Lógica para Ciencias de la Computación
Matematicas Aplicadas y Ciencias de la Computación
Escuela de Ingeniería, Ciencia y Tecnología
Universidad del Rosario

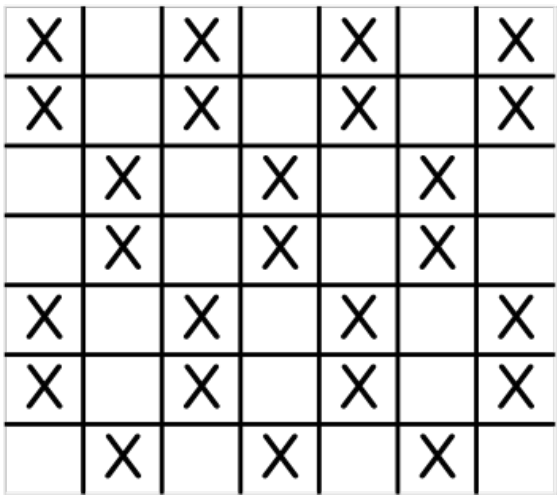
# 3 en Linea 7x7

Alvarado Ramos Juan Sebastián, Morera Torres Lukas,
Fonseca Lesmez Robert Daniel

## 1. Introducción

Este proyecto busca resolver la siguiente problemática: En un juego de "Tres en línea" se quiere ubicar cierto numero de *X* en un tablero 7x7, donde se pueden ubicar *X* de la manera que sea necesaria.

Un ejemplo de una posible solución al problema es el siguiente:



Como se ve, no hay 3 *X* consecutivas, y hay un mínimo de 3 *X* en cada línea y columna, las cuales son las reglas que se requieren cumplir para la correcta solucón de este problema.

## 2. Letras proposicionales

El proyecto se basa en una cuadrícula 7x7, es decir, con un total de 49 casillas. Por ende, se tiene un total de **49 letras proposicionales**, siendo generadas por el siguiente descriptor:

$$XenC_{x,y}$$

Siendo *X* el listado de filas enumeradas del 0 al 6, y *Y* el listado de columnas enumeradas del 0 al 6.

## 3. Reglas

La solución del problema debe satisfacer tres condiciones:

**1.** Debe haber una sola *X* por casilla

**2.** No debe haber 3 *X* seguidas en ninguna dirección. Esto incluye horizontal (→), vertical (↓), diagonal derecha (↘) y diagonal izquierda (↙)

**3.** Deben haber como mínimo 3 *X* por cada fila y columna, de modo que como mínimo deben haber 21 *X* en todo el tablero

## 4. Representación en fórmulas

**Regla 1:** Esta condición ya se ve satisfecha por el descriptor, ya que el mismo se encarga de solo poner una *X* por cada casilla:

$$XenC_{x,y}$$

**Regla 2:** En este caso fue necesario declarar inicialmente 4 fórmulas, siendo una para cada dirección, para finalmente unirlas con ANDS. La fórmula final es la siguiente:

$$\bigwedge_{\substack{x\in Filas\\ y\in\{0,4\}}}\Big(XenC_{x,y}\wedge XenC_{x,y+1}\rightarrow\neg\Big(XenC_{x,y+2}\Big)\Big)\wedge$$
$$\bigwedge_{\substack{x\in\{0,4\}\\ y\in Columnas}}\Big(XenC_{x,y}\wedge XenC_{x+1,y}\rightarrow\neg\Big(XenC_{x+2,y}\Big)\Big)\wedge$$
$$\bigwedge_{\substack{x\in\{0,4\}\\ y\in\{0,4\}}}\Big(XenC_{x,y}\wedge XenC_{x+1,y+1}\rightarrow\neg\Big(XenC_{x+2,y+2}\Big)\Big)\wedge$$
$$\bigwedge_{\substack{x\in\{0,4\}\\ y\in\{2,6\}}}\Big(XenC_{x,y}\wedge XenC_{x+1,y-1}\rightarrow\neg\Big(XenC_{x+2,y-2}\Big)\Big)$$

**Regla 3:** Por último, para esta condición se necesitó inicialmente 2 formulas, una para las filas y otra para las columnas, uniendo ambas con un AND. La fórmula final es la siguiente:

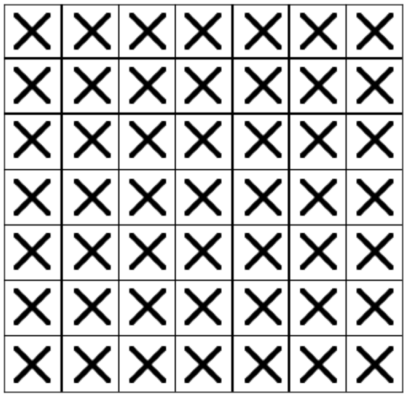
$$\bigwedge_{x\in Filas}\Big(\bigvee_{y\in Columnas}\Big(\bigvee_{\substack{w\in Columnas\\ w\neq y}}\Big(\bigvee_{\substack{z\in Columnas\\ z\neq y\\ z\neq w}}\Big(XenC_{x,y}\wedge XenC_{x,w}\wedge XenC_{x,z}\Big)\Big)\Big)\Big)\wedge$$
$$\bigwedge_{y\in Columnas}\Big(\bigvee_{x\in Filas}\Big(\bigvee_{\substack{w\in Filas\\ w\neq x}}\Big(\bigvee_{\substack{z\in Filas\\ z\neq x\\ z\neq w}}\Big(XenC_{x,y}\wedge XenC_{w,y}\wedge XenC_{z,y}\Big)\Big)\Big)\Big)$$

De esta manera se unieron las reglas con ANDS, representando la formula a solucionar en este proyecto, la cual de encontrarse un modelo permitira pintar mínimo 3 *X* en cada fila y columna, sin quedar 3 seguidas.

## 5. Visualización

Para la correcta visualización del problema en Jupyter, se empleo la libreria **matplotlib**, la cual permite hacer gráficas y cuadrículas en Python. Así, una cuadrícula 7x7 con las 49 letras verdaderas se ve de la siguiente manera:

```
I = {n.XenC.P([x,y]):True for x in range(7) for y in range (7)}
n.visualizar(I)
```



## 6. Resultados

Para llegar a la solución del proyecto se probaron los siguientes Solvers:

**1. SATtabla**

Este Solver no fue capaz de resolver el problema, ya que despues de un tiempo de ejecutarse, tira un *Memory Error*, lo que indica que este Solver no está capacitado para resolver el problema.

**2. SATtableaux**

Si bien este Solver puede correr el problema sin tirar errores como en SATtabla, el mismo toma demasiado tiempo para encontrar una solución al problema. Esto se sustenta en que se dejó ejecutando por mas de 2 horas y seguia sin encontrar un modelo. Por ende, es un Solver que no es viable para resolver el problema

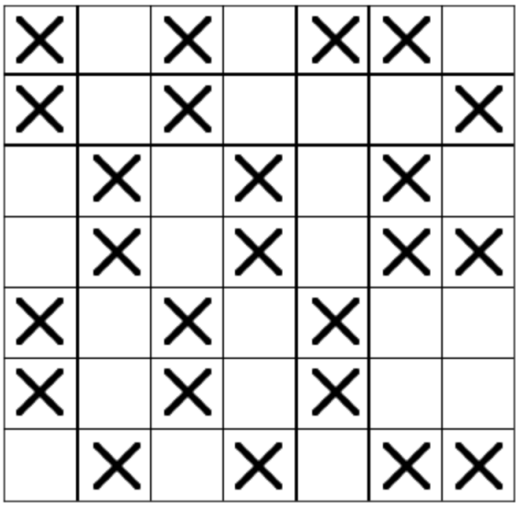
**3. DPLL**

Este fue el primer Solver que pudo arrojar un modelo de la fórmula, viendose en capacidad de resolver cada una por separado y finalmente todo el problema. Sin embargo, para que pudiese solucionarlo en un tiempo viable, fue necesario poner un total de **10 condiciones iniciales**, lo cual permite que DPLL pueda resolver en un buen tiempo. **Tiempo de resolución:** Entre 2 y 15 minutos, dependiendo de los casos que tome.

```
%%time
A = Ytoria(n.reglas)
S = tseitin(A)
S, I = dpll(S, {})
```

```
CPU times: total: 4min 19s
Wall time: 4min 19s
```

```
if I!={}: n.visualizar(I)
else: print("No hay solución!")
```



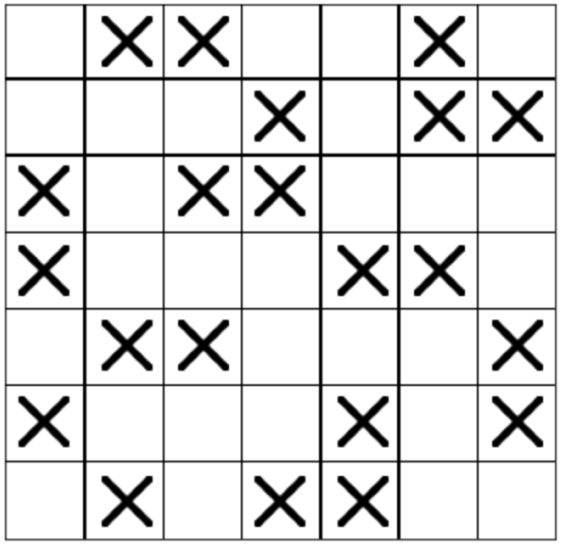
### 4. WalkSAT

Si bien este Solver es viable para resolver el problema, depende bastante de la interpretación y literales que tome en cuenta para su resolución. En nuestro caso, WalkSAT no pudo resolver el problema en un tiempo prolongado de ejecución. Es por esto que consideramos que no es un Solver viable para resolver este problema.

### 5. Minisat22

Este Solver originado de la libreria *Pysat* fue el que mejores resultados arrojó a la hora de resolver el proyecto, ya que encontró el modelo de una manera bastante rápida y **sin necesidad de condiciones iniciales**, lo cual no puede hacer DPLL. Por ello es el mejor Solver para este caso particular. **Tiempo de resolución:** Entre 5 y 6 segundos.

```
%%time
n.visualizar(SATsolver(n.rules))
```



```
CPU times: total: 5.95 s
Wall time: 6 s
```

## 7. Conclusiones

Como se pudo ver, este proyecto cuenta con un tamaño considerable, esto debido a las 49 letras prosicionales posibles gracias a las dimensiones del tablero. Por lo anterior las fórmulas de cada regla arrojaban diversas posibilidades, por lo que se requería un Solver bastante eficiente y abierto a trabajar con formulas grandes para llegar a un modelo para este problema. Es por lo anterior que Solvers ineficientes como SATtabla no funcionaban. Asi mismo, Solvers no tan rapidos como SATtableaux o en muchos casos WalkSAT, tardan mucho tiempo en encontrar la solución, haciendolos no viables. Por tanto, DPLL y Minisat22 son los mejores Solvers para este caso, ya que pueden encontrar solución y no tardan tanto tiempo en encontrarla.