick information. Equations relate cells allowing information to spread from one cell to another.

example.clpcs - /Users/wtayson/Desktop/working-clpcs - Atom

```

inspiration.clpcs • example.clpcs • review.clpcs
ample
traffic
region
direction North West Hilo
into 365 435 500
out of 385 415 500

include traffic

from r:region to :region/~r ≥ 0

into r += from _ to r
out of r += from r to _

? into West
into West = 435
into West += from North to West
into West += from Hilo to West

? from Hilo to West
from Hilo to West ≥ 0
435 = from Hilo to West + from North to West
from North to West ≥ 0
out of North += from North to West
out of Hilo += from Hilo to West

Is and Keystrokes:
borate ↕
ve-to-next-sibling ↓
ve-to-child →
uce ↵

lpcs* 23:26 (22)           UTF-8 Constraint Logic Prop...

```

435
 ┌─┐
 into West
 ┌─┐
 from North from Hilo
 to West to West

Take a look at this chart showing the relationship of `into West` to other facts. It is equal to 435, and it is the sum of the two `from _ into West` facts.

example.clpcs - /Users/wtayson/Desktop/working-clpcs - Atom

```

inspiration.clpcs • example.clpcs • review.clpcs
ample
traffic
region
direction North West Hilo
into 365 435 500
out of 385 415 500

include traffic

from _r:region to :region/~/_r ≥ 0

into _r += from _ to _r
out of _r += from _r to _

? into West
  into West = 435
  into West += from North to West
  into West += from Hilo to West

? from Hilo to West
  from Hilo to West ≥ 0
  435 = from Hilo to West + from North to West
    from North to West ≥ 0
    out of North += from North to West
    out of Hilo += from Hilo to West

Is and Keystrokes:
borate ↕
ve-to-next-sibling ↓
ve-to-child →
uce ↵

lpcs* 23:26 (22)           UTF-8 Constraint Logic Prop...

```

435
≡
into West
+
from North from Hilo
to West to West
400 35
200 235
0 435
 $0 \leq \leq 435$

If we learn the value of one of the summands, we can infer the other. Propagation goes both ways. Numbers propagate, and bounds do too.

example.clpccs - /Users/wtayson/Desktop/working-clpccs - Atom

inspiration.clpccs • example.clpccs • review.clpccs

```

ample
traffic
    region
direction North West Hilo
into   365  435  500
out of 385  415  500

include traffic

from ?r:region to :region/~_r ≥ 0

into _r += from _ to _r
out of _r += from _r to _

? into West
    into West = 435
    into West += from North to West
    into West += from Hilo to West

? from Hilo to West
    from Hilo to West ≥ 0
    from Hilo to West ≤ 435
    out of Hilo += from Hilo to West

Is and Keystrokes:
  use      ↵
  borate   ↵
  ve-to-next-sibling   ↓
  ve-to-child   →

```

435
 into West
 from North to West
 from Hilo to West
 400 35
 200 235
 0 435
 $0 \leq \leq 435$

clpccs* 22:26 (23) UTF-8 Constraint Logic Prop...

By propagating the inequality across the sum, we set an upper bound. Since addition is monotonic, a lower bound on one summand imposes an upper bound on the other.

example.clpcts - /Users/wtayson/Desktop/working-clpcts - Atom

```

inspiration.clpcts • example.clpcts • review.clpcts
direction North West Hilo
into 365 435 500
out of 385 415 500

include traffic

from _r:region to :region/~_r ≥ 0

into _r += from _ to _r
out of _r += from _r to _

? into West
  into West = 435
  into West += from North to West
  into West += from Hilo to West

? from Hilo to West
  from Hilo to West ≥ 0
  from Hilo to West ≤ 435
    435 = from Hilo to West + from North to West
      into West = 435
      into West += from Hilo to West
      into West += from North to West
      from North to West ≥ 0
      out of Hilo += from Hilo to West

Is and Keystrokes:
borate ↵
uce ↵
borate ↵
ve-to-next-sibling ↓

lpcst 23:48 (44)           UTF-8 Constraint Logic Prop...

```

435
 ┌─┐
 into West
 ┌─┐
 from North from Hilo
 to West to West
 400 35
 200 235
 0 435
 $0 \leq \leq 435$

Expand to see the derivation.

example.clp.cs - /Users/wtayson/Desktop/working-clp.cs - Atom

inspiration.clp.cs • example.clp.cs • review.clp.cs

```

ample
traffic
    region
    direction North West Hilo
    into   365  435  500
    out of 385  415  500

    include traffic

    from _r:region to :region/~/r ≥ 0

    into r += from _ to r
    out of r += from r to _

? into West
    into West = 435
    into West += from North to West
    into West += from Hilo to West

? from Hilo to West ≥ _
    from Hilo to West ≥ 135

```

Is and Keystrokes:

wer	<input type="button" value="↶"/>	<input type="button" value="↶"/>	<input type="button" value="≥"/>	<input type="button" value="↶"/>	<input type="button" value="↶"/>
porate	<input type="button" value="↶"/>				
uce	<input type="button" value="↶"/>				

clp.cs* 20:25 UTF-8 Constraint Logic Prop...

435
 \equiv
 into West
 \vdash
 from North from Hilo
 to West to West
 400 35
 200 235
 0 435
 $0 \leq \leq 435$

Having found an upper bound for `from Hilo to West`, can we improve its lower bound too?

example.clp.cs - /Users/wtayson/Desktop/working-clp.cs - Atom

```

inspiration.clp.cs • example.clp.cs • review.clp.cs
include traffic

from _r:region to :region/~_r ≥ 0

into _r += from _ to _r
out of _r += from _r to _

? into West
  into West = 435
  into West += from North to West
  into West += from Hilo to West

? from Hilo to West ≥ _
  from Hilo to West ≥ 135
    out of Hilo = 500
      into Hilo = 500
      out of Hilo = into Hilo
      out of Hilo += from Hilo to North
      out of Hilo += from Hilo to West
      from Hilo to North ≤ 365
        into North = 365
        into North += from West to North
        into North += from Hilo to North
        from West to North ≥ 0

Is and Keystrokes:
borate ↵
wer ↵
borate ↵

```

435
 into West
 from North from Hilo
 to West to West
 400 35
 200 235
 0 435
 $0 \leq \leq 435$

clp.cs* 21:21 UTF-8 Constraint Logic Prop...

How?

example.clp[cs] - /Users/wtayson/Desktop/working-clp[cs] - Atom

```

inspiration.clp[cs] • example.clp[cs] • review.clp[cs]
include traffic

from _r:region to :region/~/_r ≥ 0

into _r += from _ to _r
out of _r += from _r to _

? into West
  into West = 435
  into West += from North to West
  into West += from Hilo to West

? from Hilo to West ≥ _
  from Hilo to West ≥ 135
  out of Hilo = 500
  into Hilo = 500
  out of Hilo = into Hilo
  out of Hilo += from Hilo to North
  out of Hilo += from Hilo to West
  from Hilo to North ≤ 365
  into North = 365
  into North += from West to North
  into North += from Hilo to North
  from West to North ≥ 0

Is and Keystrokes:
borate
swer
borate
lpc[cs]* 21:21

```

UTF-8 Constraint Logic Prop...

```

graph TD
    N((North)) -- "435" --> W((West))
    W -- "385" --> N
    W -- "435" --> H((Hilo))
    H -- "500" --> W
    H -- "500" --> N
    N -- "365" --> H
    H -- "365" --> N
    H -- "365" --> W
    W -- "415" --> H
    H -- "415" --> W

```

Let's add these facts to our chart: `out of Hilo`, `into North`, and the rest.

example.clpccs - /Users/wtayson/Desktop/working-clpccs - Atom

```

inspiration.clpccs • example.clpccs • review.clpccs
include traffic

from _r:region to :region/~_r ≥ 0

into _r += from _ to _r
out of _r += from _r to _

? into West
  into West = 435
  into West += from North to West
  into West += from Hilo to West

? from Hilo to West ≥ _
  from Hilo to West ≥ 135
    out of Hilo = 500
      into Hilo = 500
      out of Hilo = into Hilo
      out of Hilo += from Hilo to North
      out of Hilo += from Hilo to West
      from Hilo to North ≤ 365
        into North = 365
        into North += from West to North
        into North += from Hilo to North
        from West to North ≥ 0

Is and Keystrokes:
borate ↵
swr ↵
borate ↵
lpcscs* 21:21

```

UTF-8 Constraint Logic Prop...

Where does the lower bound of 135 come from? It follows from there being an upper bound on `from Hilo to North`, which in turn comes from our zero bound on `from West to North`.

example.clpcts - /Users/wtayson/Desktop/working-clpcts - Atom

inspiration.clpcts • example.clpcts • review.clpcts

```

ample
traffic
    region
    direction North West Hilo
    into 365 435 200
    out of 385 415 200

    include traffic

    from _r:region to :region ~_r ≥ 0

    into _r += from _ to _r
    out of _r += from _r to _

    ? into West
        into West = 435
        into West += from North to West
        into West += from Hilo to West

    ? from Hilo to West ≥ _
        from Hilo to West ≥ 50
        into West = 435
        into West += from North to West
        into West += from Hilo to West
        from North to West ≤ 385

Is and Keystrokes:
    wer
    porate
    wer

clpcts* 5:27 (3)           UTF-8 Constraint Logic Prop...

```

What if the traffic into Hilo were different? Say, 200.

example.clp[cs] - /Users/wtayson/Desktop/working-clp[cs] - Atom

inspiration.clp[cs] example.clp[cs] review.clp[cs]

out of 385 415 200

include traffic

from _r:region to :region/ $\sim_r \geq 0$

into _r += from _ to _r
out of _r += from _r to _

? into West
into West = 435
into West += from North to West
into West += from Hilo to West

? from Hilo to West \geq
from Hilo to West \geq 50
into West = 435
into West += from North to West
into West += from Hilo to West
from North to West \leq 385
out of North = 385
out of North += from North to West
out of North += from North to Hilo
from North to West \geq 0

Is and Keystrokes:

wer ↵
borate ↵
wer ↵

clp[cs]* 5:27 (3) UTF-8 Constraint Logic Prop...

The diagram illustrates the flow of data from North to West and from North to Hilo. It features two main nodes: 'from North to West' and 'from North to Hilo'. The 'from North to West' node has an incoming edge from 'North' labeled 'from North to West' with a value of 385. It has two outgoing edges: one to 'West' labeled 'to West' with a value of 435, and another to 'Hilo' labeled 'to West' with a value of 200. The 'from North to Hilo' node has an incoming edge from 'North' labeled 'from North to Hilo' with a value of 365. It has two outgoing edges: one to 'West' labeled 'from West to Hilo' with a value of 415, and another to 'Hilo' labeled 'out of West' with a value of 200.

Not only does the lower bound drop, `into West` becomes a tight constraint and `into Hilo` becomes a loose constraint. Back in the chart, switching `into Hilo` invalidates the previous inference and updates `out of Hilo`.

example.clp[cs] - /Users/wtayson/Desktop/working-clp[cs] - Atom

inspiration.clp[cs] example.clp[cs] review.clp[cs]

out of 385 415 200

include traffic

from _r:region to :region/ $\sim_r \geq 0$

into _r += from _ to _r
out of _r += from _r to _

? into West
into West = 435
into West += from North to West
into West += from Hilo to West

? from Hilo to West \geq
from Hilo to West \geq 50
into West = 435
into West += from North to West
into West += from Hilo to West
from North to West \leq 385
out of North = 385
out of North += from North to West
out of North += from North to Hilo
from North to West \geq 0

Is and Keystrokes:

ower ↵
borate ↵
wer ↵

clp[cs]* 5:27 (3) UTF-8 Constraint Logic Prop...

435
[]
into West

385
[]
out of North \leq 385

200
[]
out of Hilo \leq 50

0 \leq

into Hilo \times
[]

365
[]
into North \times

415
[]
out of West \leq

Our new bound is not as strong – having come around from the other side.

example.clpcts - /Users/wtayson/Desktop/working-clpcts - Atom

inspiration.clpcts • example.clpcts • review.clpcts

```

ample
traffic
    region
    direction North West Hilo
    into 365 435 200
    out of 385 415 200

    include traffic

    from _r:region to :region/~/r ≥ 0

    into r += from _ to r
    out of r += from _ to _

    ?= bounds (from Hilo to West)
    50 ≤ 200

```

Is and Keystrokes:

ower	↶
te	⌘V
kspace	? = ⌘B ⌘U ⌘N ⌘D ⌘S ⌘(⌘)

clpcts* 15:31 UTF-8 Constraint Logic Prop...

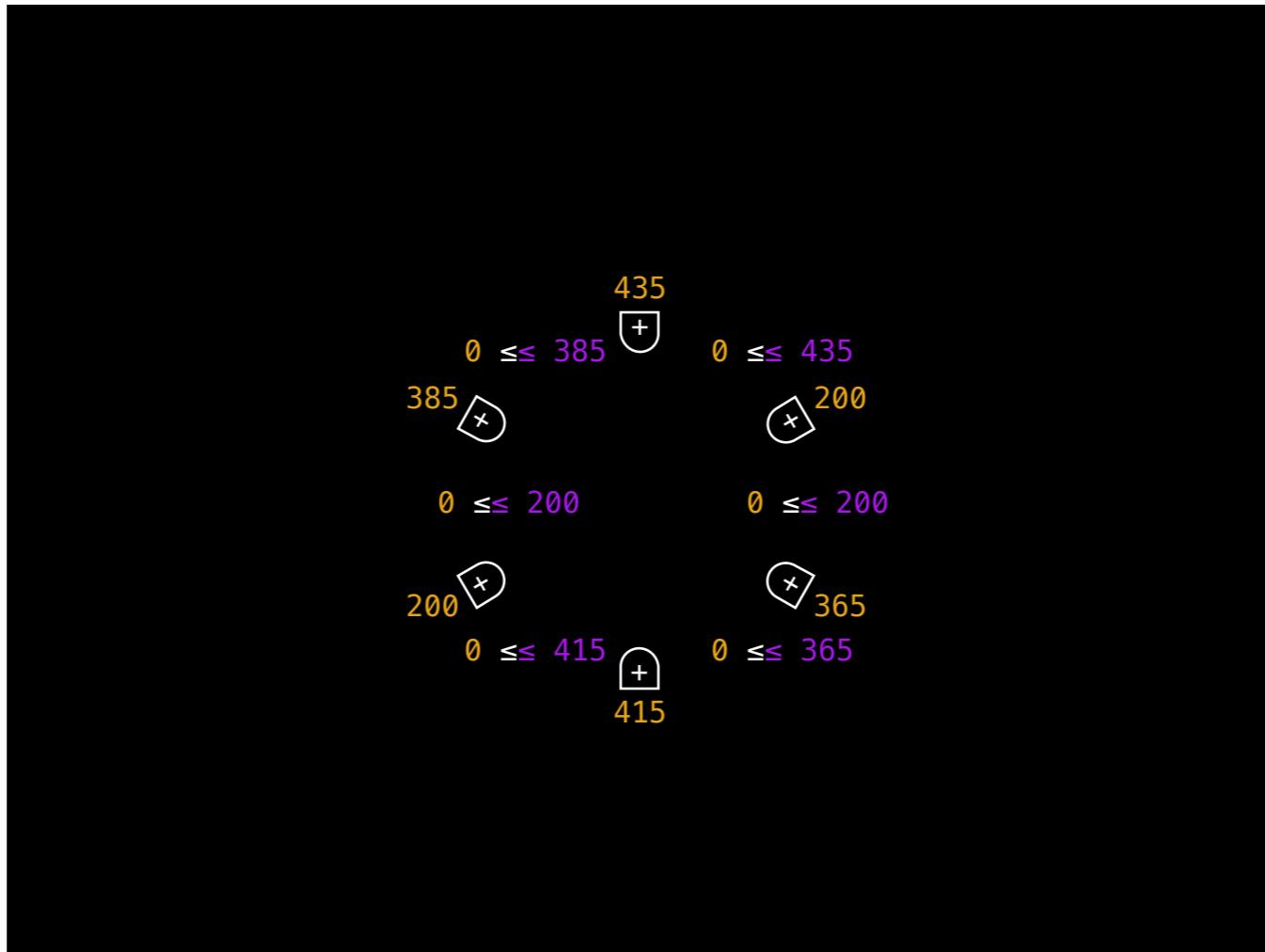
Let's ask about both the upper and lower bounds at once: the closed interval from 50 to 200. Instead of working backwards, let's see how the zero lower bounds might propagate forward.



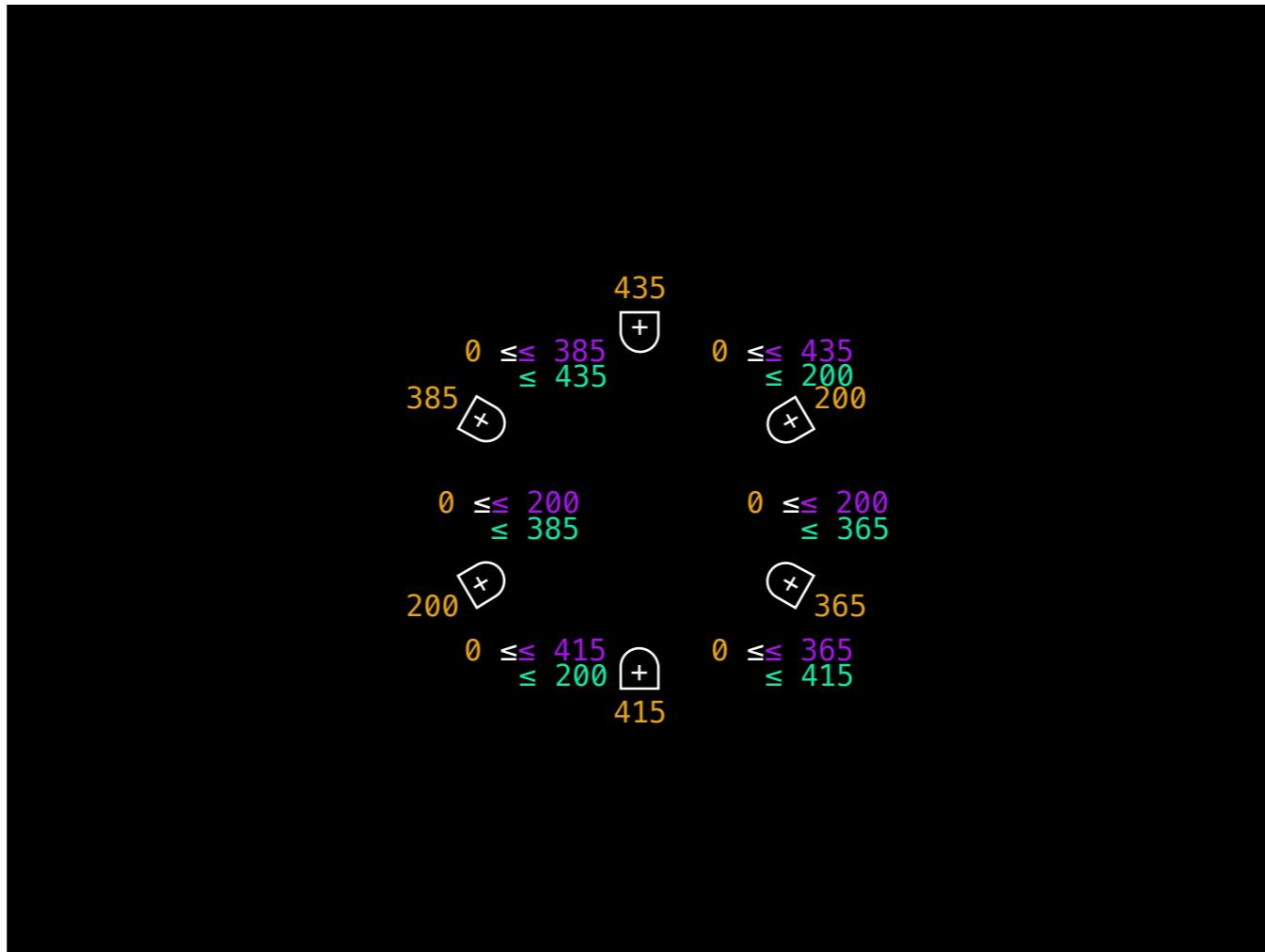
Start by stripping down to the essential relations.

$$\begin{array}{rcl} & 435 & \\ \square + & & \\ 0 \leq & & 0 \leq \\ 385 & \times & 200 \\ 0 \leq & & 0 \leq \\ 200 & \times & 365 \\ 0 \leq & & 0 \leq \\ & 415 & \end{array}$$

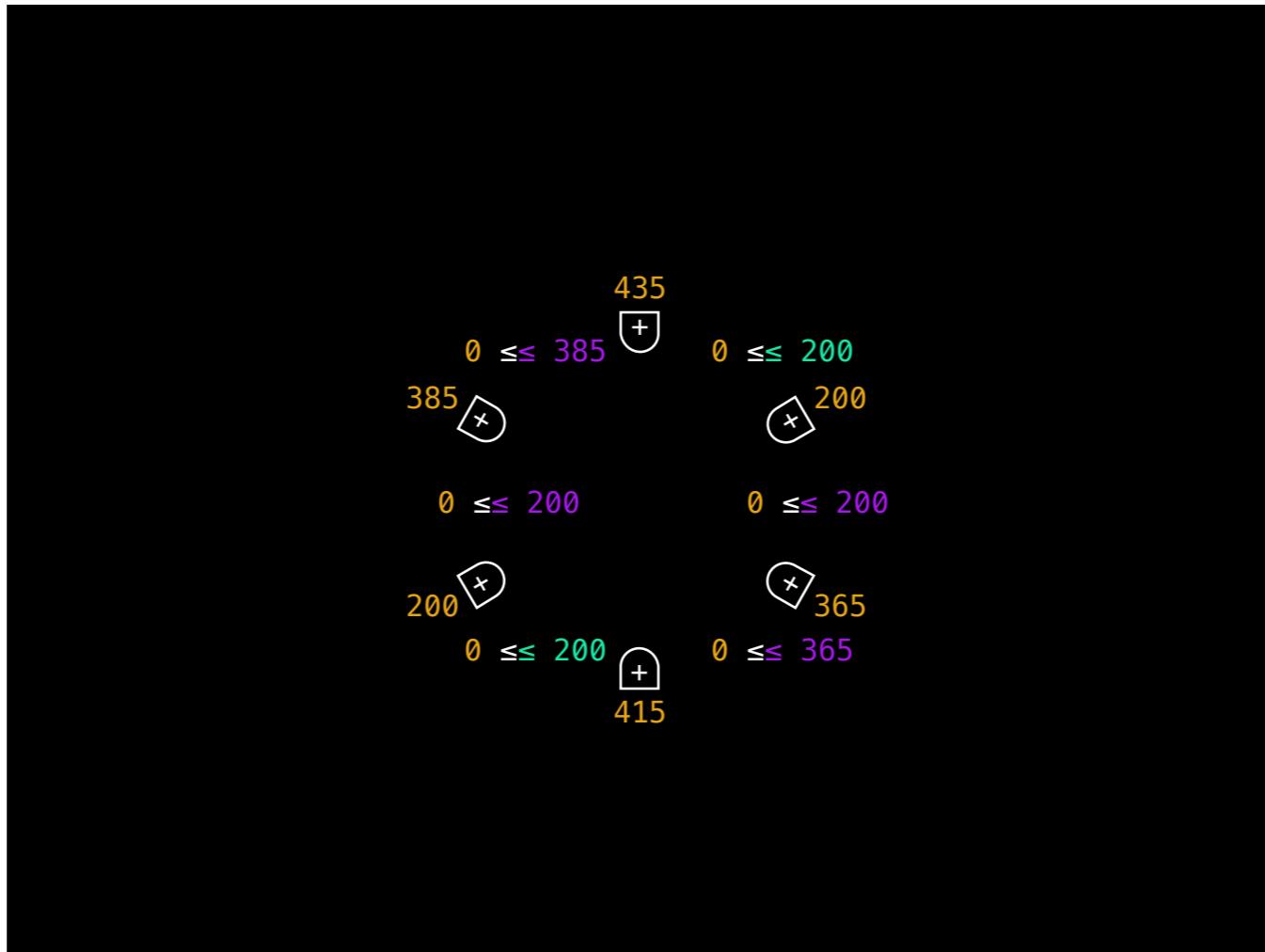
Add in the constraints.



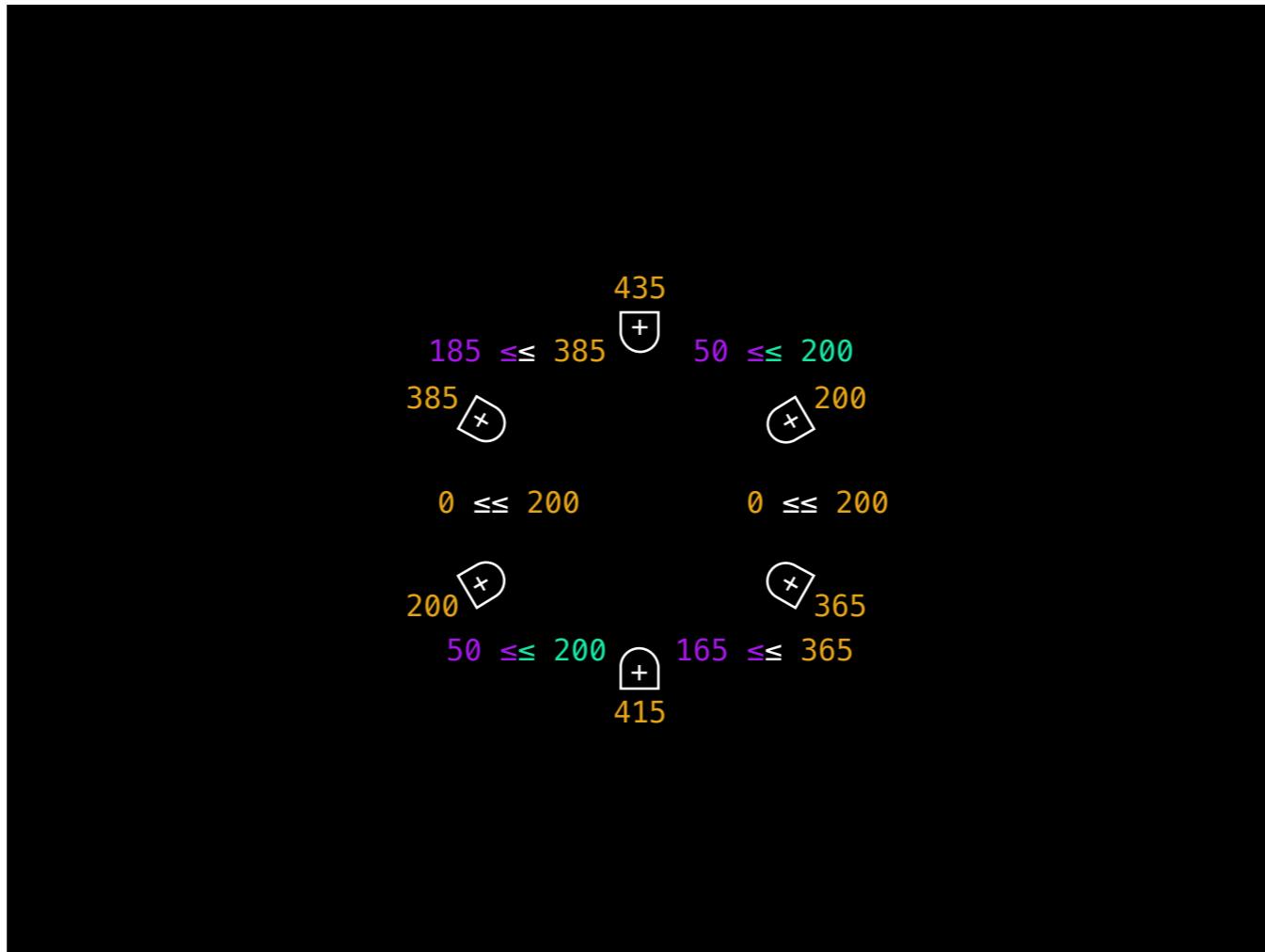
Propagate clockwise,



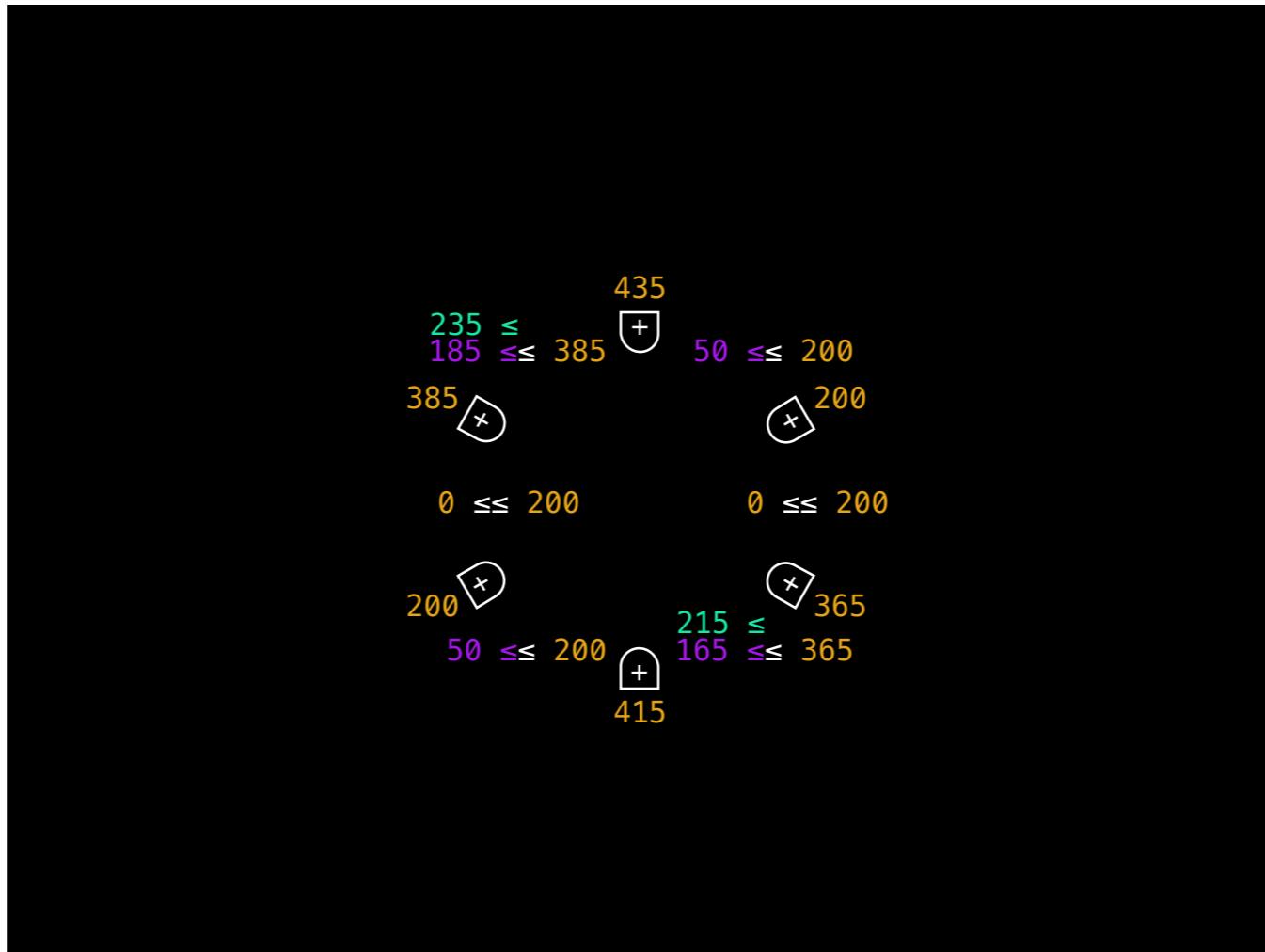
propagate counterclockwise.



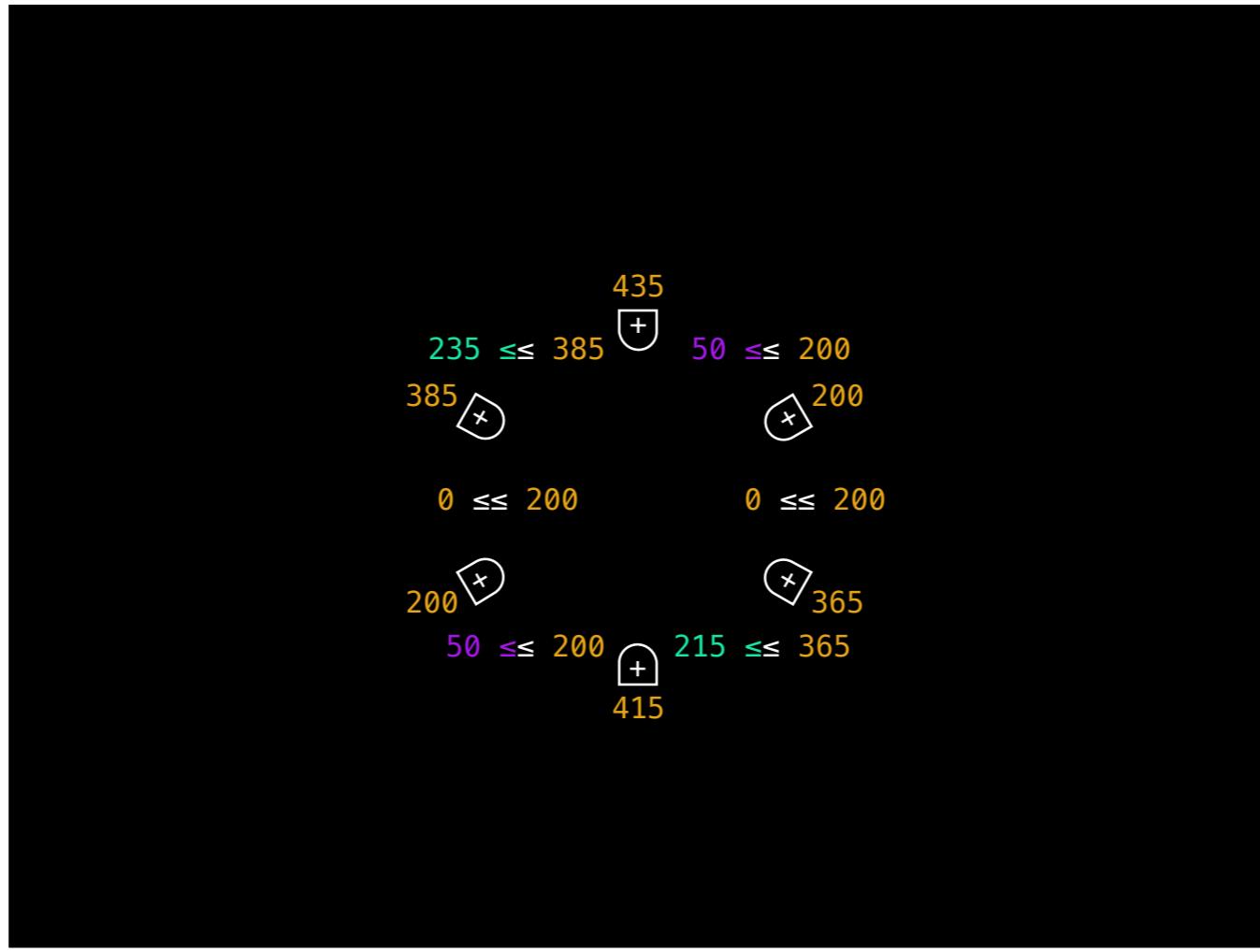
Reduce by taking the better bound in each cell.



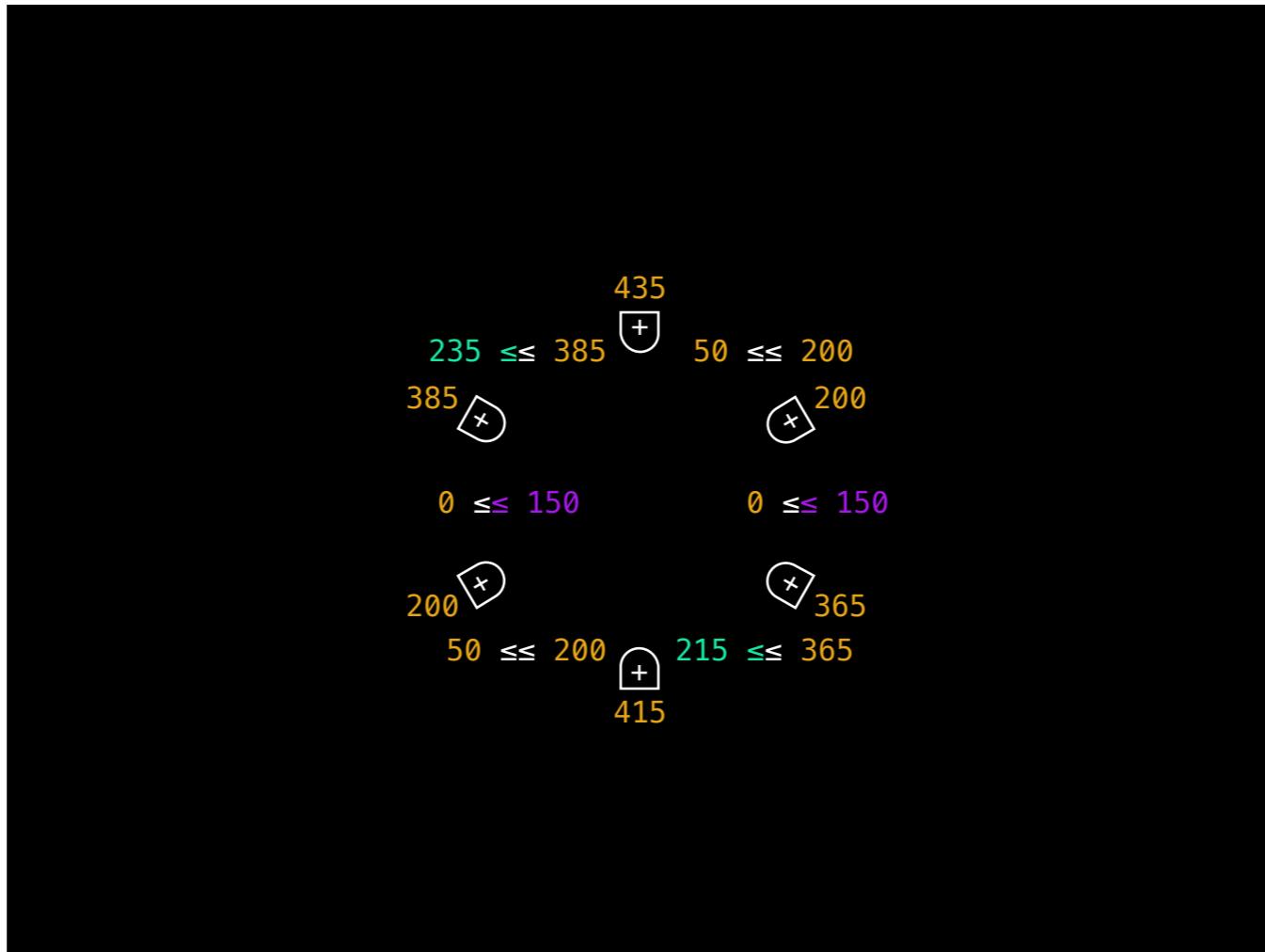
Another go. Propagate clockwise.



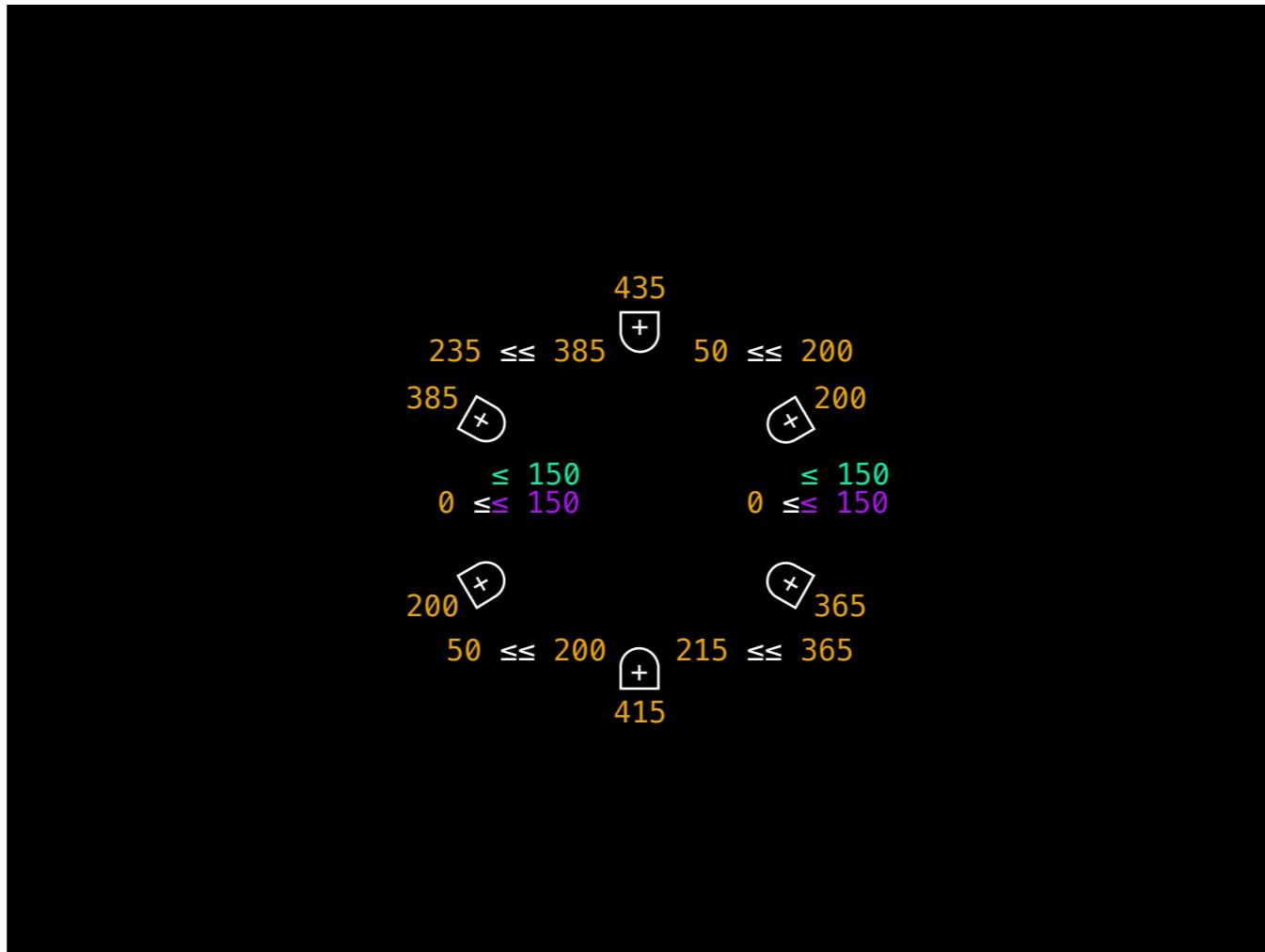
Propagate counterclockwise.



Merge again by picking the better bound.



Again, again. Clockwise.



Counterclockwise.



At last the two waves cancel. And that is forward propagation. Let's bring back the editor.

The screenshot shows the Atom code editor with three tabs open: 'inspiration.clpcs', 'example.clpcs' (which is the active tab), and 'review.clpcs'. The 'example.clpcs' tab contains the following CLP program:

```
1 Example
2 traffic
3         region
4         direction North West Hilo
5         into    365   435  50
6         out of   385   415  50
7
8 include traffic
9
10 from _r:region to :region/~/r ≥ 0
11
12 into _r += from _ to _r
13 out of _r += from _r to _
14
15 ?= bounds (from Hilo to West)
16 50 ≤ 50
```

Below the code editor, there is a 'Commands and Keystrokes' section with the following entries:

- clpcs:answer ↵ 5 0
- clpcs:answer ↵
- core:paste ⌘V

At the bottom of the screen, the status bar displays 'example.clpcs* 5:26 (2)' and 'UTF-8 Constraint Logic Prop...'.
A vertical black bar is visible on the right side of the window.

We get equality at 50.

example.clpcs - /Users/wtayson/Desktop/working-clpcs - Atom

inspiration.clpcs example.clpcs review.clpcs

```
1 Example
2 traffic
3     region
4         direction North West Hilo
5         into    365   435  49
6         out of   385   415  49
7
8     include traffic
9
10    from _r:region to :region/~/r ≥ 0
11
12    into _r += from _ to _r
13    out of _r += from _r to _
14
15    ?= bounds (from Hilo to West)
16    50 ≤ 49
```

Commands and Keystrokes:

clpcs:answer ↵
4 9

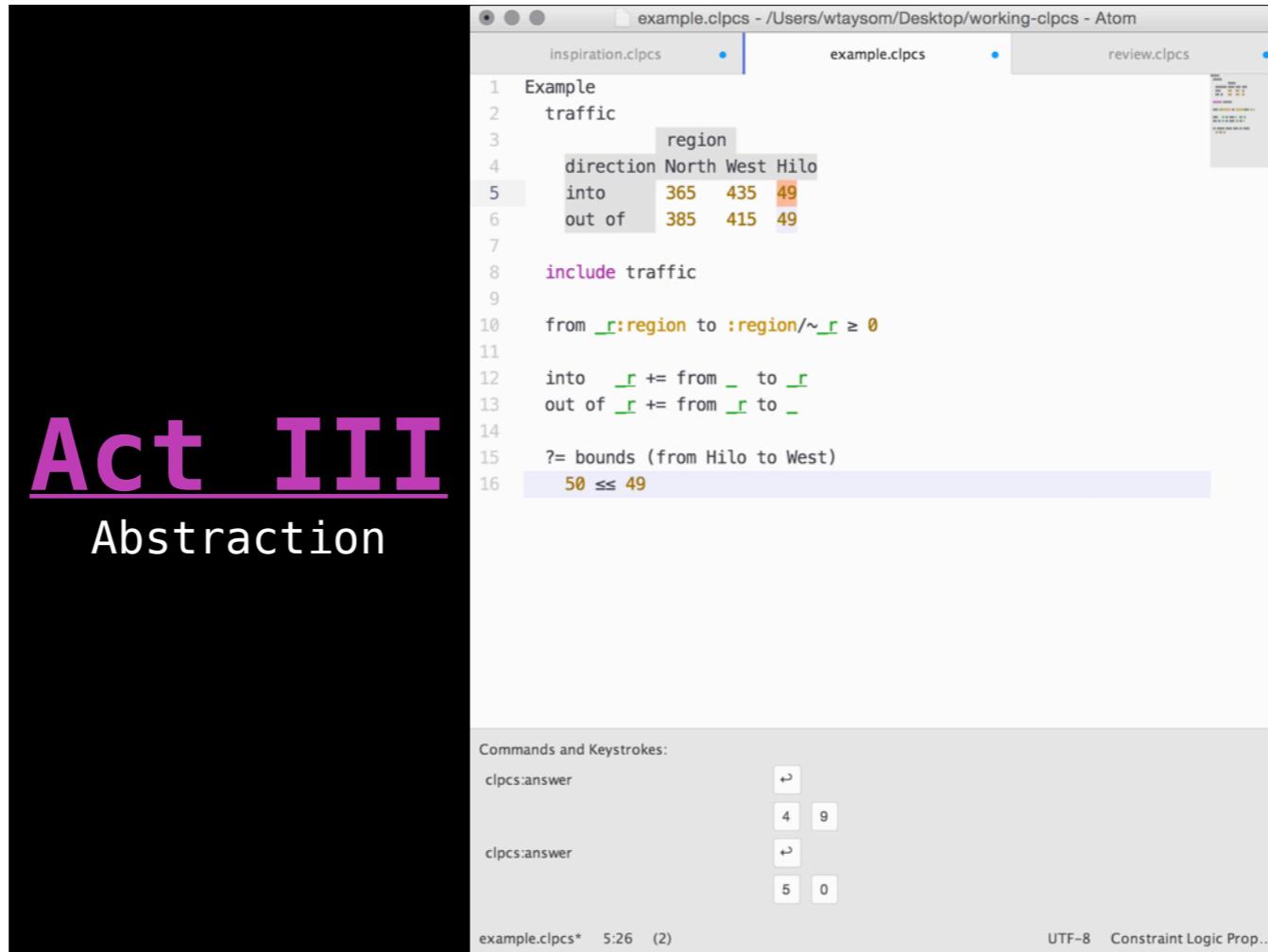
clpcs:answer ↵
5 0

example.clpcs* 5:26 (2) UTF-8 Constraint Logic Prop...

Below 50, we have conflict.

Act III

Abstraction



```
1 Example
2 traffic
3 region
4 direction North West Hilo
5 into 365 435 49
6 out of 385 415 49
7
8 include traffic
9
10 from _r:region to :region/~/r ≥ 0
11
12 into _r += from _ to _r
13 out of _r += from _r to _
14
15 ?= bounds (from Hilo to West)
16 50 ≤ 49
```

Commands and Keystrokes:

clpcts:answer	4	9
clpcts:answer	5	0

example.clpcs* 5:26 (2) UTF-8 Constraint Logic Prop...

To see how these what-if scenarios relate, let's climb up the ladder of abstraction.

The screenshot shows the Atom code editor with three tabs open: 'inspiration.clpcs', 'example.clpcs' (which is the active tab), and 'review.clpcs'. The 'example.clpcs' tab contains the following CLP CS code:

```
1 Example
2 traffic
3     region
4         direction North West Hilo
5             into    365  435  49
6             out of   385  415  49
7
8     include traffic
9
10    from _r:region to :region/~/r ≥ 0
11
12    into _r += from _ to _r
13    out of _r += from _r to _
14
15    ?= bounds (from Hilo to West)
16        50 ≤ 49
17
18    scenario = bounds (Example from Hilo to West)
```

The line '50 ≤ 49' is highlighted with a light blue background. At the bottom of the editor, there is a 'Commands and Keystrokes' section with various keyboard shortcut keys and their descriptions.

For our scenario, we're interested in the bounds on `from Hilo to West`.

The screenshot shows the Atom text editor interface with three tabs open: 'inspiration.clpcts', 'example.clpcts' (which is the active tab), and 'review.clpcts'. The code in the 'example.clpcts' tab is as follows:

```
1 Example
2 traffic
3     region
4         direction North West Hilo
5             into    365  435  49
6             out of   385  415  49
7
8     include traffic
9
10    from _r:region to :region/~/r >= 0
11
12    into _r += from _ to _r
13    out of _r += from _r to _
14
15    ?= bounds (from Hilo to West)
16        50 <= 49
17
18 scenario = bounds (from Hilo to West) :-
19     . include Example
```

Below the code editor, there is a 'Commands and Keystrokes' section with the following mappings:

clpcts:group-by	^G
editor:select-to-beginning-of-word	⌃⇧←
editor:move-to-beginning-of-word	⌃←
editor:move-to-beginning-of-word	⌃←

At the bottom of the editor window, it shows 'example.clpcts*' and the time '19:19' on the left, and 'UTF-8 Constraint Logic Prop...' on the right.

Include the `Example` module into the body of `scenario` for unqualified reference. The dot here allows a rule to have local facts. So within this rule all Example facts hold.

```
example.clp.cs - /Users/wtayson/Desktop/working-clp.cs - Atom
inspiration.clp.cs • example.clp.cs • review.clp.cs

11
12     into _r += from _ to _r
13     out of _r += from _r to _
14
15     ?= bounds (from Hilo to West)
16     50 <= 49
17
18 scenario = bounds (from Hilo to West) :-
19     . include Example
20     Example
21         traffic
22             region
23                 direction North West Hilo
24                 into    365   435   49
25                 out of   385   415   49
26
27     include traffic
28
29     from _r:region to :region/~_r >= 0
30
31     into _r += from _ to _r
32     out of _r += from _r to _
33
34     ?= bounds (from Hilo to West)
35     50 <= 49

Commands and Keystrokes:
clp.cs:elaborate           ^E
clp.cs:group-by             ^G
editor:select-to-beginning-of-word  ⌘⇧←
editor:move-to-beginning-of-word    ⌘←

example.clp.cs* 20:3          UTF-8 Constraint Logic Prop...
```

Take a look inside. Here we see the definition of the Example table. I use a gray background to suggest that the original is kept elsewhere.

The screenshot shows a window titled "example.clp.cs - /Users/wtayson/Desktop/working-clp.cs - Atom". The code editor displays a file named "example.clp.cs" with the following content:

```
11
12     into r += from _ to r
13     out of r += from r to _
14
15     ?= bounds (from Hilo to West)
16     50 ≤ 49
17
18 scenario = bounds (from Hilo to West) :-
19     . include Example
20     Example
21         traffic
22             region
23                 direction North West Hilo
24                 into    365   435 300
25                 out of   385   415 300
26
27     include traffic
28
29     from r:region to :region/~r ≥ 0
30
31     into r += from _ to r
32     out of r += from r to _
33
34     ?= bounds (from Hilo to West)
35     50 ≤ 300
```

Below the code editor, there is a "Commands and Keystrokes" section with the following entries:

clp.cs:answer	↻
	3 0 0
clp.cs:elaborate	^E
clp.cs:group-by	^G

At the bottom of the interface, it shows "example.clp.cs* 24:29 (3)" and "UTF-8 Constraint Logic Prop...".

Beyond just viewing the included table, we can customize it. When we override the included version, the original remains unchanged.

The screenshot shows a window titled "example.clp.cs - /Users/wtayson/Desktop/working-clp.cs - Atom". The code editor displays a file named "example.clp.cs" with the following content:

```
11
12     into r += from _ to r
13     out of r += from r to _
14
15     ?= bounds (from Hilo to West)
16     50 ≤ 49
17
18 scenario into_Hilo = bounds (from Hilo to West) :-  

19     . include Example
20     Example
21         traffic
22             region
23                 direction North West Hilo
24                 into 365 435 into_Hilo
25                 out of 385 415 into_Hilo
26
27         include traffic
28
29         from r:region to :region/~r ≥ 0
30
31         into r += from _ to r
32         out of r += from r to _
33
34     ?= bounds (from Hilo to West)
35     max 50 (into_Hilo - 365) ≤ min into_Hilo 435
```

Below the code editor, there is a "Commands and Keystrokes" section:

clp.cs:parameterize	⌘P
clp.cs:answer	↷
	3 0 0
clp.cs:elaborate	^E

At the bottom of the interface, it shows "example.clp.cs* 24:36 (10)" and "UTF-8 Constraint Logic Prop...".

More importantly, we can parameterize over a term.

```
1 Example
2 traffic
3     region
4         direction North West Hilo
5             into    365  435  49
6             out of   385  415  49
7
8 include traffic
9
10 from _r:region to :region/~/r >= 0
11
12 into _r += from _ to _r
13 out of _r += from _r to _
14
15 ?= bounds (from Hilo to West)
16 50 <= 49
17
18 scenario _into_Hilo = bounds (from Hilo to West) ;-
19 . include Example
20 : into Hilo = _into_Hilo
```

Commands and Keystrokes:

clp:abbreviate	^R
core:move-down	↓
clp:parameterize	⌘P
clp:answer	↶

example.clp* 20:26 UTF-8 Constraint Logic Prop...

Close up the module. See that we're simply overriding a fact. Between an overriding diff and the fully woven version with all customizing patches integrated, some things are easy to see in the derivative, some are clearer in the integral. In the future, we freely switch between the two.

example.clpcs - /Users/wtayson/Desktop/working-clpcs - Atom

inspiration.clpcs • example.clpcs • review.clpcs

```
1 Example
2 traffic
3     region
4         direction North West Hilo
5             into 365 435 49
6             out of 385 415 49
7
8     include traffic
9
10    from _r:region to :region/~/_r ≥ 0
11
12    into _r += from _ to _r
13    out of _r += from _r to _
14
15    ?= bounds (from Hilo to West)
16        50 ≤ 49
17
18    scenario into_Hilo = bounds (from Hilo to West) ;-
19        . include Example
20        : into Hilo = into_Hilo
21
22    ?= scenario 700
23        335 ≤ 435
```

Commands and Keystrokes:

clpcs:answer	↻
	7
core:backspace	⌫
core:move-left	←

example.clpcs* 22:16 UTF-8 Constraint Logic Prop...

Try out some scenarios.

example.clpcs - /Users/wtayson/Desktop/working-clpcs - Atom

```

inspiration.clpcs          example.clpcs          review.clpcs
19 . include example
20 : into Hilo = into Hilo
21
22 ?= scenario (0 ≤ 900)
23 50 ≤ **** ≤ 0
24 50 ≤ = ≤ 50
25 50 ≤ ----- ≤ 100
26 50 ≤ ----- ≤ 150
27 50 ≤ ----- ≤ 200
28 50 ≤ ----- ≤ 250
29 50 ≤ ----- ≤ 300
30 50 ≤ ----- ≤ 350
31 50 ≤ ----- ≤ 400
32 85 ≤ ----- ≤ 435
33 135 ≤ ----- ≤ 435
34 185 ≤ ----- ≤ 435
35 235 ≤ ----- ≤ 435
36 285 ≤ ----- I ≤ 435
37 335 ≤ ----- ≤ 435
38 385 ≤ ----- ≤ 435
39 435 ≤ = ≤ 435
40 485 ≤ *** ≤ 435
41 535 ≤ ***** ≤ 435
42 585 ≤ ***** ≤ 435
43 635 ≤ ***** ≤ 435

```

Commands and Keystrokes:

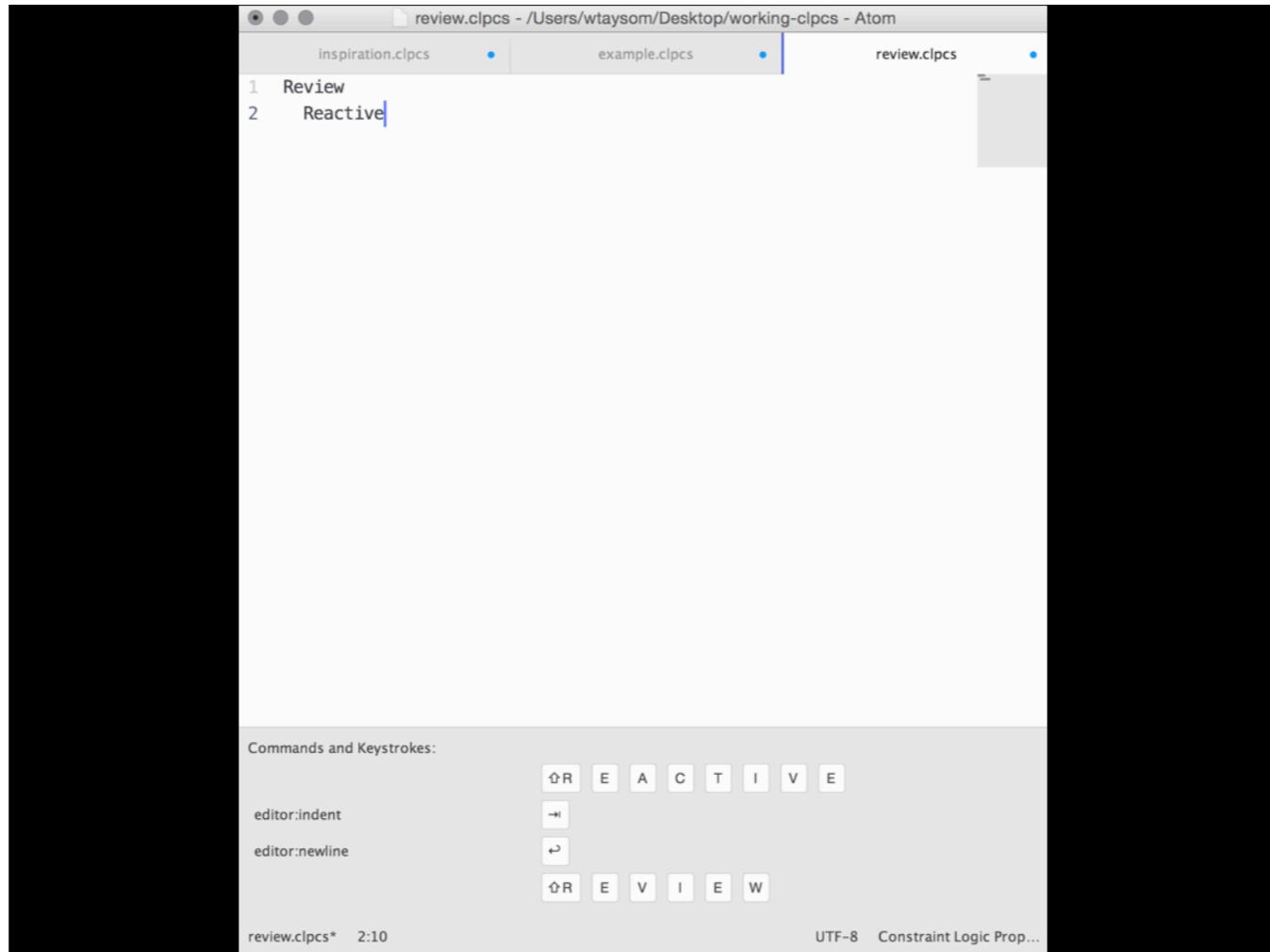
- clpcs:answer
- editor:select-to-beginning-of-word
- clpcs:answer

example.clpcs* 22:22 UTF-8 Constraint Logic Prop...

And finally, let's ask about a whole interval at once. The answer is an interval of intervals, charted crudely: dashes for standard intervals, stars for conflicted intervals, and equals sign for singletons. Although we defined the scenario with scalars in mind, we can apply it to intervals by mapping the scenario across a selection of values from the interval. Similarly, we can automatically lift other functors: sets, lists, probability distributions, time varying signals.

A generalized propagation network separates the shape of the network from the structure of data that fills its cells.

A generalized propagation network separates the shape of the network from the structure of data that fills its cells. It offers a new kind of abstraction with more potential than I'm prepared to explore today. Instead let's review what we have explored.



Being reactive is par for the course at this workshop. For every wibble, have related things wobble. Except we don't actually want every wibble to wobble.

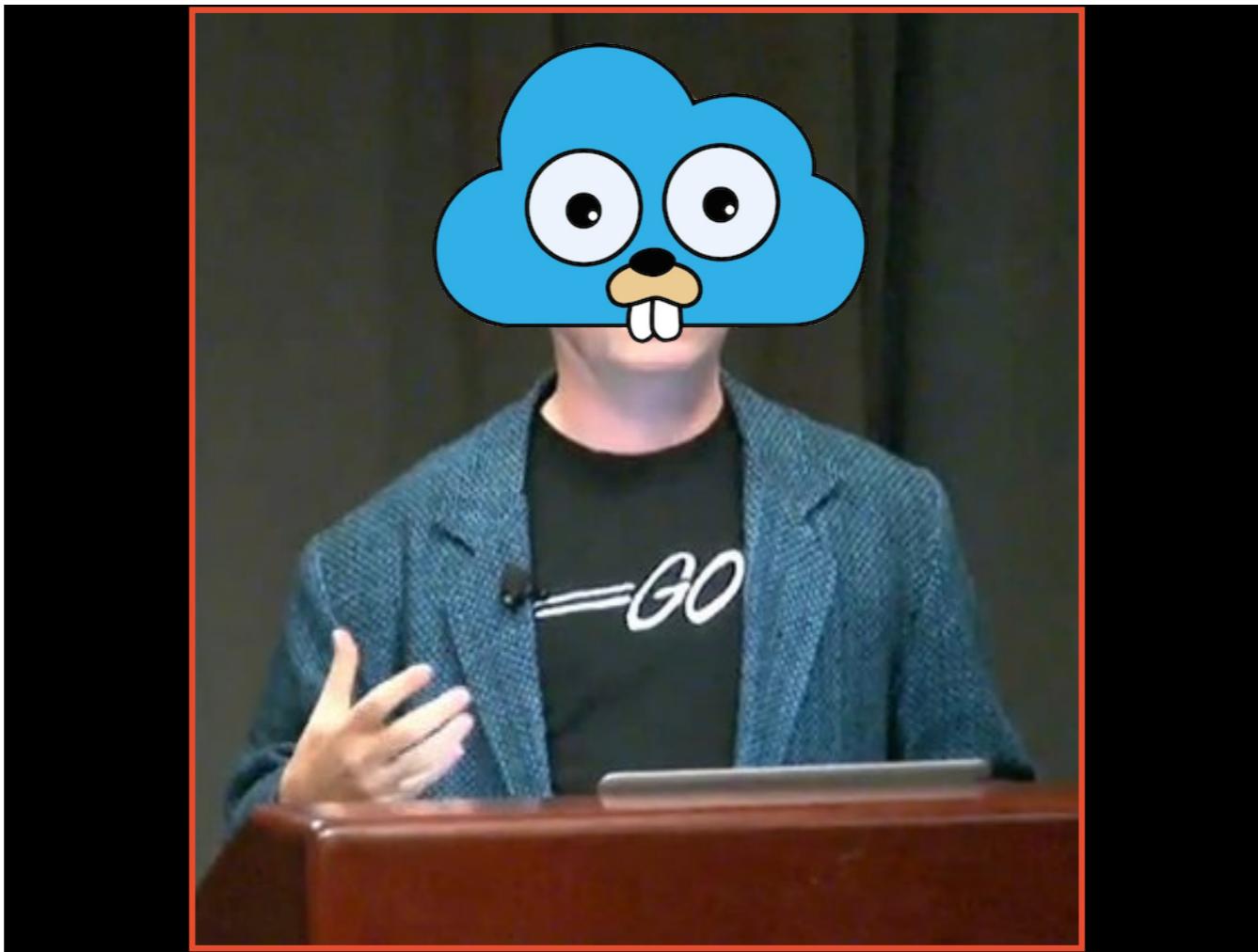


While in the middle of typing a question or making a change, a system shouldn't interrupt. Every messenger app seems to experiment with sharing your keystrokes until they realize that people put their thoughts together only wanting to send after they hit return. So in the example, I marked uncommitted changes with a light yellow background.

Some questions don't have an immediate answer. It takes time to think things through. Even with a datacenter of answers, it takes time to look them up. Build with these constraints in mind. First and always: keep the interface responsive. Process in the background. Report progress. Abort if asked. Be able to pause a process, inspect it, rewind.



Sounds like a list of operating system services. I'm not sure why process APIs remain so true to their 1970s roots.



Perhaps with the web, all effort drifted away up into the cloud.



A topic for another time.

review.clpcs - /Users/wtaysom/Desktop/working-clpcs - Atom

inspiration.clpcs example.clpcs review.clpcs

1 Review
2 Reactive Accumulative

Commands and Keystrokes:

editor:indent

review.clpcs*

2:23

UTF-8 Constraint Logic Prop...

Time. A funny dimension. A good place to keep bugs. Functional programming goes to show you can get by without time some of the time. When you do need time, be explicit about it:



"IO using monads doesn't look like purely functional programming, and it shouldn't because it isn't."

A basic spreadsheet is pure: every formula a pure function. But the people who make them are not. We build things over time, their structure at any moment being just a snapshot.

In order to work with time, we have to step out of time. I don't mean metaphysically, just that the order of events in a story doesn't need to match the order in which they were written. You can do it backwards: the resolved becomes unresolved, clues skitter away to their hiding places, foreshadows become regular back-shadows.

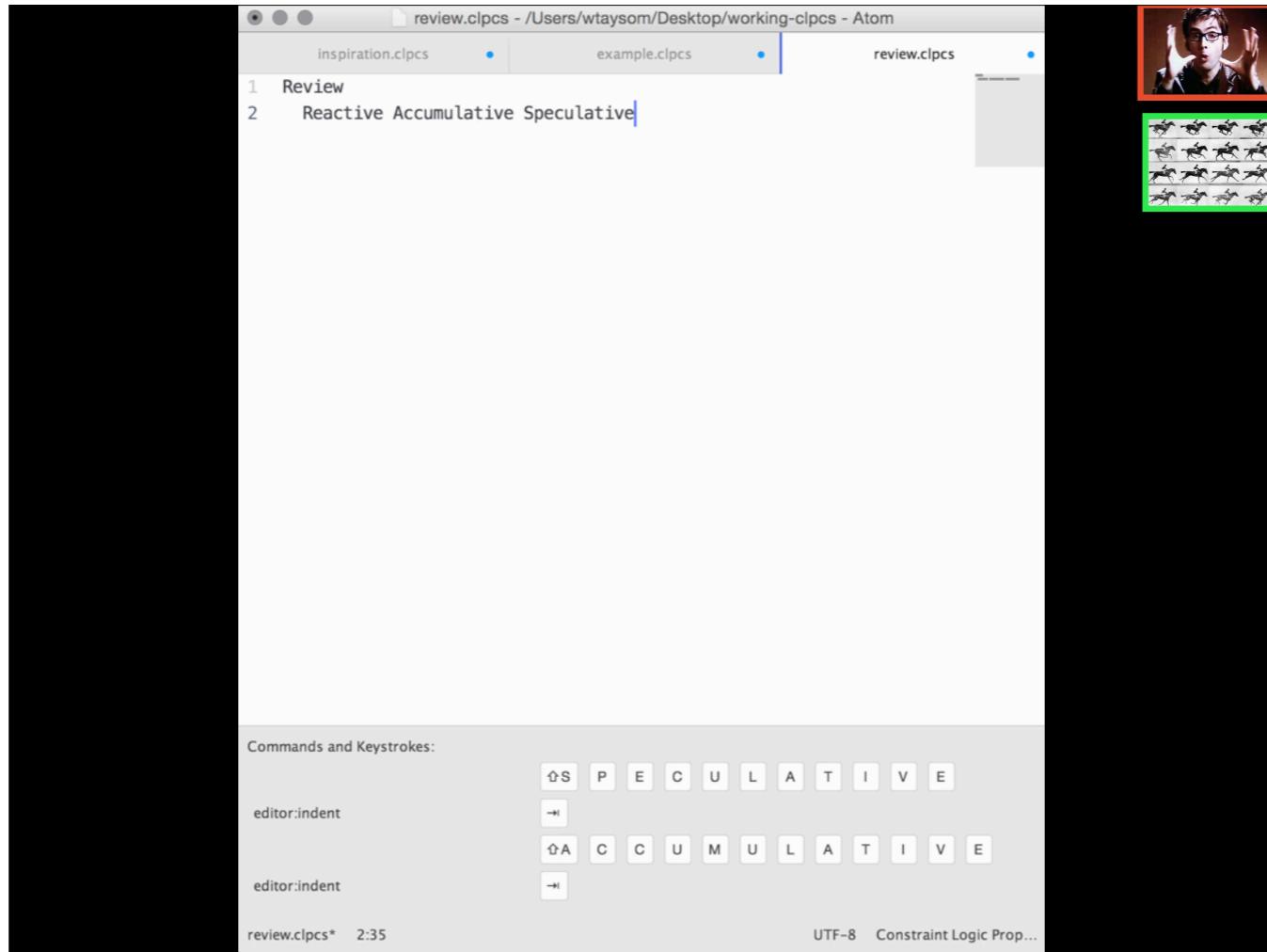


When reading, when interpreting, begin at the beginning, and go on till you come to the end: then stop. With structured programming, the key idea was to have control flow match code flow: first instruction to first line, last instruction to last line with some skipped and repeated lines in the middle. (Admittedly, that never quite worked out. The graphical user interface may be partly to blame.)



Still in our example, facts accumulated from top to bottom. A question asked early-on ignored spoilers that came later. Time unfolds strung along the vertical.

Ask a question twice, you'll get the same answer unless there are new facts written in the middle. If we quickly alternate between the two, the effect is to highlight the change. If we have a long list of answers, scrolling from one to the next gives us animation.

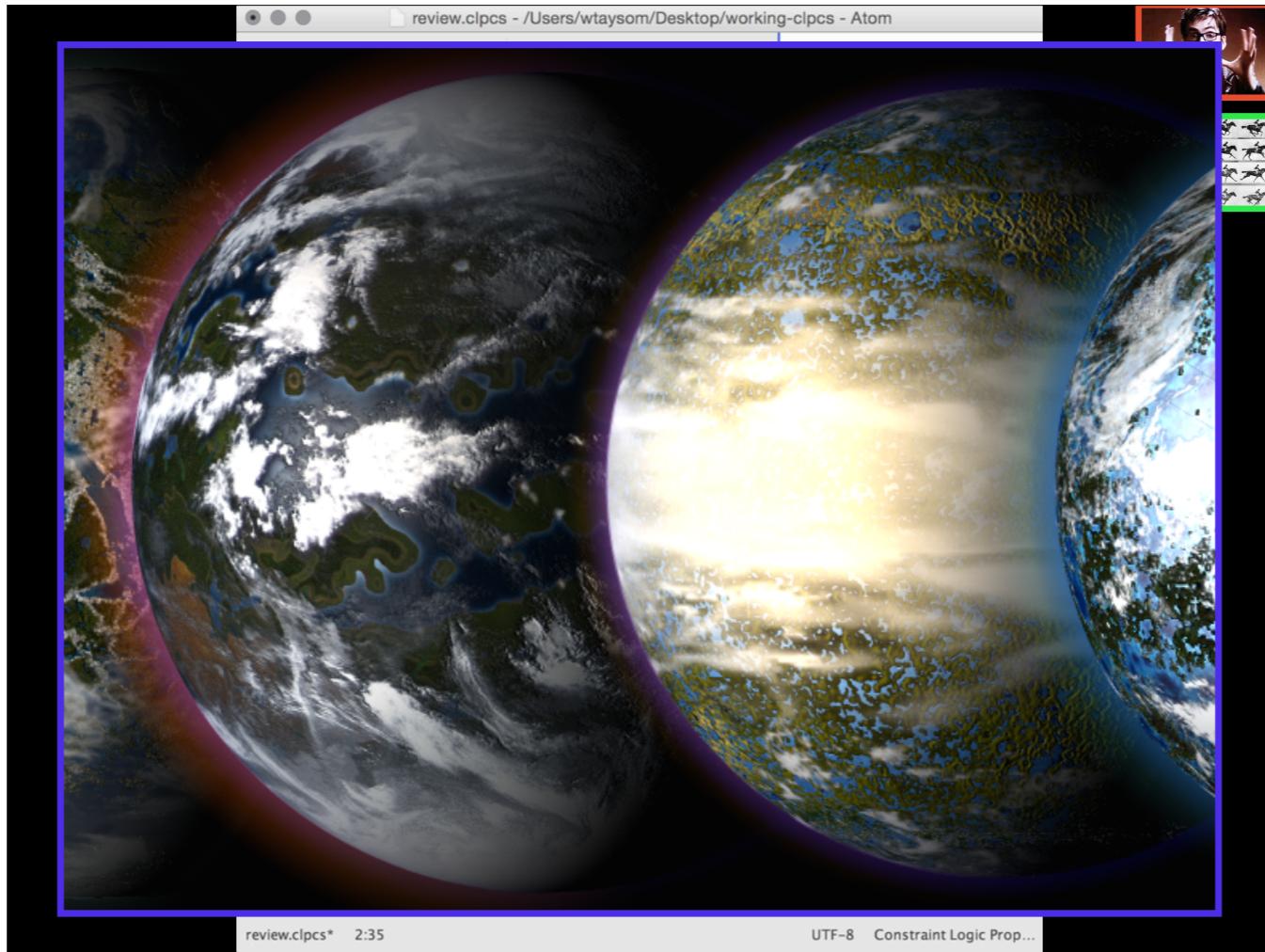


Analysts know some animator tricks. In our traffic table, we tried several values for `into Hilo` seeing changes in slow motion. In Excel to make transitions immediate, you start with a sheet. Make a copy. Update cells. Make another copy. Update cells. And so on. Switch between tabs in rapid succession:



the workbook becomes a flip book.

Of course the copies have no link back to the master, so it's easy to get out of sync. In the example, including a table made the link. Overriding customized it. Scope and variables abstracted over it.



Then with auto-lifting propagation, we speculated over a range of possible worlds, possible futures. Layer on a user driven selection function: animation becomes interaction.

review.clpcs - /Users/wtaysom/Desktop/working-clpcs - Atom

inspiration.clpcs example.clpcs review.clpcs

1 Review
2 Reactive Accumulative Speculative
3 Modular

Commands and Keystrokes:

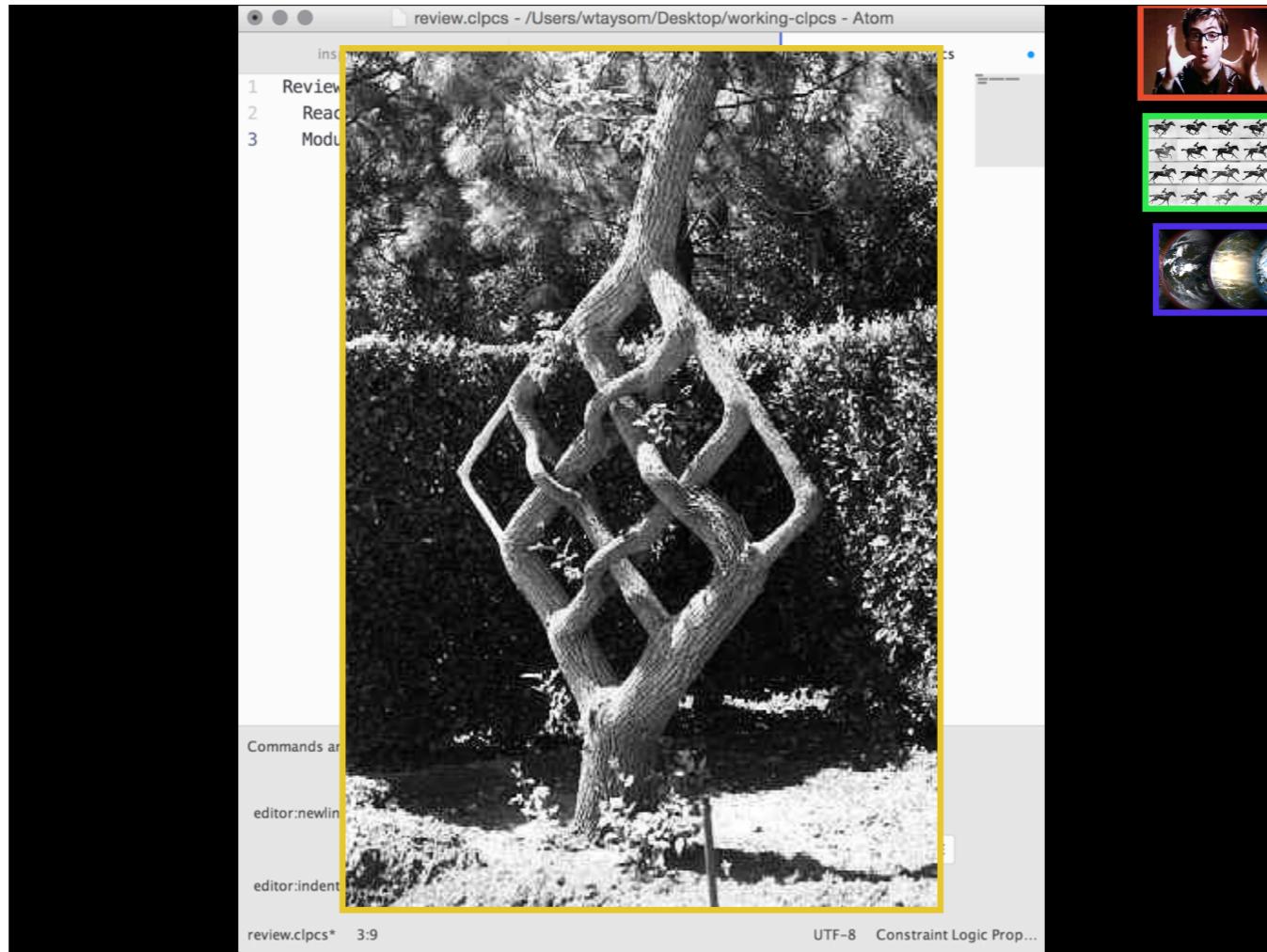
editor:newline

editor:indent

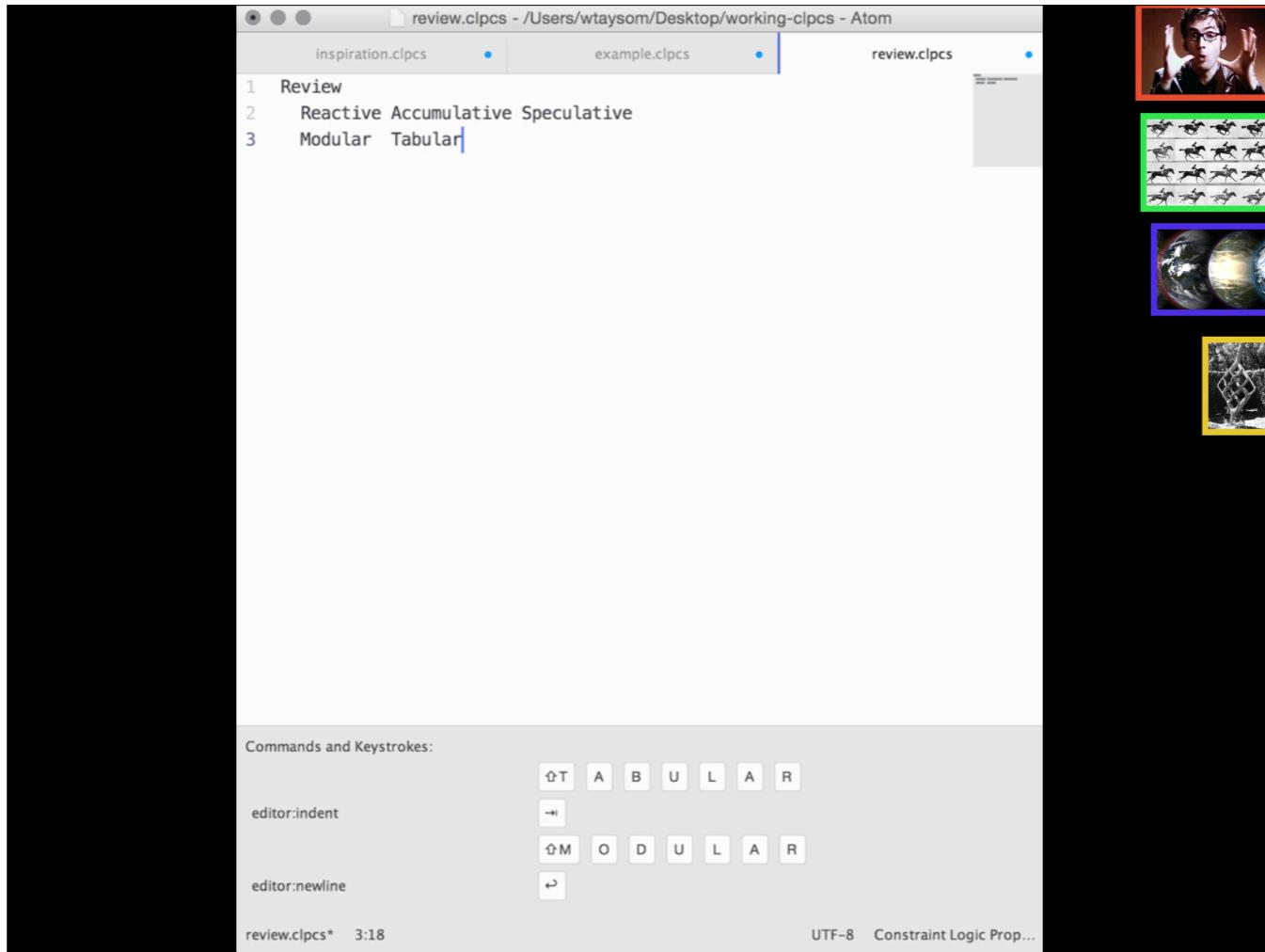
review.clpcs* 3:9

UTF-8 Constraint Logic Prop...

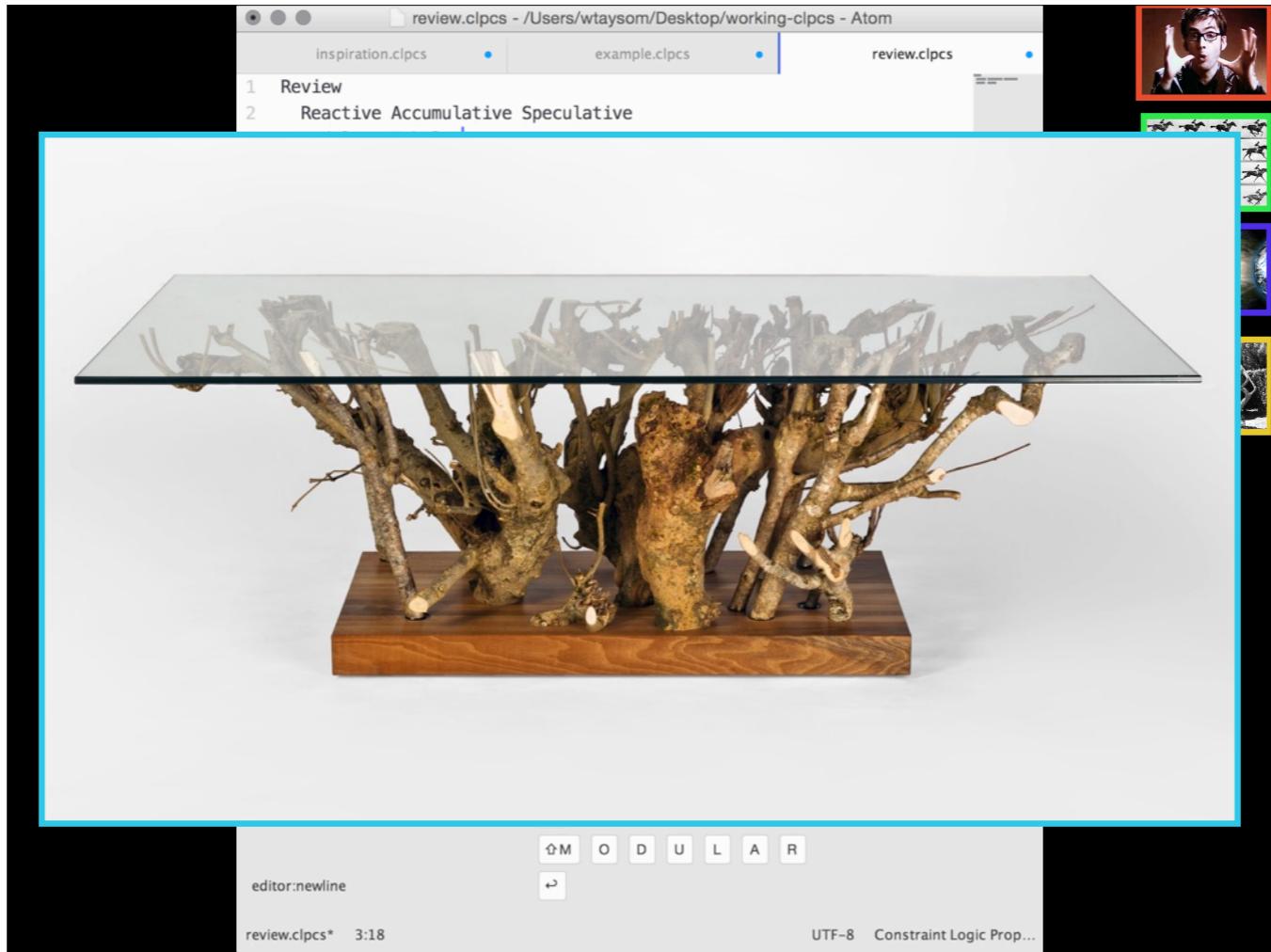
Convenient speculation comes from convenient modularity. In the example, we just indented for new scope.



Includes graft branches: multiple inheritance. How's that for a diamond problem?



Excel isn't exactly a tool for dealing with tables. Each sheet is a grid. People may treat part like a table, you can mark a box of cells as a table with sorting and filters, but maintaining the illusion requires diligence. For instance, auto-fills let you see how each cell relates to others, but they hide the fact that a parameterized formula is being applied to a whole column. Other tools take tables more seriously, often at the cost of freeform flexibility.



Moreover, not all data is tabular. We have trees too. The example showed trees and tables trying to coexist with a tree understood as a grouped table.

review.clpcs - /Users/wtaysom/Desktop/working-clpcs - Atom

inspiration.clpcs example.clpcs review.clpcs

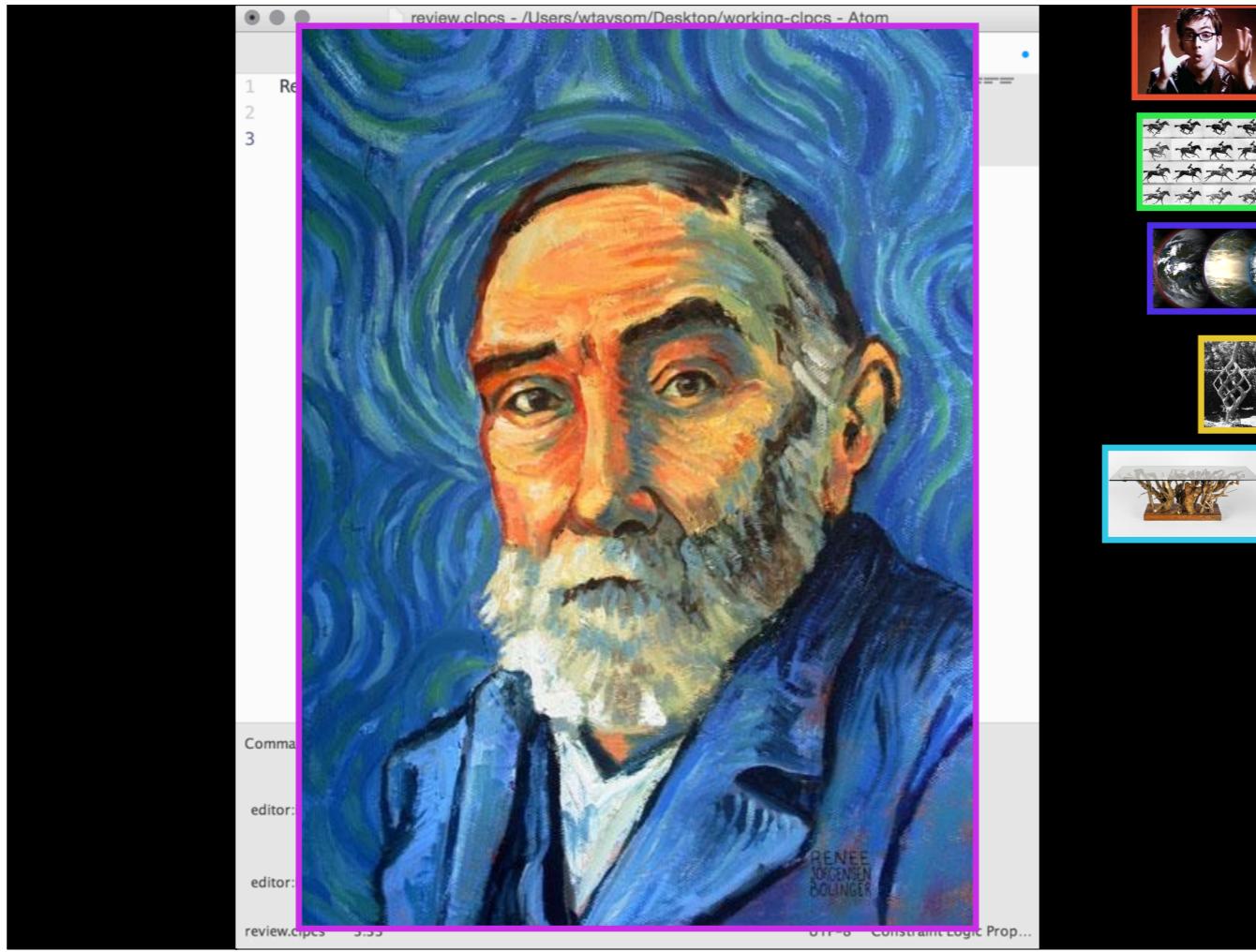
1 Review
2 Reactive Accumulative Speculative
3 Modular Tabular Binocular

Commands and Keystrokes:

editor:indent ⌘B I N O C U L A R
editor:indent ⌘T A B U L A R

review.clpcs* 3:33 UTF-8 Constraint Logic Prop...

There's more than one way to look at a thing. We've seen many examples: group-by, more or less syntactic sugar, expanding an included table. One favorite is the convention of spreadsheet cells. Enter a formula: the sense of what you want. Hit return.



See a calculated value: the reference that sense expresses in context.

It's like binocular vision: from two slightly offset planar projections, we experience a 3D scene. Any representation that emphasizes some aspect does so by obscuring another. With multiple representations, we get to see around obstacles.

review.clpcs - /Users/wtaysom/Desktop/working-clpcs - Atom

inspiration.clpcs example.clpcs review.clpcs

```
1 Review
2 Reactive Accumulative Speculative
3 Modular Tabular Binocular
4 Aggregate
```

Commands and Keystrokes:

editor:newline

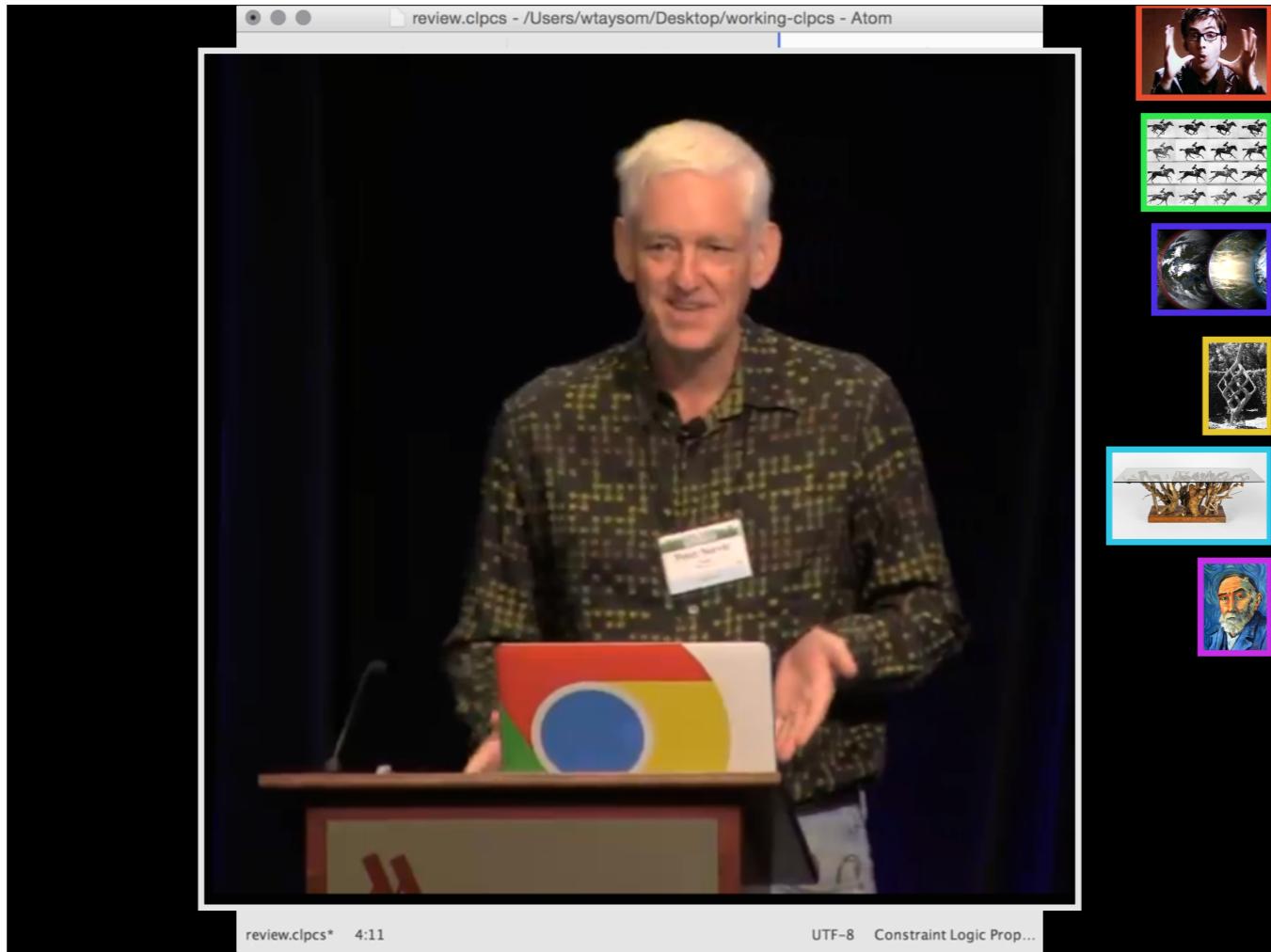
editor:indent

review.clpcs* 4:11

UTF-8 Constraint Logic Prop...

The image shows a vertical stack of six small square images. From top to bottom: 1. A man with glasses and hands raised. 2. A grid of eight small images of horses running. 3. A globe showing continents. 4. A wireframe cube or lattice structure. 5. A root system or branching structure. 6. A portrait of Vincent van Gogh.

Last year at SPLASH,



Peter Norvig remarked that mathematical specifications don't naturally translate into C++.

Program Optimization: Crossing the Math/CS Chasm

$$\begin{aligned}\mathbf{e}_{\text{best}} &= \operatorname{argmax}_{\mathbf{e}} p(\mathbf{e}|\mathbf{f}) \\ &= \operatorname{argmax}_{\mathbf{e}} p(\mathbf{f}|\mathbf{e}) p_{\text{LM}}(\mathbf{e}) \omega^{\text{length}(\mathbf{e})}\end{aligned}$$

- A few lines of math
- Tens of thousands of lines of C++

No surprise there. However, with a good declarative language, the translation can be even better than the math.

Declarative Languages for AI, ML Applications

- DYNA (Jason Eisner)
Probabilistic Context Free Parser:

```
% A single word is a phrase (given an appropriate grammar rule).
phrase(X,I,J) += rewrite(X,W) * word(W,I,J).

% Two adjacent phrases make a wider phrase (given an appropriate rule).
phrase(X,I,J) += rewrite(X,Y,Z) * phrase(Y,I,Mid) * phrase(Z,Mid,J)

% An phrase of the appropriate type covering the whole sentence is a parse.
goal           += phrase(start_nonterminal,0,length).

rewrite("S","NP","VP")=0.9
word("spring",5,6)=1
start_nonterminal="S"
```

At seeing this slide, I was awestruck: notation practically perfect in every way. Ever since, I've been pondering why I reacted so strongly.

```
% A single word is a phrase
phrase(X,I,J) += rewrite
% Two adjacent phrases
phrase(X,I,J) += rewrite
% An phrase of the appropriate length
goal += phrase
```

Earlier I said that aggregators fold over all unifying substitutions. By listing those substitutions, we enumerate all the terms in a family of sums. Unification gives us an unfolded intermediate representation. Group-by the lefthand side to see an expansion of each sum. The beauty comes from having both intensional and extensional views with regular unification serving as the lens.

review.clpcs - /Users/wtaysom/Desktop/working-clpcs - Atom

inspiration.clpcs example.clpcs review.clpcs

```
1 Review
2 Reactive Accumulative Speculative
3 Modular Tabular Binocular
4 Aggregate Propagate
```

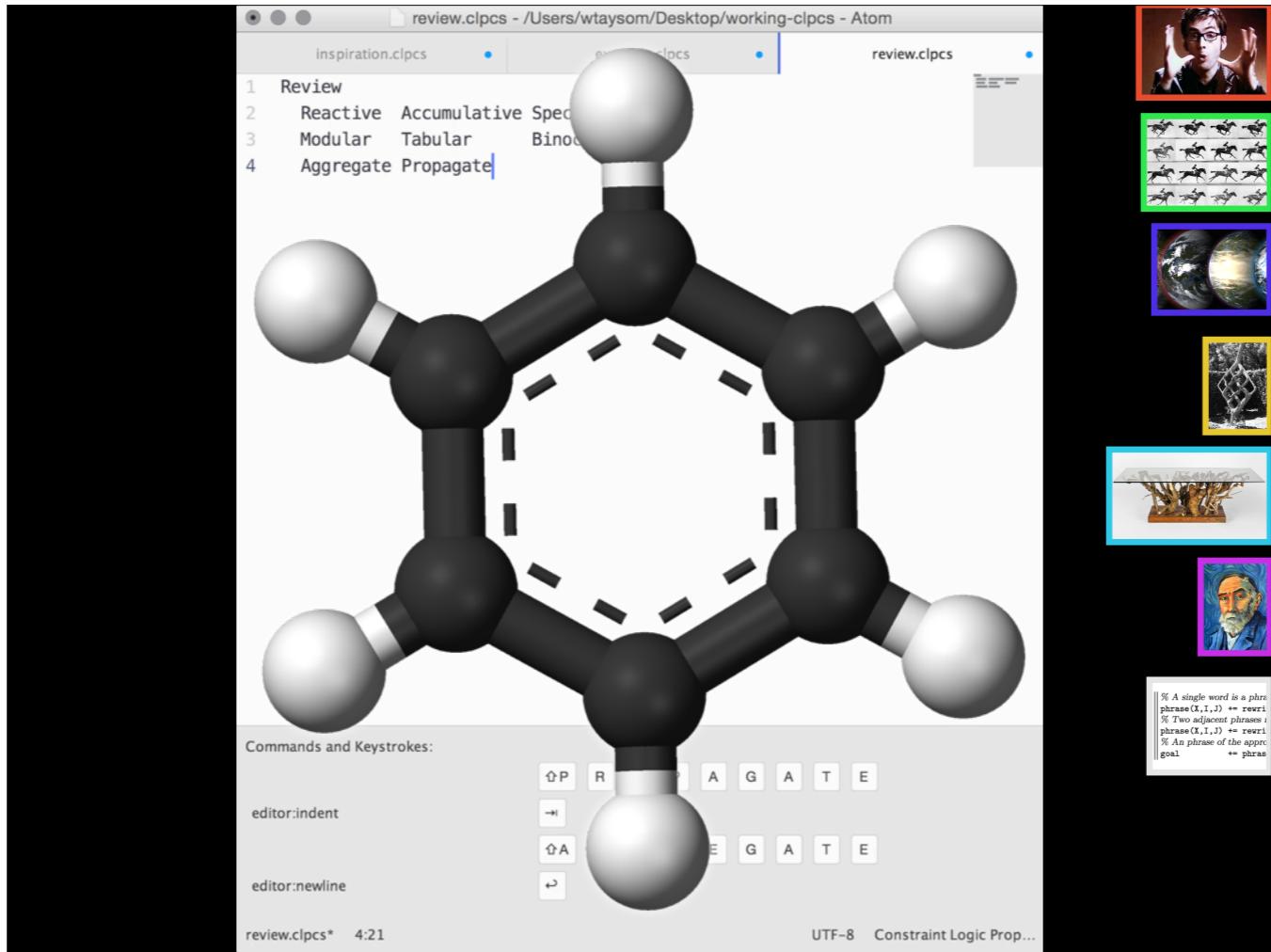
Commands and Keystrokes:

editor:indent	↑P R O P A G A T E
editor:newline	→I ↑A G G R E G A T E
	↵

review.clpcs* 4:21 UTF-8 Constraint Logic Prop...

```
% A single word is a phrase
phrase(X,I,J) :- revr1(I,J,X).
% Two adjacent phrases i
phrase(X,I,J) :- phrase(X,I,K), phrase(K,J).
% An phrase of the applic
eval(phrase(X,I,J)) :- phrase(X,I,J).
```

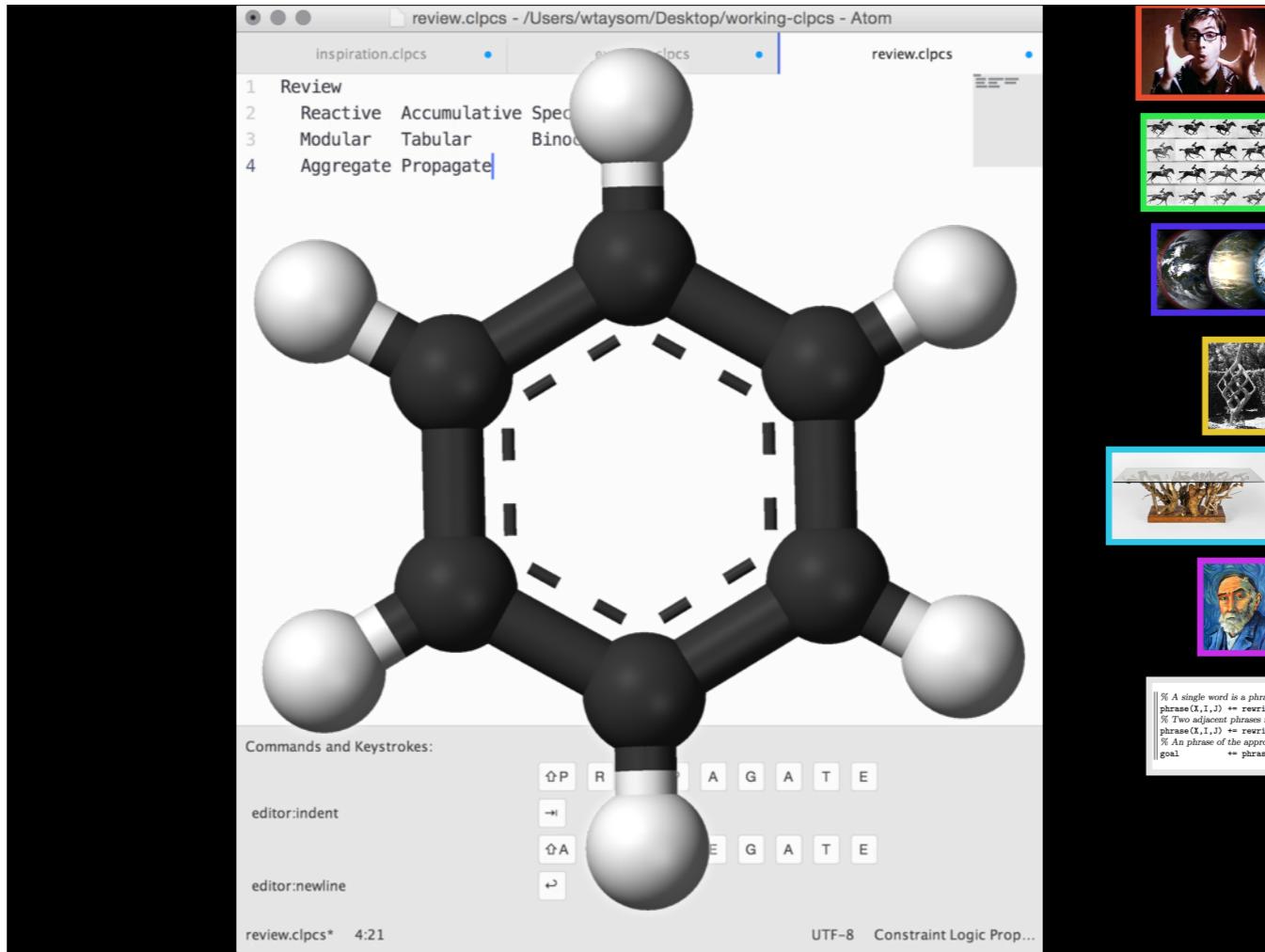
An aggregator puts each aggregate at the hub of a network with terms spreading out as spokes.



In our example, these spokes joined other hubs forming a benzene shaped ring. We propagated numbers and bounds along the network.



When Gerald Sussman talked about propagation at Strange Loop in 2011, he propagated derivatives, matrices, voltages. "The value here is that I'm allowed to dynamically change the meaning of all my operators, add new things that they can do." One new thing was to have a propagated value track its journey through the network, its province.



I've long used a weak form of this in Object Oriented settings. Proxy an object so that all its methods return a proxy whose methods all return a proxy, and so on. By the end, you have an image of data downstream from the original object, good for measuring how tangled a system is. These tracer doodads are good for view maintenance too: collect all the metadata necessary to map a view-update back to its model. Back-propagation makes tracing even better: given a wrong value find causes.

review.clpccs - /Users/wtaysom/Desktop/working-clpccs - Atom

inspiration.clpccs example.clpccs review.clpccs

```

1 Review
2 Reactive Accumulative Speculative
3 Modular Tabular Binocular
4 Aggregate Propagate Conflict

```

Commands and Keystrokes:

editor:indent	↑C O N F L I C T
	→I
editor:indent	↑P R O P A G A T E
	→I

review.clpccs* 4:33 UTF-8 Constraint Logic Prop...

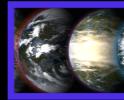
% A single word is a phrase
phrase(X,1,J) :- revr1
% Two adjacent phrases i
phrase(X,I,J) :- revr1
phrase(Y,I,J), revr1
% An phrase of the applic
eval
+ phrase

C6=C6

Sussman went on to say, "I can pay a belief in correctness for tremendous flexibility. ... The problem facing us as computer engineers is not correctness but evolvability." The whole purpose of thinking things through is to go from muddled, confused, contradictory ideas to clear, orderly, consistent ones.



Sussman went on to say, "I can pay a belief in correctness for tremendous flexibility. ... The problem facing us as computer engineers is not correctness but evolvability." The whole purpose of thinking things through is to go from muddled, confused, contradictory ideas to clear, orderly, consistent ones.



```
||% A single word is a phrase
phrase(X,I,J) :- revr1
||% Two adjacent phrases i
phrase([I,J],I,J) :- revr1
||% An phrase of the applic
phrase([I,J],I,J) :- eval
||% An phrase of the applic
phrase([I,J],I,J) :- eval
```



The degree to which our software tools can model confusion matches their usefulness in conceptualizing, brainstorming, coming to understand. A perfect language, one that only compiles correct programs, is a poor fit for a world full of inconsistency and conflict.

I'm not saying we should all be writing Ruby. A better choice may be a pure language in which chaos is a first class construct. In the example, some variables had multiple values. Instead of the conventional non-deterministic OR semantics, I experimented with a sort of over-deterministic AND semantics.

review.clp - /Users/wtaysom/Desktop/working-clp - Atom

inspiration.clp example.clp review.clp

```
1 Review
2 Reactive Accumulative Speculative
3 Modular Tabular Binocular
4 Aggregate Propagate Conflict
```

Commands and Keystrokes:

editor:indent	↑C O N F L I C T
	→i
editor:indent	↑P R O P A G A T E
	→i

review.clp* 4:33 UTF-8 Constraint Logic Prop...



Let's take one last look at the Example.

example.clp - /Users/wtayson/Desktop/working-clp - Atom

inspiration.clp • example.clp • review.clp

```

1 Example
2 traffic
3     region
4         direction North West Hilo
5         into    365  435  49
6         out of   385  415  49
7
8 include traffic
9
10 from _r:region to :region/~/r >= 0
11
12 into r += from _ to r
13 out of r += from r to _
14
15 ?= bounds (from Hilo to West)
16      50 <= 49
17
18 scenario into_Hilo = bounds (from Hilo to West) ;-
19 . include Example
20 : into Hilo = into_Hilo
21
22 ?= scenario (0 <= 900)
23      50 <= ****          <= 0
24      50 <= =              <= 50
25      50 <= -----        <= 100

```

Commands and Keystrokes:

editor:indent	↑C O N F L I C T
	→I
editor:indent	↑P R O P A G A T E
	→I

example.clp* 22:22 UTF-8 Constraint Logic Prop...

Is this the future of programming? I like the economy of expression. From a table, some equations, and a simple question, we get a nuanced answer. There may indeed be a wide and productive bridge joining logic programming to generalized propagation. This is not it. This is a curiosity, hopefully, a thought provoking one with qualities that future tools will share.

example.clp - /Users/wtayson/Desktop/working-clp - Atom

inspiration.clp • example.clp • review.clp

```

1 Example
2 traffic
3   region
4     direction North West Hilo
5     into 365 435 49
6     out of 385 415 49
7
8   include traffic
9
10  from _r:region to :region/~_r ≥ 0
11
12  into _r += from _ to _r
13  out of _r += from _r to _
14
15  ?= bounds (from Hilo to West)
16  50 ≤ 49
17
18  scenario _into_Hilo = bounds (from Hilo to West) ;-
19    . include Example
20    : into Hilo = _into_Hilo
21
22  ?= scenario {0 ≤ 900}
23  50 ≤ **** ≤ 0
24  50 ≤ = ≤ 50
25  50 ≤ ---- ≤ 100

```

Commands and Keystrokes:

editor:indent	↑C O N F L I C T
	→I
editor:indent	↑P R O P A G A T E
	→I

example.clp* 22:22 UTF-8 Constraint Logic Prop...

```

%% A single word is a phrase
phrase(X,I,J) :- revr1(X,I,J).
%% Two adjacent phrases i
phrase([X,Y],I,J) :- revr1([X,Y],I,J).
%% An phrase of the applic
phrase([X|Y],I,J) :- !, phrase(Y,I,J).
%% An phrase of the applic
phrase([X|Y],I,J) :- phrase(X,I,K), phrase(Y,K,J).

```

Reactive wibbles wobble.

example.clp - /Users/wtayson/Desktop/working-clp - Atom

inspiration.clp • example.clp • review.clp

```

1 Example
2 traffic
3   region
4     direction North West Hilo
5     into 365 435 49
6     out of 385 415 49
7
8 include traffic
9
10 from _r:region to :region/~_r ≥ 0
11
12 into _r += from _ to _r
13 out of _r += from _r to _
14
15 ?= bounds (from Hilo to West)
16 50 ≤ 49
17
18 scenario _into_Hilo = bounds (from Hilo to West) ;-
19 . include Example
20 : into Hilo = _into_Hilo
21
22 ?= scenario {0 ≤ 900}
23 50 ≤ **** ≤ 0
24 50 ≤ = ≤ 50
25 50 ≤ ---- ≤ 100

```

Commands and Keystrokes:

editor:indent	↑C O N F L I C T
	→I
editor:indent	↑P R O P A G A T E
	→I

example.clp* 22:22 UTF-8 Constraint Logic Prop...

Accumulative time as space.

example.clp - /Users/wtayson/Desktop/working-clp - Atom

inspiration.clp • example.clp • review.clp

```

1 Example
2 traffic
3   region
4     direction North West Hilo
5     into 365 435 49
6     out of 385 415 49
7
8 include traffic
9
10 from _r:region to :region/~_r ≥ 0
11
12 into _r += from _ to _r
13 out of _r += from _r to _
14
15 ?= bounds (from Hilo to West)
16 50 ≤ 49
17
18 scenario _into_Hilo = bounds (from Hilo to West) ;-
19 . include Example
20 : into Hilo = _into_Hilo
21
22 ?= scenario {0 ≤ 900}
23 50 ≤ **** ≤ 0
24 50 ≤ = ≤ 50
25 50 ≤ ---- ≤ 100

```

Commands and Keystrokes:

editor:indent	↑C O N F L I C T
	→I
editor:indent	↑P R O P A G A T E
	→I

example.clp* 22:22 UTF-8 Constraint Logic Prop...

Speculative possible futures.

example.clp - /Users/wtayson/Desktop/working-clp - Atom

inspiration.clpcs example.clpcs review.clpcs

```

1 Example
2 traffic
3   region
4     direction North West Hilo
5     into    365  435  49
6     out of   385  415  49
7
8 include traffic
9
10 from _r:region to :region/~/r >= 0
11
12 into r += from _ to r
13 out of r += from r to _
14
15 ?= bounds (from Hilo to West)
16 50 <= 49
17
18 scenario into_Hilo = bounds (from Hilo to West) ;-
19 . include Example
20 : into Hilo = into_Hilo
21
22 ?= scenario {0 <= 900}
23 50 <= **** <= 0
24 50 <= = <= 50
25 50 <= ---- <= 100

```

Commands and Keystrokes:

C	O	N	F	L	I	C	T
editor:indent	\rightarrow						
P	R	O	A	G	A	T	E
editor:indent	\rightarrow						

example.clp* 22:22 UTF-8 Constraint Logic Prop...

Modular indent and include.

example.clp - /Users/wtayson/Desktop/working-clp - Atom

inspiration.clpcs example.clpcs review.clpcs

```

1 Example
traffic
region
direction North West Hilo
into   365  435  49
out of 385  415  49
6

include traffic

from _r:region to :region/~/r >= 0
11
12 into r += from _ to r
13 out of r += from r to _
14
15 ?= bounds (from Hilo to West)
16 50 <= 49
17

scenario into_Hilo = bounds (from Hilo to West) ;-
. include Example
: into Hilo = into_Hilo
18
19
20
21
22 ?= scenario {0 <= 900}
23 50 <= **** <= 0
24 50 <= = <= 50
25 50 <= ---- <= 100
Commands and Keystrokes:
editor:indent
editor:indent
example.clp* 22:22

```

UTF-8 Constraint Logic Prop...

Tabular trees group tables.

example.clp - /Users/wtayson/Desktop/working-clp - Atom

inspiration.clpcs example.clpcs review.clpcs

```

1 Example
traffic
region
direction North West Hilo
into 365 435 49
out of 385 415 49
6
include traffic
from _r:region to :region/~/r >= 0
11
12 into r += from _ to r
13 out of r += from r to _
14
15 ?= bounds (from Hilo to West)
16 50 <= 49
17
18 scenario into_Hilo = bounds (from Hilo to West) ;-
. include Example
: into Hilo = into_Hilo
19
20
21
22 ?= scenario {0 <= 900}
23 50 <= **** <= 0
24 50 <= = <= 50
25 50 <= ---- <= 100
Commands and Keystrokes:
editor:indent
editor:indent
example.clp* 22:22

```

UTF-8 Constraint Logic Prop...

Binocular sense and reference.

example.clp - /Users/wtayson/Desktop/working-clp - Atom

inspiration.clpcs example.clpcs review.clpcs

1 Example traffic

region direction North West Hilo

into 365 435 49
out of 385 415 49

6

include traffic

from _r:region to :region/~/_r ≥ 0

11
12 into _r += from _ to _r
13 out of _r += from _r to _

14
15 ?= bounds (from Hilo to West)
16 50 ≤ 49

17

18 scenario into_Hilo = bounds (from Hilo to West) :-
. include Example
: into Hilo = into_Hilo

19

20

21

22 ?= scenario (0 ≤ 900)

23 50 ≤ **** ≤ 0
24 50 ≤ = ≤ 50
25 50 ≤ ----- ≤ 100

Commands and Keystrokes:

editor:indent

editor:indent

example.clp* 22:22

UTF-8 Constraint Logic Prop...

Aggregates unify unfolds.

example.clp - /Users/wtayson/Desktop/working-clp - Atom

inspiration.clpcs example.clpcs review.clpcs

1 Example traffic

region direction North West Hilo
into 365 435 49
out of 385 415 49

6 include traffic

from _r:region to :region/~/_r ≥ 0

11 into _r += from _ to _r
12 out of _r += from _r to _

13 % A single word is a phrase
phrase(X,1,J) :- rewr1(X,J).
% Two words phrases is
phrase(I,J,I,J) :- rewr1(I,J).
% An phrase of the approach
goal += phrase.

14
15 ?= bounds (from Hilo to West)
16 50 ≤ 49

17

18 scenario into_Hilo = bounds (from Hilo to West) :-
. include Example
: into Hilo = into_Hilo

19

20

21

22 ?= scenario (0 ≤ 900)

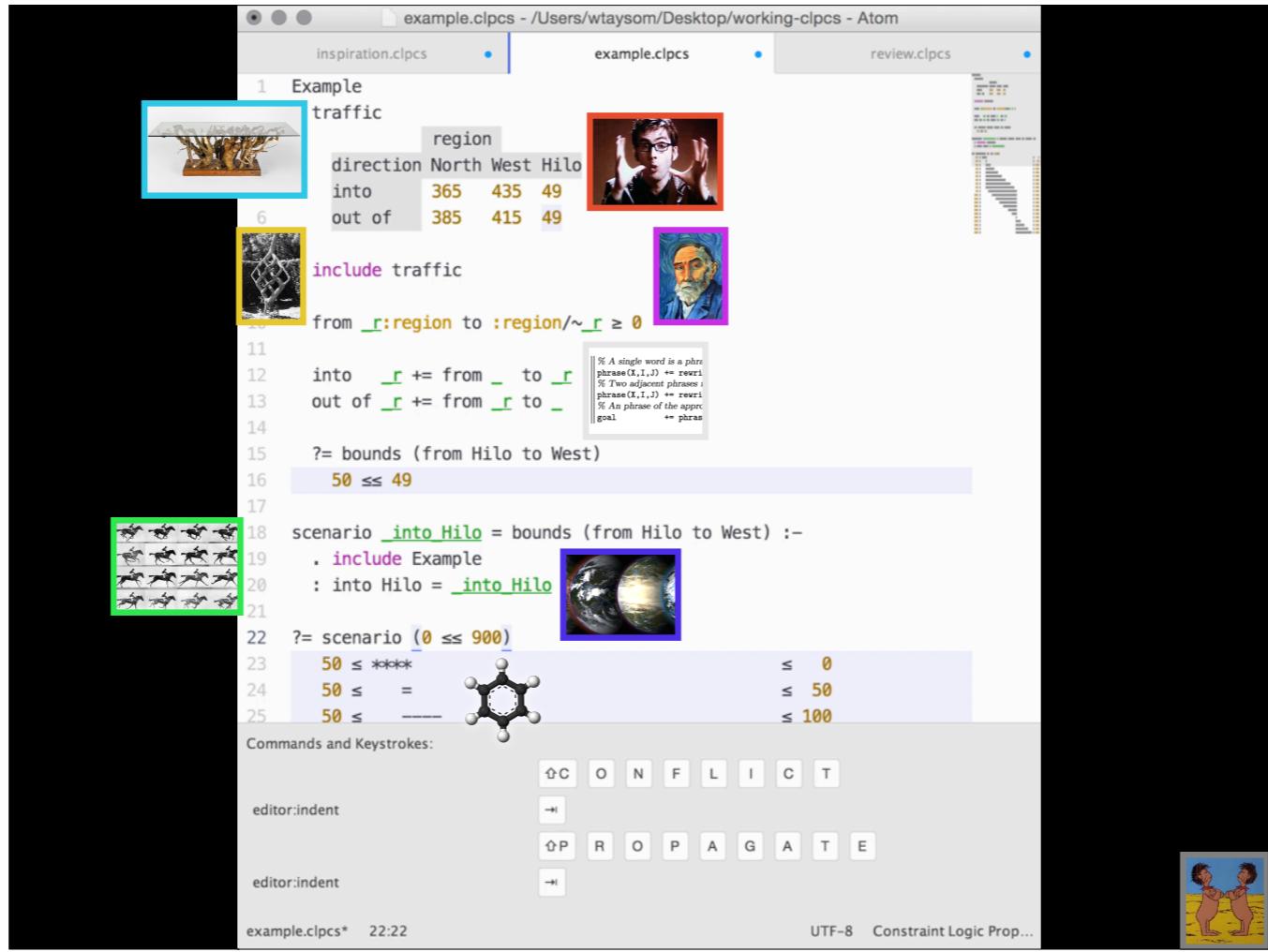
23 50 ≤ **** ≤ 0
24 50 ≤ = ≤ 50
25 50 ≤ ----- ≤ 100

Commands and Keystrokes:

editor:indent

editor:indent

UTF-8 Constraint Logic Prop...



Propagate flexible shape.

example.clp - /Users/wtayson/Desktop/working-clp - Atom

inspiration.clpcs example.clpcs review.clpcs

1 Example traffic

region
direction North West Hilo
into 365 435 49
out of 385 415 49

6 include traffic

from _r:region to :region/~_r ≥ 0

11 12 into _r += from _ to _
out of _r += from _ to _

% A single word is a phrase
phrase(X,1,J) :- rewr1
% Two words phrases is
phrase(I,1,I) :- rewr1
% An phrase of the approach
goal += phrase

?= bounds (from Hilo to West)
50 ≤ 49

17 18 scenario into_Hilo = bounds (from Hilo to West) :-
. include Example
: into Hilo = into_Hilo

19 20 21

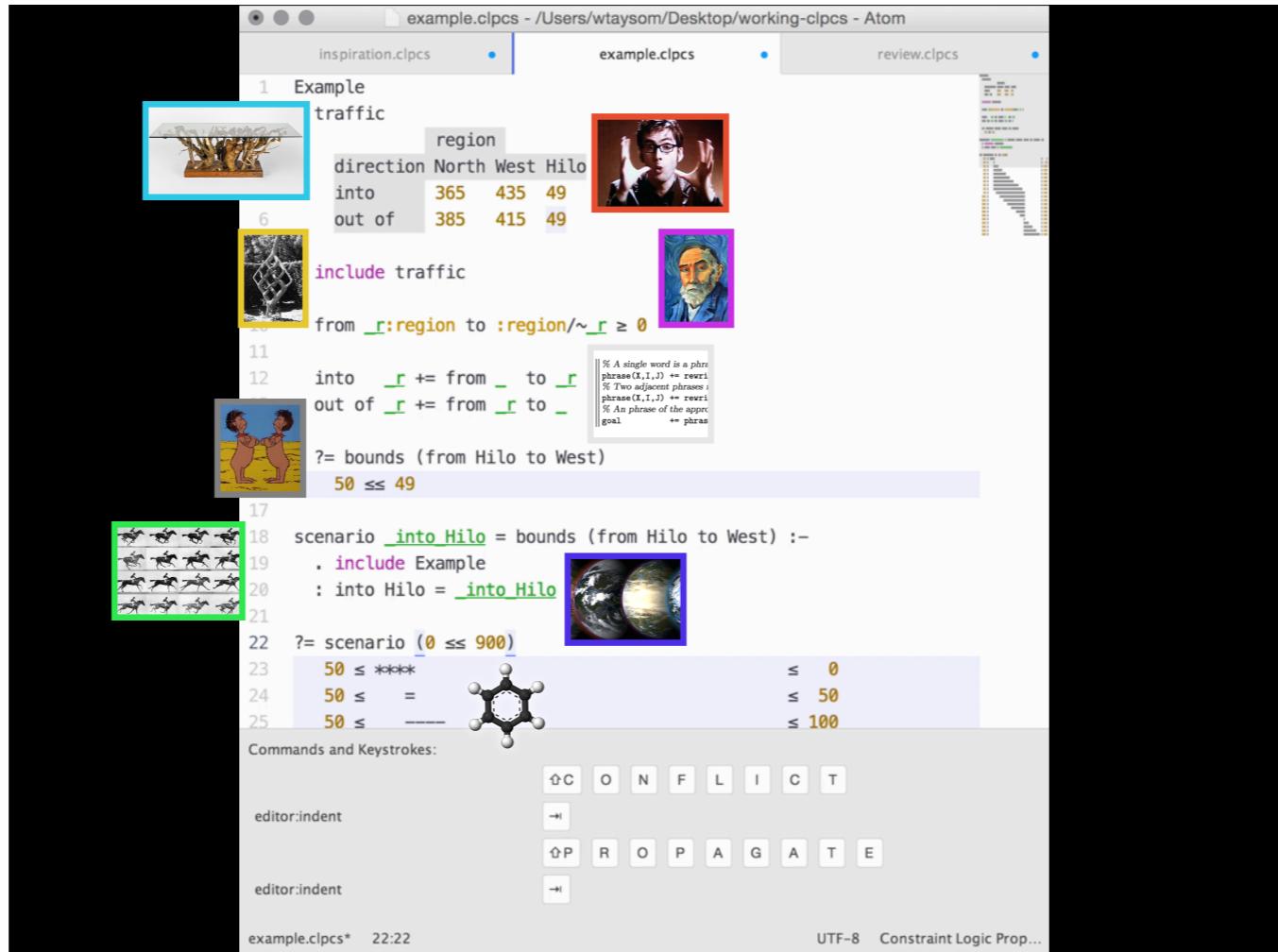
22 ?= scenario (0 ≤ 900)
50 ≤ **** ≤ 0
50 ≤ = ≤ 50
50 ≤ ----- ≤ 100

Commands and Keystrokes:

editor:indent

editor:indent

UTF-8 Constraint Logic Prop...



Conflict resolves muddled to clear.



So I leave you with these nine criteria against which you can measure your own efforts and the promising ones we'll hear about today. I'm looking forward to it. Thank you.