PROGETTO D'ESAME PER LA PARTE DI ASP

IALab A.A. 19/20

Amedeo Racanati 928995

Angelo Pio Sansonetti 928869

Introduzione

Qui di seguito segue una breve descrizione del modus operandi adottato per la progettazione del software in ASP.

All'inizio abbiamo deciso di formalizzare in modo più rigoroso le specifiche fornite dalla traccia del progetto, in maniera tale da individuare i principali predicati di interesse e i vincoli da codificare in seguito nel sistema.

Dopo la formalizzazione siamo passati alla codifica del problema, procedendo per passi incrementali. Durante le prime fasi non ci siamo preoccupati di definire l'insieme di tutti e 26 i corsi, piuttosto abbiamo definito soltanto un numero esiguo di corsi e con quelli abbiamo portato avanti la codifica dei vincoli; anche per le settimane, inizialmente, ne abbiamo stabilite relativamente poche. Ciò ci ha permesso di debuggare più efficientemente i vincoli, visto che da un lato abbiamo ridotto i tempi di attesa per la soluzione (dato che la risoluzione di un problema ridotto impiega solitamente pochi secondi), dall'altro lato è stato più facile controllare che la codifica dei vincoli stessi fosse corretta.

Dopo aver definito tutti i vincoli rigidi e aver esteso il calendario alle 24 settimane prestabilite, abbiamo introdotto gradualmente i corsi e abbiamo valutato le performance di risoluzione. Abbiamo notato che la procedura di grounding effettuata da *gringo* portava ad un utilizzo della memoria spropositato, nonché a tempi di risoluzione troppo elevati. Abbiamo analizzato l'output di gringo notando che determinati vincoli definiti in ASP portavano a generare un enorme numero di vincoli ground. Pertanto abbiamo lavorato in maniera tale da ridurre drasticamente il numero di vincoli ground (da 8 giga di utilizzo della RAM si è passati ad un centinaio di MB). Ciò ha permesso di ridurre i tempi di risoluzione da sei ore a sette minuti.

Abbiamo anche ridotto il più possibile l'utilizzo degli operatori di aggregazione, dato che in genere rallentano particolamente la procedura di risoluzione. In seguito siamo passati alla definizione di tutti i vincoli auspicabili, valutandone l'impatto sui tempi.

In ultimo abbiamo creato in Python un programma piuttosto semplice che provvede ad effettuare la formattazione dell'output di risoluzione fornito da clingo, in maniera tale da poter analizzare più facilmente i dati forniti sotto forma di calendario.

Scelte di modellazione

Dopo aver definito i predicati basilari del problema (settimane, giorni, insegnamenti, ecc...), si è pensato a come modellare l'assegnamento degli orari a calendario per ciascun insegnamento. Erano presenti due alternative:

- Modellare l'assegnamento degli orari tramite degli slot. In questo caso, ciascun slot possiede oltre alla settimana e al giorno di riferimento, l'orario di inizio e la durata dello stesso (in ore);
- Modellare l'assegnamento in maniera più granulare, assegnando a ciascuna ora un singolo insegnamento.

Il primo tipo di modellazione introduce un particolare vincolo: bisogna evitare che due slot possano sovrapporsi, perché sarebbe concettualmente scorretto che in uno stesso orario siano presenti due slot, e quindi due corsi.

La seconda scelta apparirebbe più semplice da definire, ma considerando come vengono assegnati nella realtà gli orari, è necessario che quando un corso viene erogato in un determinato giorno, le ore di questo corso siano assegnate in maniera tale da essere contigue tra loro. Pertanto è necessaria la definizione di un vincolo che faccia sì che se un insegnamento viene erogato in una giornata, le ore assegnate in questa giornata devono essere necessariamente contigue tra loro.

Dopo una valutazione che ha tenuto conto sia dei problemi dei vincoli suddetti, sia della definizione dei vincoli tassativi (e di quelli auspicabili) che si baserebbero inevitabilmente sulla concettualizzazione degli assegnamenti, abbiamo optato per la prima scelta. Questo perché ci ha permesso di definire i vincoli successivi in maniera più semplice e ha permesso anche una generazione di proposizioni ground inferiore rispetto alla seconda casistica.

Pertanto gli argomenti del predicato sono in ordine: insegnamento, numero della settimana, numero del giorno, orario di inizio (che va da 1 a 8), durata dello slot (che va da 2 a 4).

slot_assegnato(presentazione_master, 1, 5, 1, 2).

Il predicato sopra evidenziato rappresenta l'assegnamento di uno slot per la presentazione del master nel venerdì della prima settimana del calendario, il quale inizia alla prima ora e dura due ore.

0 { slot_assegnato(C,S,G,O,D) : O=1..Limite-1, D=2..4, O + D <= Limite + 1 } 1 :- insegnamento(C,_,_), orarioGiorno(S, G, Limite).

Questo aggregato permette la generazione opzionale degli slot.

Dato un insegnamento e un determinato giorno del calendario, è presente al più uno slot assegnato a quel determinato corso.

La durata del corso (D) va dalle 2 alle 4 ore. L'orario di inizio dello slot (O) va da 1 sino ad arrivare ad un numero tale che sommato alla durata dello slot non sia superiore alla durata massima delle lezioni (Limite) per il giorno in questione.

Ottimizzazione dei vincoli

Qui di seguito vengono elencati soltanto quei vincoli per i quali si è dovuto porre particolare attenzione al fine di velocizzare il processo di risoluzione nonché quello di grounding. I tempi di risoluzione indicati per ciascun caso sono stati calcolati effettuando una media tra più tempi di risoluzione, in maniera tale da tener conto della varianza dei tempi di esecuzione che si è mostrata essere rilevante. Verranno indicati diversi tempi, relativi a conseguenti ottimizzazioni applicate sul codice.

Propedeuticità

Per il vincolo della propedeuticità abbiamo formulato diverse implementazioni. La prima di queste stabiliva che se un corso C1 è propedeutico ad un corso C2, allora l'ultimo slot del corso C1 deve essere minore al primo slot del corso C2. Questo però presuppone di porre innanzitutto una relazione di ordinamento tra gli slot, nonché la definizione di un operatore di aggregazione (#min o #max) che permettesse di ottenere, per un determinato corso, il primo e l'ultimo slot assegnatoli.

Volendo ridurre però l'utilizzo degli operatori aggregati, abbiamo definito i vincoli in questa maniera:

non_soddisfa_propedeutico(C1, C2) :-

*S1 * 168 + G1 * 24 + O1 > S2 * 168 + G2 * 24 + O2,*

slot_assegnato(C1, S1, G1, O1, _), slot_assegnato(C2, S2, G2, O2, _).

:- propedeutico(C1, C2), non_soddisfa_propedeutico(C1, C2).

Questa definizione indica che se è presente anche un solo slot per il corso C1 che è superiore ad un qualsiasi slot del corso C2, allora significa che la propedeuticità non viene soddisfatta. Ciò ha permesso di rimuovere l'utilizzo degli aggregati.

Benché questo vincolo sia corretto, presenta un grosso problema: in fase di grounding, per qualsiasi corso C1 e C2 (anche se non deve essere soddisfatta la propedeuticità), vengono combinati tutti gli slot assegnabili al corso C1 con quelli del corso C2, generando così tante asserzioni che non è stato possibile effettuare la risoluzione del problema entro le 6 ore, con un utilizzo spropositato della memoria RAM (più di 8 GB).

Tramite il comando *"gringo --text Timetable.cl > output.txt"*, abbiamo potuto osservare ciò che genera gringo in fase di grounding. Abbiamo notato che la dimensione del file di output è di 7,5 GB. Ciò ha indirizzato le nostre scelte per una formulazione alternativa di questo vincolo.

```
non_soddisfa_propedeutico(C1, C2) :- propedeutico(C1, C2),
                                     S1 > S2,
                                     slot_assegnato(C1, S1, _, _, _), slot_assegnato(C2, S2, _, _, _).

non_soddisfa_propedeutico(C1, C2) :-propedeutico(C1, C2),
                                     G1 > G2,
                                     slot_assegnato(C1, S, G1, _, _), slot_assegnato(C2, S, G2, _, _).

non_soddisfa_propedeutico(C1, C2) :- propedeutico(C1, C2),
                                     O1 > O2,
                                     slot_assegnato(C1, S, G, O1, _), slot_assegnato(C2, S, G, O2, _).

:- non_soddisfa_propedeutico(C1, C2).
```

Tramite questa suddivisione delle casistiche, è stato possibile ridurre tutte quelle combinazioni di slot aventi settimana o giorno differenti.

Il predicato propedeutico(C1, C2), benché non strettamente necessario per la risoluzione in quanto ridondante, ha permesso di ridurre il numero di vincoli ground in maniera tale da generare solo quelli relativi a due corsi propedeutici tra loro, e di escludere tutti quei predicati inerenti due corsi che tra loro non sono propedeutici.

Il file generato da gringo in questo caso è di 440 MB. Provando ora ad effettuare la risoluzione, abbiamo ottenuto un tempo di circa 23 minuti con un utilizzo della RAM non superiore ai 200 MB.

Sovrapposizione slot

```
:-  slot_assegnato(C1, S, G, O1, D1),   slot_assegnato(C2, S, G, O2, D2),
    C1 != C2,    O1 != O2,
    O1 + D1 > O2,   O1 < O2 + D2.
```

:- slot_assegnato(C1, S, G, O, _), slot_assegnato(C2, S, G, O, _),

C1 != C2.

Abbiamo notato che l'output di gringo è formato per lo più dai suddetti vincoli, in quanto per due slot di uno stesso giorno di due insegnamenti differenti, sono numerose le possibili combinazioni di vincoli, tenendo anche conto degli orari e delle durate. Pertanto abbiamo fatto in modo di eliminare i vincoli qualora essi non fossero stati necessari in determinate circostanze.

Se un corso C1 è propedeutico ad un corso C2, è inutile considerare tutte quelle sovrapposizioni tali per cui uno slot del corso C1 sia susseguente ad uno slot del corso C2, in quanto esiste già il vincolo della propedeuticità che esclude questa casistica. Inoltre, se un corso C1 è propedeutico a C2, e quest'ultimo è propedeutico a C3, è inutile considerare tutte quelle sovrapposizioni di orari tra C1 e C3, in quanto gli stessi vincoli di propedeuticità escluderanno tale evenienza. Pertanto abbiamo ridefinito i vincoli nella seguente maniera:

:- slot_assegnato(C1, S, G, O1, D1), slot_assegnato(C2, S, G, O2, D2),

not inf_propedeutico(C1, C2), not inf_propedeutico(C2, C1), C1 != C2,

O1 < O2, O1 + D1 > O2.

:- slot_assegnato(C1, S, G, O1, D1), slot_assegnato(C2, S, G, O2, D2),

propedeutico(C1, C2), C1 != C2,

O1 < O2, O1 + D1 > O2.

:- slot_assegnato(C1, S, G, O, _), slot_assegnato(C2, S, G, O, _),

C1 < C2.

Con questa definizione abbiamo ridotto ulteriormente l'output del grounding portandolo a 230 MB. I tempi di esecuzione si sono ridotti a circa 10 minuti. Per i primi due vincoli, sarebbe stato possibile sostituire la variabile D2 con '_', ma si è notato che ciò porta sì ad una riduzione dell'output del grounding, ma dall'altra parte richiede maggior tempo per la risoluzione.

Altre piccole ottimizzazioni hanno portato i tempi di risoluzione intorno a 500 secondi (più di 8 minuti).

Inserimento dei vincoli soft

Partendo da un tempo di 500 secondi in presenza dei soli vincoli tassativi, ora si riepilogano i tempi di risoluzione tenendo conto dell'aggiunta di ciascun vincolo auspicabile:

- Primo vincolo: 760 secondi;
- Secondo: 668 secondi;
- Terzo: 538 secondi;
- Quarto: 541 secondi.

Considerando tutti i vincoli assieme i tempi di risoluzione si aggirano intorno ai 600 secondi con un singolo thread di esecuzione. Si noti come l'aggiunta di tutti e quattro i vincoli auspicabili porta a tempi inferiori rispetto che all'aggiunta di solo il primo (o il secondo) vincolo.

Multithreading

Infine, studiando i parametri di clingo, si è notato il parametro -t che permette l'utilizzo del multithreading per la risoluzione del problema. Ci sono due modalità: split e compete. La prima provvede a suddividere il problema in diversi sottoproblemi, e ciascuno di essi viene risolto da un singolo thread. La seconda invece prova la risoluzione del problema in modalità parallela, dove ciascun task lavora in maniera indipendente dall'altro.

Basandosi su dati empirici, si è notato che la modalità *compete* non apporta nessuna velocizzazione della risoluzione. Piuttosto la modalità *split* si è rivelata efficace, in quanto tramite l'utilizzo di 2 thread su un processore Intel dual core 2.50 Ghz ha portato i tempi di risoluzione dai 300 ai 400 secondi.

L'utilizzo di un numero di thread maggiore di 2 si è rivelata inefficace, almeno sulle nostre macchine.

Esempio di output

Nel file "Calendario.txt" è possibile consultare l'output formattato restituito dal programma Python, mostrante il calendario dei corsi.