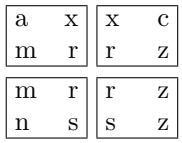
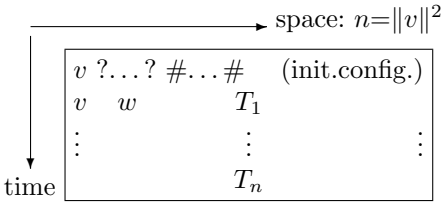
## 4.3 An NP-Complete Problem: Tiling

**Tiling Problem.** Invert the function which, given a tiled square, outputs its first row and the list of tiles used. A tile is one of the $26^4$ possible squares containing a Latin letter at each corner. Two tiles may be placed next to each other if the letters on the shared side match. (See an example at the right.) We now reduce to Tiling any search problem: given $x$, find $w$ satisfying a P-time computable property $F(x, w)$.



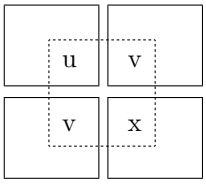

**Padding Argument.** First, we need to reduce it to some "standard" NP problem. An obvious candidate is the problem "Is there $w : U(v, w)$ ?", where $U$ is the universal Turing Machine, simulating $F(x, w)$ for $v = px$. But $U$ does not run in P-time, so we must restrict $U$ to $u$ which stops within some P-time limit. How to make this fixed degree limit sufficient to simulate any polynomial (even of higher degree) time? Let the TM $u(v, w)$ for $v$=$00 \dots 01px$ simulate $\|v\|^2$ steps of $U(px, w)$=$F(x, w)$. If the ***padding*** of 0's in $v$ is sufficiently long, $u$ will have enough time to simulate $F$, even though $u$ runs in quadratic time, while $F$'s time limit may be, say, cube (of a shorter "un-padded" string). So the NP problem $F(x, w)$ is reduced to $u(v, w)$ by mapping instances $x$ into $f(x) = 0 \dots 01px = v$, with $\|v\|$ determined by the time limit for $F$. Notice that program $p$ for $F$ is fixed. So, if some NP problem *cannot* be solved in P-time then neither can be the problem $\exists ?w : u(v, w)$. Equivalently, if the latter *IS* solvable in P-time then so is *any* search problem. We do not know which of these alternatives is true. It remains to reduce the search problem $u$ to Tiling.

**The Reduction.** We compute $u(v, w)$ (where $v = 00 \dots 01px$) by a TM represented as an array of 1-pointer cellular automata that runs for $\|v\|^2$ steps and stops if $w$ does NOT solve the relation $F$. Otherwise it enters an infinite loop. An instance $x$ has a solution iff $u(v, w)$ runs forever for some $w$ and $v = 0 \dots 01px$. Here is the space-time diagram of computation of $u(v, w)$. We set $n$ to $u$'s time (and space) $\|v\|^2$. Each row in this table represents the configuration of $u$ in the respective moment of time. The solution $w$ is filled in at the second step below a special symbol "?". If a table is filled in wrongly, i.e. doesn't reflect any actual computation, then it must have four cells sharing a corner that couldn't possibly appear in the computation of $u$ on any input.




**Proof.** As the input $v$ and the guessed solution $w$ are the same in both the right and the wrong tables, the first 2 rows agree. The actual computation starts on the third row. Obviously, in the first mismatching row a transition of some cell from the previous row is wrong. This is visible from the state in both rows of this cell and the cell it points to, resulting in an impossible combination of four cells sharing a corner.

For a given $x$, the existence of $w$ satisfying $F(x, w)$ is equivalent to the existence of a table with the prescribed first row, no halting state, and permissible patterns of each four adjacent squares (cells). Converting the table into the ***Tiling Problem***: The cells in the table are separated by "—" ; the tiles by "..."; Cut each cell into 4 parts by a vertical and a horizontal lines through its center and copy cell's content in each part. Combine into a tile each four parts sharing a corner of 4 cells. If these cells are permissible in the table, then so is the respective tile.




So, any P-time algorithm extending a given first row to the whole table of matching tiles from a given set could be used to solve any NP problem by converting it to Tiling as shown.

**Exercise:** Find a polynomial time algorithm for $n \times \log n$ Tiling Problem.