Compiling Crosswords by SAT Solving

Colin Pigden[1] and Andy King

Summer Project for MSc in Computer Science

[1]We thank Daniel Le Berre from the Université d'Artois for his help with sat4j

Outline of this talk

The crossword compiling problem

Davis-Putnam-Logemann-Loveland algorithm

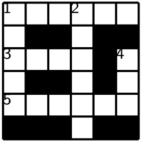
Encoding a crossword as a propositional formula

Refinements

The crossword compiling problem (fill-in crossword)

- Given a dictionary of words and a crossword grid:

torque	colon	tempt	bon	mini	pique
quirky	quay	any	encore	turkey	rue
clique	droopy	crypt	anyhow	yogi	would
loci	wreath	napkin	ugly		

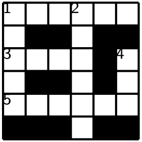


find all the ways (if any) of arranging the words into the grid.

The crossword compiling problem (fill-in crossword)

- Given a dictionary of words and a crossword grid:

torque	colon	tempt	bon	mini	pique
quirky	quay	any	encore	turkey	rue
clique	droopy	crypt	anyhow	yogi	would
loci	wreath	napkin	ugly		



  find all the ways (if any) of arranging the words into the grid.
- Can the problem of tackled with a modern SAT solver, ie, can the problem be encoded such that:
  - the size of the formulae (the number of clauses) is not $O(\ldots d^2 \ldots)$, or worse, where d is the number of words in the dictionary?
  - the number of variables does not typically exceed 1000?

Davis-Putnam-Logemann-Loveland[2] (DPLL) algorithm

- Given a propositional formula, f say, does there exist a variable assignment (a model) under which f evaluates to true?
- Although SAT is NP-complete, efficient solvers do exist for many SAT instances [Stålmarck,US Patent N527689,1995]
- A model for $f = (\neg u \vee v) \wedge (\neg w \vee u) \wedge (\neg w \vee \neg v)$ is $\theta = \{u \mapsto \mathit{false}, v \mapsto \mathit{false}, w \mapsto \mathit{false}\}$

[2] See invited paper by Zhang and Malik, "The Quest for Efficient Boolean Satisfiability Solvers", CAV, LNCS, volume 2404, 2002

Davis-Putnam-Logemann-Loveland[2] (DPLL) algorithm

- Given a propositional formula, f say, does there exist a variable assignment (a model) under which f evaluates to true?
- Although SAT is NP-complete, efficient solvers do exist for many SAT instances [Stålmarck,US Patent N527689,1995]
- A model for $f = (\neg u \vee v) \wedge (\neg w \vee u) \wedge (\neg w \vee \neg v)$ is $\theta = \{u \mapsto \mathit{false}, v \mapsto \mathit{false}, w \mapsto \mathit{false}\}$
- bool function DPLL(f, θ)

```
begin
    ⟨θ′, unsat⟩ = unit(f, θ);
    if (unsat) return false;
    else if (isSatisfied(f, θ′)) return true;
    else
        let x ∈ var(f) − var(θ′);
        if (DPLL(f, θ′ ∪ {x ↦ true})) return true;
        else return DPLL(f, θ′ ∪ {x ↦ false});
end
```

[2] See invited paper by Zhang and Malik, "The Quest for Efficient Boolean Satisfiability Solvers", CAV, LNCS, volume 2404, 2002

Walk-through for $f = (\neg u \vee v) \wedge (\neg w \vee u) \wedge (\neg w \vee \neg v)$

Consider DPLL(f, θ_1) where $\theta_1 = \emptyset$

1. unit$(f, \theta_1) = \langle \theta_1', \mathit{false} \rangle$ where $\theta_1' = \emptyset$
2. isSatisfied$(f, \theta_1') = \mathit{false}$
3. Choose $w \in \mathit{var}(f) \setminus \mathit{var}(\theta_1') = \{u, v, w\} \setminus \emptyset = \{u, v, w\}$
4. Consider DPLL(f, θ_2) where $\theta_2 = \{w \mapsto \mathit{true}\}$
   - 4.1 unit$(f, \theta_2) = \langle \theta_2', \mathit{true} \rangle$ where $\theta_2' = \theta_2 \cup \{u \mapsto \mathit{true}, v \mapsto \mathit{false}\}$
   - 4.2 Thus DPLL$(f, \theta_2) = \mathit{false}$
5. Now consider DPLL(f, θ_2) where $\theta_2 = \{w \mapsto \mathit{false}\}$
   - 5.1 unit$(f, \theta_2) = \langle \theta_2', \mathit{false} \rangle$ where $\theta_2' = \{w \mapsto \mathit{false}\}$
   - 5.2 Choose $u \in \mathit{var}(f) \setminus \mathit{var}(\theta_2') = \{u, v, w\} \setminus \{w\} = \{u, v\}$
   - 5.3 Consider DPLL(f, θ_3) where $\theta_3 = \{w \mapsto \mathit{false}, u \mapsto \mathit{true}\}$
     - unit$(f, \theta_3) = \langle \theta_3', \mathit{false} \rangle$ and $\theta_3' = \theta_3 \cup \{v \mapsto \mathit{true}\}$
     - isSatisfied$(f, \theta_3') = \mathit{true}$
     - Thus DPLL$(f, \theta_3) = \mathit{true}$
   - 5.4 Thus DPLL$(f, \theta_2) = \mathit{true}$
6. Thus DPLL$(f, \theta_1) = \mathit{true}$

Some notes on the DPLL algorithm

- Solvers usually return the model and DPLL solvers can systematically enumerate all models;

Some notes on the DPLL algorithm

- Solvers usually return the model and DPLL solvers can systematically enumerate all models;
- More variables unit assigns, the more recursive calls are avoided;

Some notes on the DPLL algorithm

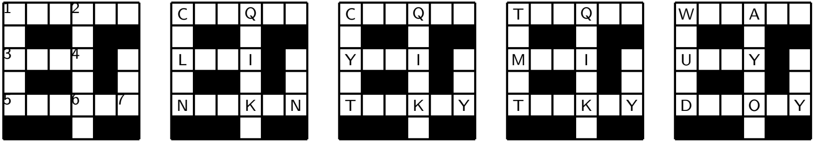
- Solvers usually return the model and DPLL solvers can systematically enumerate all models;
- More variables unit assigns, the more recursive calls are avoided;
- SAT is a "low-entry topic" because of the simplicity of DPLL;

Some notes on the DPLL algorithm

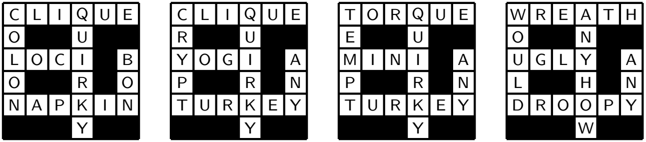
- Solvers usually return the model and DPLL solvers can systematically enumerate all models;
- More variables unit assigns, the more recursive calls are avoided;
- SAT is a "low-entry topic" because of the simplicity of DPLL;
- SAT research addresses topics such as:
  - Examining failing paths and adding new clauses to ensure that similar paths are not explored again;
  - Examining the structure of the SAT instance to assign variables in an intelligent order;
  - Investigating phase-transition behaviour;
  - SAT encoding and SAT applications

Encoding a crossword as a CNF formula (reduction)

- It is sufficient to find (encode) all combinations of characters that can arise at the intersection points between words



- Flesh out the words by searching the dictionary (note that two or more words might match the same intersection points)



Encoding a crossword as a CNF formula (compositionality)

- The 7 characters at intersection points can be represented by 35 propositional variables $x_1, \ldots, x_{35}$ where:
  - $\neg x_1 \wedge \neg x_2 \wedge \neg x_3 \wedge \neg x_4 \wedge \neg x_5$ expresses that intersection point 1 is character 0, ie, a
  - $\neg x_6 \wedge \neg x_7 \wedge \neg x_8 \wedge x_9 \wedge \neg x_{10}$ expresses that intersection point 2 is character 2, ie, c
- Suppose that:
  - $f_1(x_1, \ldots, x_{10})$ expresses the relationships between points 1 and 2 imposed by the horizontal starting at square 1;
  - $f_2(x_1, \ldots, x_5, x_{11}, \ldots, x_{15}, x_{21}, \ldots, x_{25})$ between points 1, 3 and 5 imposed by the vertical starting at square 1;
  - ...
  - $f_6(x_{30}, \ldots, x_{35})$ expresses the relationships on point 7 imposed by the vertical ending at square 7;

[Draw grid with intersection points on board]

Encoding a crossword as a CNF formula (compositionality)

- The 7 characters at intersection points can be represented by 35 propositional variables $x_1, \ldots, x_{35}$ where:
  - $\neg x_1 \wedge \neg x_2 \wedge \neg x_3 \wedge \neg x_4 \wedge \neg x_5$ expresses that intersection point 1 is character 0, ie, a
  - $\neg x_6 \wedge \neg x_7 \wedge \neg x_8 \wedge x_9 \wedge \neg x_{10}$ expresses that intersection point 2 is character 2, ie, c
- Suppose that:
  - $f_1(x_1, \ldots, x_{10})$ expresses the relationships between points 1 and 2 imposed by the horizontal starting at square 1;
  - $f_2(x_1, \ldots, x_5, x_{11}, \ldots, x_{15}, x_{21}, \ldots, x_{25})$ between points 1, 3 and 5 imposed by the vertical starting at square 1;
  - ...
  - $f_6(x_{30}, \ldots, x_{35})$ expresses the relationships on point 7 imposed by the vertical ending at square 7;
- Then $f_1(x_1, \ldots, x_{35}) \wedge \ldots \wedge f_6(x_1, \ldots, x_{35})$ is a *CNF* formula that expresses the relationships between all intersection points

[Draw grid with intersection points on board]

Generating the formula $f_1(x_1, \ldots, x_{10})$

- Scan the dictionary for all 6 letter words and extract the first and fourth characters:

torque	colon	tempt	bon	mini	pique	tq	qr	eo
quirky	quay	any	*encore*	*turkey*	rue	tk	cq	do
clique	*droopy*	crypt	*anyhow*	yogi	would	ah	wa	nk
loci	wreath	*napkin*	ugly					

- Interpret as 10-bit numbers, sort and encode as a formula:

ah	00000,00111
cq	00010,10000
...	
qr	10000,10001
wa	10111,00000

$$f_1 = \bigvee \begin{cases} \neg x_1 \wedge \neg x_2 \wedge \neg x_3 \wedge \neg x_4 \wedge \neg x_5 \wedge \neg x_6 \wedge \neg x_7 \wedge \; x_8 \wedge \; x_9 \wedge \; x_{10} & 00000, 00111 \\ \neg x_1 \wedge \neg x_2 \wedge \neg x_3 \wedge \; x_4 \wedge \neg x_5 \wedge \; x_6 \wedge \neg x_7 \wedge \neg x_8 \wedge \neg x_9 \wedge \neg x_{10} & 00010, 10000 \\ \quad \ldots & \\ x_1 \wedge \neg x_2 \wedge \neg x_3 \wedge \neg x_4 \wedge \neg x_5 \wedge \; x_6 \wedge \neg x_7 \wedge \neg x_8 \wedge \neg x_9 \wedge \; x_{10} & 10000, 10001 \\ x_1 \wedge \neg x_2 \wedge \; x_3 \wedge \; x_4 \wedge \; x_5 \wedge \neg x_6 \wedge \neg x_7 \wedge \neg x_8 \wedge \neg x_9 \wedge \neg x_{10} & 10111, 00000 \end{cases}$$

Generating the formula $f_1(x_1, \dots, x_{10})$ (reprise)

Alternatively $\neg f_1 = g_0 \vee g_1 \vee \dots \vee g_9$ where g_i are in DNF and:

$$g_0 = \bigvee \begin{cases} \neg x_1 \wedge \neg x_2 \wedge \neg x_3 \wedge \neg x_4 \wedge \neg x_5 \wedge \neg x_6 \wedge \neg x_7 \wedge \neg x_8 \wedge \neg x_9 \wedge \neg x_{10} & 00000, 00000 \\ \neg x_1 \wedge \neg x_2 \wedge \neg x_3 \wedge \neg x_4 \wedge \neg x_5 \wedge \neg x_6 \wedge \neg x_7 \wedge \neg x_8 \wedge \neg x_9 \wedge x_{10} & 00000, 00001 \\ \neg x_1 \wedge \neg x_2 \wedge \neg x_3 \wedge \neg x_4 \wedge \neg x_5 \wedge \neg x_6 \wedge \neg x_7 \wedge \neg x_8 \wedge x_9 \wedge \neg x_{10} & 00000, 00010 \\ \neg x_1 \wedge \neg x_2 \wedge \neg x_3 \wedge \neg x_4 \wedge \neg x_5 \wedge \neg x_6 \wedge \neg x_7 \wedge \neg x_8 \wedge x_9 \wedge x_{10} & 00000, 00011 \\ \neg x_1 \wedge \neg x_2 \wedge \neg x_3 \wedge \neg x_4 \wedge \neg x_5 \wedge \neg x_6 \wedge \neg x_7 \wedge x_8 \wedge x_9 \wedge x_{10} & 00000, 00100 \\ \neg x_1 \wedge \neg x_2 \wedge \neg x_3 \wedge \neg x_4 \wedge \neg x_5 \wedge \neg x_6 \wedge \neg x_7 \wedge x_8 \wedge \neg x_9 \wedge x_{10} & 00000, 00101 \\ \neg x_1 \wedge \neg x_2 \wedge \neg x_3 \wedge \neg x_4 \wedge \neg x_5 \wedge \neg x_6 \wedge \neg x_7 \wedge x_8 \wedge x_9 \wedge \neg x_{10} & 00000, 00110 \end{cases}$$

$$g_0 = \bigvee \begin{cases} \neg x_1 \wedge \neg x_2 \wedge \neg x_3 \wedge \neg x_4 \wedge \neg x_5 \wedge \neg x_6 \wedge \neg x_7 \wedge \neg x_8 \wedge \neg x_9 & 00000, 0000* \\ \neg x_1 \wedge \neg x_2 \wedge \neg x_3 \wedge \neg x_4 \wedge \neg x_5 \wedge \neg x_6 \wedge \neg x_7 \wedge \neg x_8 \wedge x_9 & 00000, 0001* \\ \neg x_1 \wedge \neg x_2 \wedge \neg x_3 \wedge \neg x_4 \wedge \neg x_5 \wedge \neg x_6 \wedge \neg x_7 \wedge x_8 \wedge \neg x_9 & 00000, 0010* \\ \neg x_1 \wedge \neg x_2 \wedge \neg x_3 \wedge \neg x_4 \wedge \neg x_5 \wedge \neg x_6 \wedge \neg x_7 \wedge x_8 \wedge x_9 \wedge \neg x_{10} & 00000, 00110 \end{cases}$$

where the second g_0 is compromised of 4 implicants.

Generating the formula $g_1(x_1, \ldots, x_{10})$

$$g_1 = \bigvee \begin{cases} \neg x_1 \wedge \neg x_2 \wedge \neg x_3 \wedge \neg x_4 \wedge \neg x_5 \wedge \neg x_6 \wedge \neg x_7 \wedge \neg x_8 \wedge \neg x_9 \wedge \neg x_{10} & 00000, 01000 \\ \neg x_1 \wedge \neg x_2 \wedge \neg x_3 \wedge \neg x_4 \wedge \neg x_5 \wedge \neg x_6 \wedge \neg x_7 \wedge \neg x_8 \wedge \neg x_9 \wedge \quad x_{10} & 00000, 01001 \\ \neg x_1 \wedge \neg x_2 \wedge \neg x_3 \wedge \neg x_4 \wedge \neg x_5 \wedge \neg x_6 \wedge \neg x_7 \wedge \neg x_8 \wedge \quad x_9 \wedge \neg x_{10} & 00000, 01010 \\ \quad \ldots & \\ \neg x_1 \wedge \neg x_2 \wedge \neg x_3 \wedge \quad x_4 \wedge \neg x_5 \wedge \neg x_6 \wedge \quad x_7 \wedge \quad x_8 \wedge \neg x_9 \wedge \quad x_{10} & 00010, 01101 \\ \neg x_1 \wedge \neg x_2 \wedge \neg x_3 \wedge \quad x_4 \wedge \neg x_5 \wedge \neg x_6 \wedge \quad x_7 \wedge \quad x_8 \wedge \quad x_9 \wedge \neg x_{10} & 00010, 01110 \\ \neg x_1 \wedge \neg x_2 \wedge \neg x_3 \wedge \quad x_4 \wedge \neg x_5 \wedge \neg x_6 \wedge \quad x_7 \wedge \quad x_8 \wedge \quad x_9 \wedge \quad x_{10} & 00010, 01111 \end{cases}$$

$$g_1 = \bigvee \begin{cases} \neg x_1 \wedge \neg x_2 \wedge \neg x_3 \wedge \neg x_4 \wedge \neg x_5 \wedge \neg x_6 \wedge \neg x_7 & 00000{,}01{*}{*}{*} \\ \neg x_1 \wedge \neg x_2 \wedge \neg x_3 \wedge \neg x_4 \wedge \neg x_5 \wedge \neg x_6 & 00000{,}1{*}{*}{*}{*} \\ \neg x_1 \wedge \neg x_2 \wedge \neg x_3 \wedge \quad x_4 \wedge \neg x_5 & 00001{,}{*}{*}{*}{*}{*} \\ \neg x_1 \wedge \neg x_2 \wedge \neg x_3 \wedge \quad x_4 \wedge \neg x_5 \wedge \neg x_6 & 00010{,}0{*}{*}{*}{*} \end{cases}$$

where the first and second g_1 compromise of $1010000_2 - 111_2 - 1 = 72$ and 4 implicants respectively.

This way of obtaining a CNF encoding cannot be novel.

Complexity of the encoding

- The formulae g_0 and g_9 consists of $\leq 2 \times 5$ implicants and all other g_i consist of $\leq 2 \times 2 \times 5$ implicants
- More generally, each $\neg g_i$ consists of $O(\lg(c)m)$ clauses where:
  - c is the number of characters in the alphabet
  - m is the maximum number of intersections for any word in the grid
- Each f_i consists of $O(\lg(c)md)$ clauses and the complete system is $\wedge_i f_i$ is $O(\lg(c)mdg)$ where:
  - d is the number of words in the dictionary
  - g is the number of words in the grid

Dictionary of 73,338 words on a 60 word grid with 132 intersections

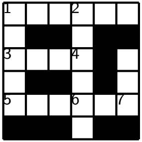
G		E		A	R	S	I	S	S	E		A	M	A	R	A	S
A	G	R	U	N		U		U		O		D		V		G	
U		E		H	A	B	E	B	A	S		F	A	U	O	R	E
D	U	P	L	E	X		A		D			O		L		U	
I		T		L	E	G	E		H	O	R	R	I	S	O	N	O
A	M	A	V	I		R			I		E		T	E	R		N
				T	R	A	N	A	B	O	Q	U	E		A	X	E
P	I	O		U		V			E		U		S	I	N		R
R		U	A	S	T	A	B	A	R		I				T	U	E
O	R	A				V		V	I	D	E	T	Q	U	E		N
P		N	I	L		I		E			S		U		S	I	T
A	S	T		A	B	S	O	L	U	O	Q	U	E				
G		E	A	T		S		L			U		R	U	E	R	E
O	B	S	T	A	V	E	R	E		S	E	S	E		X		V
	R		T		E			R		U		E	M	O	T	A	E
M	E	T	U	I	S		S	E	N	A	T	U	I		E		R
	V		L		T		P		E		U		N	O	R	I	S
R	E	G	I	A	E		E	R	U	D	I	R	I		A		A

- 1,092,868 clauses generated in 76s and sat4j solves the SAT instance in 367s $\approx$ 6m on a 3.2GHz, 1GB RAM PC
- French requires 6-bit encoding for á, â, ç, è, é, ê, ô, œ, etc

Reducing the number of propositional variables

Consider again the dictionary and the grid:

torque	colon	tempt	bon	mini	pique
quirky	quay	any	encore	turkey	rue
clique	droopy	crypt	anyhow	yogi	would
loci	wreath	napkin	ugly		

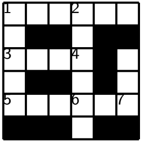


- $S_1 \subseteq \{A, C, D, E, N, Q, T, W\}$ and $S_2 \subseteq \{A, K, H, K, O, Q, R\}$

Reducing the number of propositional variables

Consider again the dictionary and the grid:

torque	colon	tempt	bon	mini	pique
quirky	quay	any	encore	turkey	rue
clique	droopy	crypt	anyhow	yogi	would
loci	wreath	napkin	ugly		

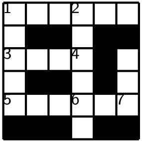


- $S_1 \subseteq \{A, C, D, E, N, Q, T, W\}$ and $S_2 \subseteq \{A, K, H, K, O, Q, R\}$
- $S_4 \subseteq \{I, Y\}$ and $S_6 \subseteq \{K, O\}$ from quirky and anyhow

Reducing the number of propositional variables

Consider again the dictionary and the grid:

torque	colon	tempt	bon	mini	pique
quirky	quay	any	encore	turkey	rue
clique	droopy	crypt	anyhow	yogi	would
loci	wreath	napkin	ugly		

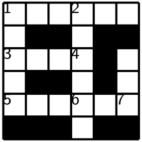


- $S_1 \subseteq \{A, C, D, E, N, Q, T, W\}$ and $S_2 \subseteq \{A, K, H, K, O, Q, R\}$
- $S_4 \subseteq \{I, Y\}$ and $S_6 \subseteq \{K, O\}$ from quirky and anyhow
- $S_5 \subseteq \{D, E, N, T\}$ and $S_7 \subseteq \{E, N, Y\}$ from encore, turkey, droopy and napkin

Reducing the number of propositional variables

Consider again the dictionary and the grid:

torque	colon	tempt	bon	mini	pique
quirky	quay	any	encore	turkey	rue
clique	droopy	crypt	anyhow	yogi	would
loci	wreath	napkin	ugly		

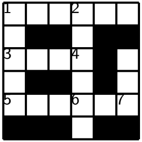


- $S_1 \subseteq \{A, C, D, E, N, Q, T, W\}$ and $S_2 \subseteq \{A, K, H, K, O, Q, R\}$
- $S_4 \subseteq \{I, Y\}$ and $S_6 \subseteq \{K, O\}$ from quirky and anyhow
- $S_5 \subseteq \{D, E, N, T\}$ and $S_7 \subseteq \{E, N, Y\}$ from encore, turkey, droopy and napkin
- $S_3 \subseteq \{L, M, Q, U, Y\}$ from mini, quay, yogi, loci and ugly

Reducing the number of propositional variables

Consider again the dictionary and the grid:

torque	colon	tempt	bon	mini	pique
quirky	quay	any	encore	turkey	rue
clique	droopy	crypt	anyhow	yogi	would
loci	wreath	napkin	ugly		



- $S_1 \subseteq \{A, C, D, E, N, Q, T, W\}$ and $S_2 \subseteq \{A, K, H, K, O, Q, R\}$
- $S_4 \subseteq \{I, Y\}$ and $S_6 \subseteq \{K, O\}$ from quirky and anyhow
- $S_5 \subseteq \{D, E, N, T\}$ and $S_7 \subseteq \{E, N, Y\}$ from encore, turkey, droopy and napkin
- $S_3 \subseteq \{L, M, Q, U, Y\}$ from mini, quay, yogi, loci and ugly
- $S_1 \subseteq \{C, T, W\}$ from colon, tempt, pique, crypt and would (note how the P is excluded)

Minimising the number of propositional variables

```
for i := 1 to 7    {    s[i] := {a, ..., z}    }
change := true
while change
    change := false
    for all w ∈ {1a, 1d, 2d, 3a, 5a, 4d}
        suppose w includes intersections i_1, ..., i_k at positions p_1, ..., p_k
        for j := 1 to k    {    t[j] = ∅    }
        read word d from dictionary until empty
            if length(d) = length(w) then
                keep := true
                for j := 1 to k
                    if char(d, p_j) ∉ s[i_j] then keep := false
                if keep then
                    for j := 1 to k    {    t[j] = t[j] ∪ {char(d, p_j)}    }
        for j := 1 to k
            if s[i_j] ∩ t[j] ⊂ s[i_j] then
                change := true; s[i_j] := s[i_j] ∩ t[j]
```

Avoiding the SAT encoding with divide-and-conquer

- Minimise S_i
- If there exists $S_i = \emptyset$ then return []
- If each $S_i = \{c_i\}$ then return $[[c_1, \ldots, c_7]]$
- Otherwise there exists $S_i = \{c_1, \ldots, c_k\}$ where $k > 1$ then
  - Put $S_i = \{c_1, \ldots, c_{\lceil k/2 \rceil}\}$ and recurse to obtain L_1
  - Put $S_i = \{c_{\lceil k/2 \rceil + 1}, \ldots, c_k\}$ and recurse to obtain L_2
- Return $\mathit{append}(L_1, L_2)$

[Relevance of principle of least commitment]

Time for a demonstration

java15 -Xmx300m -jar CrossWord.jar