1 INTRODUZIONE

Il presente lavoro di tesi nasce a seguito di un periodo di tirocinio eseguito presso i laboratori dell' università. In particolare il progetto mette le sue radici in un ambito più vasto di quello strettamente informatico, ovvero quello della biologia. Alla base del lavoro c' è la teoria riguardante lo *splicing*, ossia un tipo di comportamento ricombinante operato su molecole di DNA [2]. Ispirati da questo modello, sono stati sviluppati da alcuni colleghi una serie di algoritmi che operano appunto sullo splicing [1], e lo scopo del lavoro di tirocinio e quindi di tesi è stato quello di integrare tali algoritmi in un software esistente e gratuito, che va sotto il nome di *JFLAP*.

PROBLEMATICHE

JFLAP [3] è un software open source nato da un' idea di Susan H. Rodger, docente presso la *Duke University*; in particolare il programma si rende utile per una varietà di problematiche, come per esempio la creazione e il

test di grammatiche, automi, macchine di Turing, ecc. Nel corso degli anni il programma ha subito una serie considerevole di modifiche e cambiamenti da parte di diversi programmatori, tuttavia una cosa è rimasta costante, il linguaggio utilizzato per scrivere il programma, ovvero *Java* [4]. Si è pensato quindi di operare sulla struttura originale di JFLAP al fine di modificarlo e di estenderne le funzionalità, aggiungendo una sezione per la gestione delle parole circolari e la simulazione del processo di splicing. Si è trattato di un vero e proprio lavoro di integrazione di algoritmi atti allo scopo e sviluppati in precedenza da altri colleghi e di conseguenza della creazione di un' interfaccia grafica per
l' utente che rispettasse lo stile originale di JFLAP.

SOLUZIONI

Il lavoro svolto è stato articolato in varie "tappe". Anzitutto è stato necessario l' utilizzo di un ambiente di sviluppo Java e la scelta è ricaduta su *Eclipse* [5], un tool dal facile utilizzo, ma allo stesso tempo potente e gratuito. Per prima cosa, grazie all' ausilio di alcuni manuali, si è studiato il funzionamento di JFLAP; a ciò è seguita un' analisi approfondita della sua struttura in termini di codice sorgente Java, con lo scopo di comprendere la fattibilità dell' integrazione. In seguito si sono studiati dei vari package e delle varie classi per posizionare le classi create dai miei colleghi nei punti giusti della struttura del programma. JFLAP infatti è organizzato in modo ordinato, ogni package gestisce una sezione del programma, occupandosi del corretto funzionamento e della gestione delle interfacce grafiche per l' utente.
Prima di procedere all' inserimento vero e proprio di tali classi si è resa necessaria un' ulteriore operazione: sviluppato in America, JFLAP è completamente in lingua inglese, pertanto tutti i commenti, i nomi delle classi e le intestazioni di classi precedentemente sviluppati in lingua italiana sono stati tradotti per rendere al meglio l' integrazione.

Una volta effettuato l' inserimento di tali classi, si è passati al secondo passo: l' analisi del meccanismo di chiamate sequenziali tra classi che provoca il corretto avvio del programma e la conseguente modifica della struttura originale di tale meccanismo al fine di poter ospitare la nostra nuova sezione per la gestione delle parole circolari e dello splicing.
A questo punto ci si è trovati di fronte al passo più importante, ovvero lo sviluppo di un' interfaccia grafica per l' utente (*gui*) che rispettasse lo stile ed il funzionamento originali di JFLAP. Per fare ciò si sono utilizzate le classi *awt* e *swing* di Java per creare un' interfaccia semplice e funzionale, discostandomi dalla linea originale solo per l' inserimento di alcuni brevi dialog box che spiegano all' utente il funzionamento di ogni bottone presente nella finestra del programma. Questo perchè una delle pecche di JFLAP è la mancanza di un *help* rapido e consultabile off-line.
Come si potrà vedere in modo approfondito nella tesi, si è pensato di dividere l' interfaccia in tre sezioni principali, che si occupano rispettivamente dell' inserimento e della gestione delle parole circolari, dell' inserimento e della gestione delle regole da applicare a tali parole durante lo splicing ed infine un' ultima sezione responsabile dello splicing vero e proprio: sarà l' utente stesso a decidere il numero di iterazioni che il programma cercherà di eseguire. Ovviamente tutto questo meccanismo è sottoposto al controllo dei possibili errori (*eccezioni*): infatti le finestre di errore avvertiranno l' utente che qualcosa è stato sbagliato; come l'inserimento di una regola con un formato scorretto,oppure l' errato inserimento di parole o numero di iterazioni, ecc.
Per completare il tutto è stato messo a disposizione dell' utente un bottone che consente la copia dei risultati testuali dello splicing negli appunti (*clipboard*) del sistema operativo.
L' ultimo, ma non meno importante, passo dell' integrazione è stato il testing del programma per valutarne la funzionalità e la stabilità ed evidenziare eventuali problemi.

La fase di test ha dato esito positivo: il programma infatti risponde correttamente ai comandi impartiti dall' utente, gli errori vengono prontamente segnalati ed i risultati dello splicing sono quelli attesi.
L' unico problema riscontrato riguarda la complessità in termini computazionali del meccanismo di splicing che si basa sull' utilizzo delle liste: questo comporta un rallentamento progressivo del software all' aumentare della lunghezza delle parole e delle iterazioni fino ad arrivare ad un suo totale blocco. Possibili utilizzi futuri di questa tesi potrebbero svilupparsi in tale senso, o comunque partendo da questa tesi potranno essere apportate più facilmente ulteriori modifiche a JFLAP.

ORGANIZZAZIONE DELLA TESI

La tesi si articola in 4 capitoli: oltre a questa introduzione il capitolo 2 analizza il contesto biologico in cui si sviluppa il progetto, mostrando una serie di definizioni teoriche e formali dello splicing ed infine l' implementazione in Java di alcuni sistemi di splicing circolare. Il capitolo 3 presenta JFLAP, analizzandone il progetto, la struttura e l' utilizzo. Infine il capitolo 4 illustra il processo di integrazione, il codice sorgente sviluppato e quello modificato ed i risultati del testing.