Assignment 03 – "Prolog"

Federico Naldini, matr: 0000852918, email: federico.naldini3@studio.unibo.it
repo: https://github.com/Tirocigno/pps-17-asg03-prolog

05/12/2018

1 Description of the code

L'elaborato che ho scelto di realizzare consiste in una semplice implementazione di *Game of Life*, automa cellulare celebre in letteratura, le cui regole sono descritte alla seguente pagina. Per realizzare il progetto ho scelto di adottare un pattern *MVC*, dove per la parte di *view* e *controller* ho utilizzato i linguaggi Scala e Java, mentre per quanto riguarda il *model* ho cercato di gestire tutta la complessità del gioco utilizzando esclusivamente la programmazione logica.
Il focus dell'assignment è stato nel realizzare la soluzione a un problema ben noto, che ho avuto modo di trattare utilizzando diversi paradigmi di programmazione, applicando un paradigma totalmente differente dai precedenti, cercando di sfruttarne al meglio le potenzialità e di inquadrarne i difetti.

2 Techniques used

- **Minimalità della soluzione proposta:** In linea con la natura coincisa di Prolog, ho cercato di limitare al massimo le dimensioni del codice: grazie alle possibilità offerte dalla programmazione logica, ho limitato gli elementi di cui occorre mantenere lo stato a:
  - Le dimensioni della scacchiera di gioco.
  - Una lista contenente la posizione delle celle in vita durante la generazione corrente.
  - Il numero di generazioni trascorse.
- **Predicati Tail-Recursion:** Data la necessità per alcune regole di compiere operazioni di natura ciclica, ho scelto di realizzare tali regole e i predicati che le compongono come *tail-recursive*, in modo da massimizzare l'efficienza di tale computazione. `generate_board` e `compute_board` sono due regole realizzate seguendo quanto detto sopra.
- **Integrazione Java-Prolog:** Avendo scelto di mettere a disposizione un'interfaccia grafica scritta in Java al programma, è stato necessario integrare in qualche modo la parte di codice scritta in Prolog con le librerie Java. Per realizzare questa integrazione, ho deciso di sfruttare le API messe a disposizione dalla libreria `alice.tuprolog`, riuscendo così a ottenere l'integrazione della parte di computazione logica con il resto del codice; unica problematica non banale nella sua risoluzione è stata la conversione delle liste Prolog in Java, risolta tramite una combinazione di primitive messe a disposizione dalla libreria per scorrere gli elementi delle liste e codice appositamente scritto per effettuare la conversione del singolo elemento.
- **Performance:** Dovendo procedere a una computazione non banale per un elevato numero di celle ad ogni generazione, mi sono scontrato più volte col problema delle performance, trovandomi a dover pensare soluzioni che tenessero conto della complessità computazionale dei predicati presenti nei corpi delle varie regole. Un esempio è sicuramente il predicato `append_to_head`, che inserisce un elemento in testa a una lista, utilizzato al posto delle classiche regole di inserimento che prevedono l'aggiunta del nuovo elemento in coda: non essendo rilevante l'ordine all'interno della lista delle celle in vita, si riduce notevolmente la complessità computazionale dell'inserimento da *O(n)* a un tempo costante.

3 Self-evaluation

Personalmente credo di aver lavorato in maniera discreta per questo assignment, non ho voluto porre il focus dell'elaborato su un problema che reputavo particolarmente adatto alle potenzialità della programmazione logica, quanto più mi sono concentrato nel risolvere un problema già noto e risolto con

vari paradigmi di programmazione, ponendo l'accento sulle differenze tra la soluzione delineata con la programmazione logica e le altre già realizzate.
Sicuramente questa scelta mi ha impedito di sperimentare appieno tutte le potenzialità della programmazione logica, ma mi ha anche spinto a ragionare e progettare secondo le meccaniche di tale paradigma di programmazione.