SOME NOVEL SUDOKU PATTERNS

First version: 1 (10-2-2025)
Author: KvD

- 12-2-2025
  Added the main requirement of Flying Fry as it was missing (oops!) (thanks to Pelagic_Amber who spotted that a proof didn't make any sense without it).
- 13-2-2025
  The first chapter now lists some common terminology & notation as used in this document. Added an introduction to Newton's Cradle (NC) discussing SK-Loop (with plenty of proofs that can easily be lifted to NC). Also, some chapters have been moved up. The NC-chapter has been extended with examples.
- 16-2-2025
  Fixed various typo's (including the order of the examples of NC which didn't match the application numbering). Also added a comparison of NC with ALS-AIC and an MSLS-like approach to highlight the differences (see the example for A1).
- 24-2-2025
  Fixed some careless typo's in the STS-approach to NC's A1 (in particular, r79c4 has been corrected to r79c4 and lazily omitted slack variables have been made explicit).

Contents

About this document

This document summarizes some seemingly novel patterns that the author came up with after reading about dozens of known patterns. Only those discoveries that turned out to be potentially useful are included here (based on comparing their performance to other patterns of a similar difficulty, using 250,000 randomly generated puzzles). Often, the new patterns turned out to be more powerful than the patterns that served as their inspiration, so it seems a waste to not share the discoveries.

Due to the number of patterns that remained after vetting them, and without knowing if anyone would actually be interested in them, each pattern is only briefly described, typically by :

1. a definition of the structure,
2. a list of its main applications, and
3. an example per application.

If some patterns attract interest, more in-depth information, explanations, examples and rigorous (yet almost always short & simple) proofs can be provided.

Target audience

While several patterns of this document are quite simple, the majority is too advanced to be of (practical) interest to the average Sudoku player. If you have never heard of e.g. X-Wing, or find it hard to grasp its ideas, this document is not for you; most patterns here are considerable harder and all are described summarily.

Instead, this document targets experienced players who are familiar with many (advanced) patterns and concepts such as bilocated digits, bivalued cells, multi-cells, Hidden/Naked Almost Locked Sets, etc.

If you're the owner of some website/app that explains/implements Sudoku (patterns) and want to incorporate some of the patterns described here to make them more accessible; go ahead - no permission or attribution is required.

Common terminology & notation

ALS An Almost Locked Set, which can be naked (Naked ALS or just NALS) or hidden (Hidden ALS or just HALS).

Naked ALS A set of cells that jointly have more candidate digits than cells.

Hidden ALS A set of digits (of a house) that jointly have more candidate cells than digits (but e.g. Siamese twins takes a relaxed approach as to what a house is).

A^KLS Just an ALS, but specifically one where the difference between the number of cells and digits is K. An A^0LS is simply a plain Locked Set (e.g. a Single/Pair/Triple/Quad/… is an A^0LS).

P{D} A set of cells (P) that jointly have D as distinct candidate digits. For example, r12c3{45} refers to the Naked Pair r12c3 with candidate digits 45. r12c3{45678} refers to a NA^3LS (2 cells with 3 more candidate digits).

D{P} A set of digits (D) (of some house) that jointly have P as distinct candidate digits. For example, 45{r12c3} refers to the Hidden Pair 45 (of column 3 and/or box 1) with candidate cells r12c3.

SK-Loop

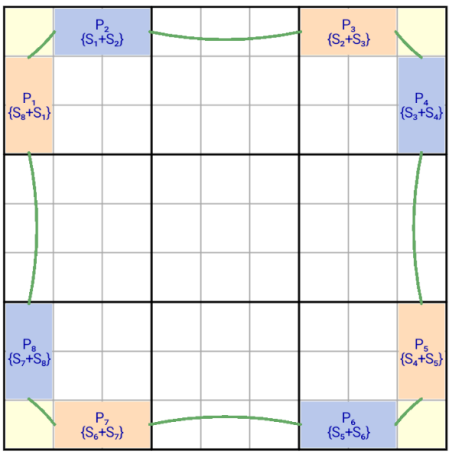
Foreword: SK-Loop is by no means a novel pattern. However, when the author stumbled upon it and its many dubious requirements and hand-wavy proofs (if any), there was no alternative but to prove it from scratch in order to understand it. In that process, novel patterns were discovered (although the level of novelty can be up to heated debates that seem to serve no real purpose and are of no interest to the author). Since someone expressed an interest in Newton's Cradle (which has since been promoted to the next main chapter), an overview of SK-Loop is given as the author has internalized it. As such it's an introduction to Newton's Cradle, Domino Chain and (an optional aspect of) Nested Cycle.

An SK-Loop is based on the intersection cells of 2 rows and 2 columns (yielding 4 "pivots"), subject to:

- **No 2 pivots may belong to the same box.**
- **The intersection of an involved line and box without its pivot are jointly called a node; the i^{th} node is denoted P_i (i = 1,…,8) and the nodes are ordered such that P_i sees P_{i+1} [1].**
- **P_i has candidate digits $S_{i-1}+S_i$, where S_{i-1} and S_i are disjunct sets of digits (+ denotes their union).**

"Officially", SK-Loop has the following additional requirements (but they are irrelevant to the applications[2] and the author sees no reason to strictly adhere to them):

- The pivots must contain a clue.
- The nodes may not contain a clue.
- The nodes must contain at least 1 unsolved cell.
- No S_i may be empty or contain more than 3 digits (in some definitions, S_i must even contain exactly 2 digits).



$P_1\{S_8+S_1\}$ - $P_2\{S_1+S_2\}$ - … - $P_8\{S_7+S_8\}$
(pivots shown in yellow)

[1] This is possible because the nodes can always form a loop, meaning that the indices wrap around at the ends, so P_8 sees P_1 (not P_9, which doesn't exist) and P_1 has candidates S_8+S_1 (not S_0+S_1).
[2] Actually, several the requirements in bold (or parts thereof) are also irrelevant; see Newton's Cradle for what's really important. Spoiler: it's the last bullet item in bold that is crucial.

Textually, this document notates[3] an SK-Loop by listing the P_i (in order), surrounded by S_{i-1} and S_i between brackets:

$\cdot S_8 \cdot P_1 \cdot S_1 \cdot P_2 \cdot S_2 \cdot \ldots \cdot S_7 \cdot P_8 \cdot S_8 \cdot$

Applications

In the following applications, #S is the total size of all S_i (i.e., their summed sizes) and #P is the total size of all P_i (i.e., the total number of cells involved, since there's no overlap):

A1) If #S = #P, every digit X of S_i can be eliminated from every cell that sees every instance of X in P_i and P_{i+1}.

Proof: every digit of S_i can solve at most 1 cell of P_i and P_{i+1}, so all digits of all S_i combined can solve at most #S cells. The application requires #S = #P, so all S_i combined can solve at most #P cells. If a digit of S_i wouldn't solve a cell of P_i or P_{i+1}, the digits of S could solve at most #P-1 cells, hence at least 1 cell (of some node) would have to be solved by a digit that doesn't occur in S. However, that's impossible as every cell only has candidates that occur in S.

For any pivot p at the center of P_k and P_{k+1}:

A2) If #S = #P+1 and candidate digit X of p occurs in S_{k-1} and S_{k+1}, then X can be eliminated from p.

Proof: pivot p sees P_{k-1}, P_k, P_{k+1}, and P_{k+2}, so p=X would eliminate X from P_{k-1}, P_k, P_{k+1}, and P_{k+2} and hence from S_{k-1}, S_k and S_{k+1} (of which 2 sets contain X). An SK-Loop (S',P) would remain in which #S' = #S-2 = #P-1 < #P, which is impossible (since there wouldn't be enough digits to solve every cell).

A3) If #S = #P+1 and the candidate digits of p are a subset of S_k, then every digit of every S_i can be eliminated from every cell r that sees P_i and P_{i+1}. For i=k, r must also see p.

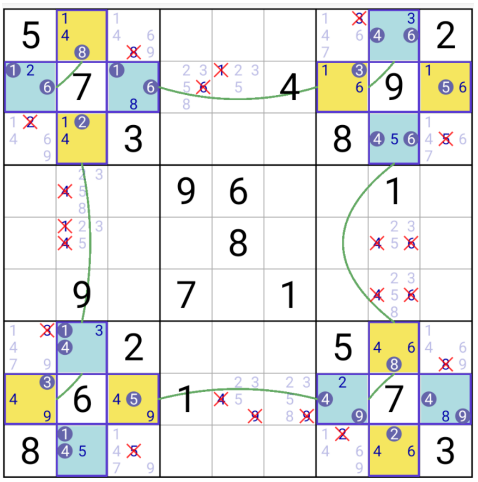
Proof: p=X for some X. This allows the removal of X from S_k, which reduces #S by 1, resulting in an SK-Loop with #S = #P. A1 then proves that every digit of S'_i can be eliminated from every cell r that sees P_i and P_{i+1}. X can be eliminated from every cell that sees p (and hence also P_k and P_{k+1}).

Alt.: the proof is even easier using A4 of Newton's Cradle: adding p to P_k doesn't affect S_k (hence #S stays the same), but it does increase #P by 1, resulting in a Newton's Cradle with #S = #P (and its A4 offers the same applications as A3 here).

[3] Perhaps some people are outraged by that notation as (apparently) convention dictates to use eureka-notation. The author isn't a fan of that notation because of its re-use from the AIC domain. At the risk of upsetting even more people: AIC (the way it is promoted) is a blunder for several reasons, and, moreover, SK-Loops simply aren't AICs (which is not the same as saying they can't be represented as such with sufficient effort).

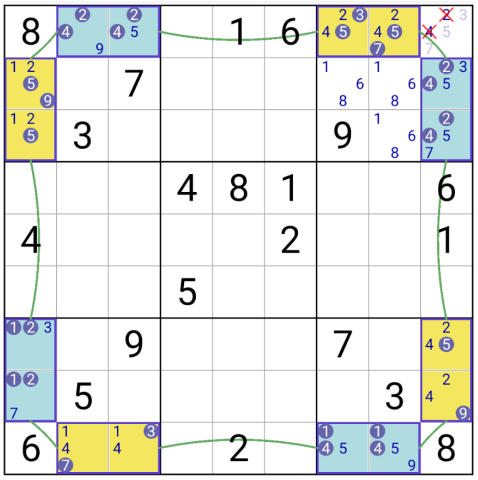
Examples

A1



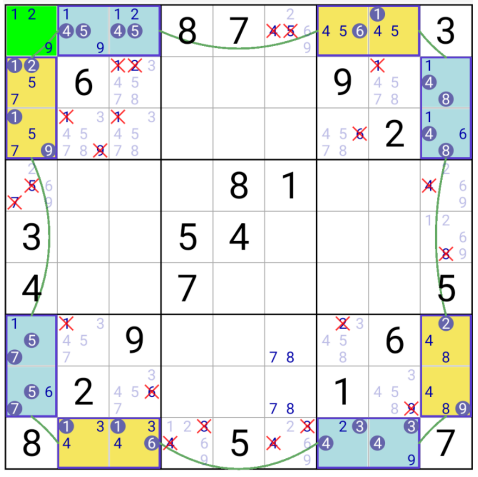
500000002070004090003000800000960010000080000907010000020005000601000708000000003

A2



800016000007000000030000900000481006400002001000500000009000700050000030600020008

A3



000870003060000900000000020000081000300540000400700005009000060020000100800050007

Good vs. bad digits

The applications barely cover #S > #P (pivots are usually already solved), and even if #S = #P no eliminations may be on offer. This and the following few sections aim to repair those shortcomings somewhat.

Letting X_{iu} stand for the digits of S_i (i.e., $S_i = \{X_{i1}, X_{i2}, \ldots\}$):

- **A digit X_{iu} is called "good" if it (eventually) solves (a cell of) P_i or P_{i+1}.**
- **Otherwise, it's called "bad".**

This offers some more applications:

A4) X_{iu} can be eliminated from every cell that sees P_i and P_{i+1}, provided that X_{iu} is a good digit.

Proof: Since X_{iu} is good, X solves P_i or P_{i+1}. Hence X doesn't solve any cell that sees P_i and P_{i+1}.

A5) If p=X establishes an SK-Loop with #S < #P + K, where K is the number digits known to be bad, then p≠X.

Proof: Every known bad digit X_{iu} can be eliminated from P_i and P_{i+1}, and hence S_i, resulting in a total decrease of #S by K. The resulting SK-Loop (S',P) satisfies #S' = #S-K < #P, which is impossible.

Finding good digits one at a time

To prove that a digit X_{iu} is good, assume that it's bad and show this leads to a contradiction.

That's hard in general, but if #S = #P+1 it's easy based on the fact that the assumption that X_{iu} is bad implies that all other digits are good.

Proof: If 2 bad digits X_{iu} and X_{jv} would exist, they could be eliminated from resp. P_iP_{i+1} and P_jP_{j+1}, and hence from S_i and S_j. This reduces #S by 2, resulting in an SK-Loop for which #S = #P-1. That's impossible.

That means that if X_{iu} is assumed to be bad, all other digits can safely be assumed to be good, which offers a comfortable number of conjuncts (16) to find a contradiction (which is likely successful or at least possible since there are only 1 bad digit if #S = #P+1 – hence the assumption is probably false).

Finding all bad digits (almost) at once

A useful observation is that if #S = #P+N, then exactly N bad digits exist.

Proof: The previous section showed that if #S = #P+1, exactly 1 bad digit X_{iu} exists. Ignoring X_{iu} as a candidate of P_i and P_{i+1} (and hence of S_i) reduces the SK-Loop to satisfy #S = #P. Exactly the same reasoning shows that if #S = #P+N, an X_{iu} exists that reduces the SK-Loop to satisfy #S = #P+N-1. By induction it follows that exactly N digits are bad.

An obvious but quite naive way to prove that some X_{iu} is a bad digit is to assume it isn't, and then show that assumption leads to a contradiction. For example, if p is some cell that sees P_i and P_{i+1}, then successfully building a chain $p \neq X_{iu} \rightarrow \ldots \rightarrow$ [contradiction] would prove that X_{iu} is a bad digit. However,

that chain also proves p=X_{iu}, which suggests that building it is probably hard (because otherwise p=X_{iu} could probably be proved without taking advantage of the SK-Loop).

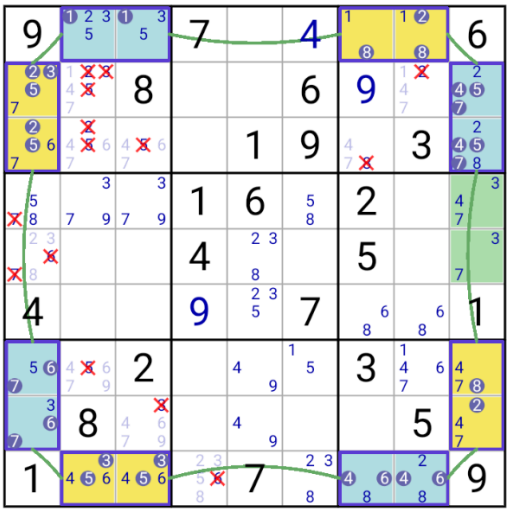
To make it easy to prove a contradiction, more digits (and cells) need to be involved. The author's favorite tactic is:

1) Assume that *all* digits are good.
2) Next, look for some obvious contradiction (e.g. an $A^{K}LS$ from which K+1 candidate digits are eliminated, or a Unique Rectangle that ends up with just 2 candidate digits, etc.).
3) Finally, consider which parts of the assumption (step 1) were actually used (step 2); the contradiction can usually be attributed to some digits of just a few segments of the SK-Loop.

This almost never identifies bad digits with pinpoint accuracy (just "this or these segments contain at least 1 bad digit"). However, that doesn't matter: once enough (N) bad digits have been found, all others are known to be good (and A4 can be used).

Example (1 segment + 1 ALS)

#S = #P+1 in ·67· r23c1 ·235· r1c23 ·1· r1c78 ·28· r23c9 ·457· r78c9 ·28· r9c78 ·46· r9c23 ·35· r78c1 ·67·:

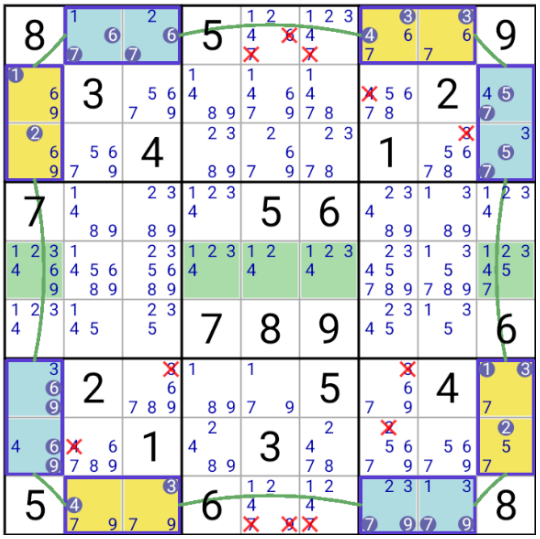


900704006008006900000019030000160200000400500400907001002000300080000050100070009

The segment r23c9 ·457· r78c9 must contain a bad digit, otherwise the ALS r45c9{347} would have to be solved without candidate digits 47, which is impossible. It isn't known exactly which digit (4, 5 or 7) of that segment is bad, but it's enough to know that one is, and hence that all other segments contain good digits and can therefore be used for the usual eliminations.

Example (2 segments + 1 ALS)

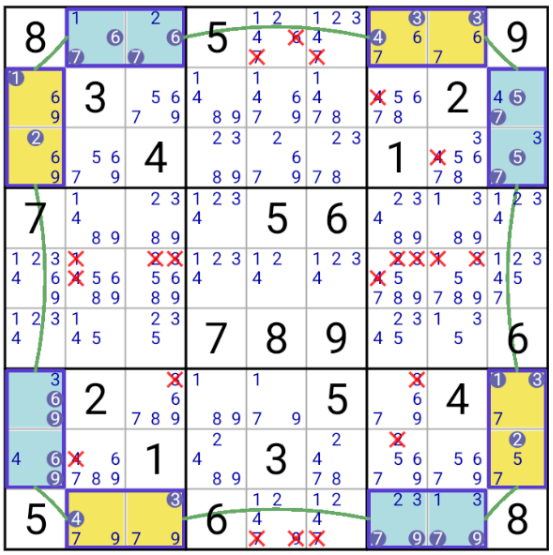
#S = #P+1 in ·12· r1c23 ·67· r1c78 ·43· r23c9 ·57· r78c9 ·123· r9c78 ·79· r9c23 ·34· r89c1 ·69· r23c1 ·12·:



900704006008006900000019030000160200000400500400907001002000300080000050100070009

Since #S = #P+1, 1 digit is bad. It must be one of “r89c1 ·69· r23c1” or “r23c9 ·57· r78c9”, otherwise r5c13569 (5 cells) would be left with candidates 1234 (4 digits), which is impossible. The remaining (good) digits offer 12 eliminations (r1c5≠67, r1c6≠7, r2c7≠4, r3c8≠4, r7c7≠3, r8c7≠2, r9c5≠79, r9c6≠7, r8c2≠4, r7c3≠3), i.e. all the usual eliminations except r1c4≠69 and r4c9≠57 which might be invalid.

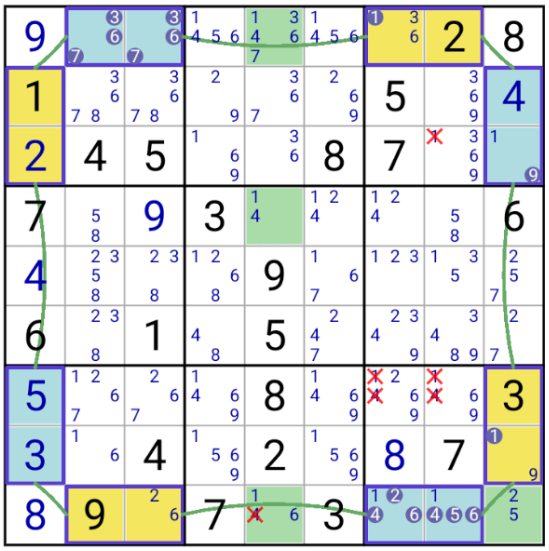
Also, since only 1 bad digit exists, 1 of the 2 suspicious segments contains only good digits. That means that either r5c1456 or r5c456c9 forms a Locked Set for 1234, which offers 9 additional eliminations in row 5 (r4c2≠14, r4c3≠23, r4c7≠234, r4c8≠13). The end-result consists of 20 eliminations:



900704006008006900000019030000160200000400500400907001002000300080000050100070009

Example (2 segments + 2 ALSes)

#S = #P+2 in ·35· r23c1 ·12· r1c23 ·367· r1c78 ·12· r23c9 ·49· r78c9 ·13· r9c78 ·2456· r9c23 ·9· r78c1 ·35·:



900000028100000504245008700709300006400090000601050000500080003304020870890703000

#S = #P+2, but this can be reduced to #S = #P because

- r9c78 ·25…· r9c23 has 1 bad digit (otherwise r9c9 would be stripped of its candidate digits).
- r1c23 ·367· r1c78 or r9c78 ·6…· r9c23 has 1 bad digit (because if they were all good, 367 could be eliminated from the NA^{2}LS r149c5{13467}, leaving just 2 digits for 3 cells in column 5).

Note: the above SK-Loop violates many traditional requirements of SK-Loop, but since they don't matter that's fine.

Finding bad digits using regularity

Here, a **regular** SK-Loop means that #S_i=#P_i=2 for all i. Such an SK-Loop has the following properties:

i) If P_i=S_i for *some* i, then P_i=S_i for *all* i.

ii) If P_i=S_{i-1} for *some* i, then P_i=S_{i-1} for *all* i.

Proof of claim i: since #P_i=#S_i and S_i and S_{i+1} don't share digits, the following inference is valid for all i:

$$P_i\{S_{i-1}+S_i\}=S_i \longrightarrow P_{i+1}\{S_i+S_{i+1}\}=S_{i+1}$$

Applying this inference 8 times in succession results in the following chain (using roll-over after index 8 back to 1):

$$P_i=S_i \longrightarrow P_{i+1}=S_{i+1} \longrightarrow \ldots \longrightarrow P_{i+8}=S_{i+8}.$$

The property follows from the observation that this is a cycle, or more readily from the fact that if the initial proposition of a chain is true, then every proposition of the chain is true.

Claim ii can be proved nearly identical, but based on the inference $P_{i+1}\{S_i+S_{i+1}\}=S_i \longrightarrow P_i\{S_{i-1}+S_i\}=S_{i-1}$. Alternatively, reversing the SK-Loop ·S_8· P_1 ·S_1· P_2 ·S_2· … ·S_7· P_8 ·S_8· to obtain ·S_8· P_8 ·S_7· … ·S_2· P_2 ·S_1· P_1 ·S_8· reduces claim ii to claim i.

As a result, every regular SK-Loop belongs to exactly 1 of the following categories:

C1) Every P_i is solved by at least 1 digit of S_{i-1} and at least 1 digit of S_i.

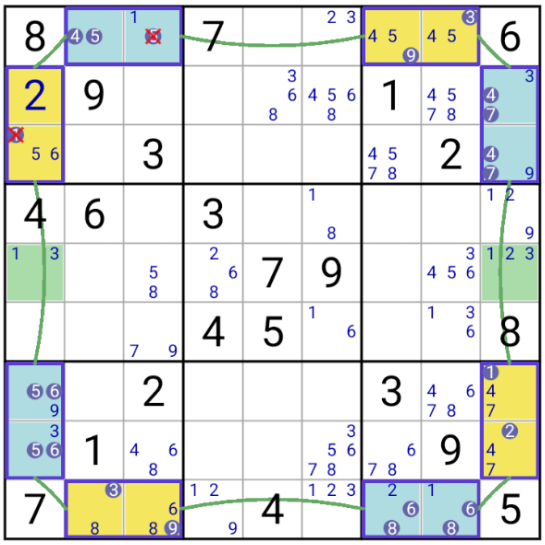
C2) Every P_i is solved by the digits of S_i.

C3) Every P_i is solved by the digits of S_{i-1}.

It's statistically most likely that a regular SK-Loop belongs to category C1. Therefore, assuming it belongs to e.g. category C2 (or C3) is likely wrong. To actually prove that it doesn't belong to C2 (or C3), it suffices to identify just 1 bad digit (as that would allow the existence of an impossible SK-Loop with #S = #P-1).

Example (regularity + 1 ALS)

#S=#P in ·56· r23c1 ·12· r1c23 ·45· r1c78 ·39· r23c9 ·47· r78c9 ·12· r9c78 ·68· r9c23 ·39· r78c1 ·56·, but the usual SK-Loop applications have already been exhausted:

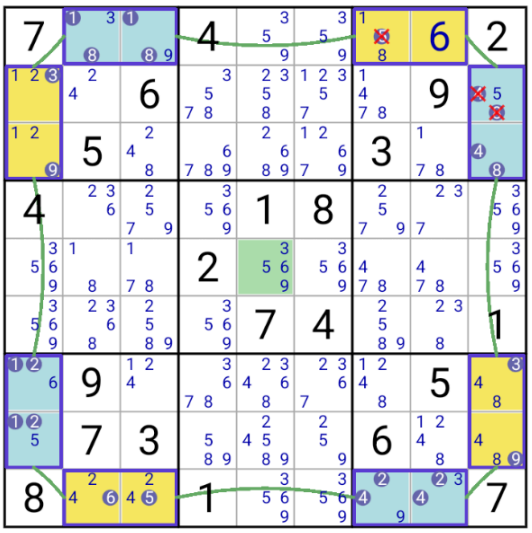


800700006290000100003000020460300000000079000000450008002000300010000090700040005

If this would be a category 2 SK-Loop, then r23c1 and r78c8 would both be solved by 12, but that's impossible as it would leave r5c19{123} with just 1 candidate digit. Hence it's not an category 2 SK-Loop, so, in particular, r1c23 and r23c1 can't be solved by resp. 45 and 12.

Example (regularity + 3 ALSes)

#S=#P in ·12· r23c1 ·39· r1c23 ·18· r1c78 ·56· r23c9 ·48· r78c9 ·39· r9c78 ·24· r9c23 ·56· r78c1 ·12·, but the basic SK-Loop applications offer no eliminations:



700400062006000090050000300400018000000200000000074001090000050073000600800100007

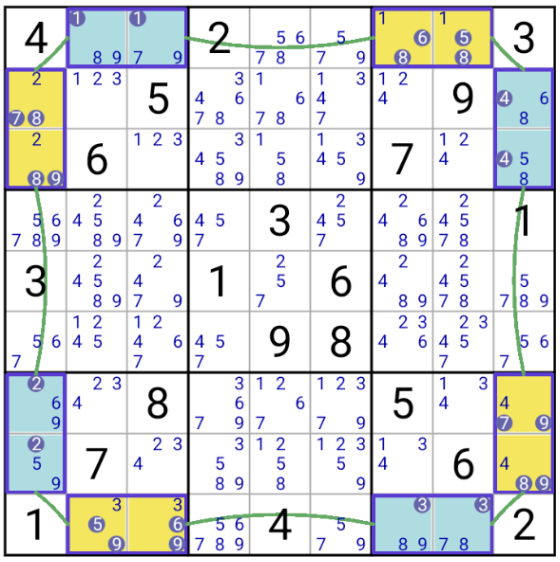
If this would be a category 2 SK-Loop, the ALS r1c59{3569} (a NA^2LS to be precise) would be reduced to the Naked Pair r1c59{39} and r5c19{3569} to the Naked Pair r5c19{56}. As a result, r5c5 (a NA^3LS) would be unsolvable.

More regularity

The kind of regularity described above occurs quite frequently, but there are other kinds. In general, if something looks regular but no eliminations are obvious, just invoke a bit of creativity to exploit the regularity and find some eliminations anyway.

Example (4 ALSes + 7 chains)

#S=#P in ·789· r1c23 ·1· r1c78 ·568· r23c9 ·2· r78c9 ·789· r9c78 ·3· r9c23 ·569· r89c1 ·4· r23c1 ·789·, but the usual SK-Loop applications have already been exhausted:



400200003005000090060000700000030001300106000000098000008000500070000060100040002

What's striking is that each box has only 2 unsolved cells that don't belong to the SK-Loop, that those 2 cells form an ALS, and moreover that in adjacent boxes those ALSes share a digit. It feels like such regularity should be exploitable.

There are some intriguing Newton's Cradles (e.g. the very regular ·24· r2c2 ·13· r2c7 ·24· r8c7 ·13· r8c3 ·24· r3c3 ·13· r3c8 ·24· r7c8 ·13· r7c2 ·24·) but the author failed to find an easy way to make use of them. A hard way is to use those Newton's Cradles as inspiration to construct an Alien Fish using base sets 13r2, 24c7, etc. with the 4 corner boxes as cover sets. However, the following chaining argument is more down to earth:

It starts by noticing the following chain (after focusing on the nodes of the SK-Loop):

C1) r1c23=1 ⟶ r1c78≠1 ⟶ r23c9=4 ⟶ r78c9≠4 ⟶ r9c78=3 ⟶ r9c23≠3 ⟶ r78c1=2 ⟶ r23c1≠2 ⟶ r1c23=1

Proof: the odd inferences are trivial and every even inference is essentially identical: it's based on the fact that 4 cells within a box must be solved by 4 distinct digits. For example, if r1c78≠1, then r1c78 is limited to the 3 candidates 568. Since r23c9 has candidates 4568, (a cell of) r23c9 must be solved by digit 4, otherwise r1c78 and r23c9 (4 cells) combined would be limited to 568 (3 digits).

The regularity invites incorporating the ALSes in boxes 1379. Referring to the pairs of unsolved cells outside the SK-Loop in boxes 1, 3 and 9 and 7 as B_1, B_3, B_9 and B_7 (the 4 ALSes), and taking the previous chain into account, it's almost unavoidable to notice that:

C2a) r1c23=1 ⟶ B_1=23

C2b) r23c9=4 ⟶ B_3=12

C2c) r9c78=3 ⟶ B_9=14

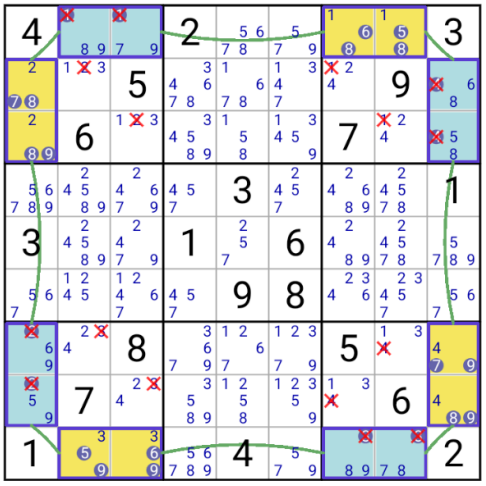
C2d) r78c1=2 ⟶ B_7=34

Since C1 is a cycle, its propositions are either all true, or all false. If they're all true, then (using C2) B_1=23, B_3=12, B_9=14 and B_7=34. The following Contradiction Chains based on those locked sets proves that r1c23=1 (which implies B_1=23, hence r2c2=2 or r3c3=2) is an incorrect assumption:

C3a) r2c2=2 ⟶ r2c7=1 ⟶ r8c7=4 ⟶ r8c3=3 ⟶ r2c2=3

C3b) r3c3=2 ⟶ r3c8=1 ⟶ r7c8=4 ⟶ r7c2=3 ⟶ r3c3=3

Hence every clause of C1 is false, which offers 16 eliminations:



Note: the above example also illustrates why "chaining" shouldn't be labeled "guessing" (or even "hard"), at least not without knowing the context. In the example, 7 chains were built around 1 SK-Loop and 4 ALSes to enable progress. Without knowing anything else, that seems ridiculously hard. However, here everything could be accomplished mentally since the SK-Loop (and regularity) guided every step, i.e. just about dictated which ALSes and which chains (including initial propositions and inferences) to consider. Having said that, there may be more pattern-like or smarter approaches that manage to find the same eliminations without having to improvise at all.

Newton's Cradle

A Newton's Cradle is a sequence of non-overlapping multi-cells P_1, …, P_{-1} in which each P_i sees its neighbor(s)[4], has candidate digits S_{i-1}+S_i and $S_0 = S_{-1}$. It's notated as:

$\cdot S_0 \cdot P_1 \cdot S_1 \cdot \ldots \cdot S_{i-1} \cdot P_i \cdot S_i \cdot \ldots \cdot S_{-1} \cdot P_{-1} \cdot S_{-1} \cdot$

Applications

A1) If #S+#S_{-1} = #P+1, then every digit X of S_{-1} can be eliminated from every cell that sees all instances of X in P_1 and P_{-1}.

A2) If a digit X occurs in K S_i's and #S+#S_{-1} < #P+K, then X can be eliminated from every cell that sees all instances of X in all P_i's.

A3) If a digit X occurs in K S_i's including S_{-1} and #S+#S_{-1} ≤ #P+K, then X can be eliminated from every cell that sees all instances of X in all P_i's.

Additional applications when P_{-1} sees P_1

A4) If P_{-1} sees P_1 and #S = #P, then every digit X of S_i can be eliminated from every cell that sees all instances of X in P_i and P_{i+1}[5] (1 ≤ i ≤ n).

A5) If P_{-1} sees P_1, #S = #P and n is even, then every digit X that belongs to every other S_i (i.e. for i=1, 3, ..., n-1, or for i=2, 4, ..., n) can be eliminated from every cell that sees all instances of X in P_i and P_{i-1} (1 ≤ i ≤ n).

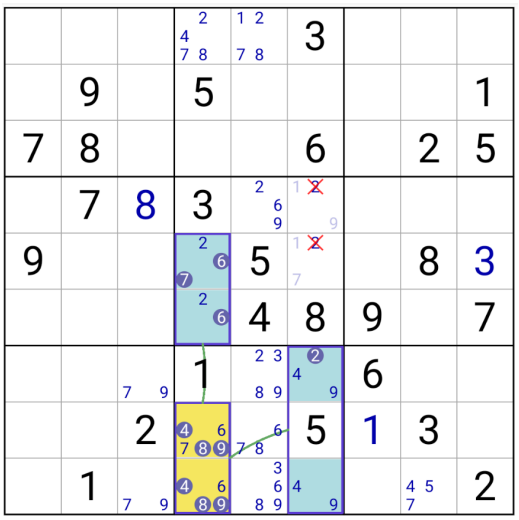
A6) If P_{-1} sees P_1, #S = #P+1 and some cell $q_u\{S_u\}$ sees P_u and P_{u+1}, then every digit X of S_i can be eliminated from every cell r that sees all instances of X in P_i and P_{i+1} (1 ≤ i ≤ n). For i=u, r must also see q_u.

A7) If P_{-1} sees P_1, #S = #P+1 and some cell $q_u\{S_u+Z\}$ sees P_u and P_{u+1} and $q_v\{S_v+Z\}$ sees P_v and P_{v+1} and q_u sees q_v, then every digit X of S_i can be eliminated from every cell r that sees all instances of X in P_i and P_{i+1} (1 ≤ i ≤ n). For i=u (or i=v), r must also see q_u (or q_v).

[4] P_1 and P_{-1} have only 1 neighbor, whereas the other P_i (1 < i < n) have 2 neighbors.
[5] For i=n, P_{i+1} is defined as (a synonym for) P_1.

Examples

A1



000003000090500001780006025078300000900050083000048907000100600002005130010000002

·2· r56c4 ·67· r89c4 ·489· r79c6 ·2·

Some alternative interpretations

The Newton's Cradle of the example above can also be expressed as an inference chain, for example (letting * stand for some variable digit that isn't relevant to know):

r56c4≠2* ⟶ r56c4=67 ⟶ r89c4=48/49/89 ⟶ r79c6=29/24 ⟶ r79c6=2*

It's possible to turn this into an alternating inference chain (making the chain needlessly longer), but it still wouldn't be a (traditional) AIC because of the presence of disjunctive clauses (as in "r79c6=29/24", which is just short for "r79c6=29 ∨ r79c6=24"). One way to get rid of them is to turn the chain into a so-called "net" (a chain that's allowed to bifurcate to later merge the resulting sub-chains into a shared conclusion). Another way is to forcibly write it in the form of a "true" AIC, but that requires cheating by blatantly ignoring that it's impossible for 2 cells to be solved by a different number of digits. Using (slightly modified) Eureka notation, it might look something like this:

(2=67)r56c4 - (67=489)r89c4 – (489=24)r79c6

Since many people expect AICs to be loopy, a final tweak can be to include r45c6 (to turn the AIC into a structure resembling a Discontinuous Alternating Nice Loop):

(179=2) r45c6 - r(2=67)r56c4 - (67=489)r89c4 – (489=24)r79c6 – (24=179)r45c6

By a flexible interpretation of the ordinary AIC rules, it follows that if 2 solves (a cell of) r45c6, then it doesn't (hence 2 can be eliminated). Of course, this isn't a proper AIC; the only reason that the conclusion is nevertheless valid, is because the total size of the "links" equals the total number of cells (+1 for the non-loopy version). Mind you, you have to count the total size of the right hand sides and not double-count the link that closes the loop.

Yet another approach is to make use of single-truth-set logic (a slightly more general version of "General Logic" that deals with relations between propositions directly rather than via the digits, cells and houses they are based on[6]); the following 3 "base" sets each contain exactly[7] 1 true proposition:

- { r79c6=2*, r79c6=49 }
- { r89c4=49, r89c4=67/68/78 }
- { r45c4=67, r45c4=2* }

Most of those individual propositions can be covered by 2 "cover" sets (which are also single-truth sets):

- { r79c6=49, r89c4=49, … } (… represents the slack variable r79c6≠49 ∧ r89c4≠49, see footnotes)
- { r89c4=67/68/78, r45c4=67, … } (idem: … = the conjunction of the negations of the propositions)

Since every proposition is covered once, except r79c6=2* and r45c4=2* which aren't covered[8], (exactly) 3-2=1 of the latter propositions must be true. Again, same eliminations.

The author has yet to find clear definitions of creatures like Multi-Fishes, Alien Fishes, (Almost) MSLSes etc. but wouldn't be surprised if the example could be expressed in some of those contexts as well (but since they use single cells and digits as building blocks it'll probably be more tedious).

Now consider the question what this example "really" is (and likewise all other examples): is it…

- A Newton's Cradle?
- A chain (that uses disjunctive clauses)?
- A nested chain?
- An AIC (that uses invalid inferences, but gets away with that by imposing a constraint on the total link size)?
- An instance of single-truth-set logic?
- …?

The answer: it's all of them, and whichever term (or more importantly: way of conceptualizing) is used is a personal preference. The author finds the case analyses of the chain-approaches somewhat inelegant, considers using AICs with logically unsound inferences an intellectual dishonesty, and doesn't like the set-approach as it's non-constructive and too powerful to apply for something as simple as this; from a practical point of view, it's easier to build a Newton's Cradle, aiming to keep the difference #S-#P small during construction (without having to be aware of things like ALSes) and stopping when an application is hit upon. People who do basically the same thing but call the end-result an (ALS-)AIC are of course free to do so.

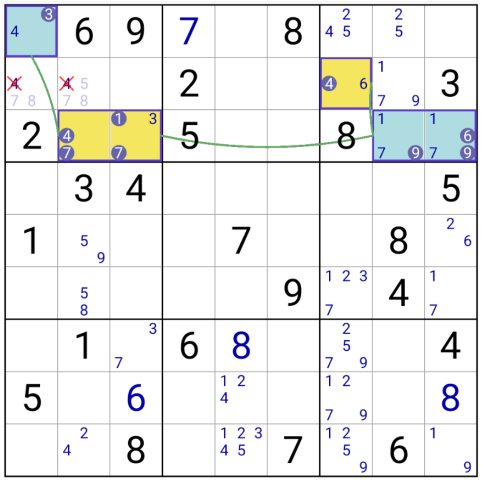
---

[6] The advantage of this is that it gets rid of GL-limitations such as that truths/links can only concern single cells or digits. Also, it enables integrating non-trivial constraints such as those based on chains or uniqueness. Of course, this enters the realm of the abstract where geometry becomes less of a feature, so the extra flexibility is not something most Sudoku players will be comfortable with.

[7] Actually, a weaker condition suffices to hold everything together, namely that base sets should contain *at least* 1 true proposition, and cover sets *at most* one (as in GL). However, looking under the hood it's doesn't really matter, as e.g. slack variables can transform any inequality constraint (like ≥1 or ≤1) to into an equality constraint (=1). Using equalities from the start is marginally simpler (and if no slack variables and suchlike are needed then no information is lost).

[8] Note that General Logic seems to abhor under-coverage (based on its better-than-usual, but still vague definition). Of course, it's possible to add a base set (involving r45c6=2*) and 2 more cover sets (involving digit 2 of the box and column of r45c6) and then appeal to the effect of double coverage (of r45c6=2*), but that seems needlessly complex.

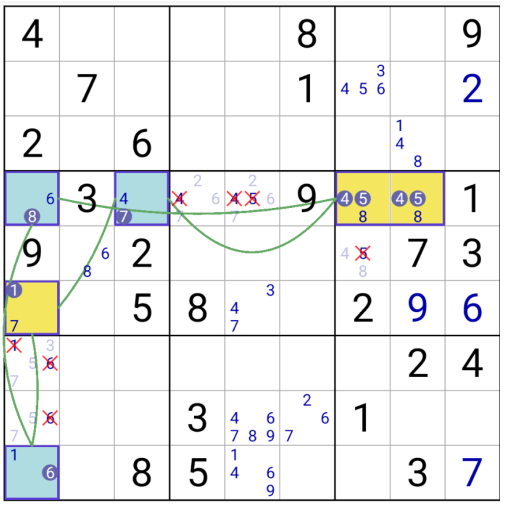
A3



069708000000200003200500800034000005100070080000009040010680004506000008008007060

·4· r1c1 ·3· r3c23 ·147· r3c89 ·69· r2c7 ·4·

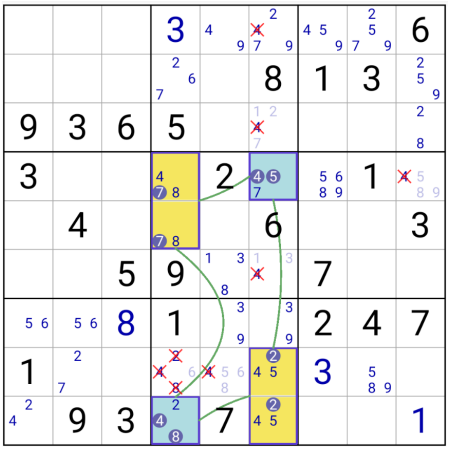
A4



400008009070001002206000000030009001902000073005800296000000024000300100008500037

·6· r4c1 ·8· r4c78 ·45· r4c3 ·7· r6c1 ·1· r9c1 ·6·

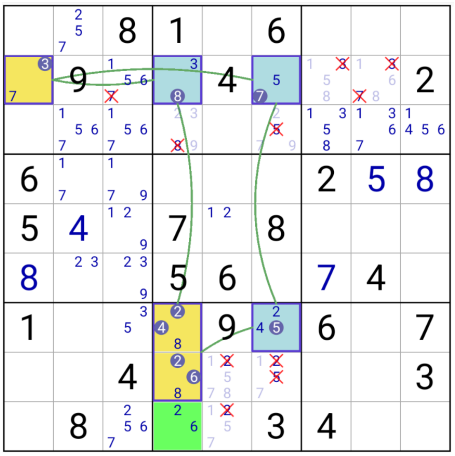
A4+A5



000300006000008130936500000300020010040006003005900700008100247100000300093070001

·2· r9c4 ·48· r45c4 ·7· r4c6 ·45· r89c6 ·2·

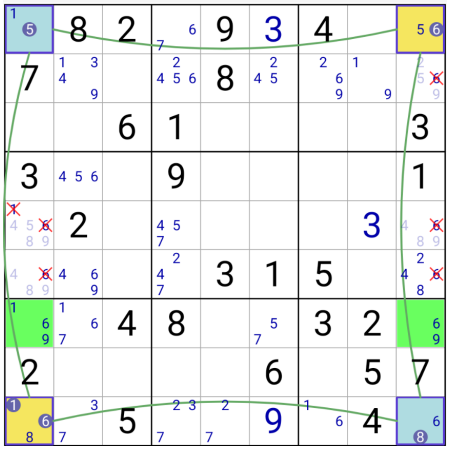
A6



008106000090040002000000000600000258540708000800560740100090607004000003080003400

·5· r2c6 ·7· r2c1 ·3· r2c4 ·8· r78c4 ·246· r7c6 ·5·

A7



082093400700080000006100003300900001020000030000031500004800320200006057005009040

·6· r9c9 ·8· r9c1 ·16· r1c1 ·5· r1c9 ·6·

Varia

- This pattern was inspired by SK-Loop, which is a special case of Newton's Cradle where n=8, P□ sees P_1, $\#P_i=\#S_i=2$ for i=1..n and several other, irrelevant constraints. Some of those constraints may make SK-Loops easier to spot, but the penalty is huge: it's rare for a puzzle to contain an SK-Loop (even when allowing for some minor relaxations). In contrast, Newton's Cradles are common.
- The name of this pattern was inspired by A1, which is based on the fact that if X is does not solve P_1, it pops out at the other end as the solution of P□ and vice versa.
- It's easy to extend the collection of applications, e.g. by involving ALSes to handle #S = #P+K for any K>0, or by considering special cases (such as a) $P_i=S_i$ for all i, b) $P_{i+1}=S_i$ for all i, c) otherwise. Introducing bad vs good digits is yet another approach.

Domino Chain

A Domino Chain is a sequence of non-overlapping[9] multi-cells[10] P_1, …, P_n (the "dominoes") in which each P_i sees its neighbor(s), has candidate digits $S_{i-1}+S_i$ [11] and $S_0 = \emptyset$ [12]. It's notated as:

$\cdot S_0 \cdot P_1 \cdot S_1 \cdot \ldots \cdot S_{i-1} \cdot P_i \cdot S_i \cdot \ldots \cdot S_{n-1} \cdot P_n \cdot S_n \cdot$

In the following, $\#S_i$ and $\#P_i$ denote the size of sets S_i and P_i. #S and #P denote the sums of those sizes over the range $1 \le i \le n$.

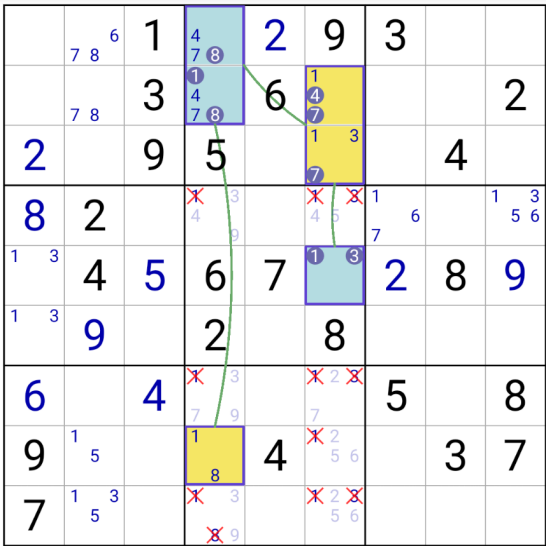
Applications

A1) If #S = #P, then every digit X of S_i can be eliminated from every cell that sees all instances of X in P_i and P_{i+1} ($1 \le i < n$).

A2) If a candidate digit Z of one or more P_i is ignored such that as a result #S < #P, then Z can be eliminated from every cell that sees every ignored instance of Z.

Examples

A1



0010293000030600022095000408200000000456702890902080006040005089000400377000000000

In ·· r5c6 ·13· r23c6 ·47· r12c4 ·18· r8c4 ··, #P=6 and #S=6, hence (A1) every digit of S_i solves a cell of P_i or P_{i+1}.

---

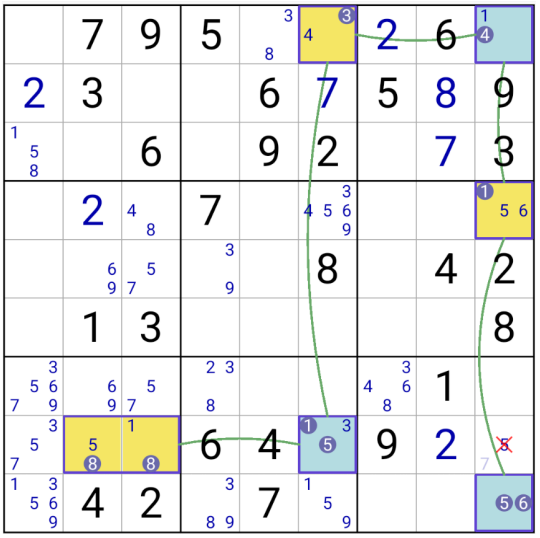
[9] The requirement that the P_i don't overlap is stronger than necessary; all that's really matters is that #P (see later) counts the number of distinct cells.

[10] A multi-cell is a set of cells that are solved by distinct digits, typically because they all see each other (i.e., share a house).

[11] $S_{i-1}+S_i$ denotes the union of 2 distinct sets of digits S_{i-1} and S_i.

[12] Ø denotes the empty set.

A2



079500260230067589006092073020700000000008042013000008000000010000640920042070000

In ·· r9c9 ·56· r4c9 ·1· r1c9 ·4· r1c6 ·3· r8c6 ·15· r8c23 ·8·, #P=7 and #S=8 (1 digit too many for A1). Ignoring candidate digit 5 in r49c9 and r8c236 results in ·· r9c9 ·6· r4c9 ·1· r1c9 ·4· r1c6 ·3· r8c6 ·1· r8c23 ·8·, resulting in #P=7 and #S=6. Hence (A2) 5 must solve a cell of r49c9 or r8c236.

Varia

- This pattern was inspired by Newton's Cradle.

Flying Fry

A Flying Fry consists of:

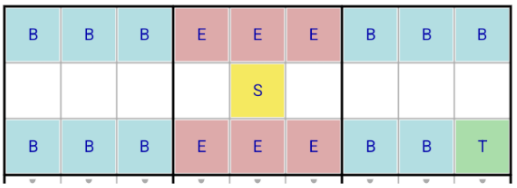
- **2 "base" rows (or columns)[13],**
- **1 “source" cell, and**
- **1 "target" cell that belongs to a base line,**

such that, for every source digit X, the base cells for X see each other.

Terminology

- **Source digits** are the candidate digits of the source cell.
- **Base cells for X** are the cells of the base lines that support X and aren't an escape or target cell.
- **Escape cells for X** are the cells of the base lines that support X and see the source cell.

In the following sketch, the *potential* base and escape cells for X are colored in resp. blue and red (they're potential because they depend on X, i.e. whether or not the cell supports some digit X).



Base lines: rows 1 and 3
Source cell: r2c5
Target cell: r3c9

Note: pictures can be misleading, especially for patterns with multiple components. For example, the sketch above might be interpreted to imply that each (colored) cell is either a B/S/T/E-cell. However, the definitions are leading and they don’t impose such a dichotomy. In particular, a source cell is allowed to belong to a base line (in which case it's also either a base cell for X or the target cell). It's also possible for the (or a) target cell to be an escape cell as well.

Applications

Flying Fry has many applications. Fortunately, all of them follow almost immediately from the following key observation:

> **The solution of the source cell is identical to the solution of the target cell.**

Since that property is crucial to every application, here’s a proof:

> *The source cell is solved by some digit, say X. X can't solve a cell that sees the source cell. Consequently, in each of the 2 base lines X solves a cell that's either the target cell or a base cell*

[13] 2 parallel lines are suggested merely because the pattern is easiest to spot in that context. However, the types of houses are irrelevant for the logic of the applications; any 2 houses (e.g. a row and a box) would suffice. With some minor tweaks to some applications, those houses can even be allowed to overlap. In fact, at a fundamental level, a “house” only needs to represent some collection of propositions of which exactly 1 is true (so e.g. “1 cell + its candidate digits” can act as a “house”).

for X. The target cell can only provide 1 of those cells (because it belongs to just 1 base line[14]). The same is true for the base cells for X (because they see each other).Hence exactly 1 target cell and 1 base cell are solved by X.

With that that fact in mind, most if not all of the applications below will hopefully be obvious. Otherwise, they might seem like rabbits pulled out of a magician's hat -- so let the above sink in before continuing.

A1) "Shared Digits"
All candidate digits that aren't shared by the source and target cell can be eliminated from both cells.

A4) "Finned Fish"
If X is a source digit, then X can be eliminated from every cell that sees the source cell and all escape cells for X.

A5) "Fish"
If X is a source digit and all escape cells X share a house, then X can be eliminated from every cell outside that house that sees all base cells for X.

A6) "Full translocation T ⇉ P"
If every candidate digit of the target translocates from that target to the same cell p, other digits can be eliminated from p.

A7) "Translocation T ↦ P"
If source digit X translocates from the target to a cell p that sees the source cell, then X can be eliminated from the target.

A8) "Translocation P ↦ T"
If source digit X translocates from a cell p that sees the source cell to the target, then X can be eliminated from p.

A10) "Known digit"
If X is a known digit[15], then X can be eliminated from a) every cell that sees the source cell, b) every cell that sees the target cell and c) every cell that sees all base cells for X.

A11) "Unique Rectangle"
If the source and target cell form 2 corners of a Unique Rectangle with p and q as the other 2 corners, and a digit translocates from p to q, then that digit can be eliminated from p.

A13) "Double Flying Fry"
Suppose a 2nd Flying Fry exists, such that the source or target cell of one Flying Fry sees the source or target cell of the other Flying Fry. If both source cells have the same 2 candidates XY, then X and Y can be eliminated from every cell that sees the source (or target) cells of both Flying Fry.

---

[14] If the target cell is a multi-cell (see later), some of its component cells might actually belong to different base lines. However, those component cells all see each other, so only 1 can be solved by X. To keep things simple, just assume that the source and target cells are single cells for now.

[15] A10 enables progress based on the knowledge that some digit X is already known to solve a source or target cell. In this pattern, the only realistic way this can be known if the source or target cell are already solved, but in generalizations (such as Exocet) some additional basic ways are available.

Note: the gaps in the application numbering are due to an alignment with another pattern known as Exocet (outside of the scope of this document) which uses 1 more source cell and 1 more base line.

Multiple targets

Using more than 1 target cell is desirable as that makes it easier to ensure that the base cells for X see each other (for every source digit X). However, if chosen arbitrarily, only a few applications would survive and moreover only in a much weakened form. For example, A1 would be reduced to "X can be eliminated from the source cells if no target cell supports X." Allowing multiple targets cells becomes much more useful if those cells aren't arbitrarily chosen but are candidate cells of a HALS[16] for non-source digits. The reason is that, no matter how the candidate cells are solved, at most 1 can be solved by a source digit. In fact, if the candidate cells of that HALS are considered to be 1 multi-cell, hardly anything needs to be changed at all:

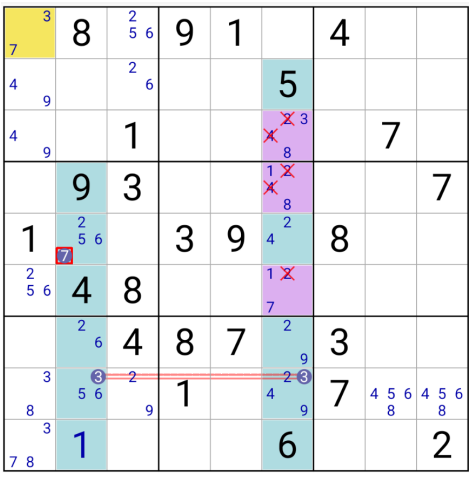
- In the key observation, it should be understood that "the" target cell refers to the unique candidate cell that isn't solved by an internal digit of the HALS.
- In A1, internal digits of the HALS are excepted: they can't be eliminated from the target (multi)cell.
- A11 becomes non-applicable (unless you can find a UR for each possible combination).

Multiple sources

Within the scope of this pattern, using multiple source cells would be useless (because if the source multi-cell pq offers progress, then so do the source cells p and/or q individually). However, increasing the number of source cells *does* make sense if the number of base lines is increased accordingly (as in Exocet which deals with 1 more source cell and 1 more base line.

Examples

A1



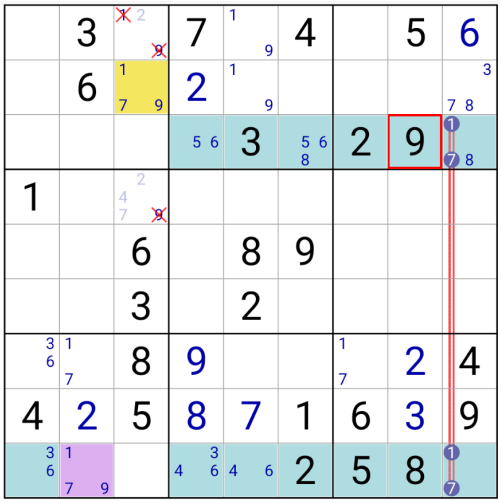
080910400000005000001000070093000007100390800048000000004870300000100700010006002

Source cells: r1c1, target cells: r346c6, base lines: columns 26

[16] Even a HALS is over-specific as a HA^{K}LS (N digits of a house with N+K candidate cells) for any K>0 would do.

- A1 "Shared Digits": r3c6≠24, r4c6≠24 and r6c6≠2 because 24 aren't shared.

A4

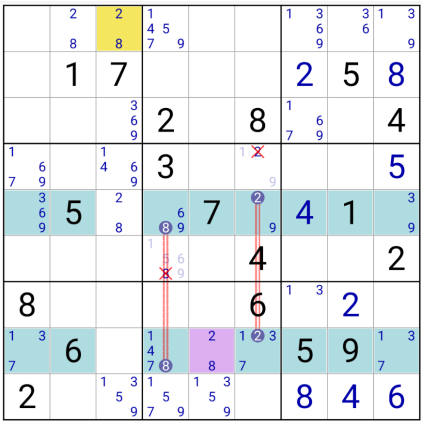


030704056060200000000030290100000000060890000030200000089000244258716390000025 80

Source cells: r2c3, target cells: r9c2, base lines: rows 39

- A4 "Finned Fish": r1c3≠1 because those cells all see [r29c3,r3c23].
- A4 "Finned Fish": r1c3≠9 and r4c3≠9 because those cells all see r29c3.

A5

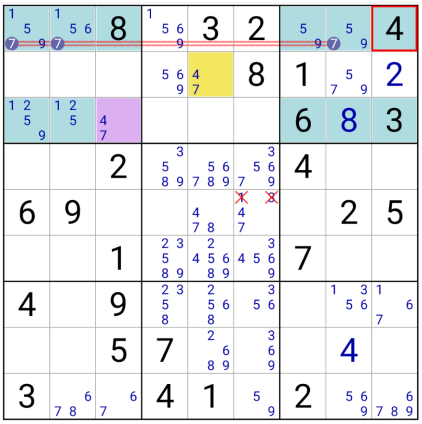


000000000017000258000208004000300005050070410000004002800006020060000590200000846

Source cells: r1c3, target cells: r8c5, base lines: rows 58

- A5 "Fish": r4c6≠2 because 2 can be fully covered by column 6.
- A5 "Fish":r6c4≠8 because 8 can be fully covered by column 4.

A6



008032004000008102000000683002000400690000025001000700409000000005700040300410200

Source cells: r2c5, target cells: r3c3, base lines: rows 13

- A6 "Full Translocation T ⇉ P": r5c6≠13 because 47 translocate from r3c3 to r5c6 in row 5.

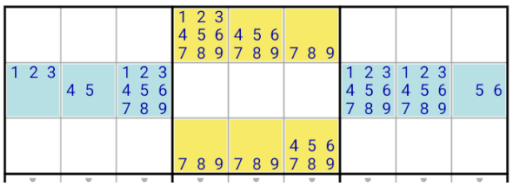
Varia

- This pattern was inspired by (Junior) Exocet (after which it's named), after realizing that seemingly important requirements of that pattern are mostly irrelevant for the logic behind its applications, especially the number of base lines.
- Flying Fry simplifies Exocet by reducing the number of base lines from 3 to 2. That turned out not to be an over-simplification, as this pattern manages to out-perform Exocet by an order of magnitude.

Anti-Pointer

An Anti-pointer consists of the cells of an intersecting box and line, excluding the intersection.

The cells of this structure can be subdivided into 2 components, namely 1) the cells of the box outside the line and 2) the cells of the line outside the box. In the following sketch, the 2 components have been colored differently:

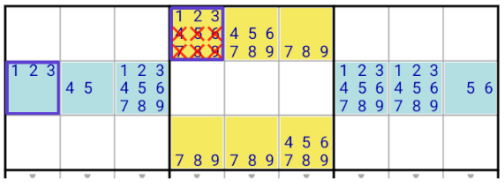


In yellow: cells of box 2 outside row 2.
In cyan: cells of row 2 outside box 2.

Applications

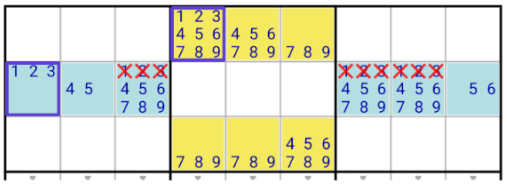
In the following applications, p is an arbitrary cell of one component and q is a cell of the other component:

A1) If q is the only cell of the other component that supports at least 1 candidate digit of p, then candidate digits not shared by p and q can be eliminated from p and q.



p = r2c1, q = r1c4 (A1)

A2) If q is the only cell of the other component that supports at least 1 candidate digit of p, then candidate digits shared by p and q can be eliminated from every cell in the component of p except p itself.



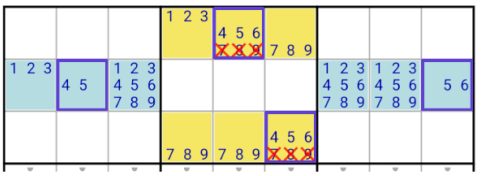
p = r2c1, q = r1c4 (A2)

Note that p and q being single cells is irrelevant; all that matters for the applications is that they have the same size.

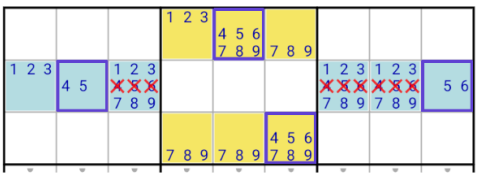
In the following, more general versions of A1 and A2, P is an arbitrary subset of cells in one component and Q is the set of all cells of the other component that support at least 1 candidate digit of P.

A1) If P contains exactly as many cells as Q, then candidate digits that are not shared by P and Q can be eliminated from P and Q.

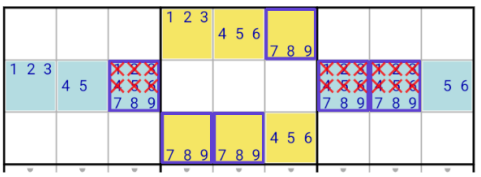
A2) If P contains exactly as many cells as Q, then candidate digits shared by P and Q can be eliminated from every cell in the component of P except P itself.



P = {r2c2, r2c9}, Q = {r1c5, r3c6} (A1)



P = {r2c2, r2c9}, Q = {r1c5, r3c6} (A2)



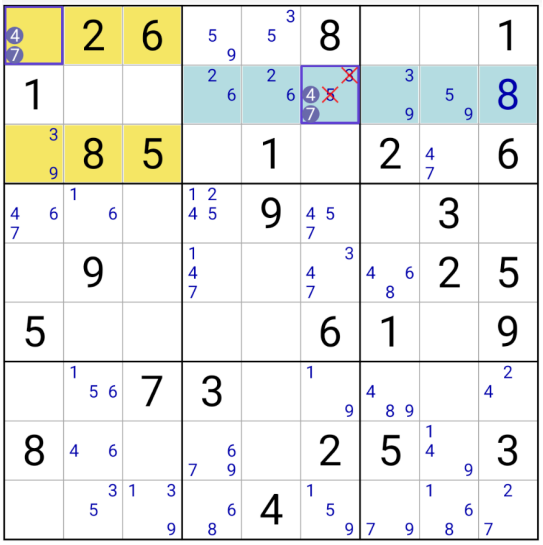
P = {r1c6, r3c4, r3c5}, Q = {r2c3, r2c7, r2c8} (A1)

Notes:

- While the running example above is of course highly contrived, it's nevertheless impressive that 2 simple applications manage to eliminate 30 digits from 12 cells without taking any other cells into account.
- It's never needed to consider multi-cells (P and Q) larger than 3 cells. While that may be intuitively true, it's not entirely trivial to prove (and strictly speaking not even completely true), so the appendix provides a proof (on page).

Examples

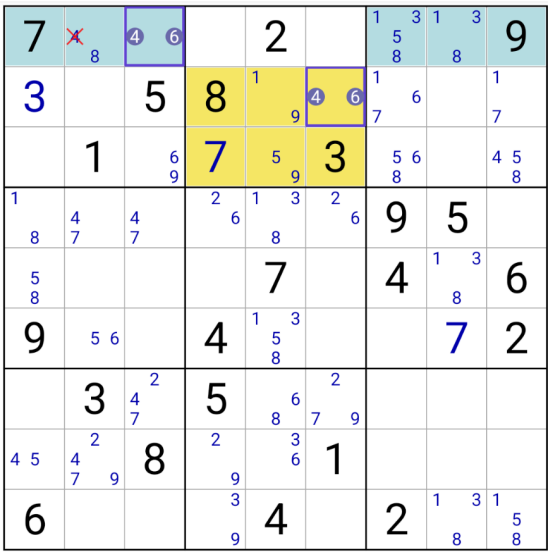
A1



026008001100000008085010206000090030090000025500006109007300000800002503000040000

The solution of r1c1 can only end up in r2c6, hence it must be solved by 4 or 7.

A2



700020009305800000010703000000000950000070406900400072030500000008001000600040200

r1c3 and cell of r1c46 must form a Naked Pair for 46.

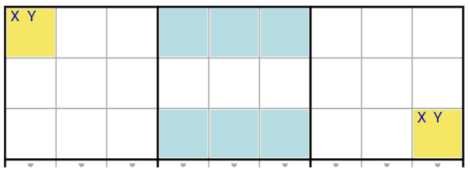
Varia

- Anti-pointer was inspired by Pointer (better known as Intersection Removal, Box-Line reduction or Pointing Pair/Triple); instead of focusing on the intersection of a box and a line, this pattern considers what can be done with the other cells of that box and line. The name was inspired by that complementary aspect (especially since Anti-pointer forces solutions into unseen cells whereas Pointer forces eliminations from seen cells).

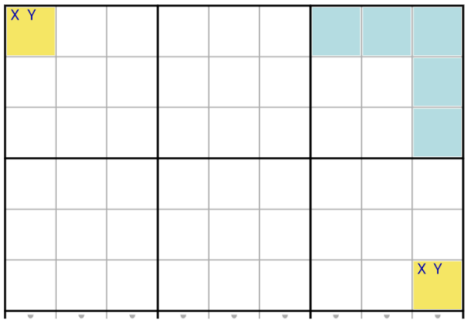
- Anti-Pointer can be interpreted as a special case of SET, although the usual applications of SET are too coarse-grained to be useful for Anti-Pointer. The idea of Anti-pointer to take a subset of cells of one component, to next determine the cells of the other component that support the same candidate digits (as in A1/A2) can also be applied to SET; however, for SET that's unpractical due to the combinatorial explosion of the number of sets P worth considering (as the number of houses increases or their overlap reduces).

XY-Flipbox

An XY-Flipbox consists of 2 bivalued cells p_1 and p_2 with candidate digits XY, such that each cell sees a mini line of the same box which doesn't support X or Y outside those mini lines.



p_1=r1c1, p_2=r3c9
(only the cyan cells of box 2 support XY)



p_1=r1c1, p_2=r6c9
(only the cyan cells of box 3 support XY)

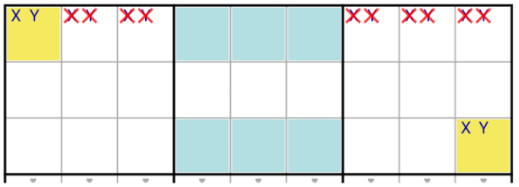
Applications

In the following, X and Y are the 2 candidate digits of p_1 and p_2 and w_1 and w_2 are the 2 mini lines that see resp. p_1 and p_2.

A1) X and Y can be eliminated from every cell that sees p_1 and p_2.

A1

A2) X and Y can be eliminated from every cell that sees p_i and w_i (i=1,2).



A2 (only shown for $p_1 \sim w_1$)

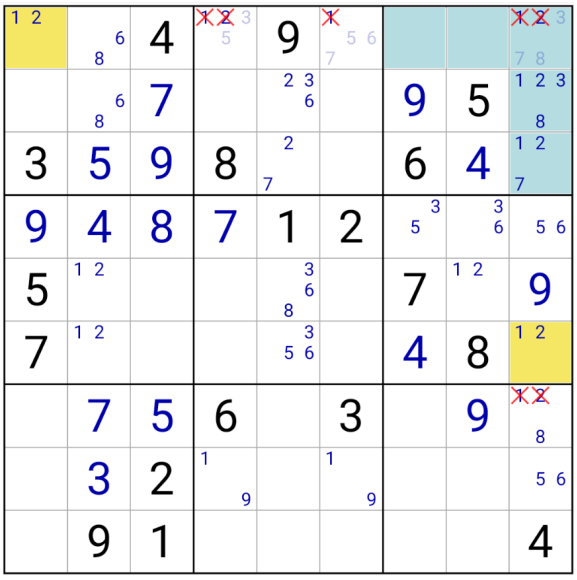
Examples

A1 or A2

005000003400056000709000050600107008000069030000030000301600200806090004042070086

In this example A1 and A2 offer exactly the same eliminations.

A1 and A2



004090000007000950359800640948712000500000709700000480075603090032000000091000004

The eliminations from r1c9 follow from A1, the other eliminations from A2.

XY-Coloring

XY-Coloring colors (some) bivalued cells using 2 colors (say, yellow and cyan) such that either all yellow or all cyan cells are solved by their smallest candidate[17].

Applications

A1) If a cell p sees 2 differently colored cells that both have Z as the smallest or largest candidate, then Z can be eliminated from p.

A2) If a cell p sees 2 cells of the same color, such that the smallest candidate Z of one is the largest candidate of the other, then Z can be eliminated from p.

A3) If a cell has 2 colors, the puzzle has no solution. This is of impossible in a proper puzzle, so any assumption that would result in a bicolored cell is known to be false.

Coloring rules

Coloring starts by identifying a Naked Pair pq{XY} (i.e. 2 cells p and q that combined have candidate digits X and Y). This allows coloring p and q respectively yellow and cyan.

Next, more cells are colored. In the following, p and q are bivalued cells in which a shared candidate X is bilocated. Cell p is already colored, q isn't.

Cell q is colored in the same color (as p) if:

- X is the smallest candidate of p and the largest of q, or
- X is the largest candidate of p and the smallest of q.

Cell q is colored in the other color if:

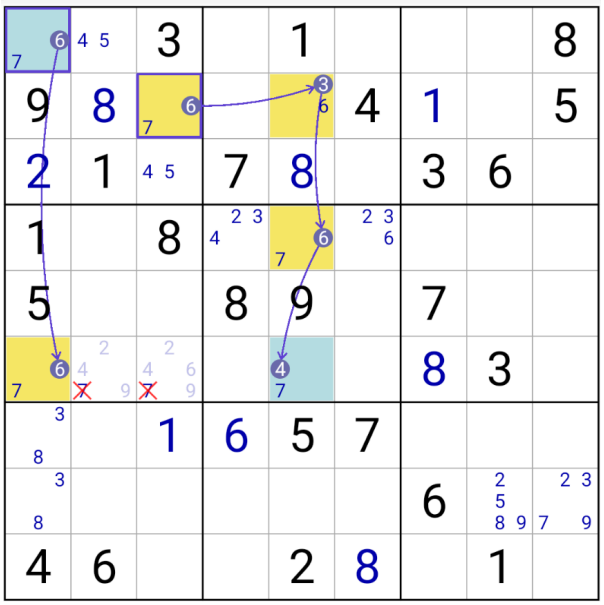
- X is the smallest candidate of both p and q, or
- X is the largest candidate of both p and q.

To summarize: the other color is used if X is the smallest or largest candidate digit of both cells, and otherwise the same color is used.

[17] Since only bivalued cells are considered, the coloring property of the definition is equivalent to the property that "either all cyan or all yellow cells are solved by their largest candidate".

Examples

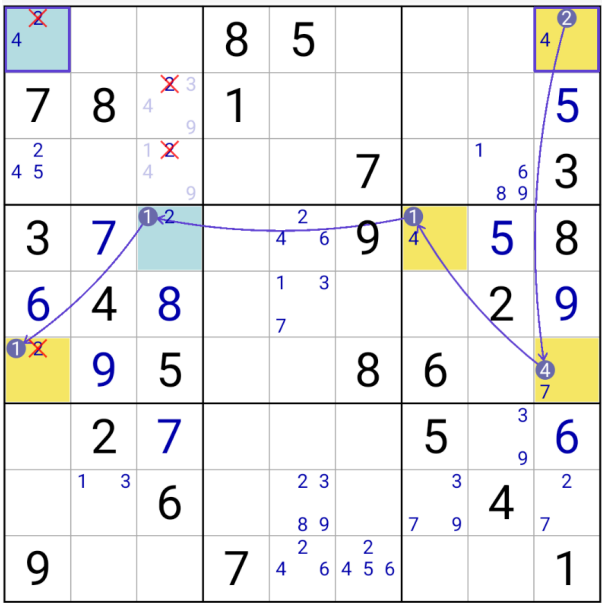
A1



003010008980004105210780360108000000500890700000000830001657000000000600460028010

Starting with [r1c1,r2c3]{67}, the resulting coloring allows (A1) the elimination of 7 from r6c23 (since 7 is the largest candidate digit of r6c1 and r6c5).

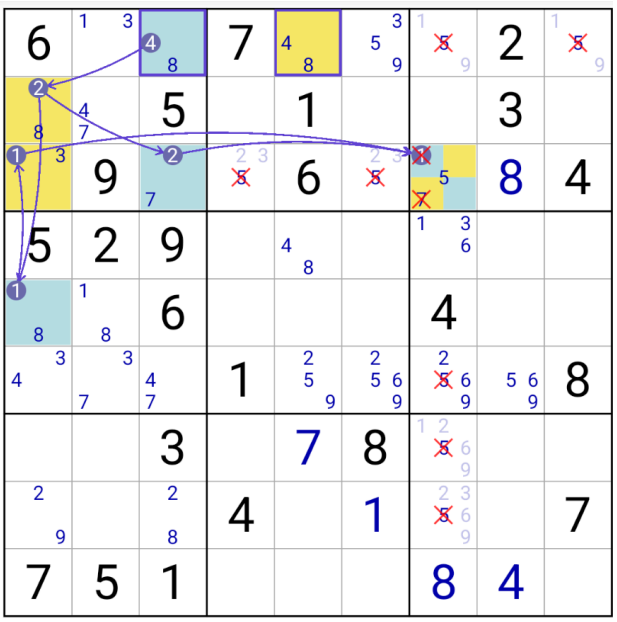
A2



000850000780100005000007003370009058648000029095008600027000506006000040900700001

Starting with r1c19{24}, the resulting coloring allows (A2) the elimination of 2 from r23c3 and r6c1 (since 2 is the smallest candidate digit of r1c1 and the largest of r4c3).

A3



000850000780100005000007003370009058648000029095008600027000506006000040900700001

Starting with r1c35{48} and the assumption that 5 doesn't solve r3c7, the resulting coloring produces a bicolored cell. Hence the assumption is false.

Varia

- This pattern was inspired by X-Coloring (aka Simple Coloring) and the result of trying to lift the relation between X-Chains and X-Coloring to XY-Chains and (some form of) XY-Coloring.
- XY-Coloring is a peculiar pattern, as digit permutations can change the min/max relations between cells. At first glance, this suggests that XY-Coloring might break the rule/guideline that patterns should be invariant under digit permutations. However, that non-invariance is just an optical illusion, thanks to the beautiful symmetries between the coloring rules and applications.

Locked Set Coloring

Locked Set Coloring is based on a Locked Set of size N with cells $p_1 \cdots p_N$. It colors cells using N colors $C_1 \cdots C_N$ such that if a cell doesn't have color C_N then it can't have the same solution as p_N.

Applications

A1) If cell p has a uniform color and supports only (some) locked digits, the locked digits it doesn't support can be eliminated from every cell that's also uniformly colored in the same color.

A2) If a cell p has no color but supports some locked digits, they can be eliminated from p.

A3) If in a house the joint set of candidate digits of cells with at least color C_N doesn't contain Z, then Z can be removed from every cell that is the only cell in a house with color C_N.

Coloring rules

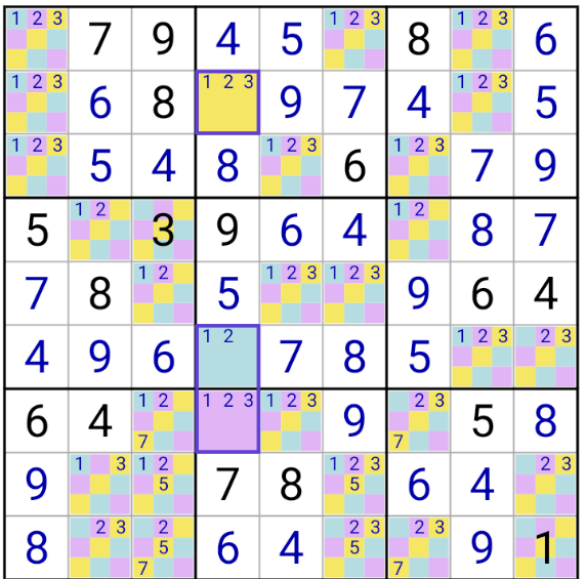
Coloring starts by applying all colors to every cell that supports at least one locked digit, except the locked cells themselves: every locked cell p_N is uniformly colored in C_N.

Next, colors are erased from cells. Any pattern can be used to eliminate colors. For example, if some cell p (initially only p_N) is uniformly colored in C_N and supports only (some) locked digits, C_N can be erased from every cell that sees p.

Examples

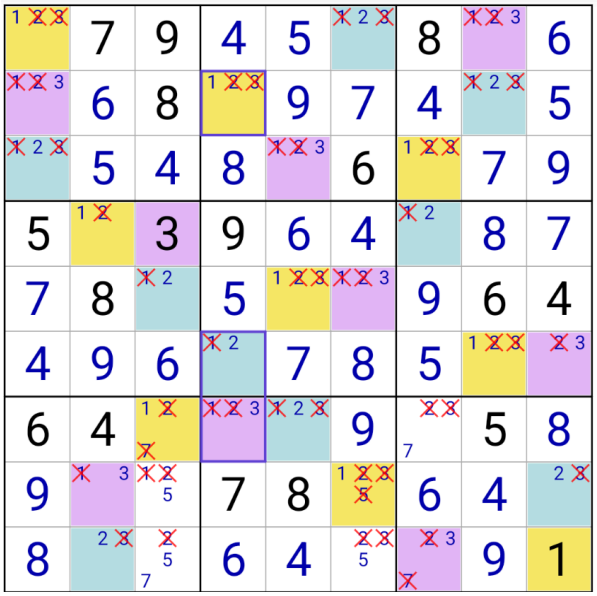
A1+A2+A3

Initial coloring based on r267c4{123}:



000850000780100005000007003370009058648000029095008600027000506006000040900700001

Next, colors are eliminated using any technique or pattern. For example, r2c4=yellow eliminates candidate yellow from r1c6, r2c1, r2c8 and r3c5 as they see r2c4. After several such elementary steps, all colored cells become monochrome (which is ideal but not necessary for A1/2/3) as follows:



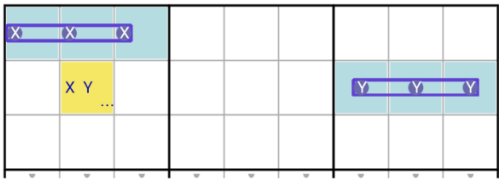
079450806068097405054806079503964087780500964496078500640009058900780640800640091

Varia

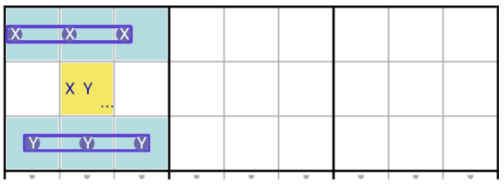
- This pattern was inspired by Anti-Pointer which also tracks the solution of one or more cells. The choice to use a coloring technique was inspired by pencil marks, which keep track of which specific digits can solve a particular cell. Rather than being explicit about what those digits are (e.g. "the digit 1"), Locked Set Coloring refers to digits indirectly, namely as the solution of a particular cell (e.g. "the digit that solves r2c4, whatever it may be").

Pointing Pair

A Pointing Pair consists of 2 Pointers W_X and W_Y for digits X and Y, under the provision that cell p (the "pivot") is solved by neither X nor Y.



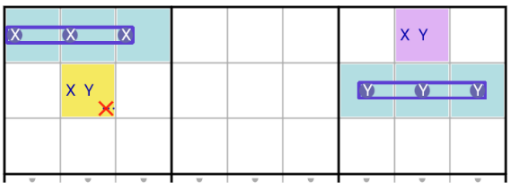
p=r2c2, w_X=r2c23, w_Y=r2c79



p=r2c1, w_X=r1c23, w_Y=r3c13

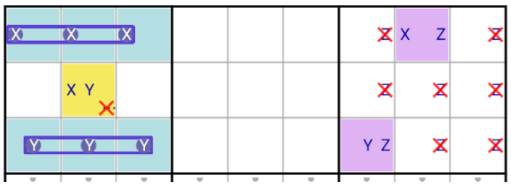
Applications

A1) If W_X and W_Y see bivalued cell u{XY}, then all candidate digits except X and Y can be eliminated from p.



A1 (u=r1c8)

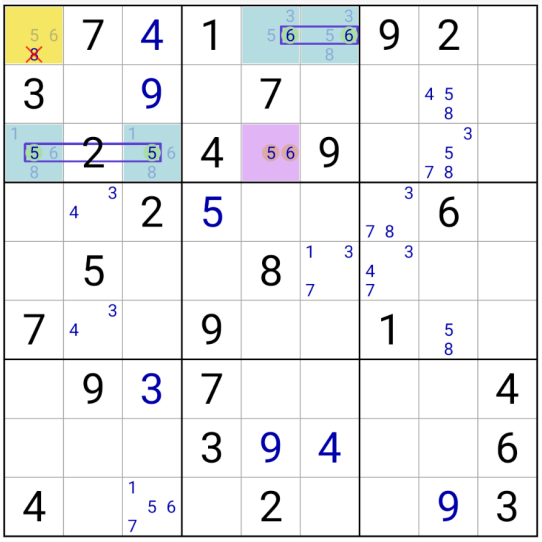
A2) If W_X sees bivalued cell u{XZ}, W_Y sees bivalued cell v{YZ} and u sees v, then all candidate digits except X and Y can be eliminated from p and Z can be eliminated from every cell that sees u and v.



A2 (u=r1c8, v=r3c7)

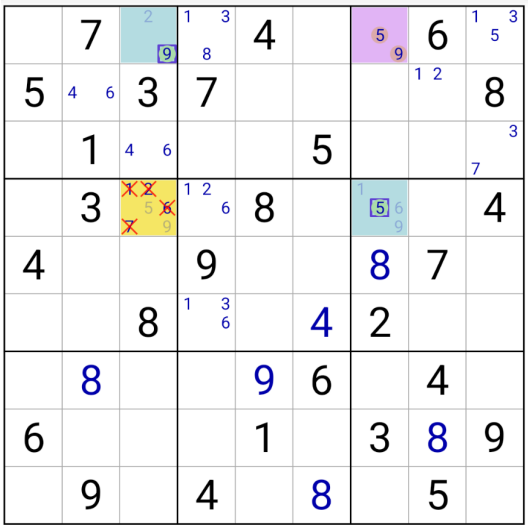
Examples

A1



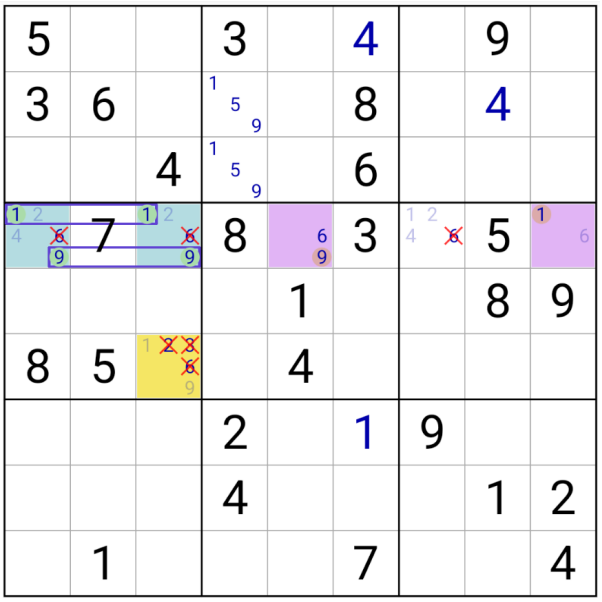
074100920309070000020409000002500060050080000700900100093700004000394006400020093

A1 (tiny pointers)



070040060503700008010005000030080004400900870008004200080096040600010389090408050

A2



500304090360008040004006000070803050000010089850040000000201900000400012010007004

Varia

- This pattern was inspired by Pointer, based on the question what can be done with a structure that is almost a Pointer.
- This pattern is surprisingly rich. For example, u in A1 can be allowed to support some more digits if they translocate to p (if so, every digit except the candidate digits of u can be eliminated from p). Another example is that A2 would remain nearly unchanged if a digit D is added to u and v, provided some cell with candidate digits DZ sees u, v and the pivot. Yet another example is based on the observation that u in A1 and u/v/uv in A2 as NALSes. Once realized, it's not hard to see that it’s possible to increase the size of the NALS(es) without invalidating the applications. It's even possible to replace u and/or v by HALSes.

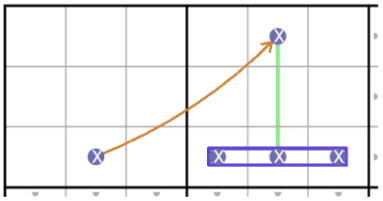
Translocation

A translocation is not a pattern but **an inference of the form p=X → q=X** ("if digit X solves cell p, then it also solves cell q", i.e. X "translocates" from p to q).

Translocations are vital to many patterns (but it doesn't seem to be recognized elsewhere as much as it deserves), so it makes sense to make the concept explicit here. Some types of translocations are described below.

Type 1 (1 bilocation)

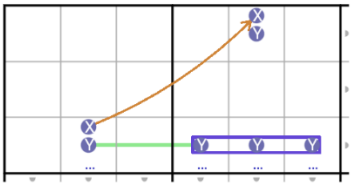
If p sees u and X is bilocated in u and q, then X translocates from p to q.



A translocation (orange)
based on 1 bilocation (green)

Type 2 (1 bilocation + 1 bivalued cell)

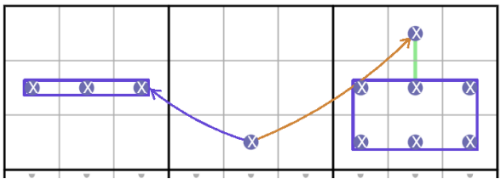
If p sees u and X is bilocated in u and q, then X translocates from p to q.



A translocation (orange), based on
1 bilocation (green) and 1 bivalued cell (r1c5)

Type 3 (1 translocation + 1 bilocation)

If X translocates from p to u and only 1 cell q of some house doesn't see p or u but supports X, then X translocates from p to q.



A translocation (orange) based on
1 translocation (blue) and 1 bilocation (green)

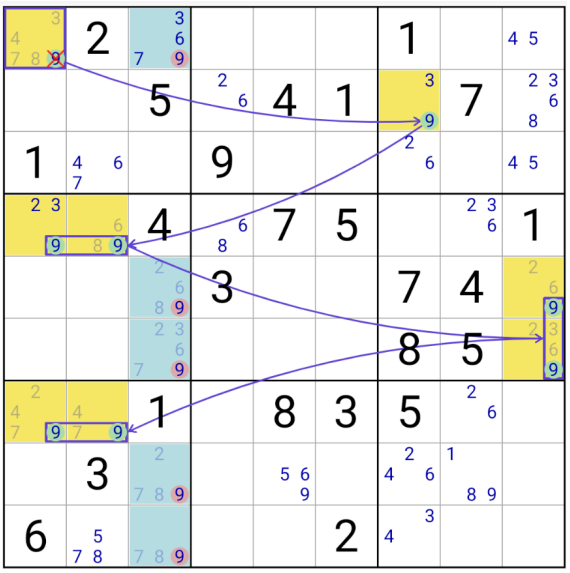
Note: if the sub-translocation is based on a bilocation (type 1) as in the sketch above, it's easy to spot. However, sub-translocations of types 2 or 3 are for some reason quite unintuitive (at least to the author).

More types

- For every cell p and digit X, X translocates from p to p. This is of course a trivial type of translocation but it broadens the scope of some patterns (like Hidden SM-Wing) a bit without having consider “degenerate” cases.
- If X translocates from p to q and from q to r, then X also translocates from p to r.
- In general, any Inference Chain of the form p=X → … → q=X offers the translocation p=X → q=X.

Translocation Chain

A Translocation Chain is an Inference Chain[18] in which every inference is a translocation.



Applications

For any Translocation Chain $p_1=X \rightarrow p_2=X \rightarrow \ldots \rightarrow p_n=X$:

A1) If every cell of some house that supports X sees a p_i, then X can be eliminated from p_1.

A2) If every cell of some house that supports X sees a p_i, and X translocates from p_n to p_1, then X can be eliminated from every p_i.

A3) If X translocates from p_n to p_1, then X can be eliminated from every cell that sees 1) some p_i and 2) every cell in a house of some p_n except p_n itself that supports X.

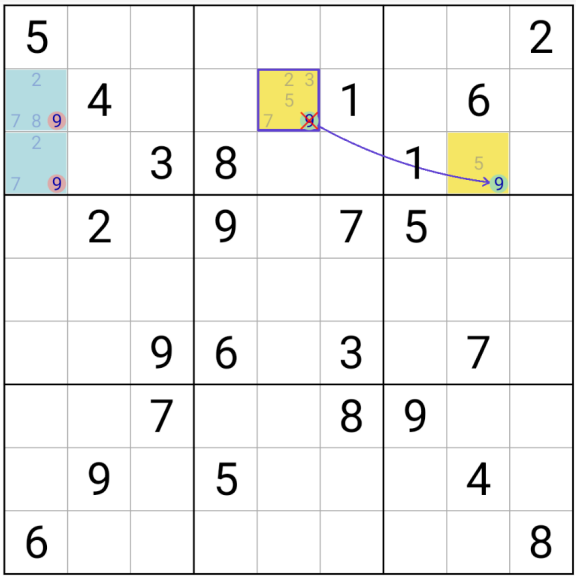
A4) If p_n sees some or all cells of p_1, then X can be eliminated from those cells.

Note: A3 captures the same eliminations as those by a cycle in an (Alternating) Inference Chain. Since all inferences are based on equalities in a Translocation Chain, some extra work has to be done to extract the eliminations.

[18] An Inference Chain is just a sequence of inferences, such that at each step the premise is a sufficient condition for the consequent to be true. It has no irrelevant and frankly silly requirements such as that the inferences should alternate between “strong links” and “weak links” as in AIC. In fact, the author considers AIC to be one of the least (properly) understood, and worst explained Sudoku concepts, not in the least because of how its pictures use undirected links (sets of propositions of which exactly 1 is true), but it’s textual explanations invariably use a degenerate form (binary, directed inferences).

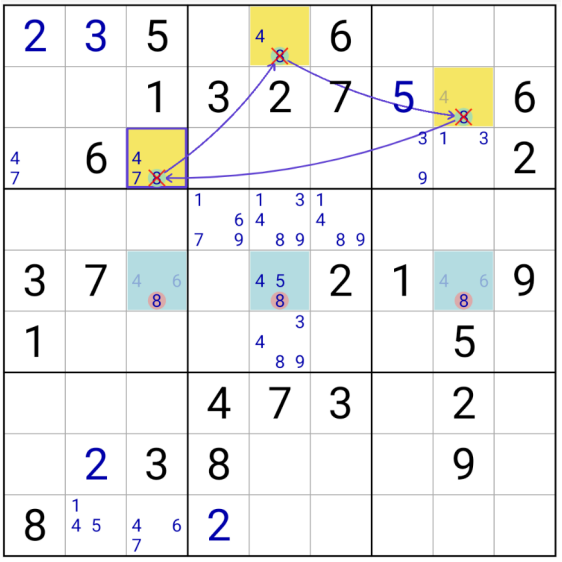
Examples

A1



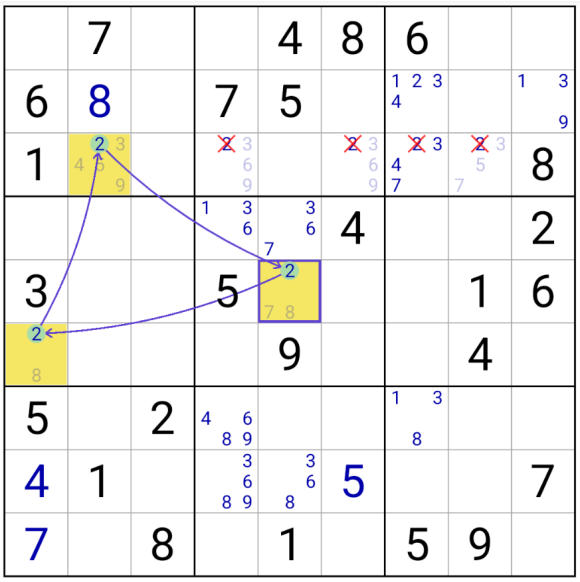
500000002040001060003800100020907500000000000009603070007008900090500040600000008

A2



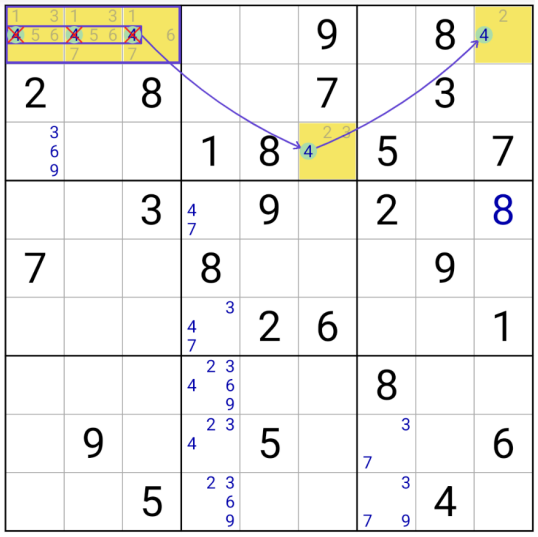
235006000001327506060000002000000000370002109100000050000473020023800090800200000

A3



070048600680750000100000008000004002300500016000090040502000000410005007708010590

A4



000009080208007030000180507003090208700800090000026001000000800090050006005000040

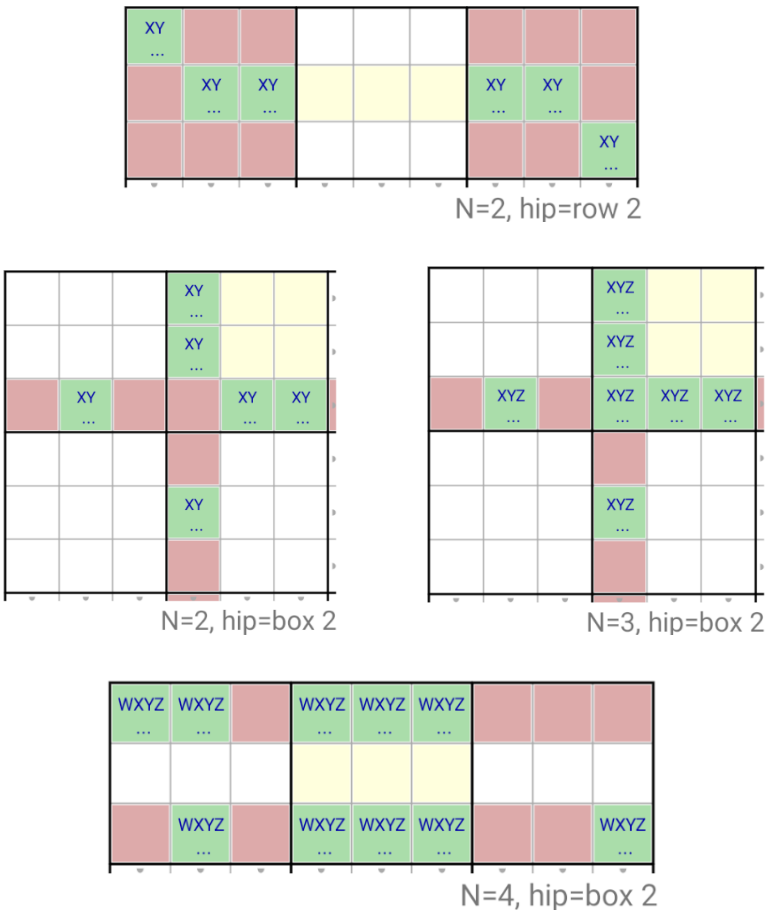
Varia

- This pattern does not have a particular inspiration, it's just an example of successful solving approach ("tracking a potential solution of a cell") I've been using for as long as I can remember.

Siamese Twins

Siamese Twins consist of 2 Hidden A⋆LSes[19] (the "twins") for the same N digits that share a house (the "hip") for some of their candidate cells (the "internal cells"). The twins are allowed to share at most 1 candidate cell.

A few of the many configurations possible are shown below:



Terminology

For each twin (a HA⋆LS of size N):

- The **internal digits** are the N digits of the Hidden A⋆LS.
- The **external digits** are the candidate digits of the candidate cells that are not internal.
- The **internal cells** are the candidate cells of the Hidden A⋆LS that belong to the hip.
- The **external cells** are the candidate cells of the Hidden A⋆LS that do not belong to the hip.

The **internal/external digits/cells** of Siamese Twins are the internal/external digits/cells of the twins combined.

[19] An Hidden A⋆LS (of size N) is a Hidden A^KLS (N digits of a house with N+K candidate cells) for some arbitrary K≥0. In the sketches, K=1 is used.

Applications

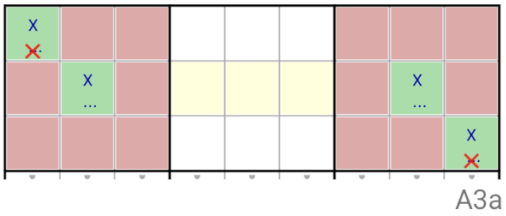
A1) Internal digit X can be eliminated from every cell that sees every instance of X in the external or shared cells.

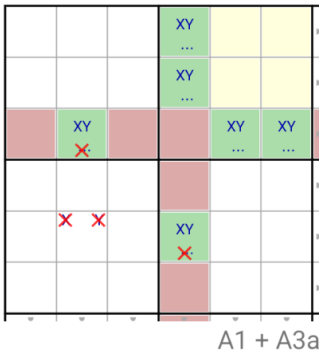
A2) If the number of external cells + the number of shared internal cells is at most N, all internal digits can be eliminated from every cell of the hip that does not belong to a twin.

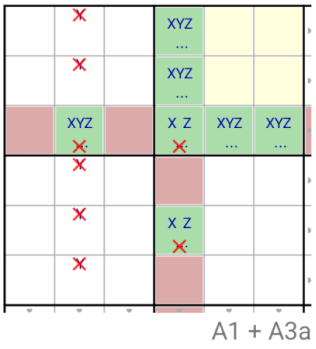
A3) If the number of external cells + the number of shared internal cells is N+K, the external or shared cells form a Hidden A^KLS for the internal digits.

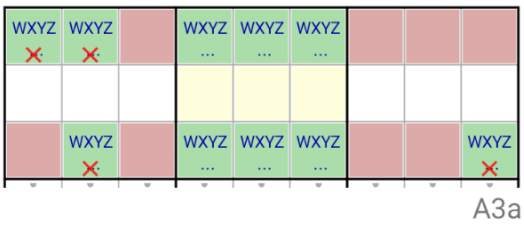
A3 doesn't explicitly describe eliminations but rather a useful fact to potentially deduce eliminations[20]. However, it takes only a small step to specialize A3 into more concrete applications, for example:

A3a) If K=0, every external digit can be eliminated from every external or shared cell.









The following application is also based on A3 for K=0, but it employs 2 Siamese Twins:

A3b) If 2 Siamese Twins reveal (via A3) the Hidden ALSes XY{pqr} and UV{pqs} (where pqrs are 4 distinct cells and XYUV are 4 distinct digits), r can be restricted to XY, s to UV and p and q to XYUV, i.e. other candidate digits can be eliminated.

[20] It's like the difference between "p or q are solved by Z" and "Z can be eliminated from every cell that sees p and q", only less trivial.

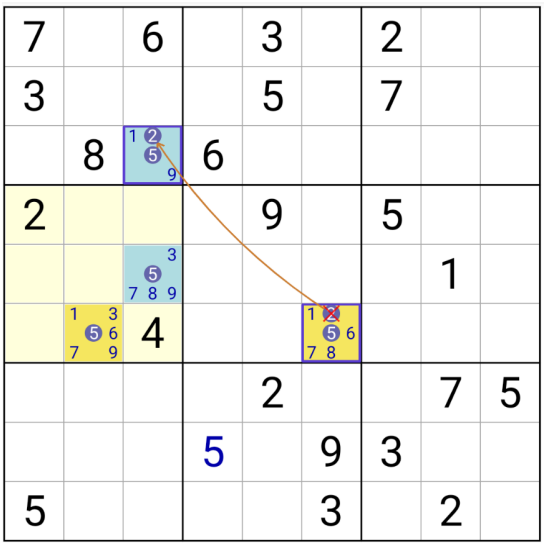
If K=1 in A3, the external cells + the shared cell (N+1 in total) form a Hidden ALS for N internal digits. Hence at most 1 candidate cell of that Hidden ALS (i.e., the external cells plus the shared internal cell) can be solved by an external digit.

A3c) If K=1 and an external digit translocates from one candidate cell to another, that digit can be eliminated from the former candidate cell.

A3d) If K=1 and p=X causes the elimination of all internal digits from 2 distinct candidate cells, X can be eliminated from p.

Examples

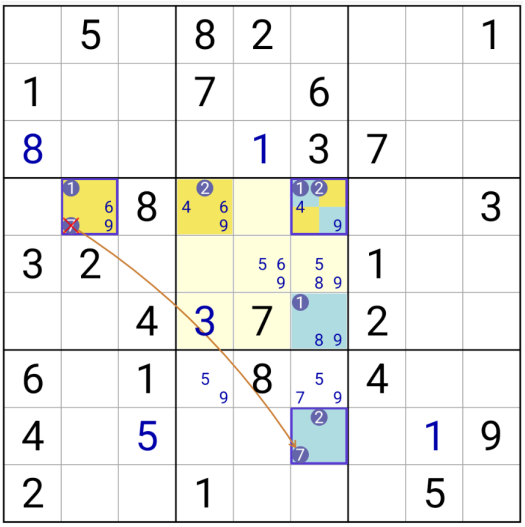
hip=box, N=1



706030200300050700080600000200090500000000010004000000000020075000509300500003020

The Hidden ALS 5{r6c26} and the Hidden ALS 5{r35c3} are conjoined by box 4, revealing the Hidden ALS 5{r3c3,r6c6}. Hence r6c6≠2, based on A3c (using r6c6=2 ⟶ r3c3=2).

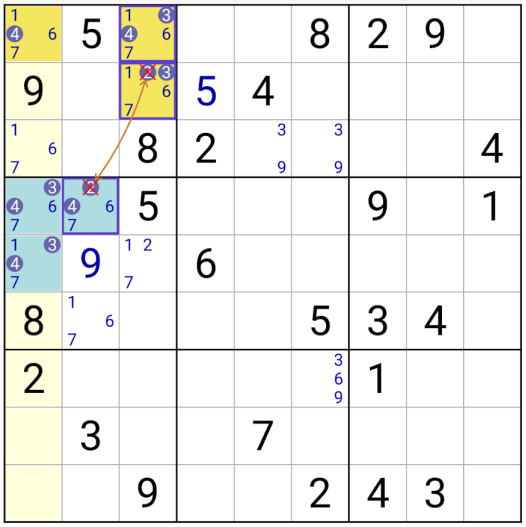
Hip=box, N=2



050820001100706000800013700008000003320000100004370200601080400405000019200100050

The Hidden ALS 12{r4c246} and the Hidden ALS 12{r468c6} are conjoined by box 5, revealing the Hidden ALS 12{r4c26,r8c6}. Hence r4c2≠7, based on A3c (using r4c2=7 ⟶ r8c6=7).

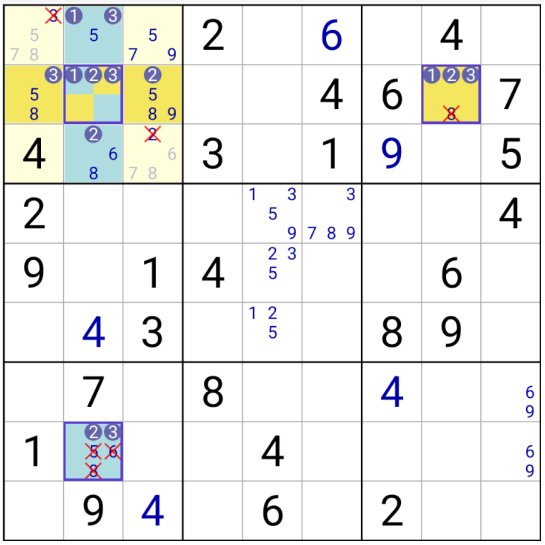
Hip=column, N=2



050008290900540000008200004005000901090600000800005340200000100030070000009002430

The Hidden ALS 34{r1c13,r2c3} and the Hidden ALS 34{r4c12,r5c1} are conjoined by column 1, revealing the Hidden ALS 34{r12c3,r4c2}. Hence r4c2≠2 and r2c3≠2, based on A3c (using r4c2=2 ⟶ r2c3=2 and r2c3=2 ⟶ r4c2=2).

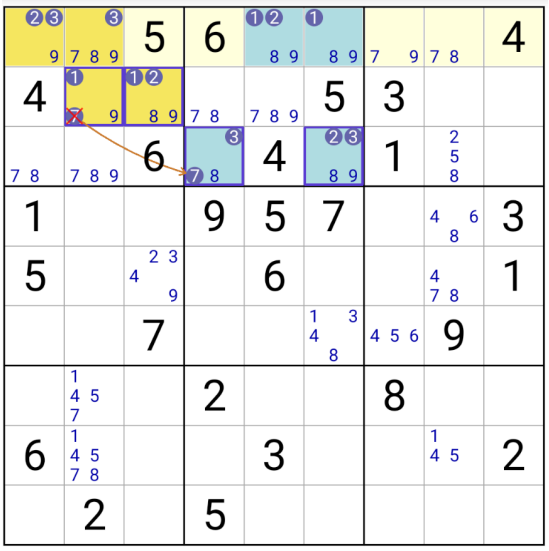
Hip=box, N=3 ("Triple Firework")



000206040000004607400301905200000004901400060043000890070800400100040000094060200

The Hidden ALS 123{r2c1238} and the Hidden ALS 123{r1238c2} are conjoined by box 1, revealing the Hidden LS 123{r2c28,r8c2}. Hence r2c8≠8 and r8c2≠568, based on A3a. The eliminations from r1c1 and r3c3 follow from A2.

Hip=row, N=3



005600004400005300006040100100957003500060001007000090000200800600030002020500000

The Hidden ALS 123{r1c12,r2c23} and the Hidden ALS 123{r1c56,r3c46} are conjoined by row 1, revealing the Hidden ALS 123{r2c23,r3c46}. Hence r2c2≠7, based on A3c (using r2c2=7 → r3c4=7).

Varia

- Siamese Twins was inspired by Triple Firework.

Naked SM-Wing

A Naked SM-Wing consists of 2 translocations for 2 different digits from 2 pivots to the same wing.

Applications

In the following applications, the 2 translocations are $p_1=X_1 \rightarrow w=X_1$ and $p_2=X_2 \rightarrow w=X_2$ ($X_1 \neq X_2$).

A1) If p_2 supports only candidate digits X_1 and X_2 and sees p_1, then X_1 can be eliminated from p_1.

A2) If X_2 is bilocated in p_1 and p_2, then X_1 can be eliminated from p_1.

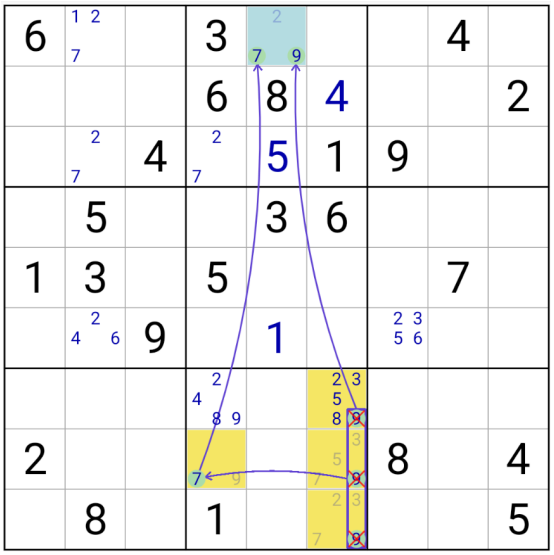
Note that the above applications are specific instances of a more abstract application:

A1) If $p_1=X_1 \rightarrow p_2=X_2$, then X_1 can be eliminated from p_1.

Also note that in A0 (and A1), p_1 and p_2 can be multi-cells. However, for A2 it makes no sense to consider multi-cells p_2 (since p_2 only supports X_1 and X_2 it would have to form a Naked Pair, which would trivially eliminate X_1 from p_1).

Examples

A1



600300040000684002004051900050036000130500070009010000000000002000008040801000005

Varia

- Naked SM-Wing was inspired by M-Ring which can be considered to be a special case of Naked SM-Wing (where $p_1=p_2$ and the 2 translocations are equivalences). Also, if the translocations are of type 1 only, applications A1 and A2 correspond to resp. M-Wing and S-Wing.

Hidden SM-Wing

A Hidden SM-Wing consists of 2 translocations for the same digit from 2 pivots to 2 wings that see each other.

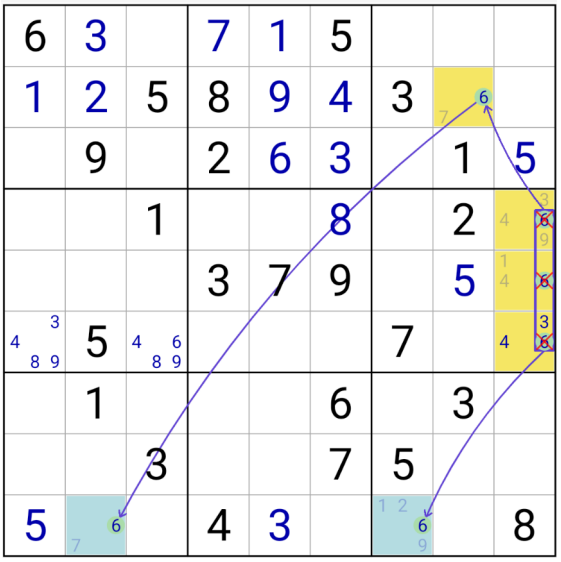
Applications

In the following application, the 2 translocations are p_1=X ⟶ w_1=X and p_2=X ⟶ w_2=X ($p_1 \neq p_2$).

A2) If X translocates from p_1 to p_2, then X can be eliminated from p_1.

Examples

A1



630715000125894300090263015001008020000379050050000700010006030003007500500430008

Varia

- Hidden SM-Wing was inspired by Naked SM-Wing.

Half Naked ALS Pair

A Half Naked ALS Pair consists of a HALS (Hidden ALS) and a NALS (Naked ALS) that interact via a SECP.

In the pattern known as ALS Pair, an RCC (Restricted Common Candidate) enables 2 NALSes to directly interact with each other. For the interaction between a HALS and a NALS, a similar concept is introduced:

A "Common Pair" (CP) is defined as a pair (p,X) such that:

- p is a candidate cell of the HALS that supports X, and
- X is a candidate digit of the NALS and of p.

An "entangled" CP (ECP) satisfies:

- Every cell of the NALS that supports X sees p.

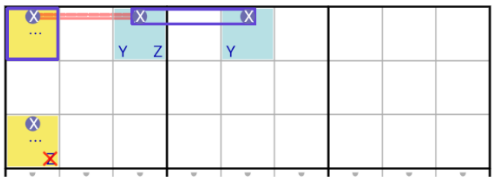
A "smallest" CP (SCP) satisfies:

- The only internal digit supported by p is X.

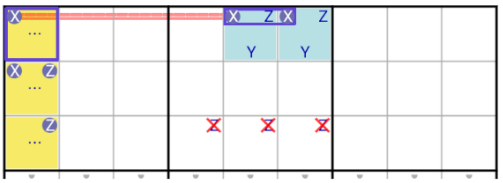
A CP that is both entangled as well as smallest is abbreviated SECP.

Applications

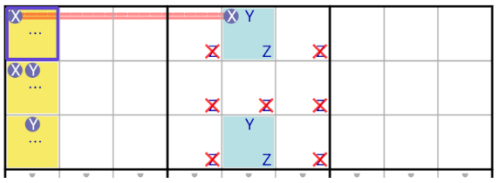
A1) If (p,X) is a SECP and (q,Z) an ECP where Z is an external digit of the HALS, then Z can be eliminated from q.



A2) If (p,X) is a SECP and (q,Z) an SCP, then Z can be eliminated from every cell that sees q and every instance of Z in the NALS.

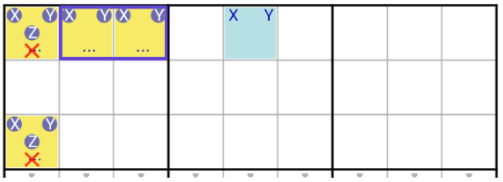


A3) If (p,X) is a SECP, (q,Y) an SCP and q sees K cells of the NALS that combined have K+1 candidate digits {Y,Z,…}, then Z can be eliminated from every cell r that sees every instance of Z in the K cells.



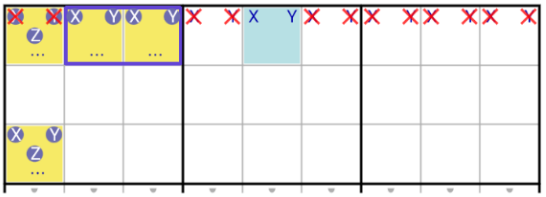
XY{r123c1}, r13c5{XYZ}
A3 (p=r1c1, q=r3c1, K=1)

A4) If (pq,XY) is a SECP, then all external digits can be removed from every candidate cell of the HALS except p and q.



XYZ{r1c123,r3c1}, r1c5{XY}
A4 (pq=r1c23)

A5) If (pq,XY) is a SECP, then for every candidate digit Z of the NALS, Z can be eliminated from every cell r that sees every instance of Z in the NALS. For Z=X and Z=Y, r should see p and q as well.



XYZ{r1c123,r3c1}, r1c5{XY}
A5 (pq=r1c23)

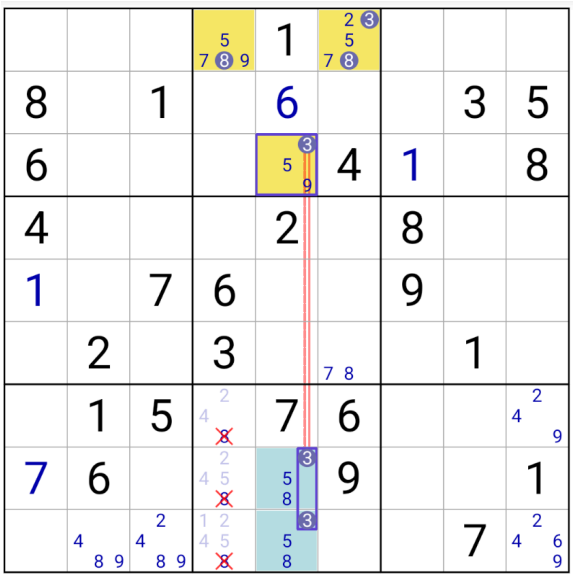
Examples

A1

7	5 8			9				6
	3			4				
	9	4			7		5	
	5 6	7	4				8	
4 8 9	6	4 8			5	2		
	1				8	6		
	2		7					
		8		5			9	
3								1

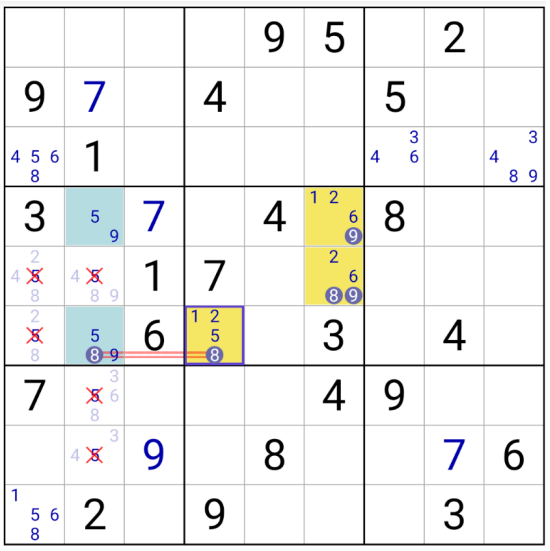
700090006030040000094007050007400080000005200010008600020700000008050090300000001

A2



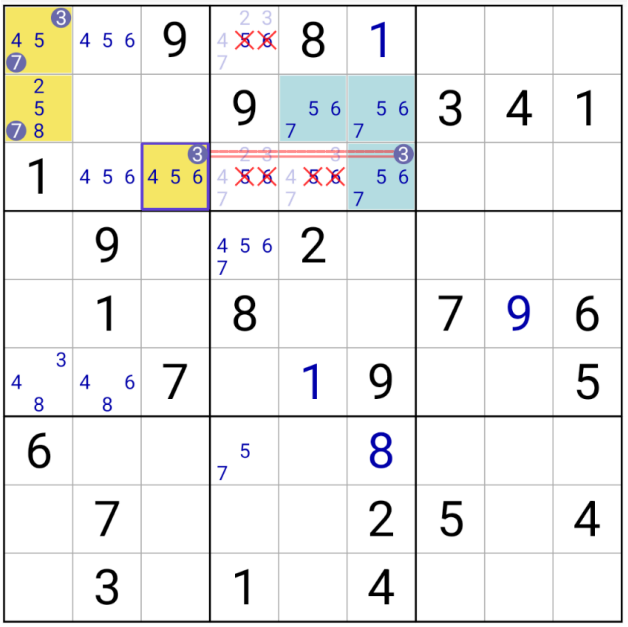
000010000801060035600004108400020800107600900020300010015076000760009001000000070

A3 (K=1)



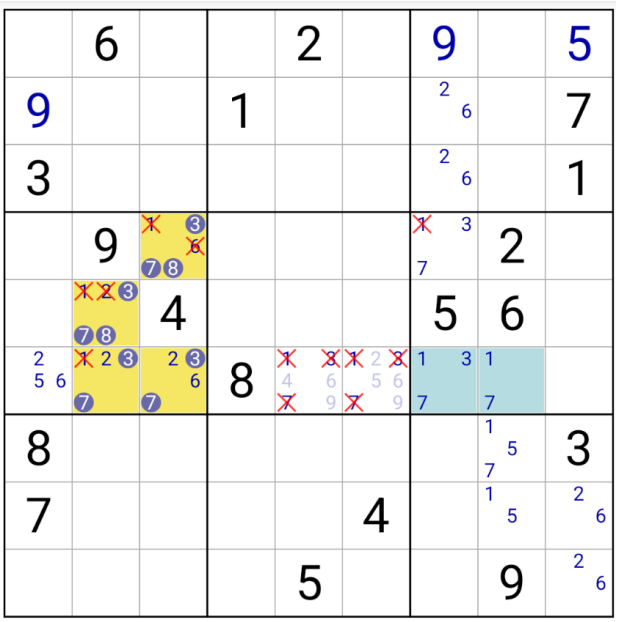
000095020970400500010000000307040800001700000006003040700004900009080076020900030

A3 (K=2)



009081000000900341100000000090020000010800796007019005600008000070002504030104000

A4 + A5



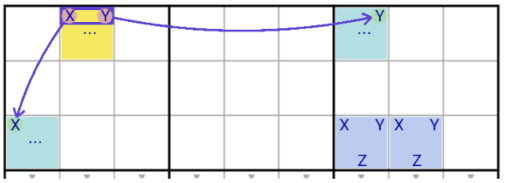
060020905900100007300000001090000020004000560000800000800000003700004000000050090

Varia

- Half Naked ALS Pair was inspired by (Naked) ALS Pair which uses 2 NALSes. Half Naked ALS Pair improves upon the performance of ALS Pair by about 70% (in terms of occurrences in puzzles and average number of eliminations). Obviously, a variant that uses 2 HALSes was also considered. However, the performance of that “Hidden ALS Pair” pattern turned out to be abysmal.

Puppet Master

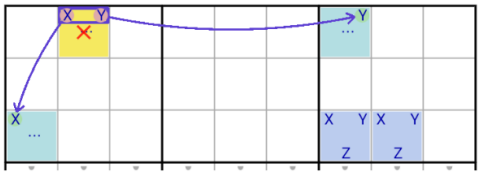
A Puppet Master consists of a cell (the "pivot") with candidate digits XY⋯, such that X and Y are bilocated in the pivot and 2 cells (the "wings", one for X and one for Y) that are able to eliminate X or Y from the same NALS (the "target").



Pivot: r1c2{XY⋯}
Wings: r3c1{X⋯} and r1c7{Y⋯}
Target: r3c78{XYZ}

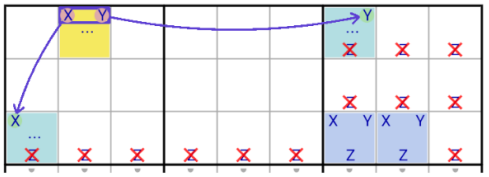
Applications

A1) Every candidate except X and Y can be eliminated from the pivot.



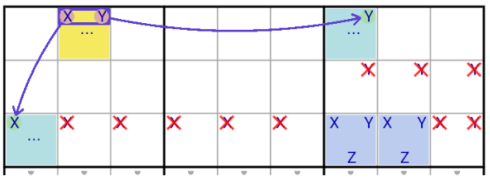
A1

A2) Every candidate Z of the target except X and Y can be eliminated from every cell sees every instance of Z in the target.



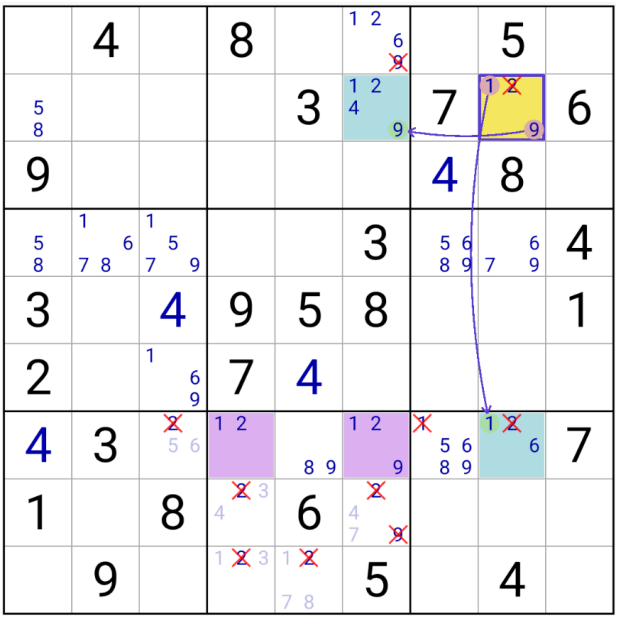
A2

A3) X (or Y) can be eliminated from every that cell sees the target and the wing for X (or Y).



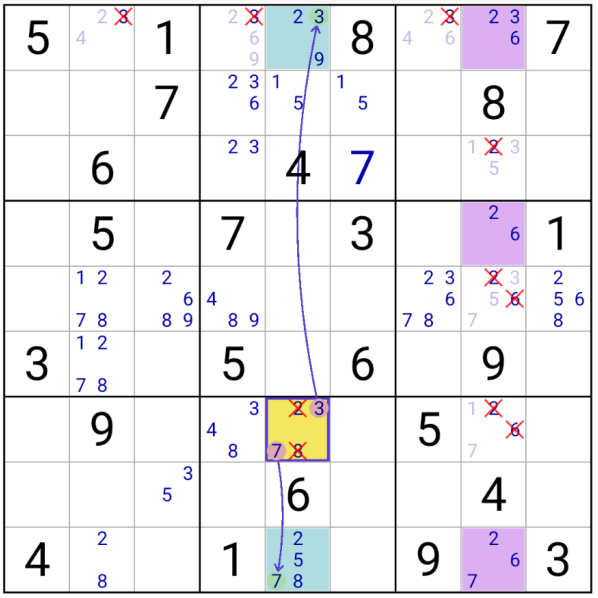
A3

Examples



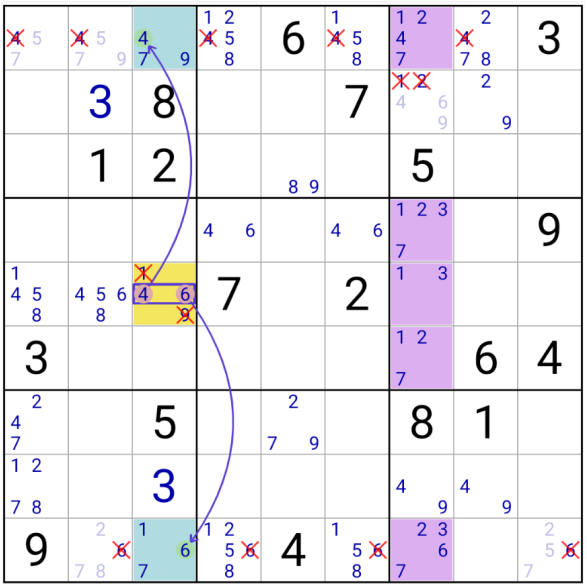
040800050000030706900000480000003004304958001200740000430000007108060000090005040

Candidates 19 are bilocated in the pivot r2c8 and the wings r2c6 and r7c8=1, which locks the target Naked ALS r7c46{129} (since r2c6=9 or r7c8=1).



501008007007000080060047000050703001000000000300506090090000500000060040400100903

Candidates 37 are bilocated in the pivot r7c5 and the wings r9c5 and r1c5=3, which locks the target Naked ALS r149c8{2367} (since r9c5=7 or r1c5=3).



000060003038007000012000500000000009000702000300000064005000810003000000900040000

Candidates 46 are bilocated in the pivot r5c3 and the wings r9c3 and r1c3=4, which locks the target Naked ALS r14569c7{123467} (since r9c3=6 or r1c3=4).

Varia

- Puppet Master was inspired by XY-Wing (instead of instead of considering the cases P=X and P=Y, it was considered how the cases P≠X and P≠Y could be exploited).

Near-death Blossom

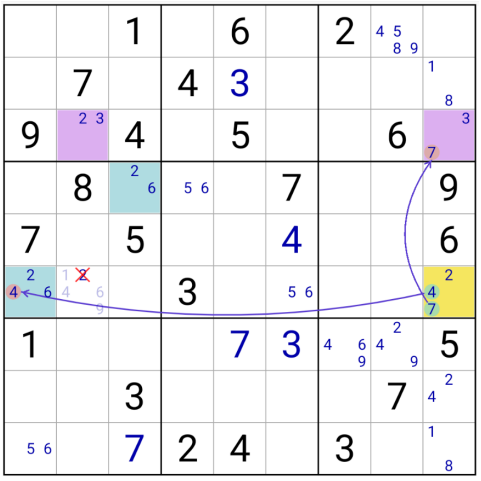
A Near-death Blossom consists of a cell (the "pivot") of which each candidate digit X except Z is associated with a Naked ALS (the "wing" for X) such that 1) X and Z are candidate digits of the wing, and 2) the pivot sees every instance of X in the wing.

Applications

A1) Z can be eliminated from every cell that sees every instance of Z in the Near-death Blossom.

Examples

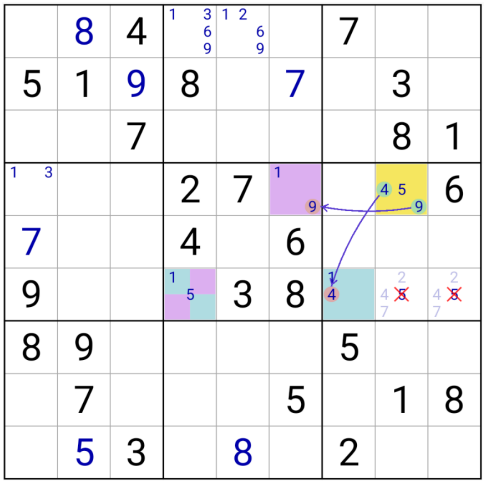
A1



0010602000704300009040500600800070097050040060003000001000730050030000700072403 00

r6c9≠2 locks Naked ALSes r3c29{237} or [r4c3,r6c1]{246}. Regardless of which ALS (if any) gets locked, r6c2≠2.

A1 (with overlapping wings)



0840007005198070300070000810002700067004060009000380008900005000700050180530802 00

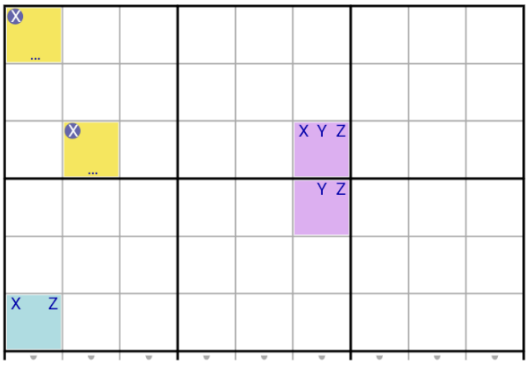
r4c8≠5 locks Naked ALSes [r4c6,r6c4]{159} or r6c47{145}. Regardless of which ALS (if any) gets locked, r6c8≠5 and r6c9≠5.

Varia

- Near-death Blossom was inspired by Death Blossom.

ALS-PQ Wing

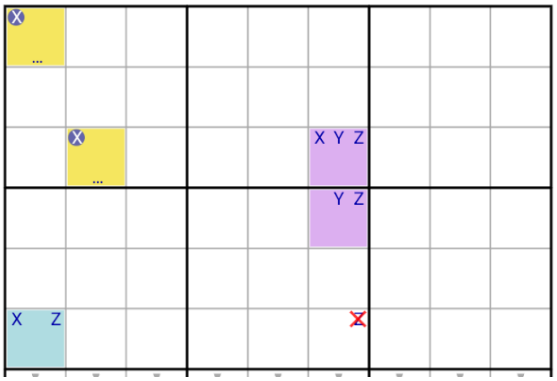
An ALS-PQ Wing consists of a Hidden ALS (the "pivot") with 2 candidate cells p_1 and p_2 that support only 1 internal digit (X_1 and X_2) and have a wing. That wing is a Naked ALS in which every instance of X_i (at least 1) sees p_i.



Pivot: X{r1c1,r3c2}
Wings: r6c1{XZ} and r34c6{XYZ}

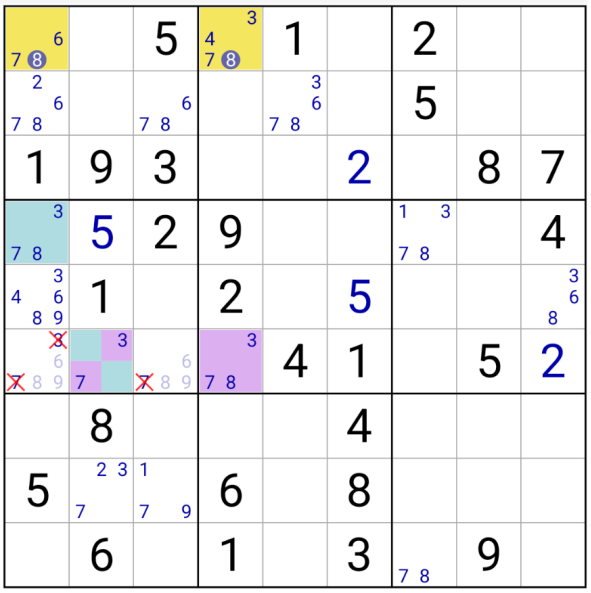
Applications

A1) If Z is a shared candidate digit of both wings other than X_1 and X_2, Z can be eliminated from every cell that sees every instance of Z in both wings.



Examples

A1



005010200000000500193002087052900004010205000000041052080004000500608000060103090

The Hidden ALS 8{r1c14} supports only 1 internal digit in r1c1{8…} and r1c4{8…}, which locks at least 1 wing as follows: [r4c1,r6c2]{378} = 37 or r6c24{378} = 37. Hence r6c1≠37 and r6c3≠7.

Varia

- ALS-PQ Wing was inspired by ALS-XY Wing (simply by letting the pivot be a HALS instead of NALS).
- It's of course possible to use other types of wings; all that's relevant is that the wing for p_i is affected by p_i=X_i (in this pattern, the wing is a NALS that turns into a Locked Set).

Digit Comb

A Digit Comb consists of a digit X with candidate cells p_1, …, p_N such that each p_i has a wing w_i. All individual wings w_i should offer the same elimination under the assumption that p_i=X is true.

Various types of wings can be used:

- A Naked ALS w_i with candidate digit X, such that every instance of X sees p_i.
- A Hidden ALS w_i with a candidate cell q that only supports X as an internal digit and sees p_i.
- A Hidden ALS w_i with external digit X and p_i as a candidate cell.
- A cell w_i that is p_i itself.
- A cell w_i such that X translocates from p_i to w_i.
- A cell w_i such that a digit Y other than X translocates from w_i to p_i.

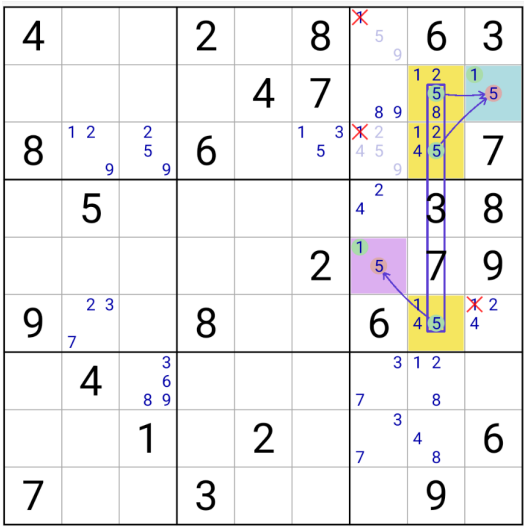
Of course, any Inference Chain of the form p_i=X → … → q≠Z can serve as a wing for p_i (and the pattern would be known as Digit Forcing Chain). However, building a chain for every p_i without constraints is costly. The interesting aspect of ALS Comb is that relying on just a few simple, almost "predefined" structures turned out to offer a surprisingly good performance (compared to single-celled digit forcing chains of unrestricted lengths).

Applications

A1) Every shared elimination is valid.

Examples

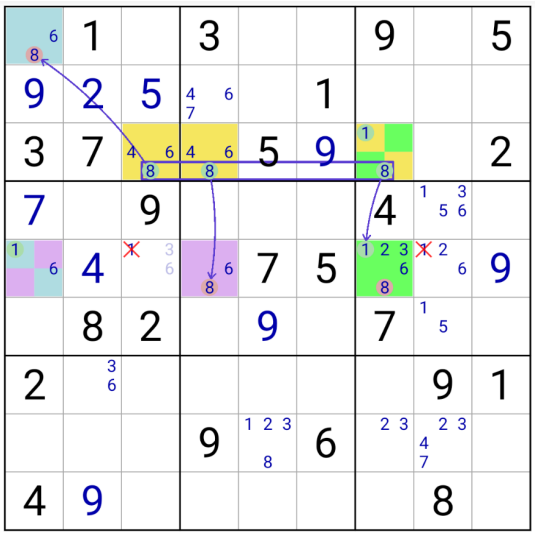
2 NALSes



400208063000047000800600007050000038000002079900800600040000000001020006700300090

5{r236c8} locks 1 of the following ALSes: r2c9{15} or r5c7{15}

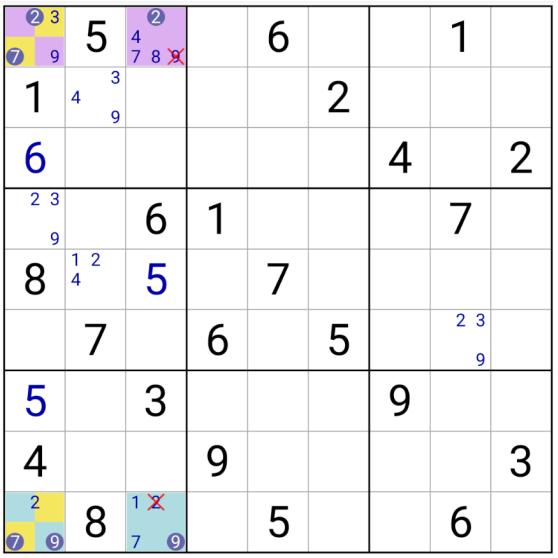
2 NALSes + 1 HALS.



0103009059250010003700590027090004000400750090820907002000000910009060004900000 80

8{r3c347} locks 1 of the following ALSes: r15c1{168}, r5c14{168} or 1{r35c7}.

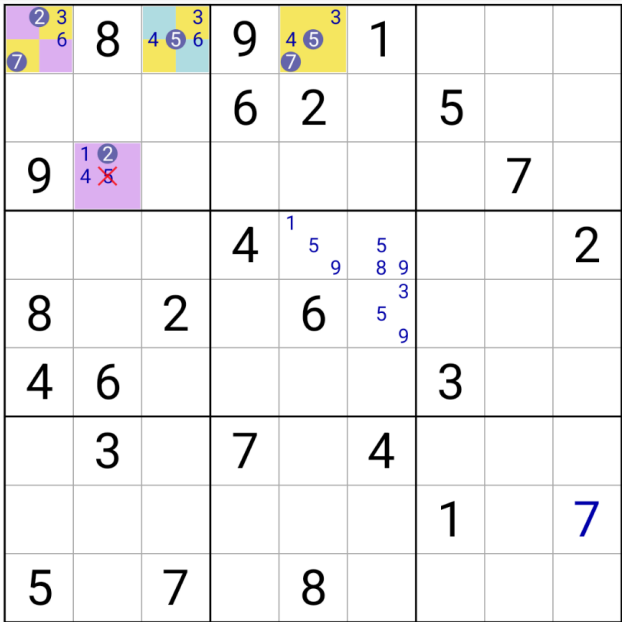
2 HALSes



0500600101000020006000004020061000708050700000706050005030009004009000030800500 60

7{r19c1} locks 1 of the following ALSes: 9{r9c13} or 2{r1c13}

HALS + cell



080901000000620500900000070000400002802060000460000300030704000000000107507080000

57{r1c135} locks at least 1 wing as follows: 2{r1c1,r3c2}=r3c2 or r1c3=5

Note that this example uses 2 digits for the pivot and therefore is not an instance of Digit Comb but rather some other, unknown pattern (that author hasn't bothered to define). The reason that the same ideas of Digit Comb can be used is that it's impossible for r1c1≠7 and r1c3≠5 to be both true (because r1c1 and r1c3 support only 1 digit of the HALS 57{r1c135}), hence r1c1=7 V r1c3=5 is a verity.

Nested Cycle

Nested Cycle uses Inference Chains to derive progress from sets of cycles.

If $P_1 \leftrightarrow \ldots \leftrightarrow P_N$ is a cycle then exactly 1 of the following falsity chains is valid and hence can be built (ignoring the effort this may require):

- $P_1 \,\&\, \ldots \,\&\, P_N \rightarrow \ldots \rightarrow \text{false}$
- $\neg P_1 \,\&\, \ldots \,\&\, \neg P_N \rightarrow \ldots \rightarrow \text{false}$

If successful, the resulting progress consists of N eliminations and solutions in total (for a simple cycle).

To minimize the difficulty of arriving at a falsity and to maximize progress, N shouldn't be tiny: a cycle with N<4 is almost always (too) hard, N=8 is good, N>16 is great. If no large enough cycle is available, it's possible to combine multiple cycles to achieve the same effect as 1 large cycle.

Combining cycles

Inference Chains are a powerful tool to combine cycles. Since chains are all about propositions, to apply them to cycles they must first be translated to propositions (via $\langle\!\langle \cdot \rangle\!\rangle$):

> **For any cycle C, $\langle\!\langle C \rangle\!\rangle$ is the proposition that "every proposition of C is true".**

More formally, $\langle\!\langle P_1 \leftrightarrow \ldots \leftrightarrow P\square \rangle\!\rangle = P_1 \,\&\, \ldots \,\&\, P\square$. The reason this pattern/technique uses cycles is that their propositions are either all true, or all false, i.e. either $\langle\!\langle C \rangle\!\rangle$ or $\langle\!\langle \neg C \rangle\!\rangle$, which gives the useful identity $\langle\!\langle \neg C \rangle\!\rangle = \neg\langle\!\langle C \rangle\!\rangle$.

Cycles can be combined by using just 2 (meta) inferences:

<u>Inference 1: “Inflation”</u>

> If $\langle\!\langle \neg P \rangle\!\rangle \wedge \ldots \wedge \langle\!\langle \neg Q \rangle\!\rangle \rightarrow \ldots \rightarrow \text{false}$,
> then $\langle\!\langle P \rangle\!\rangle \vee \ldots \vee \langle\!\langle Q \rangle\!\rangle$

<u>Inference 2: “Deflation”</u>

> If $\langle\!\langle P \rangle\!\rangle \vee \ldots \vee \langle\!\langle Q \rangle\!\rangle \vee \langle\!\langle Z \rangle\!\rangle$
> and $\langle\!\langle P \rangle\!\rangle \vee \ldots \vee \langle\!\langle Q \rangle\!\rangle \vee \langle\!\langle \neg Z \rangle\!\rangle$,
> then $\langle\!\langle P \rangle\!\rangle \vee \ldots \vee \langle\!\langle Q \rangle\!\rangle$

Inflation provides a conceptually easy way to construct sets of cycles P, …, Q such that $\langle\!\langle P \rangle\!\rangle \vee \ldots \vee \langle\!\langle Q \rangle\!\rangle$ is true. Deflation reduces pairs of those sets to smaller sets, until --ideally-- a set consisting of just 1 cycle P (for which $\langle\!\langle P \rangle\!\rangle$ is true) is discovered.

A minor simplification

Before inflating cycles, it's convenient to merge any overlapping cycles (just to reduce the number of cycle combinations to consider during inflation):

- If cycles P and Q share a proposition, then $\langle\!\langle P \rangle\!\rangle = \langle\!\langle Q \rangle\!\rangle = \langle\!\langle P \leftrightarrow Q \rangle\!\rangle$.
- If cycle P contains some proposition X and cycle Q contains proposition ¬X, then $\langle\!\langle P \rangle\!\rangle = \neg\langle\!\langle Q \rangle\!\rangle = \langle\!\langle P \leftrightarrow \neg Q \rangle\!\rangle$.

In both cases, P and Q can be abandoned and replaced by a larger cycle (either $P \leftrightarrow Q$ or $P \leftrightarrow \neg Q$).

Seeding cycles

Nested Cycles can be based on any set of "seeding" cycles, but preferably the total number of propositions (of all cycles combined) is large compared to the total number of cycles.

Just picking elementary propositions (like p=X) and plugging them in as cycles of length 1 is very generic, but it offers an impractical size ~ amount ratio. Using bilocated digits and/or bivalued cells is a bit better, but still hard for human solvers.

As a rule of thumb, cycles with less than 4 (distinct) propositions aren't worth the effort it takes to deal with them. If there are numerous cycles of length 4+, it's hard to know where to begin (and investigating all combinations would be a time-consuming task). Fortunately, by the time Nested Cycle becomes worth considering, puzzles tend to have few simple yet interesting cycles left (about 3-4 cycles, and almost never more than 8).

Seeding cycles based on SK-Loop

Nested Cycle doesn't address the problem of finding seeding cycles; its focus is on how to extract progress from them after they've been found. As a curiosity, there happens to be a pattern that yields interesting cycles (almost) instantly.

A category I SK-Loop $\cdot S_8 \cdot P_1 \cdot S_1 \cdot P_2 \cdot S_2 \cdot \ldots \cdot S_7 \cdot P_8 \cdot S_8 \cdot$ has the properties that $\#S_i=\#P_i=2$ and that every P_i is solved by 1 digit of S_{i-1} and 1 digit of S_i.

Equivalently, 1 digit of S_i is solved by a cell of P_i, and the other digit is solved by a cell of P_{i+1}. Naming the 2 digits of S_i X_i and Y_i, this property corresponds to the following multi-celled cycles:

$$P_i=X_i \leftrightarrow P_i \neq Y_i \leftrightarrow P_{i+1}=Y_i \leftrightarrow P_{i+1} \neq X_i$$

Naming the 2 cells of P_i q_i and r_i, and considering only those i for which X_i and Y_i are bilocated in resp. q_iq_{i+1} and r_ir_{i+1} (which is very common for the pivotal boxes of the SK-Loop), the following single-celled cycles are available:

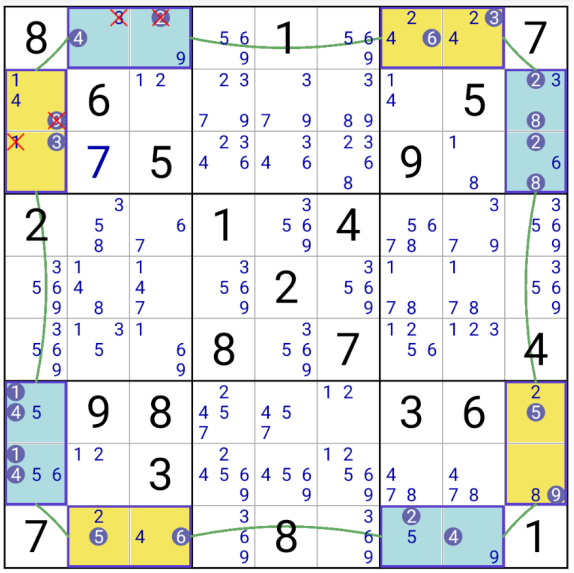
$$q_i=X_i \leftrightarrow q_{i+1} \neq X_i \leftrightarrow r_{i+1}=Y_i \leftrightarrow r_i \neq Y_i$$

This means that if an SK-Loop is present and determined to belong to category I[21], it can be used to provide up to 4 cycles of length 4 as input to Nested Cycle.

[21] It suffices to build 2 falsity chains that start with plentiful (16) conjuncts, with the aim of proving 1) $P_i \neq S_i$ and 2) $P_i \neq S_{i+1}$.

Examples

1 cycle, seeded by SK-Loop



800010007060000050075000900200104000000020000000807004098000360003000000700080001

This puzzle contains the SK-Loop:

·14· r23c1 ·39· r1c23 ·24· r1c78 ·36· r23c9 ·28· r78c9 ·59· r9c78 ·24· r9c23 ·56· r78c1 ·14·

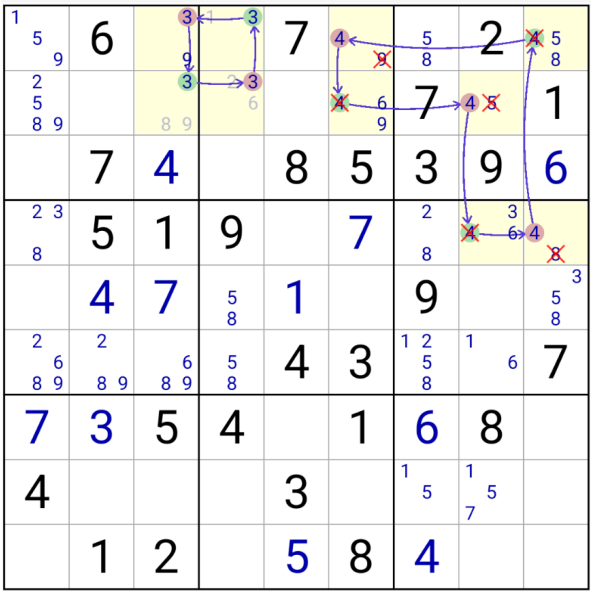
It can't the true that $P_i=S_i$ for all i, otherwise r2c1=9 ∧ r8c9=9, hence r5c19≠9 ∧ r28c5≠9, hence r5c46=9 ∧ r46c5=9 (which is impossible). Similarly, it can't the true that $P_i=S_{i+1}$ for all i (as that results in the contradiction r5c46=6 ∧ r46c5=6). Hence some P_i is solved by a digit of S_i and of S_{i+1}. It's easy to prove that this must then be true not just for any but for every i (see the analysis of SK-Loop).

Hence this SK-Loop is a category 1 SK-Loop that offers 4 cycles, 1 of which is:

r3c1=3 ↔ r1c2≠3 ↔ r1c3=9 ↔ r2c1≠9

This cycle (or rather set of 1 cycle) is trivial to inflate: every proposition must be true, otherwise they'd all be false, hence r3c1≠3 ∧ r1c3≠9, hence r3c1=1 ∧ r1c3=2, hence r2c3≠12 (which is impossible).

2 cycles



060070020000000701074085396051907000047010900000043007735401680400030000012058400

This puzzle contains a few simple cycles, 2 of which are:

- r1c6≠4 ↔ r2c6=4 ↔ r2c8≠4 ↔ r4c8=4 ↔ r4c9≠4 ↔ r1c9=4
- r1c3≠3 ↔ r2c3=3 ↔ r2c4≠3 ↔ r1c4=3

The number of propositions available (10) makes it easy to inflate those cycles:

- ⟪r1c6=4 ↔ r2c6≠4 ↔ r2c8=4 ↔ r4c8≠4 ↔ r4c9=4 ↔ r1c9≠4⟫ ∨ ¬⟪r1c3≠3 ↔ r2c3=3 ↔ r2c4≠3 ↔ r1c4=3⟫
  Proof: suppose that's false; then r1c6≠4 ∧ r1c3≠3, hence r1c6=9 ∧ r1c3=9, which is impossible.
- ⟪r1c6=4 ↔ r2c6≠4 ↔ r2c8=4 ↔ r4c8≠4 ↔ r4c9=4 ↔ r1c9≠4⟫ ∨ ⟪r1c3≠3 ↔ r2c3=3 ↔ r2c4≠3 ↔ r1c4=3⟫
  Proof: suppose that's false; then r2c6=4 ∧ r4c8=4 ∧ r2c4=3 (eliminating candidate cells of digit 6 in column 5 and row 4), hence r2c5=6 ∧ r4c5=6, which is impossible.

Deflation is even simpler and results in:

- ⟪r1c6=4 ↔ r2c6≠4 ↔ r2c8=4 ↔ r4c8≠4 ↔ r4c9=4 ↔ r1c9≠4⟫

Appendix (some deferred proofs)

Anti-Pointer

It’s never needed to consider sets P larger than 3 cells in Anti-pointer.

The applications only deal with Anti-Pointers for which P and Q have the same size and no other cell in the component of Q supports a candidate digit of P. For the sake of brevity, let's denote those Anti-Pointers P ↣ Q, the "other cells in the component of P and Q" P' and Q', and "the candidate digits of some set of cells S" C(S).

Now consider P ↣ Q, where P and Q consists of N cells. P' and Q' both consist of 6-N cells, so also have equal size. Furthermore, Q' supports no digits of C(P), hence P' contains the only cells of its component that support C(Q'). In other words, if P ↣ Q is valid then so is Q' ↣ P'. P and Q' can't both consist of more than 3 cells (otherwise 6 = #P+#P' = #P+#Q' > 6), so the smallest of them uses at most 3 cells.

That takes care of the boring bit. The interesting bit left to prove is that if P ↣ Q offers an elimination, then so does Q' ↣ P' (and vice versa, but that trivially follows on account of symmetry).

Case 1: suppose A2 in P ↣ Q eliminates x from p', for some p' in P' and x in C(P). Then x is a digit of C(P') (because x can be eliminated from p'). Also, x is not a digit in C(Q') (because Q contains every cell of its component that supports x). Hence A1 in Q' ↣ P' also establishes p'≠x

Case 2: suppose A1 in P ↣ Q eliminates y from q, for some q in Q and y in C(Q) but not C(P). It's safe to assume that y is in C(P'), as otherwise y is neither in C(P) nor in C(P'), and so y would be an ordinary Pointer in the intersection of the box and line. If indeed it was overlooked earlier, it can be used to eliminate y from q (rendering case 2 inapplicable).

Subcase 2a: If y is not in C(Q'), then A1 of Q' ↣ P' eliminates y from P'.

Subcase 2b: If y is in C(Q'), then A2 of Q' ↣ P' eliminates y from q.