

Progetto di Ragionamento Automatico

Elia Piacentini - VR448249

23 gennaio 2020

Indice

1	Introduzione	2
1.1	Consegna del progetto	2
1.2	Scelta del linguaggio per lo sviluppo	2
2	Analisi del solutore	3
2.1	L'idea alla base	3
2.2	Il solutore	3

Capitolo 1

Introduzione

1.1 Consegna del progetto

Implementare in un linguaggio di programmazione a scelta la procedura di decisione, basata su chiusura di congruenza, della soddisfacibilità di un insieme di letterali nel frammento senza quantificatori dell'unione delle teorie dell'uguaglianza, delle liste non vuote possibilmente cicliche, e degli array senza estensionalità. Si implementi l'algoritmo di chiusura di congruenza su grafo diretto aciclico, considerando le varianti viste in classe (e.g., lista proibita, scelta non arbitraria del rappresentante della classe generata da union, aggiornamento del campo find di tutti i nodi di una classe).

1.2 Scelta del linguaggio per lo sviluppo

Ottenuta la consegna, il primo dubbio da risolvere era relativo al linguaggio in cui implementare il solutore; l'unico requisito per il linguaggio era che fosse eseguibile su Linux. Degli aspetti da non sottovalutare erano la velocità di sviluppo (visto il tempo a disposizione) e la velocità di esecuzione (che sarebbe potuta essere un parametro di valutazione). I linguaggi presi in considerazione erano tre:

- Java
- Python
- Dart

JAVA, il linguaggio che conosco meglio, è abbastanza veloce in esecuzione ma avrei dovuto spendere troppo tempo per strutturare il codice. Dart, simile a JAVA come prestazioni, avrebbe potuto rappresentare una scelta interessante: la compatibilità sarebbe stata garantita tramite un'interfaccia web e questo mi avrebbe anche agevolato nel creare una UI; purtroppo la mia conoscenza di Dart era troppo bassa per cimentarmi in un progetto come questo. Python non è certo il linguaggio più veloce, ma sicuramente è uno dei più versatili. La velocità di sviluppo e di debug sono alte ed esistono parecchie librerie per gestire la parte grafica. Ho quindi scelto di sviluppare il progetto in Python, nello specifico usando la versione 3.7.

Capitolo 2

Analisi del solutore

2.1 L'idea alla base

Il progetto, fin dall'inizio, è stato pensato diviso in due parti: il solutore ed il parser. Prima di analizzare il solutore, è necessario soffermarsi sulla struttura base dell'algoritmo di congruence closure: il *Node*. La classe `Node.py` definisce il mattone fondamentale dell'algoritmo; i principali elementi della classe sono:

- `int id`: id numerico del nodo
- `String fn`: simbolo della funzione del nodo
- `List<int> args`: id dei nodi argomenti
- `int find`: id del rappresentante della classe a cui appartiene il nodo
- `Set<int> ccpair`: id dei nodi genitori

I nodi vengono creati durante la fase di preprocessing dal parser: in questo modo, il solutore deve preoccuparsi solamente di eseguire la chiusura di congruenza tra gli elementi.

2.2 Il solutore

Il solutore consiste in una classe, `Graph.py`, contenente i metodi necessari all'implementazione della chiusura di congruenza; i metodi presenti sono:

Metodo	Descrizione
node (int id): Node n	Restituisce un nodo dato l'id.
find (int id): int id	Restituisce il find del nodo: se questo è diverso da se stesso, chiama la funzione find sul nuovo id.
ccpar (int id): Set<int> id	Restituisce gli id dei nodi genitori del rappresentante della classe.
congruent (int id_1, int id_2): boolean result	Confronta due nodi e restituisce true se sono congruenti (simbolo di funzione ed argomenti uguali), false altrimenti.
union (int id_1, int id_2): void	Unisce due classi di congruenza, accorpando i ccpar di un nodo nell'altro e cambiando il riferimento del find del nodo svuotato verso l'altro nodo.
merge (int id_1, int id_2): void	Metodo che permette di lanciare la union sui termini che devono essere messi nella stessa classe di congruenza.

- **node**(int id):Node n restituisce un nodo dato l'id.
- **find**: restituisce il find del nodo