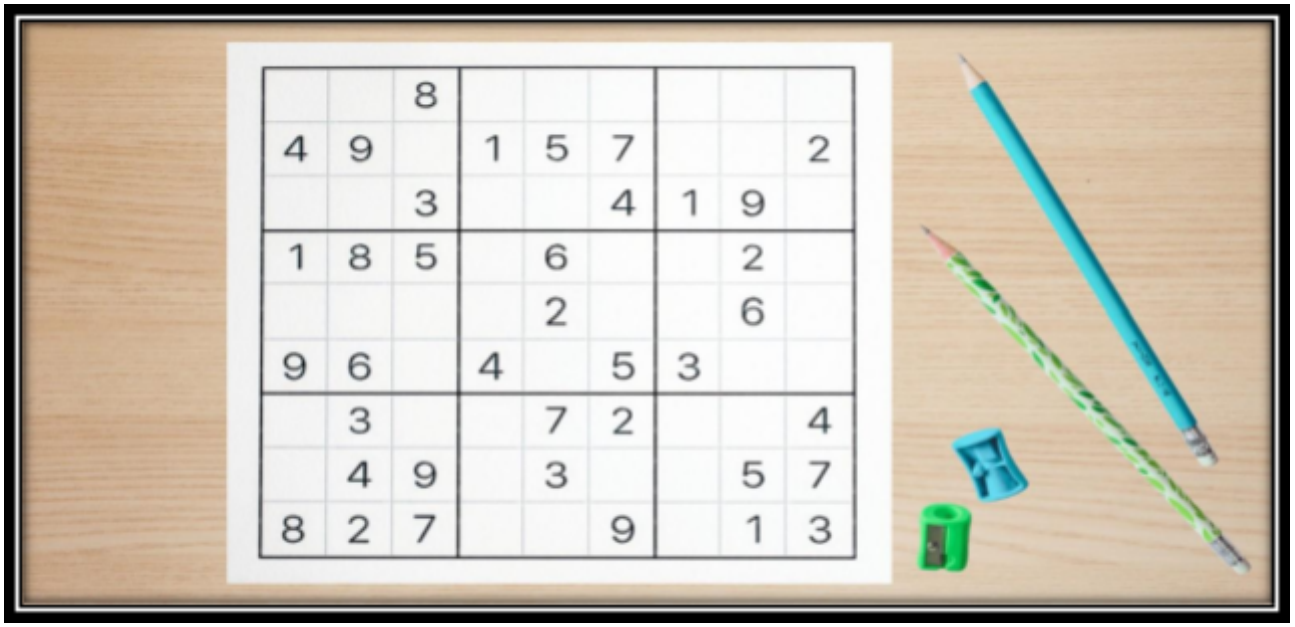
Progetto di Elementi di Intelligenza Artificiale

Per il professor Sperlì Giancarlo

di Eugenio Iandoli (N46070067), Gaetano Emanuele Izzo (N46007081), Aniello Gaudino (N4600)

SUDOKU



Problema

Il Sudoku è uno dei puzzle logici più popolari al mondo, nato in Giappone negli anni '80 ma con radici che risalgono ai quadrati latini di Eulero del XVIII secolo. Il gioco si presenta come una griglia 9×9 suddivisa in 9 sottogriglie 3×3, dove l'obiettivo è riempire tutte le celle con numeri da 1 a 9, rispettando tre vincoli fondamentali:

1. **Vincolo di riga**: ogni riga deve contenere tutti i numeri da 1 a 9, senza ripetizioni
2. **Vincolo di colonna**: ogni colonna deve contenere tutti i numeri da 1 a 9, senza ripetizioni
3. **Vincolo di blocco**: ogni sottogriglia 3×3 deve contenere tutti i numeri da 1 a 9, senza ripetizioni

La sfida del Sudoku risiede nell'applicazione simultanea di questi vincoli per dedurre logicamente i valori delle celle vuote. A differenza di altri puzzle, il Sudoku non richiede

conoscenze matematiche avanzate, ma si basa puramente su logica deduttiva e ragionamento per vincoli.

Il fascino di questo gioco sta nella sua semplicità di regole unite alla complessità delle strategie risolutive, che vanno dalle tecniche base come "naked singles" e "hidden singles" fino a strategie avanzate come "X-wing" e "swordfish". Questo lo rende un candidato ideale per l'implementazione in Prolog, linguaggio naturalmente orientato alla risoluzione di problemi basati su vincoli.

Modalità di implementazione

La scelta di implementare il Sudoku in Prolog (Programming in Logic) è stata naturale e motivata da diverse considerazioni tecniche. Prolog è un linguaggio di programmazione logica basato su predicati, fatto e regole, che eccelle nella risoluzione di problemi di soddisfacimento di vincoli (CSP - Constraint Satisfaction Problems).

Il Sudoku è per sua natura un CSP: abbiamo un insieme di variabili (le 81 celle della griglia), un dominio per ciascuna variabile (numeri da 1 a 9), e un insieme di vincoli che devono essere simultaneamente soddisfatti. Prolog, attraverso la libreria **clpfd** (Constraint Logic Programming over Finite Domains), fornisce strumenti potenti per modellare e risolvere esattamente questo tipo di problemi.

Knowledge Base (KB), Regole di Inferenza e Motore Inferenziale

La **Knowledge Base (KB)** è una raccolta strutturata di informazioni, fatti e regole che rappresentano la conoscenza di un dominio specifico. In un sistema basato su regole, come quelli implementati in Prolog, la KB contiene tutti i dati e le relazioni necessarie per risolvere problemi o rispondere a domande all'interno di quel dominio. Nel contesto del Sudoku, la KB è rappresentata dalla griglia del gioco, dove ogni cella può contenere un valore numerico o essere vuota, e dai vincoli che definiscono le regole del gioco.

Nel nostro progetto, la KB del sistema è costituita dai fatti e dalle regole che descrivono il dominio del Sudoku. In particolare, la KB è rappresentata dalla griglia del Sudoku, dove ogni cella contiene un valore o è vuota. I vincoli del Sudoku,

come l'unicità dei numeri in ogni riga, colonna e blocco 3x3, sono rappresentati come regole nella KB.

Regole di Inferenza

Le **regole di inferenza** sono meccanismi che permettono di dedurre nuove informazioni dalla KB. Queste regole sono solitamente espresse come implicazioni logiche che definiscono come nuove conoscenze possono essere derivate dai fatti esistenti. Nel nostro progetto Sudoku, le regole di inferenza sono utilizzate per determinare se una mossa è valida, generare la griglia del gioco e verificare la correttezza delle soluzioni.

Nel nostro progetto, le regole di inferenza sono rappresentate dalle procedure che controllano la validità delle mosse e generano la griglia del Sudoku. Ad esempio, la regola `valid_move/4` è una regola di inferenza che verifica se un numero può essere inserito in una specifica cella della griglia senza violare i vincoli del Sudoku:

```prolog
valid_move(Grid, R, C, V) :-
   between(1, 9, R), between(1, 9, C), between(1, 9, V),
   nth1(R, Grid, Row), nth1(C, Row, 0),
   \+ member(V, Row),
   transpose(Grid, Cols), nth1(C, Cols, Col),
   \+ member(V, Col),
   block_coordinates(R, C, BR, BC),
   get_block(Grid, BR, BC, Block),
   \+ member(V, Block).
```

Questa regola deduce che una mossa è valida se il valore `V` non è già presente nella riga, colonna o blocco 3x3 corrispondente.

Motore Inferenziale

Il **motore inferenziale** è il componente di un sistema basato su regole che applica le regole di inferenza alla KB per dedurre nuove informazioni o prendere decisioni. In altre parole, il motore inferenziale utilizza le regole definite per manipolare e interrogare la KB al fine di risolvere problemi o rispondere a query.

Nel nostro progetto, il motore inferenziale è responsabile dell'applicazione delle regole di inferenza alla KB per dedurre nuove informazioni. Il motore inferenziale è rappresentato dalle procedure che generano la griglia del Sudoku e validano le mosse dell'utente. Ad esempio, la procedura `generate_grid/1` utilizza il modulo `library(clpfd)` di Prolog per generare una griglia valida:

```prolog
generate_grid(NumRemove, Puzzle) :-
   length(Rows, 9), maplist(same_length(Rows), Rows),
   append(Rows, Vs), Vs ins 1..9,
   maplist(all_distinct, Rows),
   transpose(Rows, Cols), maplist(all_distinct, Cols),
   Rows = [R1, R2, R3, R4, R5, R6, R7, R8, R9],
   blocks(R1, R2, R3), blocks(R4, R5, R6), blocks(R7, R8, R9),
   label(Vs),
```

...
```

Questa procedura deduce una griglia valida utilizzando vincoli logici per assicurarsi che ogni riga, colonna e blocco 3x3 contenga numeri univoci da 1 a 9. Nel progetto Sudoku, il motore inferenziale è responsabile della generazione della griglia, della validazione delle mosse dell'utente e della risoluzione del puzzle.

## Migliorie implementare e possibili migliorie future

Le principali estensioni che abbiamo identificato includono:
```

Livelli di Difficoltà

- **Principiante**: 30-35 celle rimosse, pattern simmetrici
- **Intermedio**: 40-45 celle rimosse, tecniche logiche standard
- **Esperto**: 50+ celle rimosse, richiedono tecniche avanzate

Interfaccia Grafica

Sviluppo di una GUI utilizzando:

- **XPCE** (libreria grafica di SWI-Prolog)
- **Highlight** delle celle correlate quando si seleziona una posizione
- **Validazione visuale** in tempo reale con evidenziazione degli errori

Funzionalità Avanzate (con cui poter migliorare il programma in futuro)

- **Sistema di hint** che suggerisce mosse basate su tecniche logiche standard
- **Salvataggio/caricamento** delle partite in corso
- **Statistiche di gioco** (tempo impiegato, numero di errori, tecniche utilizzate)
- **Modalità competitiva** con classifiche e sfide temporizzate

Per questi motivi, Prolog si dimostra un linguaggio estremamente adatto per modellare e risolvere il Sudoku, valorizzando la natura puramente logica del problema e facilitando l'esplorazione sistematica dello spazio delle soluzioni.

Lista comandi di gioco

Abbiamo deciso di creare un'applicazione con il minimo dei comandi, per non rendere difficile l'esperienza di gioco all'utente.

Difatti, l'unico vero e proprio comando è :

1) **play.** per iniziare una partita.
   In seguito al comando di play verrà richiesto all'utente di selezionare una difficoltà: 1. (Principiante) , 2. (Intermedio), 3. (Esperto).

   Dopo la sezione della difficoltà verrà generato e mostrato a schermo la griglia
   del sudoku all'utente, ed un menù numerato con il quale effettuare le scelte.

Risultati

Il sistema implementato dimostra l'efficacia di Prolog nella gestione di problemi basati su vincoli. Di seguito alcuni esempi rappresentativi dell'utilizzo del programma:

Esempio 1: Partita Completata con Successo

```
Benvenuto al Sudoku!
Generazione griglia in
corso... Ecco il puzzle:
    1 2 3   4 5 6   7 8 9
  +-------+-------+-------+
1 | 5 3 . | . 7 . | . . . |
  |       |       |       |
2 | 6 . . | 1 9 5 | . . . |
  |       |       |       |
3 | . 9 8 | . . . | . 6 . |
  +-------+-------+-------+
...

Tentativi rimanenti: 3
=== MENU SUDOKU ===
1. Inserisci numero
2. Mostra griglia
3. Esci
Scelta: 1
Riga (1-9): 1
Colonna (1-9): 3
Valore (1-9): 4
Inserimento
completato.

Complimenti, Sudoku completato!
```

Esempio 2: Gestione degli Errori

```
Scelta: 1
Riga (1-9): 1
Colonna (1-9): 4
Valore (1-9): 5
```

```
Mossa non valida!

Tentativi rimanenti: 2
```

Il sistema dimostra efficacemente la rilevazione di violazioni dei vincoli, penalizzando appropriatamente l'utente per mosse non valide.

Esempio 3: Validazione Input

```
Scelta: 1
Riga (1-9): 12
Colonna (1-9): 5
Valori fuori dal
range!

Tentativi rimanenti: 3
```

La robustezza del sistema nella gestione di input non validi garantisce un'esperienza utente fluida.