Discovering Relational Structure in Program Synthesis Problems with Analogical Reasoning

Jerry Swan and Krzysztof Krawiec

University of York
Poznan University of Technology
jerry.swan@york.ac.uk, krawiec@cs.put.poznan.pl

Genetic Programming Theory and Practice
Ann Arbor, MI

May 20, 2016

Outline

$$\boxed{\text{Analogies} \leftrightarrows \text{GP}}$$

1. Introduction
2. Analogies $\leftarrow$ GP
   - 'Solving' analogies using GP
   - Some experimental evidence
3. Analogies $\rightarrow$ GP
   - Using analogical reasoning in GP
   - Some experimental evidence
4. Discussion

Motivation

Analogy:

- *'A mapping between systems or processes'*.
- A mechanism for re-contextualising situations in terms of prior experience.
- Pervades natural language [Lakoff and Johnson, 1980]
- *'The core of cognition'* [Chalmers et al., 1992].

Computational models of analogy:

- Geometric reasoner [Evans, 1964].
- Structure Mapping Engine [Falkenhainer et al., 1989, Turney, 2008]
- Matching techniques in Case-Based Reasoning [Aamodt and Plaza, 1994].
- Heuristic-Driven Theory Projection [Schmidt et al., 2014].
- Connectionist models
  [Holyoak and Thagard, 1989, Hummel and Holyoak, 1997]

Proportional Analogy

Questions of the form:

`Gills are to fish as` *what* `are to mammals?`

or more generally:

`A : B :: C : D`

Proportional Analogy

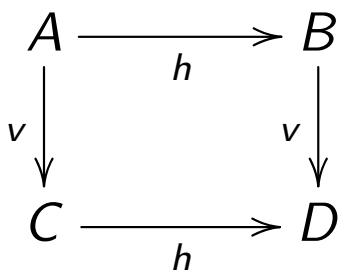
Questions of the form:

`Gills are to fish as` *what* `are to mammals?`

or more generally:

`A : B :: C : D`

Can be framed using commutativity:



In contrast to traditional optimization problems:

- More than one 'right' answer possible
- Not easy to define objective quality measure

The 'Letter String Analogy' (LSA) domain

[Hofstadter, 1995]:

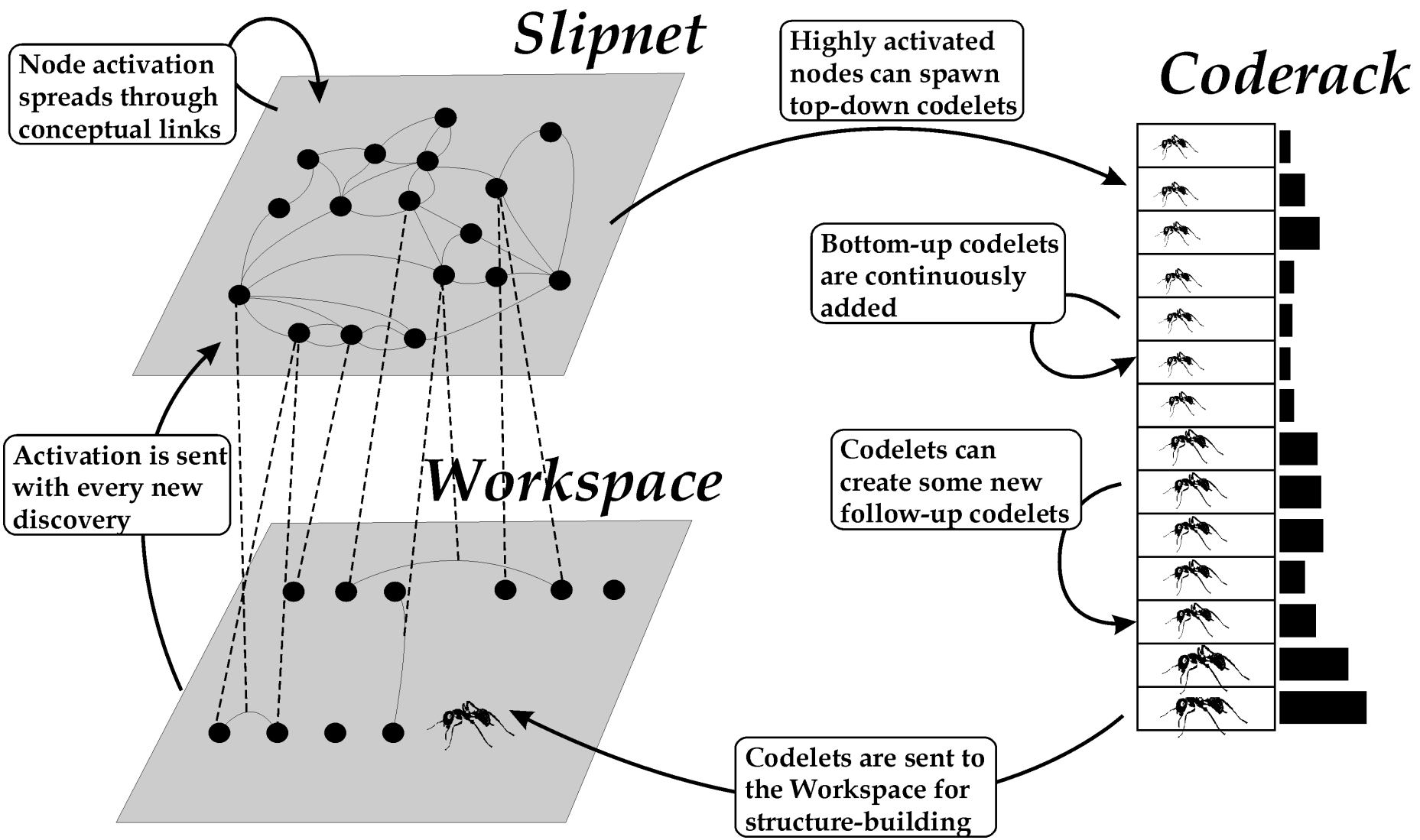
If **abc** maps to **abd**, what does **ijk** map to?

- 'Most naïve' answer is **ijd**.
- Claim the 'most compelling' answer is **ijl**.
  - Humans recognize the 'successorship' relation in both **abc** and **ijk**.
  - The most compelling rule: "increment the last letter".

Notable work in the LSA domain:

- Hofstadter and Mitchell's CopyCat [Hofstadter, 1995].
- The algebraic approach of Dastani [Dastani et al., 2003].
- Schmid's use of E-generalization [Schmid, 2003].

CopyCat Architecture



Features of CopyCat

- Highly domain-driven
- Intended as a cognitively-plausible model of analogy-making.
- Can be seen as performing 'Artificial Chemistry' on symbols.
- Feedback mechanisms complex, but meticulously engineered
  - *"Each additional mechanism and interaction is well-motivated by deeper considerations — there is very little that is ad hoc"* [Holland, 1998].

Other approaches

Dastani's Algebraic Approach [Dastani et al., 2003]

- Uses *Structural Information Theory* (SIT): `Symmetry`, `Iteration` and `Alternation`, e.g.
  - `Sym(a,bb)` → `abba`,
  - `Alt(a,bb)` → `abab`,
  - `Iter(a,Succ,3)` → `abc`.
- Expressions using these primitives are intended to correspond to *Gestalt* preferences for human perception [Ehrenfels, 1890].
- *Information Load* to measure SIT's quality:
  - E.g., *Iter*`(ab,id,2)` preferred to *Alt*`(a,bb)`

Schmid's E-generalization [Schmid and Burghardt, 2003]

- Also based on SIT
- Uses *Regular Tree Grammars* — a set of production rules over SIT expressions (a 'tree of trees').
- Performs Anti-Unification on the tree grammars representing the analogy.

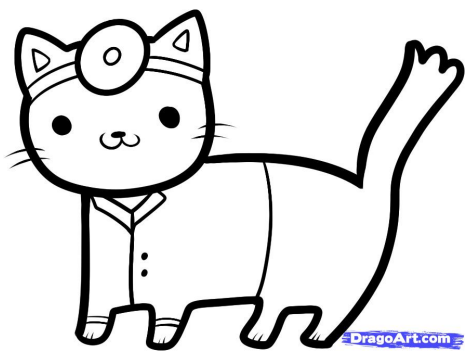
GPCat - A GP-based alternative to CopyCat

Analogies ← GP

GPCat combines both formal and generative techniques.

Given: Some analogy of the form $A : B :: C : ?$

1. **Evolve** a population of *triples* of SIT terms, (t_A, t_B, t_C).
2. **Generate** solutions to analogy from the best-of-run individual.



GpCat: Fitness function

Fitness = an aggregate of:

- Agreement with the known elements of analogy (A,B,C):
  - Total Levensthein distance $Lev(t_A, A) + Lev(t_B, B) + Lev(t_C, C)$ (min)
- Complexity:
  - Total information load $InfLoad(t_A) + InfLoad(t_B) + InfLoad(t_C)$ (min)
- Structural consistency between the SITs for B and C:
  - The total number of variables in (max)
  - The number of mappings to null value (i.e. $\$j \mapsto \epsilon$) (min)

Anti-unification example:

`Seq(Iter(a,Succ,3),g)`

`Seq(Iter(Group(c,c),Pred,3),h)`

the AU is `Seq(Iter($1,$2,3), $3)`, with substitutions:

$$\sigma_B = \{\$1 \mapsto a, \$2 \mapsto Succ, \$3 \mapsto g\}$$
$$\sigma_C = \{\$1 \mapsto Group(c, c), \$2 \mapsto Pred, \$3 \mapsto h\}$$

GpCat: Generating solutions to analogy problem

Given best-of-run individual:

1. Perform AU of σ_B and σ_C.
2. Combinatorially perform all alignments of variables in AU result
3. The resulting letter strings are the proposed values of D.

GpCat - Results

GPCat	Most frequent answers (top 5 over 30 runs)				
abc:abd::ijk	ijl:100	ik:7	bcd:7	abbd:7	ac:7
abc:abd::xyz	xya:100	bcd:7	abbd:7	xz:0.07	ac:7
abc:abd::kji	ijl:70	cba:57	kln:17	bce:10	jl:7
abc:qbc::iijjkk	aabbcc:53	ijl:43	ab:23	ij:23	ik:10
abc:abd::mrrjjj	jkm:67	iiaaa:33	rrjjj:33	jrrjjj:17	diiaaa:17

CopyCat	Most frequent answers (percent)				
abc:abd::ijk	ijl:96.9	ijd:2.7	ijk:0.2	hjk:0.1	ijj:0.1
abc:abd::xyz	xyd:81.1	wyz:11.4	yyz:6	dyz:0.7	xyz:0.4
abc:abd::kji	kjh:56.1	kjj:23.8	lji:18.6	kjd:1.1	kki:0.3
abc:abd::iijjkk	iijjll: 81.0	iijjkl: 16.5	iijjdd: 0.9	iikkll: 0.9	iijkll: 0.3
abc:abd::mrrjjj	mrrkkk:70.5	mrrjjk:19.7	mrrjkk:4.8	mrrjjjj:4.2	mrrjjd:0.6

- Outcomes partially coincide with those of CopyCat.
- Parameter-tuning needed for correspondence against human bias.
- Next step: Regress GPCat parameters so that outputs are closer to the distribution of human answers (York Summer internship project).

Analogical Reasoning for Program Synthesis

$$\boxed{\text{Analogies} \longrightarrow \text{GP}}$$

Analogical Reasoning for Program Synthesis

$$\boxed{\text{Analogies} \longrightarrow \text{GP}}$$

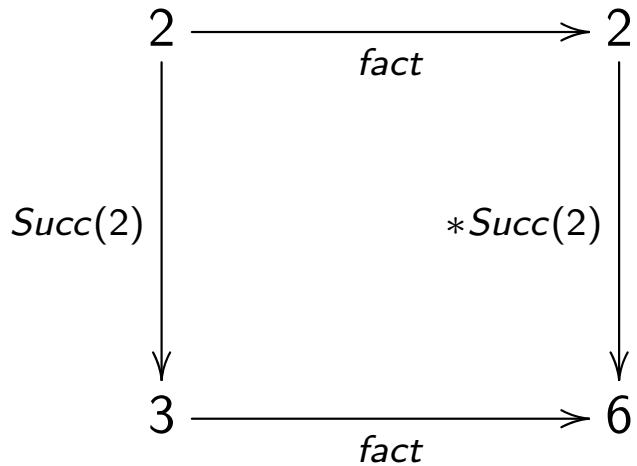
- Where to look for analogies in program synthesis?
- Several possibilities:
  - Tests
  - Program traces
  - ...

Example 1: Factorial

Fitness cases:

`fact(2)` $\rightarrow$ `2`

`fact(3)` $\rightarrow$ `6`

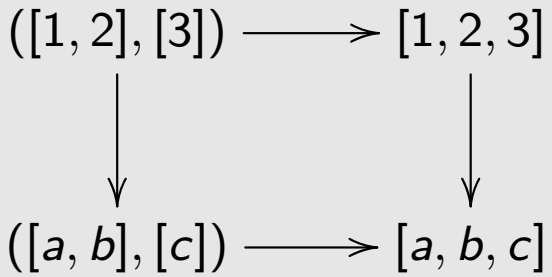


Pair of tests = **I/O analogy**.

Example 2: Append

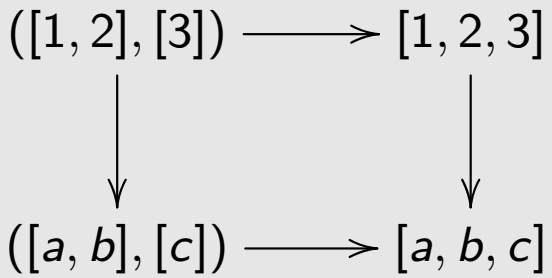
```
append([1,2], [])       = [1,2]
append([1,2], [3])      = [1,2,3]
append([1,2,3], [])     = [1,2,3]
append([a,b], [c])      = [a,b,c]
```

Example 2: Append

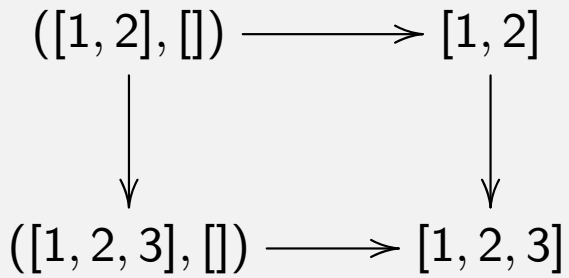


Type abstraction

Example 2: Append

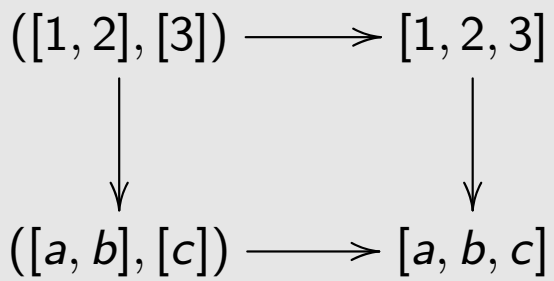


Type abstraction

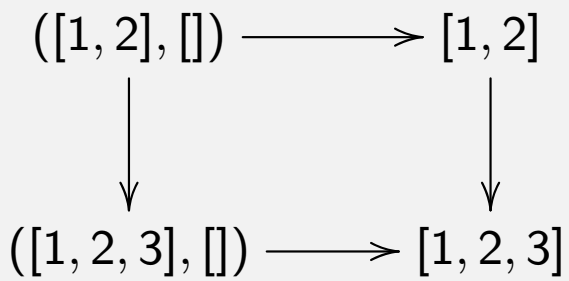


Neutrality

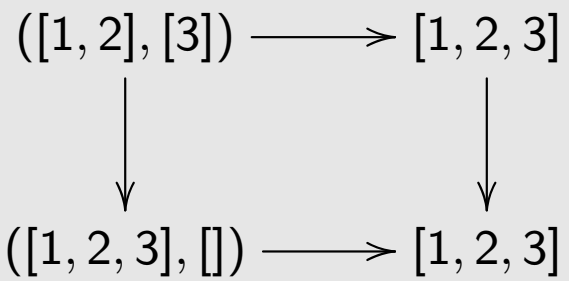
Example 2: Append



Type abstraction

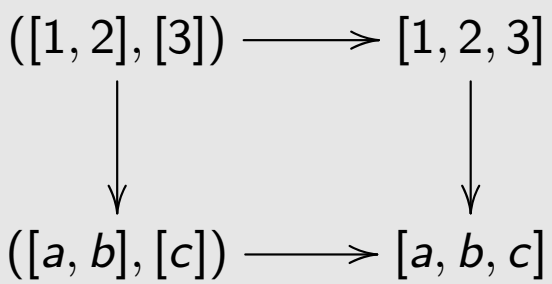


Neutrality

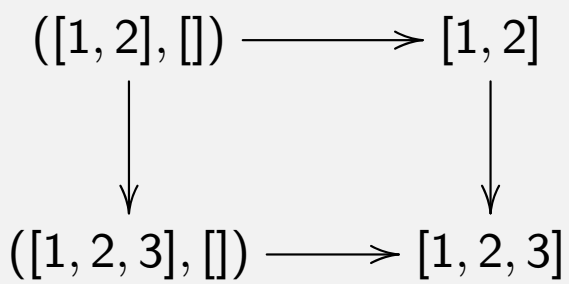


Invariance

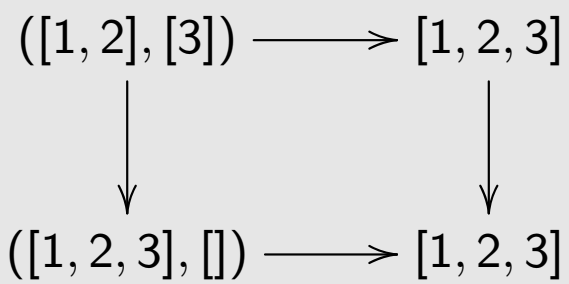
Example 2: Append



Type abstraction



Neutrality



Invariance

I/O analogies capture three unrelated characteristics of the task.

Example 2: Append

- `[1,2,3]` can be expressed as:

`Cons(1,Cons(2,Cons(3,Nil)))`

- Using SIT-style relations, this can be represented as

`Iter(Nil,Succ,3)`

- `Iter` here expresses a *catamorphism*.
- Hence, analogy #2 can be represented by the AU:

`App(Iter(Nil, succ, $1), Nil), Iter(Nil, succ, $1))`

- The substitutions $\sigma_1 = \{\$1 \mapsto 2\}$, $\sigma_2 = \{\$1 \mapsto 3\}$, reflect the fact that appending `Nil` preserves structure.

Promoting analogies in GP: A naive approach

Intent: Make GP pay more attention to analogies between tests.

- **Given:** Set of tests $T = \{(I_i, O_i)\}$.
- *Structural Analogy* (SA): a pair of tests $((I_1, O_1), (I_2, O_2)) \in T \times T$, such that:
  1. $\mathit{Hamming}(I_1, I_2) = 1$, and
  2. $O_1 \neq O_2$

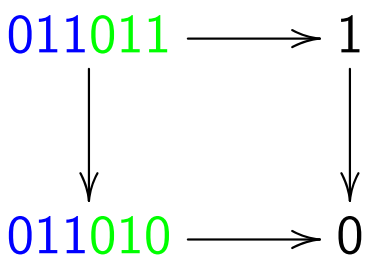
Promoting analogies in GP: A naive approach

Intent: Make GP pay more attention to analogies between tests.

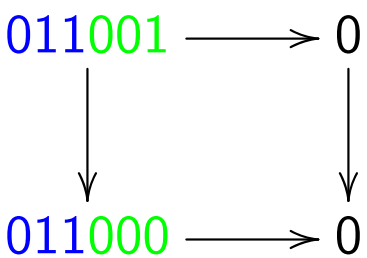
- **Given:** Set of tests $T = \{(I_i, O_i)\}$.
- *Structural Analogy* (SA): a pair of tests $((I_1, O_1), (I_2, O_2)) \in T \times T$, such that:
  1. $\mathit{Hamming}(I_1, I_2) = 1$, and
  2. $O_1 \neq O_2$

Examples for 6-bit comparator problem:

Structural analogy:



No structural analogy:



The method

Idea: Evaluate solutions on structural analogies instead of tests.

- A = the set of all analogies built from the given set of tests T
  - For n binary variables, $|A| \leq |T|n$
- Fitness = the number of passed analogies.
  - A program p passes an analogy $a = (t_1, t_2)$ if it passes both t_1 and t_2.
- Some $t \in T$ may be absent in A
  - For such ts, we extend A with (t, t)

Results

SA vs. GP: success rate (200 runs):

	cmp6	disc1	disc2	disc3	disc4	disc5	maj5	maj6	maj7	mal1	mal2	mal3	mal4	mal5	mux6	par5
GP	6.5	0.0	0.0	0.0	0.0	0.0	81.0	35.5	**2.5**	12.0	3.0	13.5	0.5	52.5	78.5	0.0
SA	**32.0**	**1.5**	**1.0**	**9.5**	**0.0**	**0.5**	**83.0**	**51.0**	1.5	**17.5**	4.5	**15.0**	2.0	**62.0**	**93.5**	0.0

- SA approach clearly better.
- Same computational cost as GP.
- Alternative explanations?

Could this be due to finer granularity of fitness?

Number of tests $|T|$ vs. number of structural analogies $|A|$:

	cmp6	disc1	disc2	disc3	disc4	disc5	maj5	maj6	maj7	mal1	mal2	mal3	mal4	mal5	mux6	par5
$\|T\|$	64	27	27	27	27	27	32	64	128	15	15	15	15	15	64	32
$\|A\|$	72	42	42	42	42	42	42	89	198	24	24	24	24	24	64	80

- $|A|$ only moderately greater than $|T|$ (yet greater)

Could this be due to finer granularity of fitness?

Number of tests $|T|$ vs. number of structural analogies $|A|$:

	cmp6	disc1	disc2	disc3	disc4	disc5	maj5	maj6	maj7	mal1	mal2	mal3	mal4	mal5	mux6	par5
$\|T\|$	64	27	27	27	27	27	32	64	128	15	15	15	15	15	64	32
$\|A\|$	72	42	42	42	42	42	42	89	198	24	24	24	24	24	64	80

- $|A|$ only moderately greater than $|T|$ (yet greater)
- Hence control configurations:
  - SA-rand: SA with second tests in analogies randomly shuffled
  - H1: no requirement of different output in analogies
  - H1-rand: H1 + random shuffling

Could this be due to finer granularity of fitness?

Number of tests $|T|$ vs. number of structural analogies $|A|$:

	cmp6	disc1	disc2	disc3	disc4	disc5	maj5	maj6	maj7	mal1	mal2	mal3	mal4	mal5	mux6	par5
$\|T\|$	64	27	27	27	27	27	32	64	128	15	15	15	15	15	64	32
$\|A\|$	72	42	42	42	42	42	42	89	198	24	24	24	24	24	64	80

- $|A|$ only moderately greater than $|T|$ (yet greater)
- Hence control configurations:
  - SA-rand: SA with second tests in analogies randomly shuffled
  - H1: no requirement of different output in analogies
  - H1-rand: H1 + random shuffling

	cmp6	disc1	disc2	disc3	disc4	disc5	maj5	maj6	maj7	mal1	mal2	mal3	mal4	mal5	mux6	par5
GP	6.5	0.0	0.0	0.0	0.0	0.0	81.0	35.5	**2.5**	12.0	3.0	13.5	0.5	52.5	78.5	0.0
SA	**32.0**	**1.5**	**1.0**	**9.5**	**0.0**	**0.5**	**83.0**	**51.0**	1.5	**17.5**	4.5	**15.0**	2.0	**62.0**	**93.5**	0.0
SA-rand	15.0	0.0	0.0	7.5	0.0	**0.5**	64.5	35.0	**2.5**	9.5	**5.0**	12.0	**2.5**	38.0	77.5	0.0
H1	5.0	0.0	0.0	0.0	0.0	0.0	79.5	34.5	**2.5**	**17.5**	4.5	**15.0**	2.0	**62.0**	86.5	0.0
H1-rand	7.0	0.0	0.0	0.5	0.0	0.0	75.5	37.5	**2.5**	10.0	3.5	7.5	1.5	45.5	71.5	0.0

Conclusions

Analogies capture structure in problem formulation (e.g., build on tests), which:

- Captures regularities/patterns
- An induction algorithm exploits those regularities.

Consequences:

- More fine-grained information about the problem and candidate solutions
- Natural means for prioritizing and diversifying search.
- Better-informed search algorithm.
- Moving from blackbox to whitebox setting.

Future work

- Translating the observed analogies into search drivers that help deciding *what* to modify in a candidate program and *how*
  - *Learn* how to transform the observed patterns/analogies into moves.
- Applying analogical reasoning to *program traces* (cf. PANGEA [Krawiec and Swan, 2013])
- AU on higher-order objects.

References I

[Aamodt and Plaza, 1994] Aamodt, A. and Plaza, E. (1994).
Case-based reasoning: Foundational issues, methodological variations, and system approaches.
AI Commun., 7(1):39–59.

[Chalmers et al., 1992] Chalmers, D. J., French, R. M., and Hofstadter, D. (1992).
High-level perception, representation, and analogy: a critique of artificial intelligence methodology.
J. Exp. Theor. Artif. Intell., 4(3):185–211.

[Dastani et al., 2003] Dastani, M., Indurkhya, B., and Scha, R. (2003).
Analogical projection in pattern perception.
J. Exp. Theor. Artif. Intell., 15(4):489–511.

[Ehrenfels, 1890] Ehrenfels, C. v. (1890).
Über Gestaltqualitäten.
Vierteljahresschr. für Philosophie, 14, 249-292.

References II

[Evans, 1964] Evans, T. G. (1964).
A heuristic program to solve geometric-analogy problems.
In *Proceedings of the April 21-23, 1964, spring joint computer conference*, AFIPS '64 (Spring), pages 327–338, New York, NY, USA. ACM.

[Falkenhainer et al., 1989] Falkenhainer, B., Forbus, K. D., and Gentner, D. (1989).
The Structure-Mapping Engine: Algorithm and Examples.
Artificial Intelligence, 41(1):1 – 63.

[Hofstadter, 1995] Hofstadter, D. R. (1995).
Fluid Concepts and Creative Analogies: Computer Models of the Fundamental Mechanisms of Thought.
Basic Books, New York.

[Holland, 1998] Holland, J. H. (1998).
Emergence: From chaos to order.
J. Artificial Societies and Social Simulation, 1(4).

References III

[Holyoak and Thagard, 1989] Holyoak, K. J. and Thagard, P. (1989).
Analogical mapping by constraint satisfaction.
Cognitive Science, 13(3):295–355.

[Hummel and Holyoak, 1997] Hummel, J. E. and Holyoak, K. J. (1997).
Distributed representations of structure: A theory of analogical access and mapping.
Psychological Review, pages 427–466.

[Krawiec and Swan, 2013] Krawiec, K. and Swan, J. (2013).
Pattern-guided genetic programming.
In *Proceedings of the 15th Annual Conference on Genetic and Evolutionary Computation*, GECCO '13, pages 949–956, New York, NY, USA. ACM.

[Lakoff and Johnson, 1980] Lakoff, G. and Johnson, M. (1980).
Metaphors we Live by.
University of Chicago Press, Chicago.

References IV

[Schmid, 2003] Schmid, U. (2003).
Inductive Synthesis of Functional Programs, Universal Planning, Folding of Finite Programs, and Schema Abstraction by Analogical Reasoning, volume 2654 of *Lecture Notes in Computer Science*.
Springer.

[Schmid and Burghardt, 2003] Schmid, U. and Burghardt, J. (2003).
An algebraic framework for solving proportional and predictive analogies.
In *In F. Schmalhofer, et. al. (Eds.), Proceedings of the European Conference on Cognitive Science*, pages 295–300. Erlbaum.

[Schmidt et al., 2014] Schmidt, M., Krumnack, U., Gust, H., and Kühnberger, K. (2014).
Heuristic-Driven Theory Projection: An overview.
In Prade, H. and Richard, G., editors, *Computational Approaches to Analogical Reasoning: Current Trends*, volume 548, pages 163–194. Springer.

References V

[Stewart and Cohen, 1999] Stewart, I. and Cohen, J. (1999).
Figments of Reality: The Evolution of the Curious Mind.
Cambridge University Press.

[Turney, 2008] Turney, P. D. (2008).
The latent relation mapping engine: Algorithm and experiments.
J. Artif. Intell. Res. (JAIR), 33:615–655.

Anti-Unification

- Extracts the common substructure of a set of terms T.
- AU of T is a term with
  - some subterms replaced with *variables*, and
  - a *substitution* σ (i.e. a mapping from variables to terms) for each $t \in T$, such that when applied to u, it makes it equal to t, i.e., $u\sigma = t$.
- Expressiveness of AU depends on how equality between terms is defined: for *syntactic* AU, function symbols are labels with no intrinsic meaning.

Algorithm 1 Anti-unification algorithm for two terms (Reynolds,Plotkin).

function AU(x,y)
  if $x = y$ **then**
    return x
  else if $x = f(x_1, \ldots, x_n) \wedge y = f(y_1, \ldots, y_n)$ **then**
    return $f(\mathrm{AU}(x_1, y_1), \ldots, \mathrm{AU}(x_n, y_n))$
  else
    return ϕ
  end if
end function

Cognitive bias

Q: Do we need cognitive bias for program synthesis?
A: We actually may need it.

The arguments:

- Synthesized programs often need to be legible for humans.
- Human cognitive biases largely coincide with universal biases [Stewart and Cohen, 1999]:
  - A program preferred by a human will often be the 'right one' (i.e. generalize well).