

Synthesis

Reading Reflection

Discuss in groups

- If you could express your intent to the computer in any way at all, how would you want to write programs?
 - What input would you have the computer take?
 - How would the interaction between you and the computer work?
- What was confusing about synthesis from the first reading/your understanding of synthesis so far?
 - It's ok if this is lots of things! We'll be getting hands-on soon, which should clear up a lot of confusions. :)
- Are there applications that you'd expect are amenable to synthesis but that haven't made it into the literature yet? (Weren't mentioned in Chapter 2.)

Reading Key Takeaways

- The core challenges in synthesis:
 - Scalability/size of the program space
 - Capturing user intent—What's a good spec? How do we get it?
- The variety of plausible specs we can get from users
 - I/O examples, demonstrations, logical specs, natural language, programs with holes, equivalent programs (!)
- The variety of search techniques
 - Enumerative, constraint-based, deductive, statistical
 - And at a higher level, the fact that synthesis is not just **one** technique
- A general sense of the problems to which synthesis has been applied so far

Thank you for your survey
answers!

Why synthesis?



There are a few PL techniques that just keep coming up in HCI tasks!

- Program synthesis
- Projection/Structure editors
- Program slicing


Others come up, but these seem to come up all the time.

Demo time

FlashFill

Do you have Excel installed? You can probably run this demo on your own laptop while I run it on mine!

B2		fx Prof. Cheung	
	A	B	
1	Whole Name		
2	Alvin Cheung	Prof. Cheung	
3	Armando Fox		
4	Jonathan Ragan-Kelley		
5	Koushik Sen		
6	Sanjit A. Seshia		
7	Katherine A. Yelick		
8			

B3		fx	
	A	B	C
1	Whole Name	Prof. Name	
2	Alvin Cheung	Prof. Cheung	
3	Armando Fox	Prof. Fox	
4	Jonathan Ragan-Kelley	Prof. Kelley	
5	Koushik Sen	Prof. Sen	
6	Sanjit A. Seshia	Prof. Seshia	
7	Katherine A. Yelick	Prof. Yelick	
8			

CTRL + E

Scythe

To run this one, head to: <https://scythe.cs.washington.edu/demo>

i Click the Synthesis button to synthesize Queries from the example!

+ Empty Panel
- Remove Panel
Load An Example Panel ▾
Connect to Database ▾

Offline (No backend DB connected)

✦ Example Task: Find the span of career peak (the year when the first paper and most cited papers are published) of computer scientists given the list of their pushlished papers.

papers				
author	title	year	citation	
H. Simon	Understanding wi	1974	277	X
H. Simon	Organization	1997	20057	X
H. Simon	The sciences of tl	1996	17561	X
R. Tibshirani	Developmental re	2004	50	X
R. Tibshirani	Flexible discrimin	1995	51	X
R. Tibshirani	IRF9 and STAT1 ε	2008	47	X
P. Bork	Automated pair-w	1998	51	X
P. Bork	UN targets top kil	2011	19	X

Add Row
Add Col
Del Col

Constants None ?

Aggregators (Optional) ?

+ Add Table
- Remove Table

output			
author	min_year	peak_year	
H. Simon	1974	1997	X
R. Tibshirar	1995	1995	X
P. Bork	1998	1998	X

Add Row
Add Col
Del Col

Synthesize

```

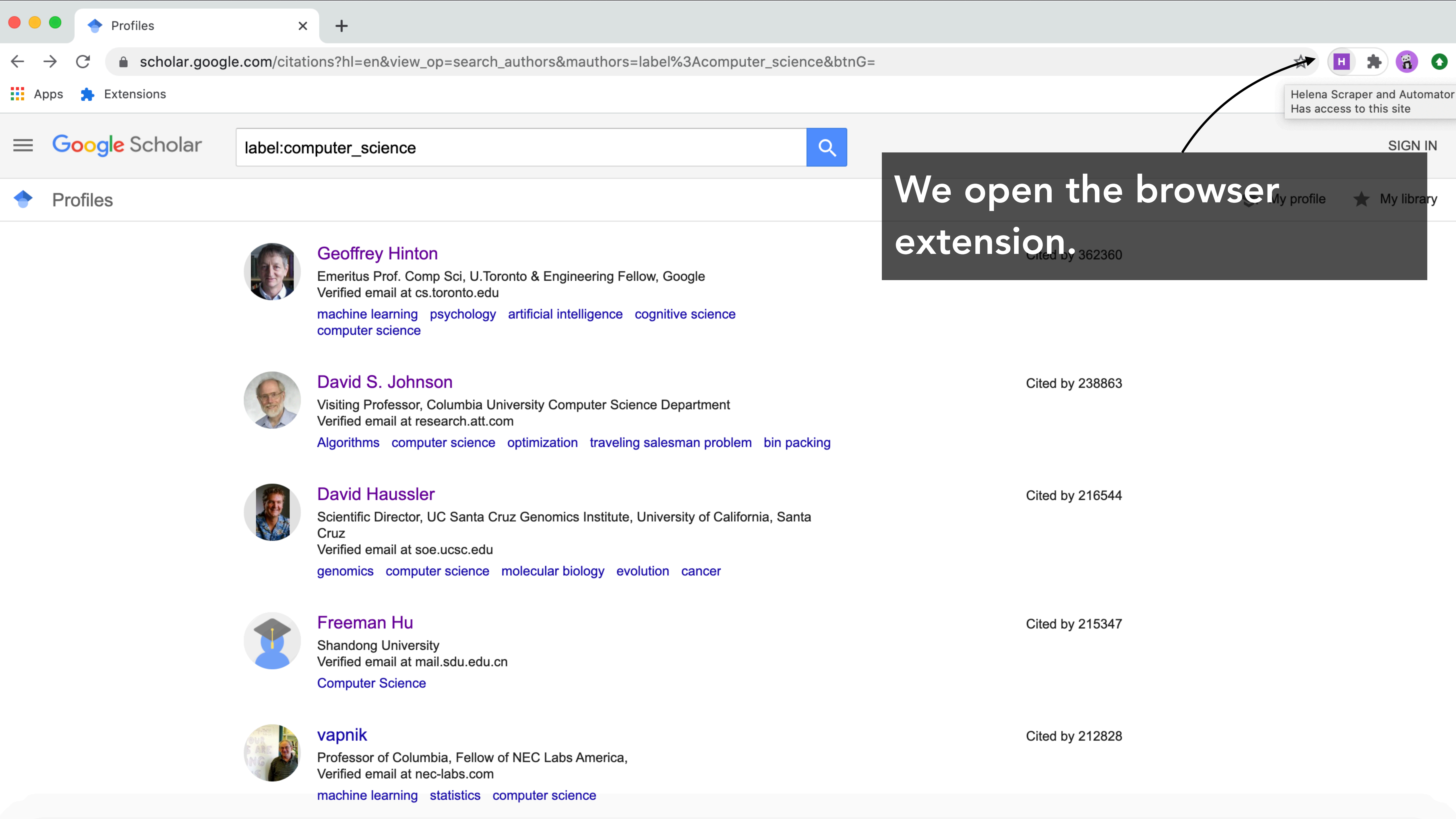
Select t6.author,t6.min_year,t6.year
From
  (Select t5.author, t5.min_year, t4.author As author1, t4.year
  From ((Select
    t3.author, Min(t3.year) As min_year
  From
    papers As t3
  Group By
    t3.author) As t5 Join
  (Select t7.author,t7.year
  From
    (Select t1.author, t1.max_citation, t8.author As author1, t8.ti
    tle, t8.year, t8.citation
  From ((Select
    t2.author, Max(t2.citation) As max_citation
  From
    papers As t2
  Group By
    t2.author) As t1 Join
    papers As t8)) As t7
  Where t7.max_citation = t7.citation
  And t7.author = t7.author1) As t4)) As t6
Where t6.author = t6.author1;
  
```

Synthesized Query 1 ▾
Run on DB
Visualize ▾

Helena

If you want to run this one, you have to install an extension:

<http://helena-lang.org/install>



We open the browser extension.



Geoffrey Hinton

Emeritus Prof. Comp Sci, U.Toronto & Engineering Fellow, Google
Verified email at cs.toronto.edu

machine learning psychology artificial intelligence cognitive science
computer science

Cited by 362360



David S. Johnson

Visiting Professor, Columbia University Computer Science Department
Verified email at research.att.com

Algorithms computer science optimization traveling salesman problem bin packing

Cited by 238863



David Haussler

Scientific Director, UC Santa Cruz Genomics Institute, University of California, Santa Cruz
Verified email at soe.ucsc.edu

genomics computer science molecular biology evolution cancer

Cited by 216544



Freeman Hu

Shandong University
Verified email at mail.sdu.edu.cn

Computer Science

Cited by 215347



vapnik

Professor of Columbia, Fellow of NEC Labs America,
Verified email at nec-labs.com

machine learning statistics computer science

Cited by 212828

Current Script | Saved Scripts | Scheduled Runs

We're recording! Remember, collect ONLY the FIRST ROW of data. When you're ready to add a new cell, hover over the text you want, then press **ALT** + click. We'll show the data you've collected right here: Cancel Recording

Collect the FIRST ROW of your target dataset.

 Stop Recording



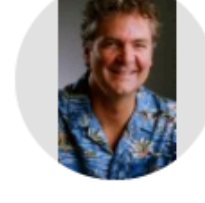



Profiles

scholar.google.com/citations?hl=en&view_op=search_authors&mauthors=la...

Apps Extensions

label:computer_science

Profiles

	Geoffrey Hinton Geoffrey Hinton, Comp Sci, U. Toronto & Engineering Fellow, Google verified email at cs.toronto.edu machine learning psychology artificial intelligence cognitive science computer science	Cited by 362360
	David S. Johnson Visiting Professor, Columbia University Computer Science Department Verified email at research.att.com Algorithmic combinatorics on words combinatorics on words parsing	Cited by 238863
	David Haussler Scientific Director, UC Santa Cruz Genomics Institute, University of California, Santa Cruz Verified email at soe.ucsc.edu genomics computer science molecular biology evolution cancer	Cited by 216544
	Freeman Hu Shandong University Verified email at mail.sdu.edu.cn Computer Science	Cited by 215347
	vapnik Professor of Columbia, Fellow of NEC Labs America, Verified email at nec-labs.com machine learning statistics computer science	Cited by 212828
	Joerg Meyer	Cited by 185932

We demonstrate how to find information that goes in the first row of our target dataset.

We're recording! Remember, collect ONLY the FIRST ROW of data. When you're ready to add a new cell, hover over the text you want, then press **ALT** + click. We'll show the data you've collected right here:

Cancel Recording

Geoffrey Hinton

Stop Recording

Profiles Geoffrey Hinton - Google Scho

scholar.google.com/citations?hl=en&user=JicYPdAAAAAJ

Google Scholar

Geoffrey Hinton FOLLOW

Emeritus Prof. Comp Sci, U.Toronto & Engineering Fellow, Google
Verified email at cs.toronto.edu - [Homepage](#)

machine learning psychology artificial intelligence cognitive science computer science

ARTICLES CITED BY CO-AUTHORS

TITLE	CITED BY	YEAR
Imagenet classification with deep convolutional neural networks	66206	2012
Deep learning	28294	2015
Learning internal representations by error-propagation	26773	1986
Learning internal representations by error propagation	26468	1986
Learning internal representations by error propagation	26422	1986

We continue on another page (and another table).

Current Script | Saved Scripts | Scheduled Runs

We're recording! Remember, collect ONLY the FIRST ROW of data. When you're ready to add a new cell, hover over the text you want, then press **ALT** + click. We'll show the data you've collected right here: Cancel Recording

Geoffrey Hinton | Imagenet classification with deep convolutional neural networks
66206 | 2012



Stop Recording

We're done demonstrating.

Profiles | Geoffrey Hinton - Google Scho

scholar.google.com/citations?hl=en&user=JicYPdAAAAAJ

Google Scholar

 **Geoffrey Hinton** FOLLOW

Emeritus Prof. Comp Sci, U.Toronto & Engineering Fellow, Google
Verified email at cs.toronto.edu - [Homepage](#)

[machine learning](#) [psychology](#) [artificial intelligence](#) [cognitive science](#) [computer science](#)

ARTICLES | CITED BY | CO-AUTHORS

TITLE	CITED BY	YEAR
Imagenet classification with deep convolutional neural networks A Krizhevsky, I Sutskever, GE Hinton Advances in neural information processing systems, 1097-1105	66206	2012
Deep learning Y LeCun, Y Bengio, G Hinton Nature 521 (7553), 436-444	28294	2015
Learning internal representations by error-propagation DE Rumelhart, GE Hinton, RJ Williams Parallel Distributed Processing: Explorations in the Microstructure of ...	26773	1986
Learning internal representations by error propagation DE Rumelhart, GE Hinton, RJ Williams Learning internal representations by error propagation	26468	1986
Learning internal representations by error propagation DE Rumelhart, GE Hinton, RJ Williams MIT Press, Cambridge, MA 1 (318)	26422	1986

Current Script | Saved Scripts | Scheduled Runs

Save and Run Script

program_name Save Script

Advanced Options

Start New Script

```

load https://scholar.google.com/citations?hl=en&view_... into p
for each row in list_1 in page1 ( ✓ for all rows, for the first 2
do
  scrape list_1_item_1 in page1
  click list_1_item_1 in page1 , load page into page2
  for each row in list_3 in page2 ( ✓ for all rows, for the f
  do
    scrape title in page2
    scrape cited_by in page2
    scrape year in page2
    add dataset row that includes: list_1_item_1 TE

```

Relevant Tables

Troubleshooting
What kind of problem are you having?

New Recording Window | Profiles | Geoffrey Hinton - Google Sc

scholar.google.com/citations?hl=en&user=JicYPdAAAAAJ

Google Scholar

Geoffrey Hinton
Emeritus Prof. Comp Sci, U.Toronto & Engineering Fellow, Google
Verified email at cs.toronto.edu - [Homepage](#)
machine learning psychology artificial intelligence cognitive science computer science

[ARTICLES](#) | CITED BY | CO-AUTHORS

TITLE	CITED BY	YEAR
Imagenet classification with deep convolutional neural networks A Krizhevsky, I Sutskever, GE Hinton Advances in neural information processing systems, 1097-1105	66206	2012
Deep learning Y Lecun, Y Bengio, GE Hinton 2015	28294	2015
Learning internal representations by error-propagation DE Rumelhart, GE Hinton, RJ Williams Parallel Distributed Processing: Explorations in the Microstructure of ...	26773	1986
Learning internal representations by error propagation DE Rumelhart, GE Hinton, RJ Williams Learning internal representations by error propagation	26468	1986
Learning internal representations by error propagation DE Rumelhart, GE Hinton, RJ Williams MIT Press, Cambridge, MA 1 (318)	26422	1986

The synthesizer writes our program.

Current Script Saved Scripts Scheduled Runs Script Run 1

Pause Script Resume Script Restart From Beginning Cancel Script Run

Download Data (This Scrape) Download Data (All Scrapes)

Note: the downloaded dataset may be slightly out of date if we haven't saved all data yet. Rows so far: 40

Geoffrey Hinton	Imagenet classification with deep convolutional neural networks	66206	2012	1
Geoffrey Hinton	Deep learning	28294	2015	2
Geoffrey Hinton	Learning internal representations by error-propagation	26773	1986	3
Geoffrey Hinton	Learning internal representations by error propagation	26468	1986	4
Geoffrey Hinton	Learning internal representations by error propagation	26422	1986	5
Geoffrey Hinton	Learning representations by back-propagating errors	21859	1986	6
Geoffrey Hinton	Dropout: a simple way to prevent neural networks from overfitting	21365	2014	7
Geoffrey Hinton	Visualizing data using t-SNE	14439	2008	8
Geoffrey Hinton	A fast learning algorithm for deep belief nets	13397	2006	9
Geoffrey Hinton	Reducing the dimensionality of data with neural networks	12573	2006	10
Geoffrey Hinton	Rectified linear units improve restricted boltzmann machines	10069	2010	11
Geoffrey Hinton	Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups	8233	2012	12
Geoffrey Hinton	Learning multiple layers of features from tiny images	8034	2009	13
Geoffrey Hinton	Speech recognition with deep recurrent neural networks	5975	2013	14
Geoffrey Hinton	Improving neural networks by preventing co-adaptation of feature detectors	5308	2012	15
Geoffrey Hinton	Training products of experts by minimizing contrastive divergence	4574	2002	16
Geoffrey Hinton	Adaptive mixtures of local experts	4263	1991	17
Geoffrey Hinton	A learning algorithm for Boltzmann machines	4171	1985	18
Geoffrey Hinton	Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude	3924	2012	19
Geoffrey Hinton	Distilling the knowledge in a neural network	3887	2015	20

This page is being controlled by Helena. If you want to interact with this page anyway, click here to remove the overlay. Keep in mind that navigating away from the current page may disrupt the Helena process.



David Haussler

Scientific Director, UC Santa Cruz Genomics Institute, University of California, Santa Cruz

Verified email at soe.ucsc.edu

genomics computer science molecular biology evolution cancer

FOLLOW

ARTICLES CITED BY

TITLE CITED BY YEAR

Initial sequencing and analysis of the human genome 19484 2001 ES Lander, LM Linton, B Birren, C Nusbaum, MC Zody, J Baldwin, ... Macmillan Publishers Ltd.

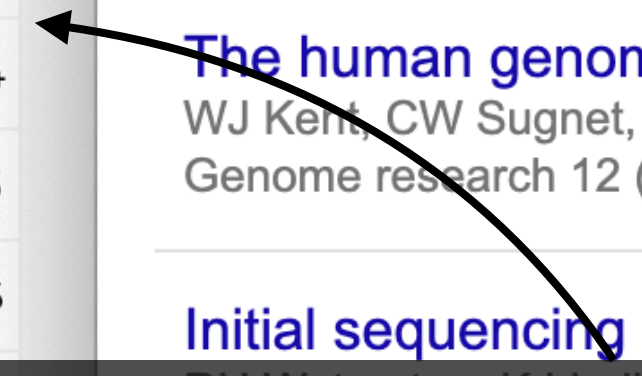
An integrated encyclopedia of DNA elements in the human genome 10539* 2012 ENCODE Project Consortium Nature 489 (7414), 57-74

The human genome browser at UCSC 8020 2002 WJ Kent, CW Sugnet, TS Furey, KM Roskin, TH Pringle, AM Zahler, ... Genome research 12 (6), 996-1006

Initial sequencing and comparative analysis of the mouse genome 7069 2002 RH Waterston, K Lindblad-Toh, E Birney, J Rogers, JF Abril, P Agarwal, ... Nature 420 (6915), 520-526

A map of human genome variation from population-scale sequencing 7053 2010 1000 Genomes Project Consortium Nature 467 (7319), 1061

The program collects our data.



Browser tabs: Extensions, Profiles






Address bar: https://scholar.google.com/citations?hl=en&view_op=search_authors&mauthors=label%3Acomputer_science&btnG=

Navigation: Apps, Extensions, Other Bookmarks

Google Scholar logo

Search bar:

Navigation: Profiles, My profile, My library

	Geoffrey Hinton Emeritus Prof. Comp Sci, U.Toronto & Engineering Fellow, Google Verified email at cs.toronto.edu machine learning neural networks artificial intelligence cognitive science computer science	Cited by 246012
	DEYWIS MORENO High Energy Physicist, Universidad Antonio Narino Verified email at uan.edu.co High Energy Physics Computer science	Cited by 206009
	David S. Johnson Visiting Professor, Columbia University Computer Science Department Verified email at research.att.com Algorithms computer science optimization traveling salesman problem bin packing	Cited by 176731
	David Haussler Scientific Director, UC Santa Cruz Genomics Institute, University of California, Santa Cruz Verified email at soe.ucsc.edu genomics computer science molecular biology evolution cancer	Cited by 174202
	vapnik	Cited by 170728

LENS

	before		after
	<pre>cmp r1, #0 mov r3, r1, asr #31 add r2, r1, #7 mov r3, r3, lsr #29 movge r2, r1 ldrb r0, [r0, r2, asr #3] bic r1, r2, #248 sub r3, r1, r3 asr r1, r0, r3 and r0, r1, #1</pre>		<pre>asr r3, r1, #2 add r2, r1, r3, lsr #29 ldrb r0, [r0, r2, asr #3] and r3, r2, #248 sub r3, r1, r3 asr r1, r0, r3 and r0, r1, #1</pre>
	(b)		(c)

I know, I know, not as photogenic, but it makes programs much faster!!

Falx

<https://falx.cs.washington.edu/tool>

5 min break

Back up. What's program synthesis?

Find a program P that meets a spec $\phi(\text{input}, \text{output})$:

Correctness Condition

$$\exists P. \forall x. \phi(x, P(x))$$

Find P

- When to use synthesis:
 - **Ease-of-use/productivity:** When writing ϕ is faster or easier than writing P
 - **Correctness:** when proving ϕ is easier than proving P

Hey, I've seen this before

I give computer a high-level description of what I want it to do

Computer gives me back a low-level program for doing it



Synthesis vs. compilation

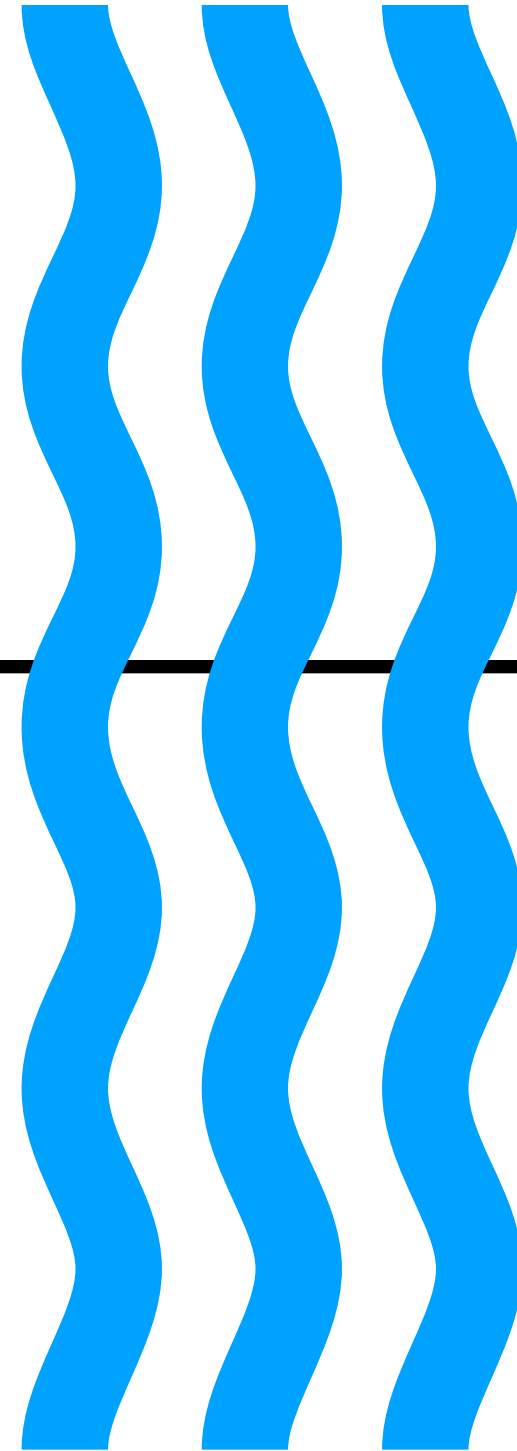
Compilation

Synthesis



Typically deterministic

Typically performs lowering
via a sequence of rewrite
rules



Searches a space of
possible programs

...or sometimes a space of
possible sequences of
rewrite rules! look, the line
is blurry $\backslash_(_)_/$

If it involves search, we
usually call it synthesis

Even if you don't take away anything else from today's lecture, take away that you can write a synthesizer!

Even if you don't take away anything else from today's lecture, take away that you can write a synthesizer!



What do we need to decide to make a synthesizer?

Hint: 3 things

How does the user express what they want the program to do?

What space of programs is the synthesizer allowed to use?

What algorithm will the synthesizer use to search that space?

What do we need to decide to make a synthesizer?

Hint: 3 things For today's sample synthesizer, let's pick...

How does the user express what they want the program to do?

Input-Output examples

What space of programs is the synthesizer allowed to use?

Anything in a Domain-Specific Language (DSL) of our choice

What algorithm will the synthesizer use to search that space?



Enumeration

Which is to say...generating programs until we find one that works

Input-Output Examples

- Any work here?
- Nah, this is going to be pretty straightforward.

- Example:

({ "x" → 3, "y" → 7 } , 23)

({ "x" → 4, "y" → 4 } , 19)

({ "x" → 2, "y" → 12 } , 31)

**Can you guess it?? Did you already
synthesize this in your head?**

Domain-Specific Language

- This one's a classic, but for another domain we might design something more customized

$$\begin{aligned} \text{expr} := & \mathcal{N} \\ & | v \\ & | (\text{expr} + \text{expr}) \\ & | (\text{expr} - \text{expr}) \\ & | (\text{expr} * \text{expr}) \end{aligned}$$

Enumeration

Spec:

({ "x" → 3, "y" → 7 } , 23)
({ "x" → 4, "y" → 4 } , 19)
({ "x" → 2, "y" → 12 } , 31)

Space of programs:

$expr := \mathcal{N}$
| v
| $(expr + expr)$
| $(expr - expr)$
| $(expr * expr)$

level 0:

[0, 1, 2, 3, 4, y, x]

count: 7

level 1 :

[0, 1, 2, 3, 4, y, x, (0+0), (0*0), (0-0), (0+1), (0*1), (0-1), (0+2), (0*2), (0-2), (0+3), (0*3), (0-3), (0+4), (0*4), (0-4), (0+y), (0*y), (0-y), (0+x), (0*x), (0-x), (1+0), (1*0), (1-0), (1+1), (1*1), (1-1), (1+2), (1*2), (1-2), (1+3), (1*3), (1-3), (1+4), (1*4), (1-4), (1+y), (1*y), (1-y), (1+x), (1*x), (1-x), (2+0), (2*0), (2-0), (2+1), (2*1), (2-1), (2+2), (2*2), (2-2), (2+3), (2*3), (2-3), (2+4), (2*4), (2-4), (2+y), (2*y), (2-y), (2+x), (2*x), (2-x), (3+0), (3*0), (3-0), (3+1), (3*1), (3-1), (3+2), (3*2), (3-2), (3+3), (3*3), (3-3), (3+4), (3*4), (3-4), (3+y), (3*y), (3-y), (3+x), (3*x), (3-x), (4+0), (4*0), (4-0), (4+1), (4*1), (4-1), (4+2), (4*2), (4-2), (4+3), (4*3), (4-3), (4+4), (4*4), (4-4), (4+y), (4*y), (4-y), (4+x), (4*x), (4-x), (y+0), (y*0), (y-0), (y+1), (y*1), (y-1), (y+2), (y*2), (y-2), (y+3), (y*3), (y-3), (y+4), (y*4), (y-4), (y+y), (y*y), (y-y), (y+x), (y*x), (y-x), (x+0), (x*0), (x-0), (x+1), (x*1), (x-1), (x+2), (x*2), (x-2), (x+3), (x*3), (x-3), (x+4), (x*4), (x-4), (x+y), (x*y), (x-y), (x+x), (x*x), (x-x)]

count: 154

level 2 :

[0, 1, 2, 3, 4, y, x, (0+0), (0*0), (0-0), (0+1), (0*1), (0-1), (0+2), (0*2), (0-2), (0+3), (0*3), (0-3), (0+4), (0*4), (0-4), (0+y), (0*y), (0-y), (0+x), (0*x), (0-x), (1+0), (1*0), (1-0), (1+1), (1*1), (1-1), (1+2), (1*2), (1-2), (1+3), (1*3), (1-3), (1+4), (1*4), (1-4), (1+y), (1*y), (1-y), (1+x), (1*x), (1-x), (2+0), (2*0), (2-0), (2+1), (2*1), (2-1), (2+2), (2*2), (2-2), (2+3), (2*3), (2-3), (2+4), (2*4), (2-4), (2+y), (2*y), (2-y), (2+x), (2*x), (2-x), (3+0), (3*0), (3-0), (3+1), (3*1), (3-1), (3+2), (3*2), (3-2), (3+3), (3*3), (3-3), (3+4), (3*4), (3-4), (3+y), (3*y), (3-y), (3+x), (3*x), (3-x), (4+0), (4*0), (4-0), (4+1), (4*1), (4-1), (4+2), (4*2), (4-2), (4+3), (4*3), (4-3), (4+4), (4*4), (4-4), (4+y), (4*y), (4-y), (4+x), (4*x), (4-x), (y+0), (y*0), (y-0), (y+1), (y*1), (y-1), (y+2), (y*2), (y-2), (y+3), (y*3), (y-3), (y+4), (y*4), (y-4), (y+y), (y*y), (y-y), (y+x), (y*x), (y-x), (x+0), (x*0), (x-0), (x+1), (x*1), (x-1), (x+2), (x*2), (x-2), (x+3), (x*3), (x-3), (x+4), (x*4), (x-4), (x+y), (x*y), (x-y), (x+x), (x*x), (x-x), (0+0), (0*0), (0-0), (0+1), (0*1), (0-1), (0+2), (0*2), (0-2), (0+3), (0*3), (0-3), (0+4), (0*4), (0-4), (0+y), (0*y), (0-y), (0+x), (0*x), (0-x), (1+0), (1*0), (1-0), (1+1), (1*1), (1-1), (1+2), (1*2), (1-2), (1+3), (1*3), (1-3), (1+4), (1*4), (1-4), (1+y), (1*y), (1-y), (1+x), (1*x), (1-x), (2+0), (2*0), (2-0), (2+1), (2*1), (2-1), (2+2), (2*2), (2-2), (2+3), (2*3), (2-3), (2+4), (2*4), (2-4), (2+y), (2*y), (2-y), (2+x), (2*x), (2-x), (3+0), (3*0), (3-0), (3+1), (3*1), (3-1), (3+2), (3*2), (3-2), (3+3), (3*3), (3-3), (3+4), (3*4), (3-4), (3+y), (3*y), (3-y), (3+x), (3*x), (3-x), (4+0), (4*0), (4-0), (4+1), (4*1), (4-1), (4+2), (4*2), (4-2), (4+3), (4*3), (4-3), (4+4), (4*4), (4-4), (4+y), (4*y), (4-y), (4+x), (4*x), (4-x), (y+0), (y*0), (y-0), (y+1), (y*1), (y-1), (y+2), (y*2), (y-2), (y+3), (y*3), (y-3), (y+4), (y*4), (y-4), (y+y), (y*y), (y-y), (y+x), (y*x), (y-x), (x+0), (x*0), (x-0), (x+1), (x*1), (x-1), (x+2), (x*2), (x-2), (x+3), (x*3), (x-3), (x+4), (x*4), (x-4), (x+y), (x*y), (x-y), (x+x), (x*x), (x-x)]

count: 71,302

Ok, no luck so far. Let's just mash these together! In every possible combination!

Hm, still no luck. Keep mashing.

Enumeration...pruned with Operational Equivalence

←Which is the fancy program synthesis way of saying "they do the same thing on the inputs we care about."

Spec:

({ "x" → 3, "y" → 7 } , 23)
 ({ "x" → 4, "y" → 4 } , 19)
 ({ "x" → 2, "y" → 12 } , 31)

Space of programs:

$expr := \mathcal{N}$
 | v
 | $(expr + expr)$
 | $(expr - expr)$
 | $(expr * expr)$

Ok, these are all just 0...which we already have. Why'd you give me these???

level 0:
 [0, 1, 2, 3, 4, y, x]
 count: 7

level 1 :
 [0, 1, 2, 3, 4, y, x, (0+0), (0*0), (0-0), (0+1), (0*1), (0-1), (0+2), (0*2), (0-2), (0+3), (0*3), (0-3), (0+4), (0*4), (0-4), (0+y), (0*y), (0-y), (0+x), (0*x), (0-x), (1+0), (1*0), (1-0), (1+1), (1*1), (1-1), (1+2), (1*2), (1-2), (1+3), (1*3), (1-3), (1+4), (1*4), (1-4), (1+y), (1*y), (1-y), (1+x), (1*x), (1-x), (2+0), (2*0), (2-0), (2+1), (2*1), (2-1), (2+2), (2*2), (2-2), (2+3), (2*3), (2-3), (2+4), (2*4), (2-4), (2+y), (2*y), (2-y), (2+x), (2*x), (2-x), (3+0), (3*0), (3-0), (3+1), (3*1), (3-1), (3+2), (3*2), (3-2), (3+3), (3*3), (3-3), (3+4), (3*4), (3-4), (3+y), (3*y), (3-y), (3+x), (3*x), (3-x), (4+0), (4*0), (4-0), (4+1), (4*1), (4-1), (4+2), (4*2), (4-2), (4+3), (4*3), (4-3), (4+4), (4*4), (4-4), (4+y), (4*y), (4-y), (4+x), (4*x), (4-x), (y+0), (y*0), (y-0), (y+1), (y*1), (y-1), (y+2), (y*2), (y-2), (y+3), (y*3), (y-3), (y+4), (y*4), (y-4), (y+y), (y*y), (y-y), (y+x), (y*x), (y-x), (x+0), (x*0), (x-0), (x+1), (x*1), (x-1), (x+2), (x*2), (x-2), (x+3), (x*3), (x-3), (x+4), (x*4), (x-4), (x+y), (x*y), (x-y), (x+x), (x*x), (x-x)]
 count: 154

And these are the same on all inputs.

And eventually we'll find some that aren't the same on *all* inputs, but are the same on {"x" → 3, "y" → 7}, {"x" → 4, "y" → 4}, and {"x" → 2, "y" → 12}

This is exactly as simple as it looks. Seriously, you can write this synthesizer in vanilla Python in one page. Let's see it!

```

1 import itertools
2 class Op:
3     ops = {"+": lambda a,b: a+b, "-": lambda a,b: a-b, "*": lambda a,b: a*b}
4     def __init__(self, a, op, b):
5         self.a = a; self.op = op; self.b = b
6     def __repr__(self):
7         return "(" + str(self.a) + self.op + str(self.b) + ")"
8     def interpret(self, argDict):
9         return Op.ops[self.op](self.a.interpret(argDict), self.b.interpret(argDict))
10 class Val:
11     def __init__(self, v):
12         self.v = v
13     def __repr__(self):
14         return str(self.v)
15     def interpret(self, argDict):
16         return self.v
17 class Var:
18     def __init__(self, n):
19         self.n = n
20     def __repr__(self):
21         return self.n
22     def interpret(self, argDict):
23         return argDict[self.n]
24
25 spec = [{"x": 3, "y": 7}, 23], # our input-output pairs
26         {"x": 4, "y": 4}, 19),
27         {"x": 2, "y": 12}, 31]]
28 expected_outputs = [output for inputDict, output in spec] # for convenience, the outputs we expect for our i-o pairs
29 def test_against_spec(expr):
30     outputs = [expr.interpret(inputDict) for inputDict, output in spec] # run the expression on our target inputs
31     if (outputs == expected_outputs):
32         print "found it!", expr
33         exit()
34
35 exprs = [Val(x) for x in range(5)] + [Var(x) for x in spec[0][0].keys()] # the starting set is 0 through 5 and our args
36 print "level 0:\n", exprs, "\ncount:", len(exprs)
37 for expr in exprs:
38     test_against_spec(expr) # let's just see if any of our starting exprs do the trick...
39
40 ops = Op.ops.keys() # what operators are we allowed to use?
41 level = 0
42 while(True):
43     level += 1
44     print "level", level, ":"
45     for pair in itertools.product(exprs, exprs): #let's make bigger expressions!
46         for op in ops:
47             new_expr = Op(pair[0], op, pair[1])
48             test_against_spec(new_expr)
49             exprs.append(new_expr)
50 print exprs, "\ncount:", len(exprs)

```

This one isn't pruning at all.
What do we do to prune with OE?

Just an extra 6 lines!

Pruning based on Operational
Equivalence can cut down our
search space dramatically!

And this is just at level 2!

```
[Sarahs-MBP:othermaterials schasins$ python onePageSynthesizer.py
level 0:
[0, 1, 2, 3, 4, y, x]
count: 7
level 1 :
count: 154
level 2 :
count: 71302
level 3 :
found it! (3+(2*(y+x)))
[Sarahs-MBP:othermaterials schasins$ python onePageSynthesizer0E.py
level 0:
[0, 1, 2, 3, 4, y, x]
count: 7
level 1 :
count: 63
level 2 :
count: 2051
level 3 :
found it! (3+(2*(y+x)))
```

So if you're ever watching a synthesis talk and get confused...just remember enumeration. At a sufficiently high level of abstraction, it's just going through programs until it finds one that works.

We can make enumeration smarter

- Doesn't have to be just start with the smallest program, then list all the programs in order of size until you find one that works
- We can have heuristics or language models that let us explore better/likelier programs first instead of smaller programs first
- There are other ways of pruning (other than Operational Equivalence) that let us cut out much more of the space
- We can make smart choices about what constants to include
- This was the easy-to-write version, but there are many ways to make it more effective
 - For a long time, the winner of the SyGuS competition (the primary competition for people who write synthesizers) was an enumerative solver!
 - This is a real technique!

Quick brainstorm. What would
you like to synthesize?

Synthesis is like a buffet



- This is not one technique that either applies or doesn't apply to your problem
- It's a whole family of techniques
- Tackling a new problem, you'll probably be looking through a host of existing approaches and tools...
 - If you read synth literature, you'll see very different domains formalized in very different ways. This isn't accidental!
- ...and maybe inventing your own. Custom synthesizers are still common

To think about for next reading

- The issue of ambiguous specs. As designers of usable tools, do we want to prevent ambiguous specs? If yes, how? Do we want to allow them? If yes, how does this affect our synthesizer?
- What constrains the design of a our target languages for synthesis?
- What's the tradeoff between designing for making the synthesizer's task easier vs. designing for the user of the tool?

Please install before next class

https://docs.racket-lang.org/rosette-guide/ch_getting-started.html#%28part._sec~3aget%29



The Rosette Language

ABOUT

DOWNLOAD

DOCS

APPS

COURSES

PAPERS

About Rosette

Rosette is a solver-aided programming language that extends [Racket](#) with language constructs for program synthesis, verification, and more. To verify or synthesize code, Rosette compiles it to logical constraints solved with off-the-shelf [SMT](#) solvers. By combining virtualized access to solvers with Racket's metaprogramming, Rosette makes it easy to develop synthesis and verification tools for new languages. You simply

A brilliant language from
Emina Torlak