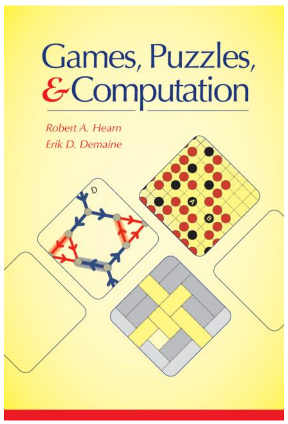
Game Complexity

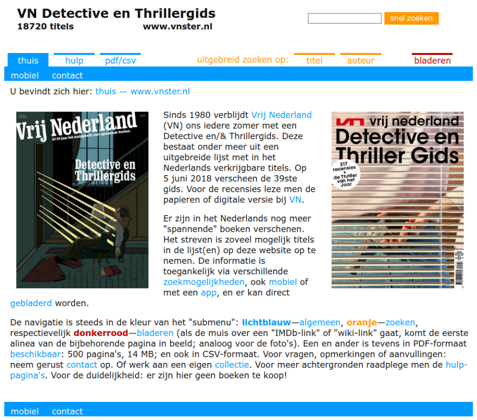
Gadgets in the Rush Hour



Walter Kosters, Universiteit Leiden

www.liacs.leidenuniv.nl/~kosterswa/

IPA, Eindhoven; Friday, January 25, 2019



VN Detective en Thrillergids
18720 titels www.vnster.nl

snel zoeken

thuis hulp pdf/csv uitgebreid zoeken op: titel auteur bladeren
mobiel contact

U bevindt zich hier: thuis — www.vnster.nl

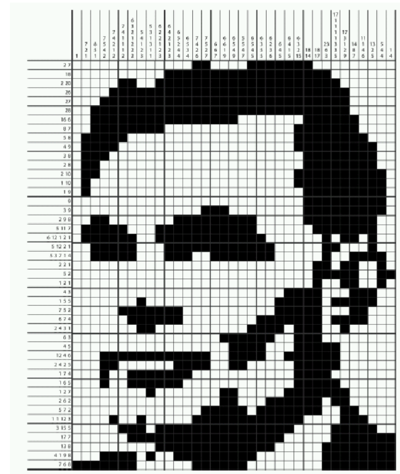
Sinds 1980 verblijdt Vrij Nederland (VN) ons iedere zomer met een Detective en/& Thrillergids. Deze bestaat onder meer uit een uitgebreide lijst met in het Nederlands verkrijgbare titels. Op 5 juni 2018 verscheen de 39ste gids. Voor de recensies leze men de papieren of digitale versie bij VN.

Er zijn in het Nederlands nog meer "spannende" boeken verschenen. Het streven is zoveel mogelijk titels in de lijst(en) op deze website op te nemen. De informatie is toegankelijk via verschillende zoekmogelijkheden, ook mobiel of met een app, en er kan direct gebladerd worden.

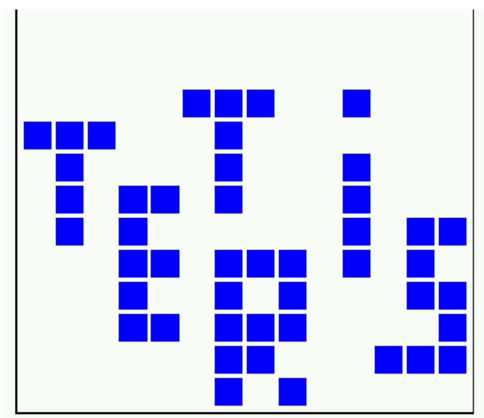
De navigatie is steeds in de kleur van het "submenu": **lichtblauw**—algemeen, **oranje**—zoeken, respectievelijk **donkerrood**—bladeren (als de muis over een "IMDb-link" of "wiki-link" gaat, komt de eerste alinea van de bijbehorende pagina in beeld; analoog voor de foto's). Een en ander is tevens in PDF-formaat beschikbaar: 500 pagina's, 14 MB; en ook in CSV-formaat. Voor vragen, opmerkingen of aanvullingen: neem gerust contact op. Of werk aan een eigen collectie. Voor meer achtergronden raadplege men de hulp-pagina's. Voor de duidelijkheid: er zijn hier geen boeken te koop!

mobiel contact

link



link



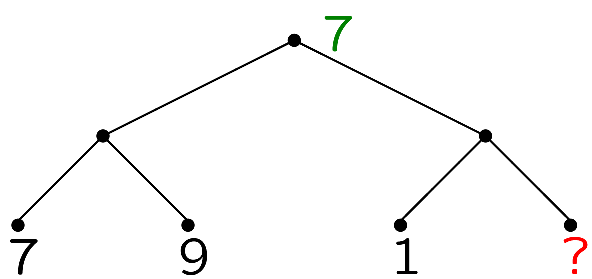
link

mystery novels,

tomography

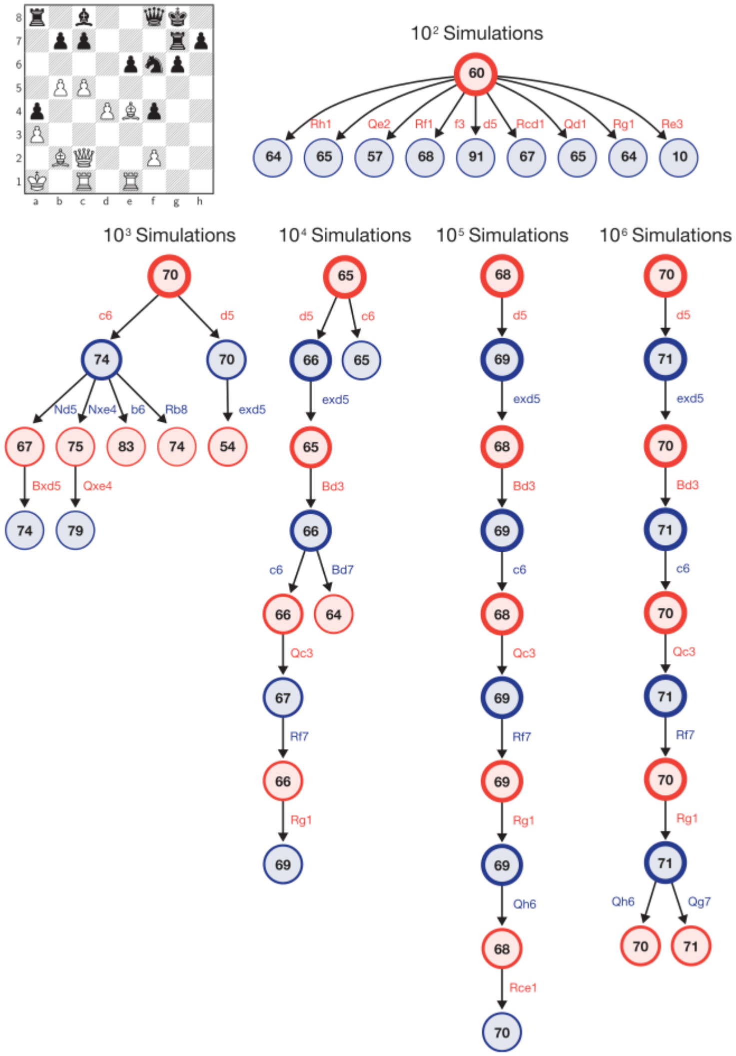
and Tetris

Deep Blue (with minimax/α-β) vs. Garry Kasparov, 1997



← MAX to move

← MIN to move

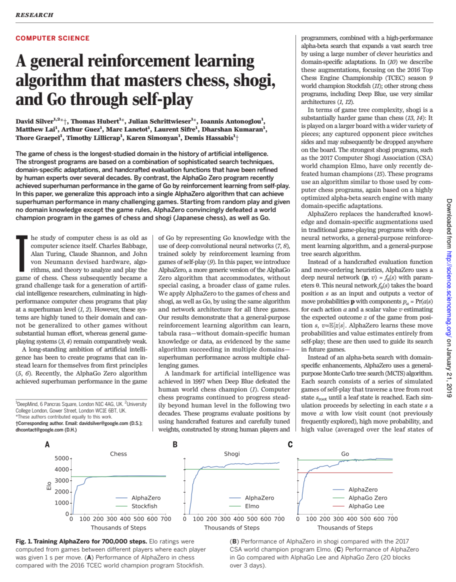


December 2018

AlphaZero

Silver et al.

Science 362, 1140–1144



RESEARCH

COMPUTER SCIENCE

A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play

David Silver[1,2]*†, Thomas Hubert[1]*, Julian Schrittwieser[1]*, Ioannis Antonoglou[1], Matthew Lai[1], Arthur Guez[1], Marc Lanctot[1], Laurent Sifre[1], Dharshan Kumaran[1], Thore Graepel[1], Timothy Lillicrap[1], Karen Simonyan[1], Demis Hassabis[1]†

The game of chess is the longest-studied domain in the history of artificial intelligence. The strongest programs are based on a combination of sophisticated search techniques, domain-specific adaptations, and handcrafted evaluation functions that have been refined by human experts over several decades. By contrast, the AlphaGo Zero program recently achieved superhuman performance in the game of Go by reinforcement learning from self-play. In this paper, we generalize this approach into a single AlphaZero algorithm that can achieve superhuman performance in many challenging games. Starting from random play and given no domain knowledge except the game rules, AlphaZero convincingly defeated a world champion program in the games of chess and shogi (Japanese chess), as well as Go.

The study of computer chess is as old as computer science itself. Charles Babbage, Alan Turing, Claude Shannon, and John von Neumann devised hardware, algorithms, and theory to analyze and play the game of chess. Chess subsequently became a grand challenge task for a generation of artificial intelligence researchers, culminating in high-performance computer chess programs that play at a superhuman level (*1*, *2*). However, these systems are highly tuned to their domain and cannot be generalized to other games without substantial human effort, whereas general game-playing systems (*3*, *4*) remain comparatively weak.

A long-standing ambition of artificial intelligence has been to create programs that can instead learn for themselves from first principles (*5*, *6*). Recently, the AlphaGo Zero algorithm achieved superhuman performance in the game of Go by representing Go knowledge with the use of deep convolutional neural networks (*7*, *8*), trained solely by reinforcement learning from games of self-play (*9*). In this paper, we introduce AlphaZero, a more generic version of the AlphaGo Zero algorithm that accommodates, without special casing, a broader class of game rules. We apply AlphaZero to the games of chess and shogi, as well as Go, by using the same algorithm and network architecture for all three games. Our results demonstrate that a general-purpose reinforcement learning algorithm can learn, tabula rasa—without domain-specific human knowledge or data, as evidenced by the same algorithm succeeding in multiple domains—superhuman performance across multiple challenging games.

A landmark for artificial intelligence was achieved in 1997 when Deep Blue defeated the human world chess champion (*1*). Computer chess programs continued to progress steadily beyond human level in the following two decades. These programs evaluate positions by using handcrafted features and carefully tuned weights, constructed by strong human players and programmers, combined with a high-performance alpha-beta search that expands a vast search tree by using a large number of clever heuristics and domain-specific adaptations. In (*10*) we describe these augmentations, focusing on the 2016 Top Chess Engine Championship (TCEC) season 9 world champion Stockfish (*11*); other strong chess programs, including Deep Blue, use very similar architectures (*1*, *12*).

In terms of game tree complexity, shogi is a substantially harder game than chess (*13*, *14*): It is played on a larger board with a wider variety of pieces; any captured opponent piece switches sides and may subsequently be dropped anywhere on the board. The strongest shogi programs, such as the 2017 Computer Shogi Association (CSA) world champion Elmo, have only recently defeated human champions (*15*). These programs use an algorithm similar to those used by computer chess programs, again based on a highly optimized alpha-beta search engine with many domain-specific adaptations.

AlphaZero replaces the handcrafted knowledge and domain-specific augmentations used in traditional game-playing programs with deep neural networks, a general-purpose reinforcement learning algorithm, and a general-purpose tree search algorithm.

Instead of a handcrafted evaluation function and move-ordering heuristics, AlphaZero uses a deep neural network $(\mathbf{p}, v) = f_\theta(s)$ with parameters θ. This neural network $f_\theta(s)$ takes the board position s as an input and outputs a vector of move probabilities $\mathbf{p}$ with components $p_a = \Pr(a|s)$ for each action a and a scalar value v estimating the expected outcome z of the game from position s, $v \approx \mathbb{E}[z|s]$. AlphaZero learns these move probabilities and value estimates entirely from self-play; these are then used to guide its search in future games.

Instead of an alpha-beta search with domain-specific enhancements, AlphaZero uses a general-purpose Monte Carlo tree search (MCTS) algorithm. Each search consists of a series of simulated games of self-play that traverse a tree from root state s_{root} until a leaf state is reached. Each simulation proceeds by selecting in each state s a move a with low visit count (not previously frequently explored), high move probability, and high value (averaged over the leaf states of

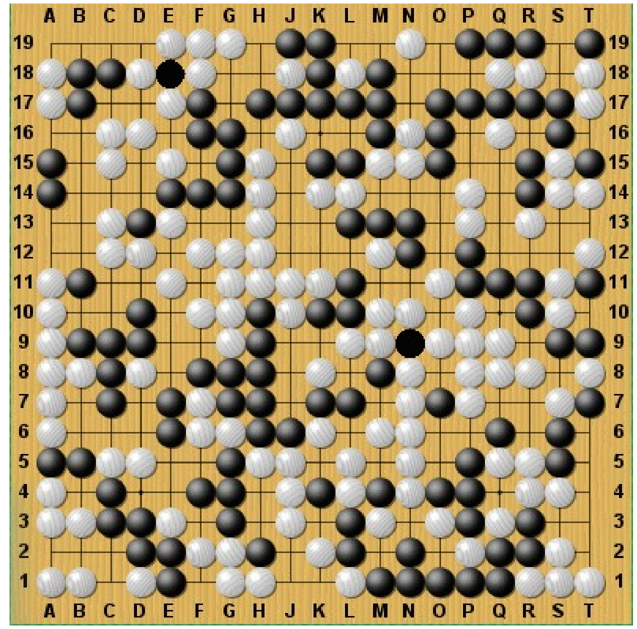
[1]DeepMind, 6 Pancras Square, London N1C 4AG, UK. [2]University College London, Gower Street, London WC1E 6BT, UK.
*These authors contributed equally to this work.
†Corresponding author. Email: davidsilver@google.com (D.S.); dhcontact@google.com (D.H.)

A Chess B Shogi C Go
Elo; Thousands of Steps
AlphaZero, Stockfish; AlphaZero, Elmo; AlphaZero, AlphaGo Zero, AlphaGo Lee

Fig. 1. Training AlphaZero for 700,000 steps. Elo ratings were computed from games between different players where each player was given 1 s per move. (**A**) Performance of AlphaZero in chess compared with the 2016 TCEC world champion program Stockfish. (**B**) Performance of AlphaZero in shogi compared with the 2017 CSA world champion program Elmo. (**C**) Performance of AlphaZero in Go compared with AlphaGo Lee and AlphaGo Zero (20 blocks over 3 days).

In 2016 John Tromp showed at CG2016 that there are

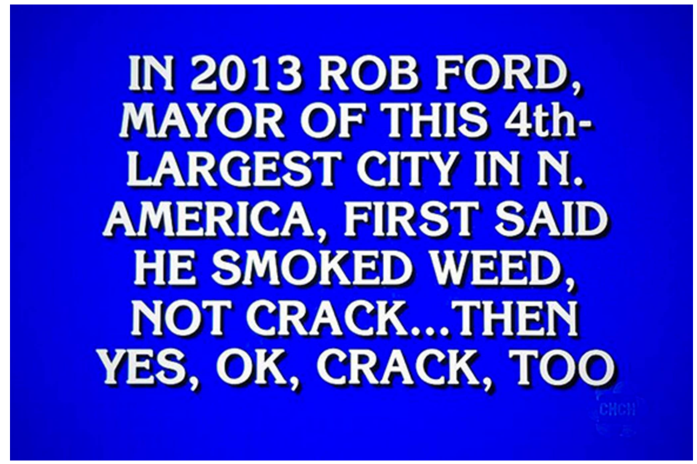
208168199381979984699478633344862770286522453884530548425639456820927419612738015378525648451698519643907259916015628128546089888314427129715319317557736620397247064840935



$\approx 2 \cdot 10^{170}$ legal positions in 19×19 Go, using dynamic programming and HARDWARE.

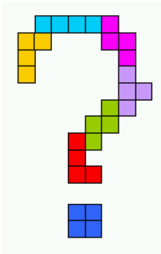
https://tromp.github.io/go/legal.html

In 2011 IBM used a computer to play "Jeopardy!":



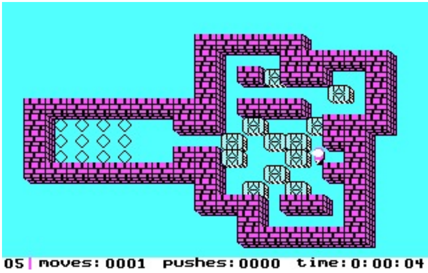
We study the complexity of games (puzzles, ...). We want to make statements like

Tetris is NP-complete.



In order to do so, we examine reductions between appropriate games, with the help of gadgets.

Games studied include TipOver, Plank puzzles, Sokoban$\rightarrow$, Rush Hour, Mahjongg, ...

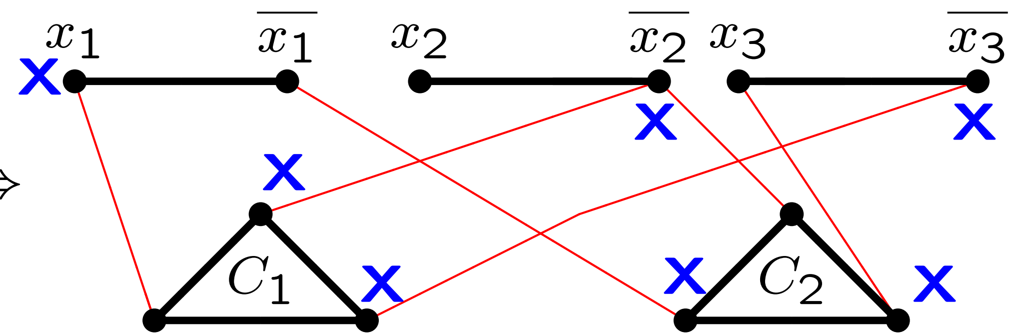


We want to reduce a known problem to a new one, for example, 3SAT to VC (so **V**ertex**C**over is NP-hard).

For every Boolean variable x_i we make a variable gadget (left) and for every clause C_j a clause gadget (right):

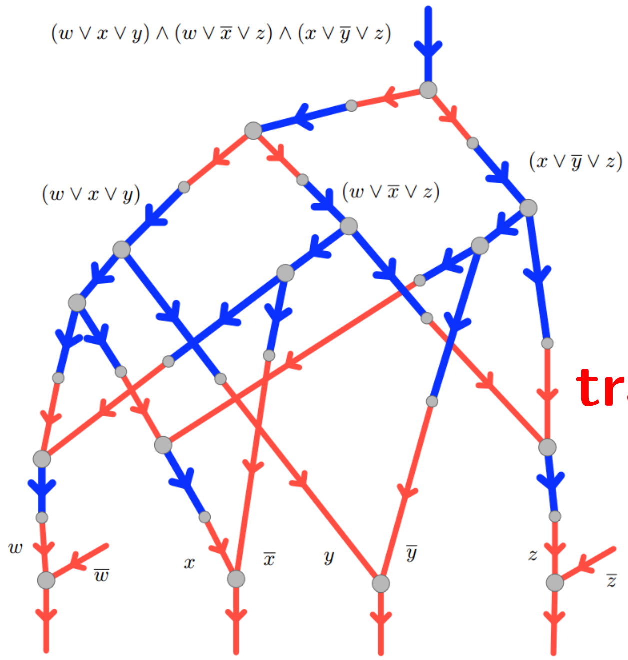
We connect these gadgets in the intuitive way; satisfying assignments (left) correspond to vertex covers (right):

$$\overbrace{(x_1 \vee \overline{x_2} \vee \overline{x_3})}^{C_1} \wedge \underbrace{(\overline{x_1} \vee \overline{x_2} \vee x_3)}_{C_2} \Rightarrow$$

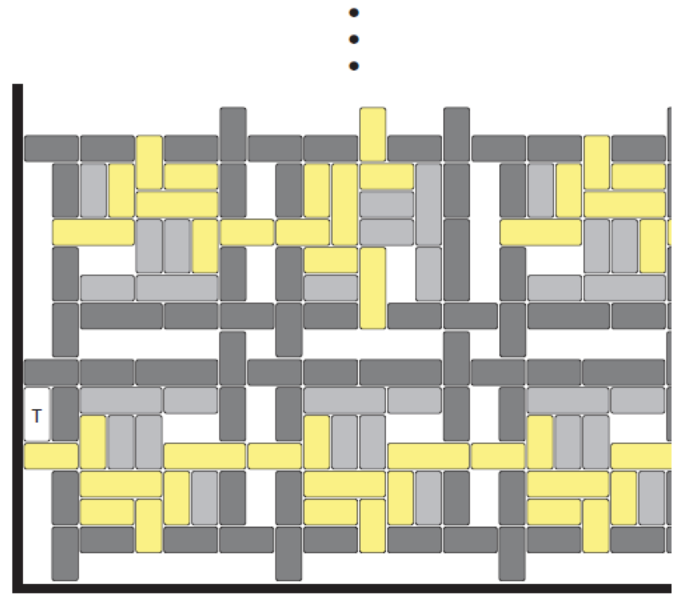


Satisfying assignment $x_1 = \texttt{true}$, $x_2 = x_3 = \texttt{false}$ gives a VC **X** of size $3 + 2 \cdot 2 = 7$, for 3 literals and 2 clauses.

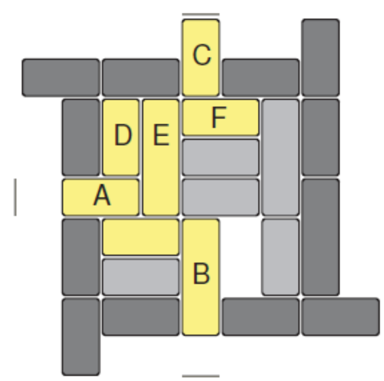
Basic idea: gadgets



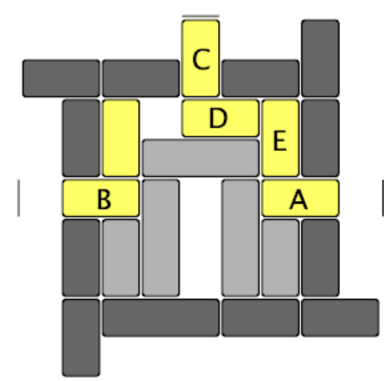
translate into



(a) Layout



(b) AND



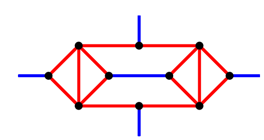
(c) Protected OR

Suppose we want to show a game to be NP/PSPACE-hard (formally: some related (y/n)-decision problem Π).

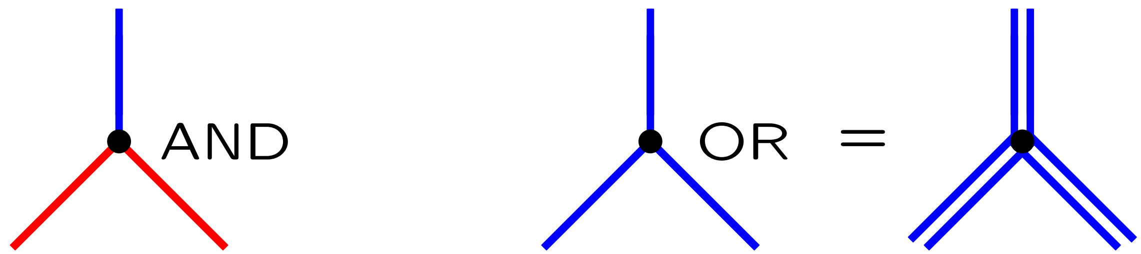
For this purpose we produce a reduction from a known well-chosen *graph* game (formally: some related (y/n)-decision problem Π', hopefully with planar graphs) to Π.

The less complicated Π' is, the better. If we are lucky, we only have to show how certain basic constructs are "emulated" by means of gadgets. Plus many details ...

We also have gadgets to emulate certain (sub)graph behaviour in the graphs themselves.

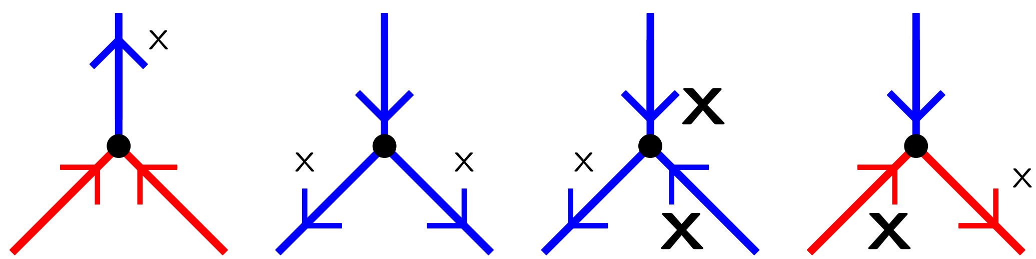


Constraint graphs consist of AND- and OR-nodes:



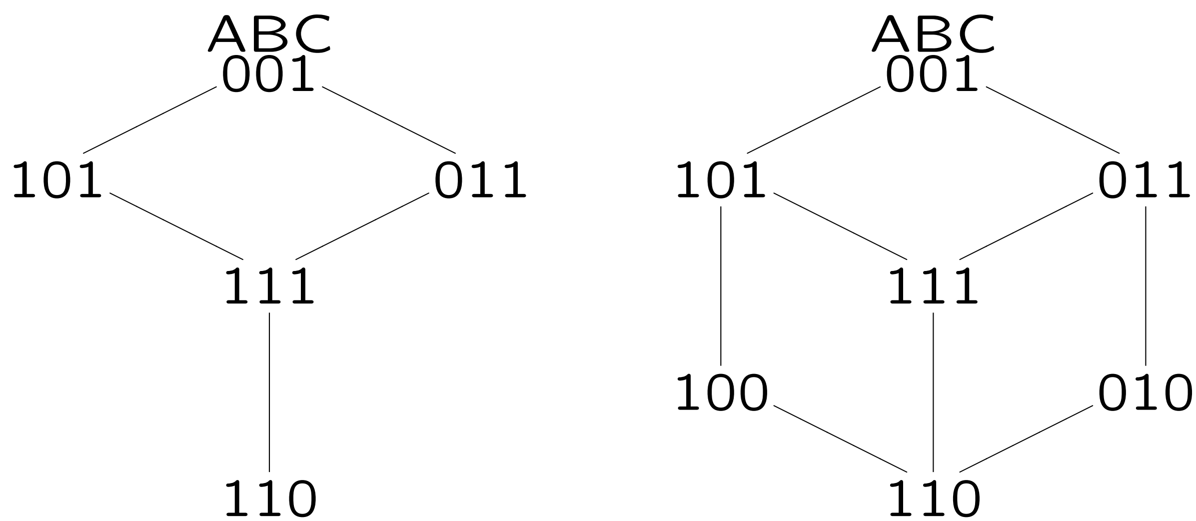
Edges are always directed such that every node = vertex receives a total input ≥ 2, where incoming blue edges contribute 2 and incoming red edges 1.

Examples:



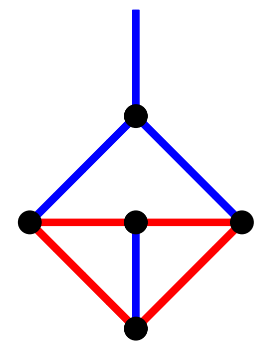
An edge can be reversed if all total inputs remain ≥ 2 (**X**).

External behavior of these gadgets can be described by the statespaces below (where 1: points in; 0: points out):



We have several simple gadgets available:

- free blue-edge terminator (FBET)
- *constrained blue-edge terminator* (CBET):

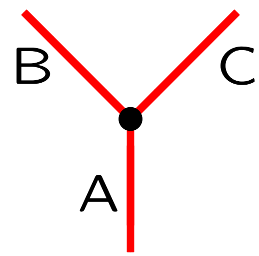


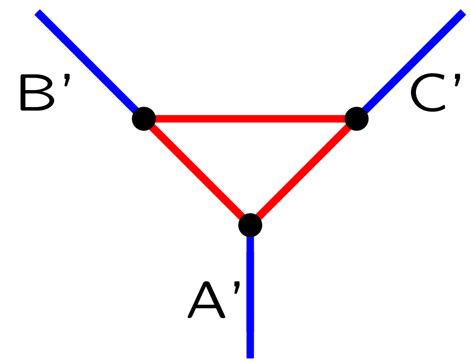
- free red-edge terminator (do we need this?)

Exercise: $\boxed{\text{Explain the CBET (arrows? statespace?).}}$

Exercise: $\boxed{\text{Develop a FBET.}}$

The *CHOICE-vertex* (left) can be emulated by the gadget on the right:





Exercise: $\boxed{\text{Show that the emulation works.}}$

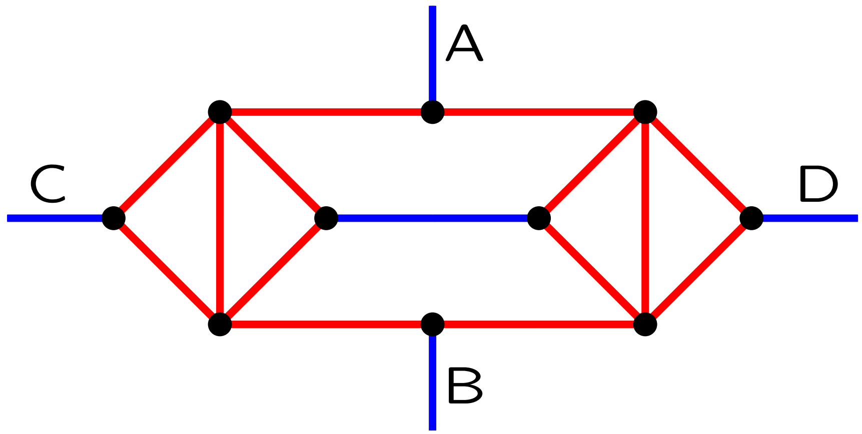
Don't worry about the fact that A, B and C are all red or blue. What matters now is whether they point in or out.

And in reality edges are always directed (have arrows)!

In many graphs we have (unavoidable) *edge crossings*.

We now want a gadget that can replace such a crossing. So assume that we have two crossing blue edges. (There is no node where the edges cross.)

If we have such a gadget, we need only emulate *planar graphs* in our reductions to specific games — and these are often planar (flat)!



Exercise: $\boxed{\text{Show that A and B may not both point out.}}$

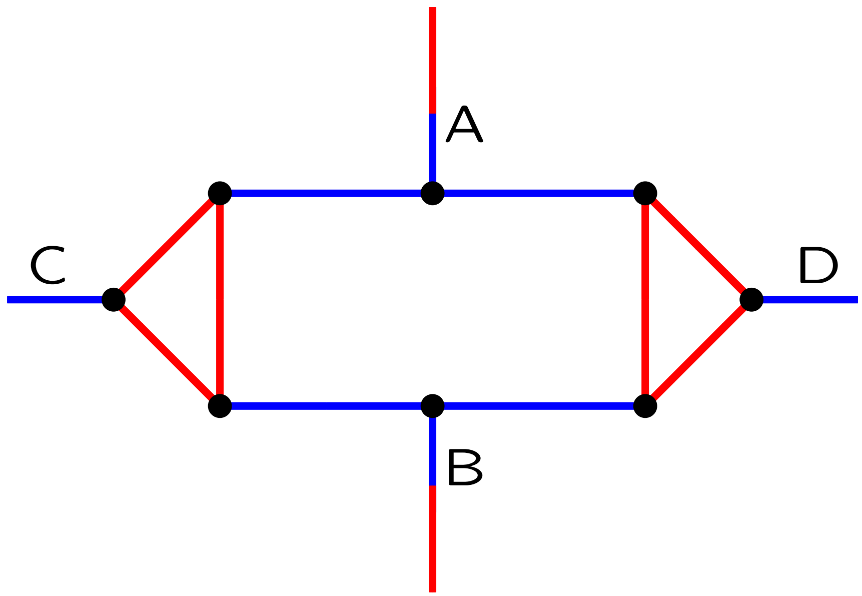
Exercise: $\boxed{\text{Show there are } \leq 2^4 - 7 = 9 \text{ states for ABCD.}}$

Exercise: $\boxed{\text{Show that this emulates two crossing edges.}}$

Exercise: $\boxed{\text{And if each edge may be reversed at most once?}}$

Wait a minute: did we just use “4-red-nodes”!?

This gadget requires any 2 from A/B/C/D to go in:

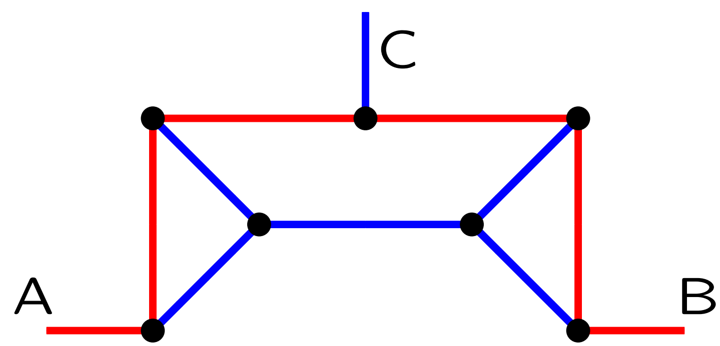


Exercise: Show that this can replace a “4-reds-node”.

Exercise: Still OK if each edge may be reversed at most once?

In that case we (unfortunately) need a “race condition”.

For a *protected-OR vertex* two of the three incident edges are special: they are *not both* allowed to be directed inward (by some outside force).



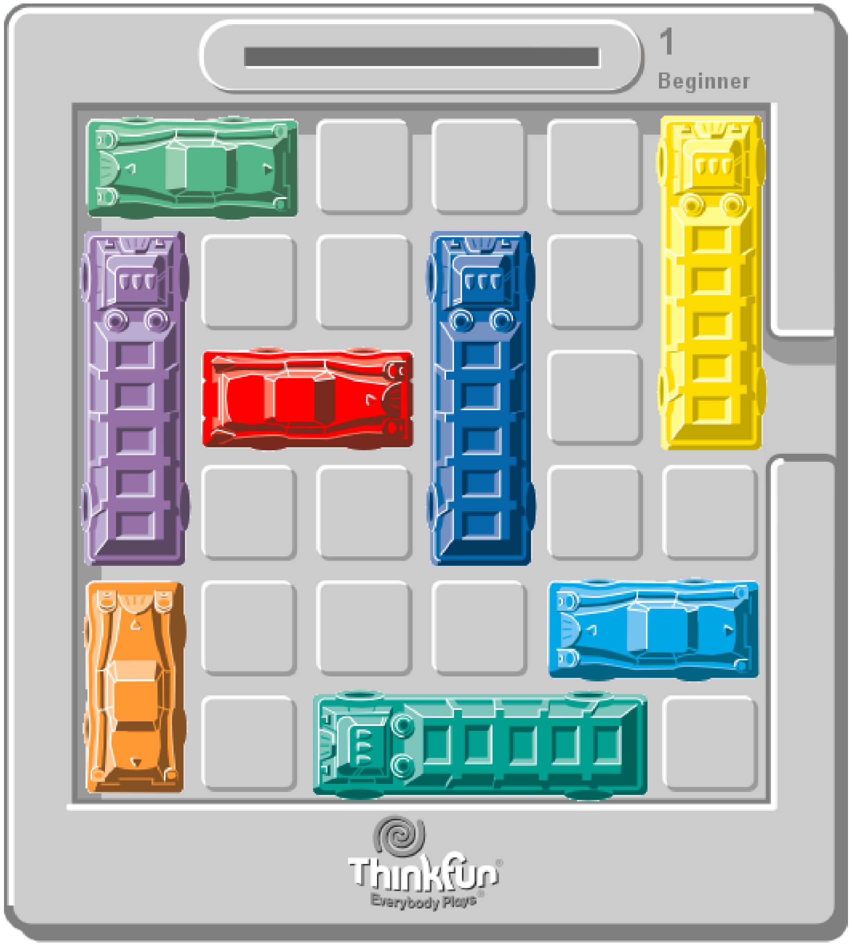
Exercise: Show that this emulates an OR-node.

Remember again that A and B can “change to blue”.

Exercise: Where are the protected-OR-nodes in the gadgets?

Exercise: Describe the statespace of a protected-OR-node.

Having seen the general picture and some gadgetry, we now examine particular games and puzzles, like Rush Hour®:



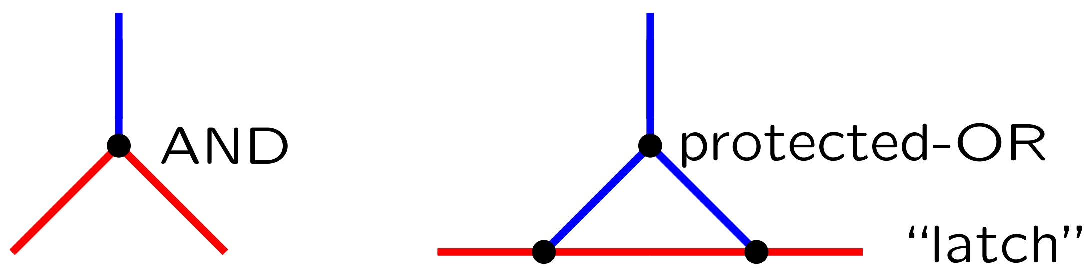
www.puzzles.com/products/rushhour.htm

The rules of Rush Hour are easy: cars may move either horizontally or vertically (left/right and up/down), in their natural direction, as long as they do not bump/crash through other cars or the walls.

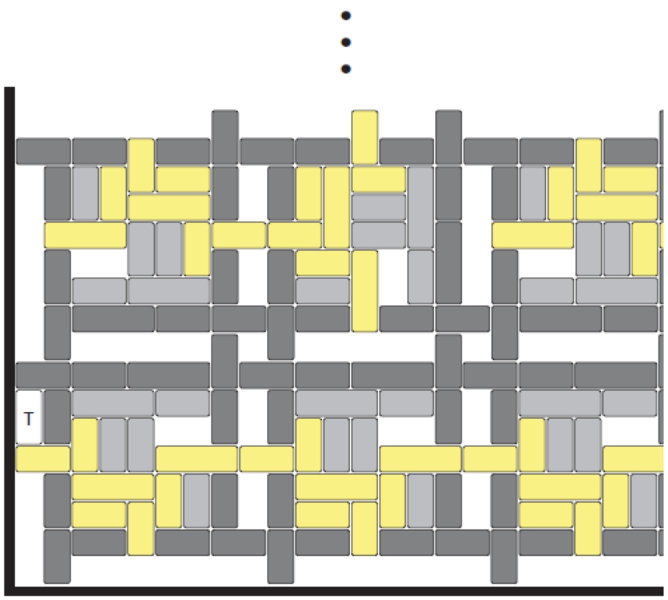
Target: get the red car out of the garage through the exit.

Theorem Rush Hour is PSPACE-complete.
(Remember Savitch: PSPACE = NPSPACE.)
$\uparrow$
non-deterministic Turing machine with polynomial space

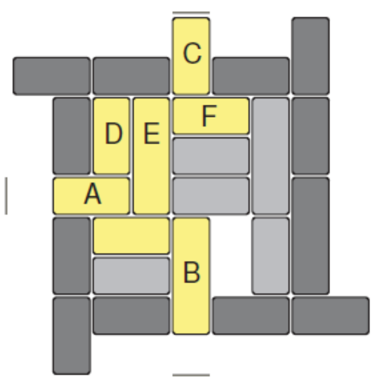
The proof proceeds by reduction from Nondeterministic Constraint Logic (NCL): NCL is PSPACE-complete for planar graphs using only ANDs and protected-ORs.



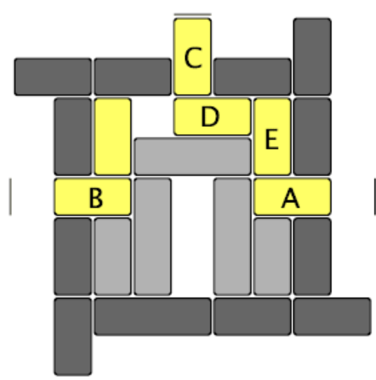
The decision problem is: Given a constraint graph G (including arrows) and a distinguished edge e in G; is there a sequence of edge reversals that eventually reverses e? Moves may be repeated: it is an *unbounded game*.



(a) Layout



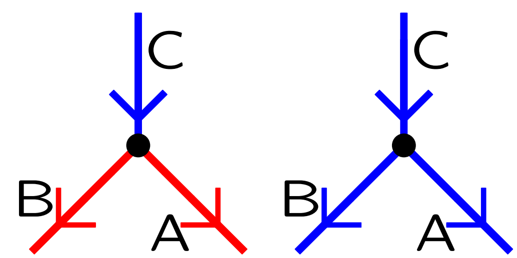
(b) AND



(c) Protected OR

target “car” T
must go down

“car” is in
$\Leftrightarrow$
edge points out

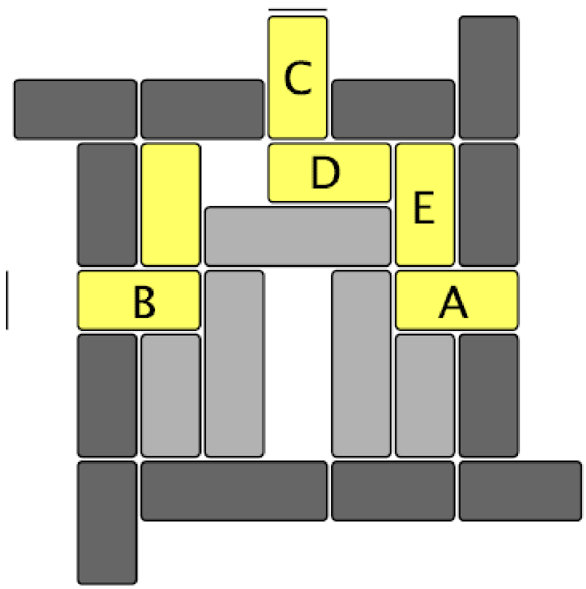


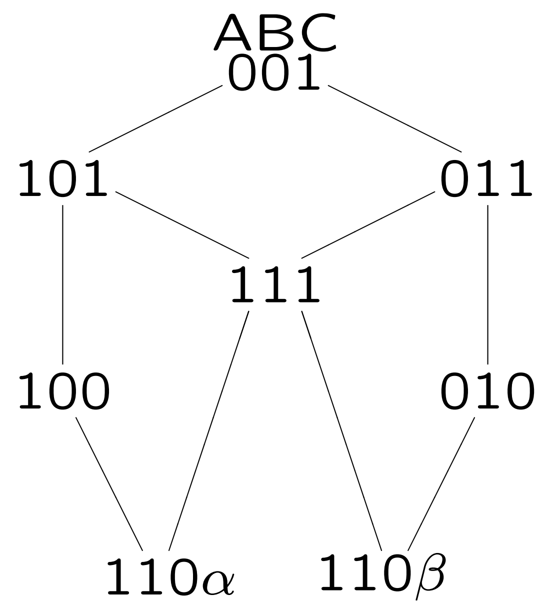
Exercise: $\boxed{\text{Fill in the proof details.}}$

This includes

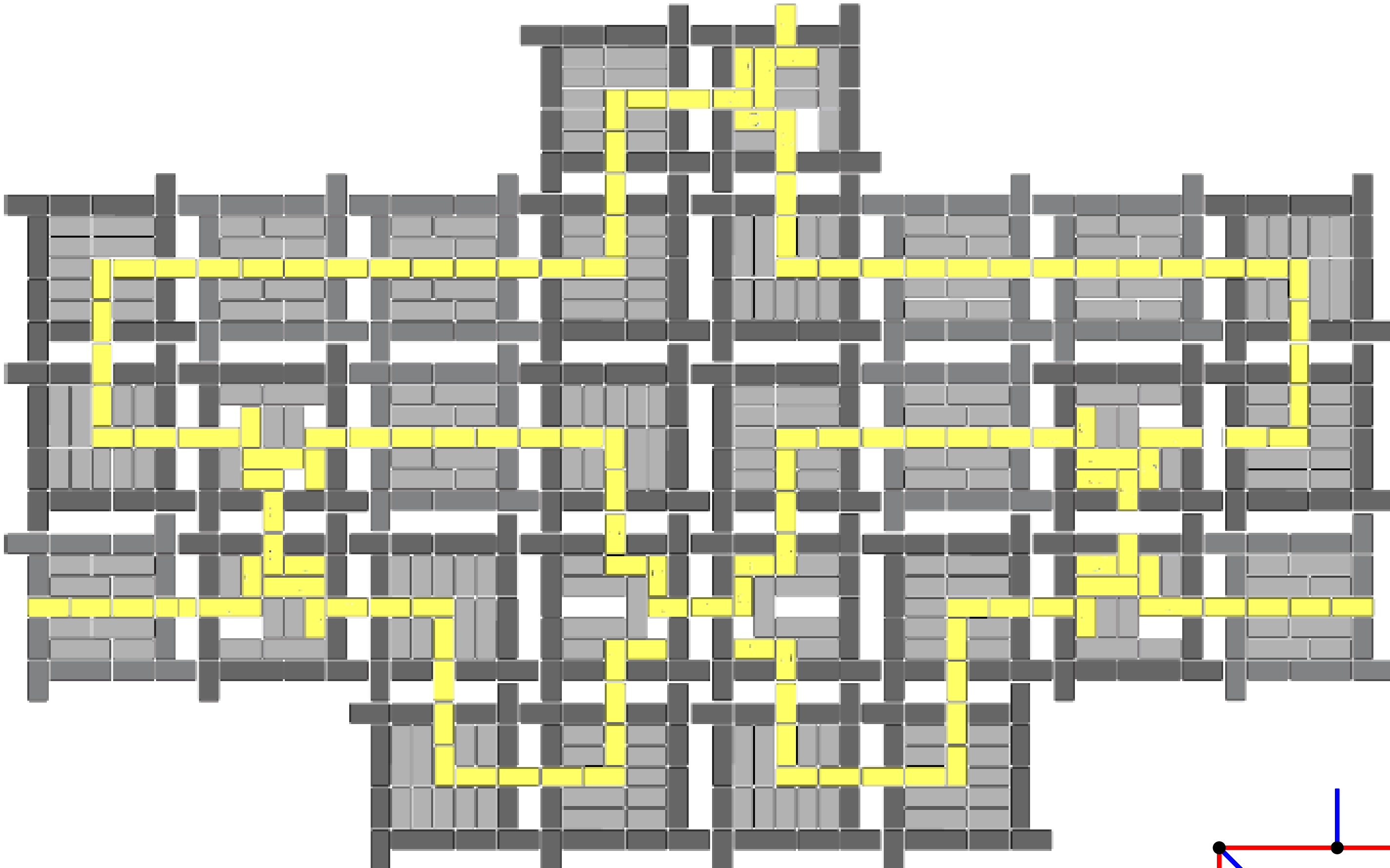
- proper inner working of the gadgets,
- proper communication between gadgets,
- proper glueing together (in polynomial space),
- check that walls do not move (or hardly),
- ...

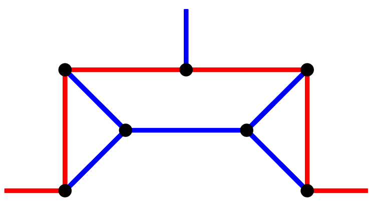
The statespace for the Rush-Hour protected-OR gadget is somewhat strange (where again 1: car out; 0: car in):



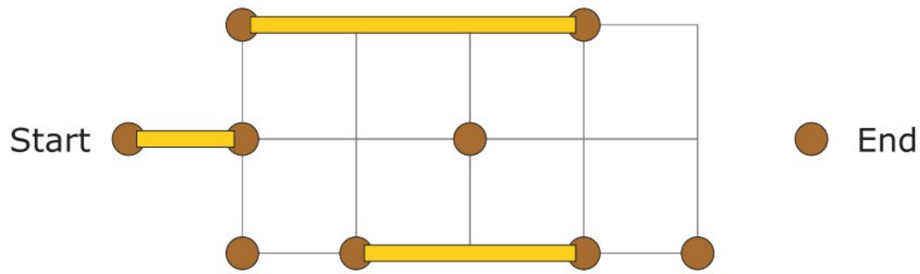


Rush-OR



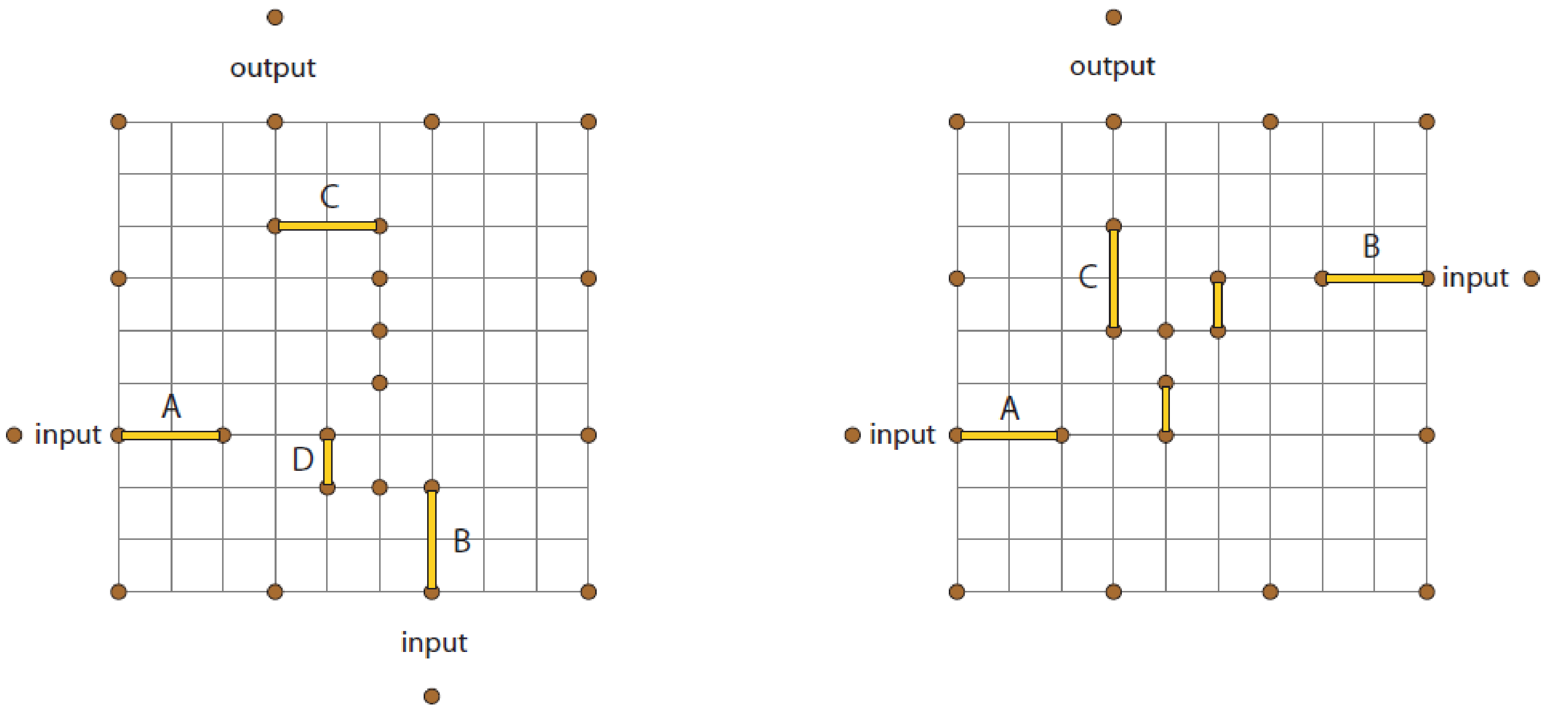


And how about Plank puzzle = River Crossing^{TM} (link)?



You must travel from Start to End; you can carry and move one plank at a time (if you "have" it), and traverse them in the obvious way.

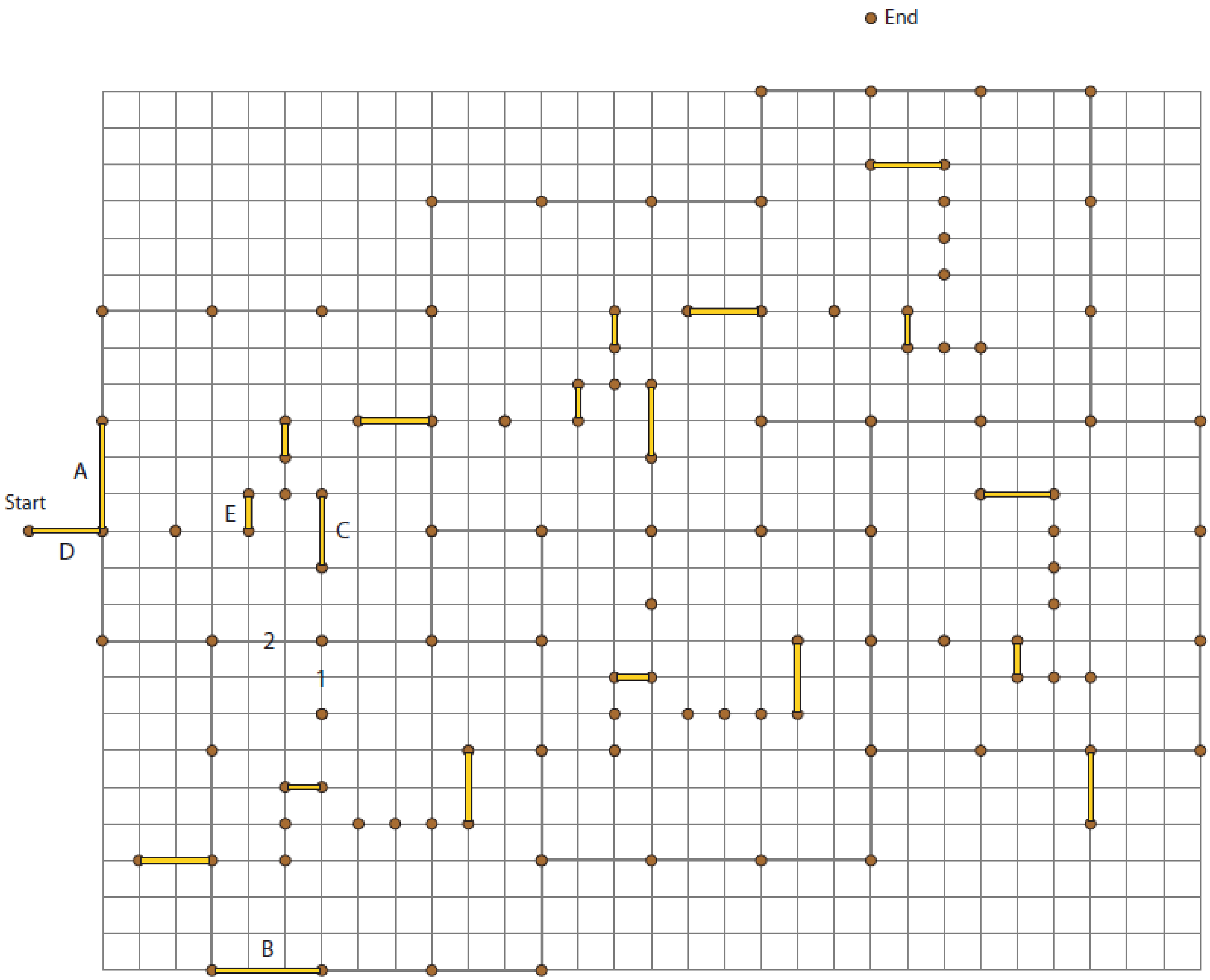
The Plank puzzle is also PSPACE-complete:

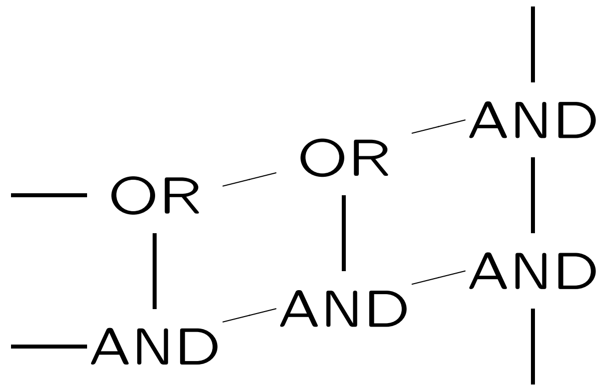


(a) AND (b) OR

In these gadgets, for the correct behavior it is important that plank A and/or B are inside. You can freely walk around the squares with a length 3 plank.

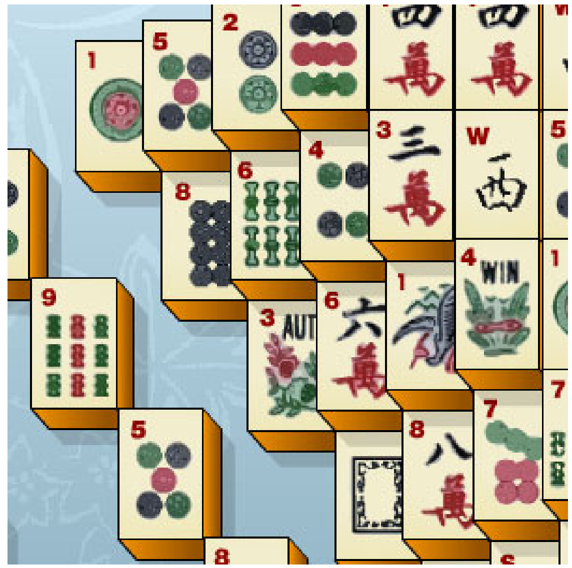
Plank puzzle — 3





Game rules: two visible stones may be removed if they are the same *and* they are “free” to one or two sides.

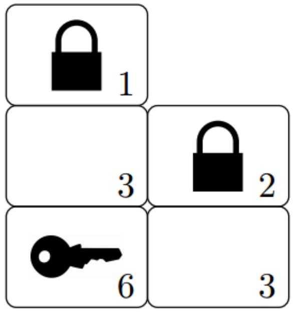
Exercise: $\boxed{\text{Provide AND- and OR-gadgets for Mahjongg.}}$



Hint: keep it simple; find a small set of stones, such that a special one can be "opened" exactly if one (for OR, or both for AND) of two others can be removed.

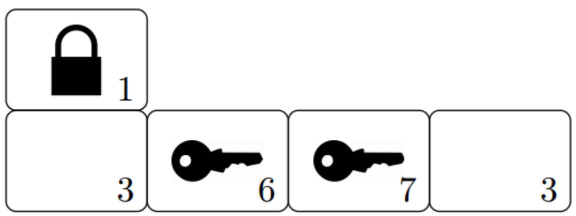
Exercise: $\boxed{\text{And a CHOICE-gadget?}}$

Mahjongg — gadgets continued

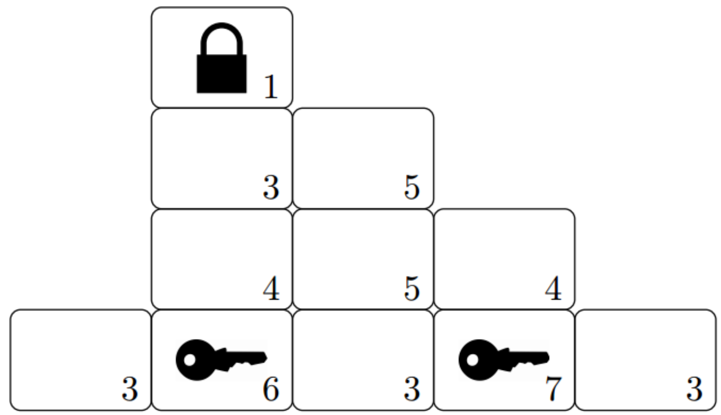


(a) AND gadget

(b) OR gadget

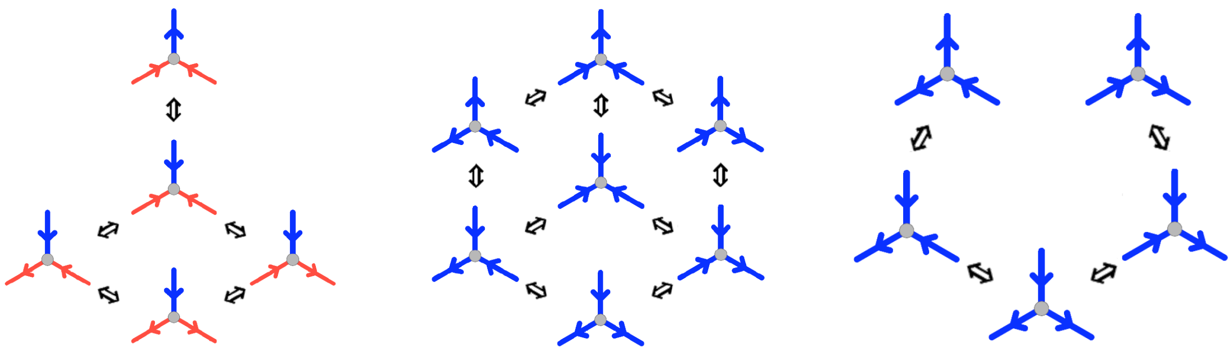


(c) FANOUT gadget



(d) CHOICE gadget

The statespaces for AND, OR and protected-OR:



Reductions between problems concerning games are based on simple gadgets, technique and peculiarities. Many games can be proven to be NP-hard, PSPACE-hard, etc., using the Constraint Logic machinery.

Thanks: Erik Demaine & Bob Hearn (book: Games, Puzzles & Computation, AK Peters, 2009) and Jan van Rijn.

www.liacs.leidenuniv.nl/~kosterswa/19gadgets.pdf