Generazione automatica di manovre per lo Space Shuttle mediante la Programmazione Logica Disgiuntiva*

Stefania Galizia

Department of Mathematics, Università della Calabria,
Via Pietro Bucci, 30B
87036 Rende (CS), Italy
galizia@mat.unical.it

Sommario La programmazione logica disgiuntiva (PLD) costituisce un potente strumento per la rappresentazione della conoscenza e del ragionamento di senso comune.
Un'importante applicazione della PLD è l'*answer set planning*.
Nell'answer set planning un problema di pianificazione P è codificato in un programma PL in PLD, tale che ogni answer set di PL corrisponda ad una soluzione del problema di planning P.
Nel presente lavoro abbiamo preso in considerazione il sistema *RCS*, il sistema di controllo di reazione dello Space Shuttle.
Abbiamo progettato una tecnica, basata su answer set planning, per la pianificazione delle azioni da svolgere per effettuare le manovre dello Space Shuttle, partendo da un precedente lavoro realizzato dal gruppo di ricerca KRLab del Texas Tech University.
La nostra tecnica supera alcuni limiti riscontrati nel precedente metodo garantendo, in particolare, una migliore "qualità" dei piani calcolati.
Abbiamo implementato la nostra tecnica facendo uso del sistema di programmazione logica disgiuntiva DLV.

1 Introduzione

La programmazione logica disgiuntiva (PLD) costituisce un formalismo efficace per il trattamento del ragionamento di senso comune[2,4,9]. Essa consente, ad esempio, di rappresentare, in maniera semplice e naturale, forme di ragionamento non monotono, conoscenza incompleta, problemi diagnostici, di pianificazione e, più in generale, problemi di elevata complessità computazionale. Grazie ad un elevato potere espressivo[4] la PLD permette di rappresentare anche problemi Σ_2^P-completi (in modalità 'brave reasoning'). Assumiamo che il lettore conosca i principali costrutti sintattici della PLD e la semantica dei modelli stabili. Per maggiori approfondimenti consigliamo i seguenti riferimenti: [1,2,10,11].
Tra le applicazioni più interessanti della PLD, c'è, come già menzionato, quella della

* Questo lavoro è stato finanziato dalla Commissione Europea nell'ambito dei progetti IST-2001-37004 WASP e IST-2001-33570 INFOMIX.

pianificazione. Nel presente lavoro abbiamo studiato l'impiego della programmazione logica disgiuntiva per la risoluzione di problemi di Pianificazione in un complesso dominio applicativo: il sistema di controllo di Reazione dello Space Shuttle (*RCS, Reaction Control System*).
RCS è un complesso sistema con vari compiti da gestire: controllo delle manovre di atterraggio e di decollo della navicella, supporto alle manovre nello spazio, controllo dei processi di ossidazione del carburante per fornire il necessario propellente nel corso delle diverse manovre della navicella.
E' necessario capire come le manovre vengano eseguite; in particolare, ogni manovra dipende dal percorso seguito dal carburante, il quale riceve una spinta nella direzione opportuna e, quindi, arrivato nel giusto punto, viene incendiato. Per controllare i percorsi che il carburante deve seguire per giungere nella direzione più idonea all'esecuzione della manovra desiderata, *RCS* è dotato di un circuito elettronico. Nel momento in cui lo Space Shuttle arriva in orbita, agli astronauti è assegnato il compito di eseguire delle operazioni per realizzare le manovre. Il sistema proposto in questo lavoro è in grado di generare la sequenza di operazioni da compiere in funzione delle manovre da effettuare. Tali operazioni possono essere azioni manuali o digitali su interruttori; entrambe le tipologie di operazioni controllano le valvole poste lungo i percorsi coperti dal carburante e dall'ossidante. Tali percorsi, talvolta, riportano dei guasti(valvole malfunzionanti, interruttori bloccati, ecc...); bisogna, dunque, pianificare la serie di operazioni da effettuare per l'esecuzione delle manovre, tenendo conto di eventuali malfunzionamenti.
Nel presente lavoro abbiamo realizzato, mediante l'uso della PLD, un sistema di planning per *RCS*. In sintesi, abbiamo studiato in dettaglio il sistema *RCS*, analizzando una tecnica di pianificazione, realizzata dal gruppo di ricerca KRLab del Texas Tech University[15] e implementata con un programma in Smodels[16]. Abbiamo, dunque, progettato un nuovo metodo di pianificazione per *RCS* che supera alcuni limiti riscontrati nel precedente metodo. Abbiamo realizzato concretamente il suddetto pianificatore implementandolo nel sistema di programmazione logica disgiuntiva, DLV[1,2,3,5].
DLV e Smodels sono, senza dubbio, i due principali sistemi basati sulla programmazione logica. DLV, al momento, è l'unico sistema che supporta pienamente la programmazione logica disgiuntiva[3], consentendo, inoltre, l'uso dei "weak constraints"[1]; ciò ha permesso di ottimizzare il problema. Ottimizzare, nel nostro caso, è significato trovare il miglior piano da far eseguire al fine di ottenere la giusta manovra, ma più in particolare fornire nuove strade attraverso le quali scartare piani non efficienti.
Il seguito del lavoro è organizzato come segue: la sezione 2 illustra brevemente il problema classico dell Pianificazione; la sezione 3 contiene una descrizione del sistema *RCS*; la sezione 4 illustra il programma logico disgiuntivo per la pianificazione delle manovre dello Space Shuttle; la sezione 5 trae alcune conclusioni discutendo brevemente i miglioramenti ottenuti nella generazione dei piani.

2 Il Problema della Pianificazione

Supponiamo che un sistema sia caratterizzato da un insieme di stati e da una relazione di transizione che descrive le possibili evoluzioni del sistema, ovvero i possibili cambia-

menti di stato. Dato lo stato iniziale del sistema e uno stato finale desiderato, il planning consiste, informalmente, nella determinazione di un insieme di passi che fa evolvere il sistema dallo stato iniziale allo stato finale desiderato, compatibilmente con la relazione di transizione.

Più formalmente[12], siano Act e Flu due insiemi disgiunti di atomi ground, chiamati, rispettivamente, *azioni* e *fluents*; uno stato è un sottoinsieme $S \subseteq Flu$. Intuitivamente, gli atomi contenuti in S sono l'insieme delle proprietà vere in questo stato.

Una Specifica di Planning PS è una tupla $\langle I, S, A, G\rangle$, dove I è uno stato, chiamato *stato iniziale*; S è l'insieme degli *stati ammissibili*; $A : S\times Act \rightarrow S$ è una funzione parziale, chiamata *funzione di transizione*, che definisce l'effetto delle azioni. Se (S, a) appartiene al dominio di A, allora l'azione a è applicabile allo stato S; G è uno stato, chiamato *goal*.

Un completamento dello stato iniziale I è uno stato ammissibile $\bar{S}$ (completo) tale $I \subseteq \bar{S}$. Sia PS=$\langle I, S, A, G\rangle$ una specifica di planning e $\bar{S}$ un completamento dello stato iniziale I. Una sequenza finita $A : a_0, a_1, ...a_{n-1}$ di azioni è un piano per PS se esiste una sequenza di stati $S_0, S_1, ..., S_n$ tali che: $S_0 = \bar{S}$, $G \subseteq S_n$, per ogni $0 \leq i \leq n-1$, a_i è applicabile a S_i e $A(S_i, a_i) = S_{i+1}$.

Il numero n di elementi di A è chiamato *lunghezza* del piano A.
Un problema di planning (detto "Planning Classico"), in letteratura, è definito come segue:
Dato un intero n e una specifica di planning PS=$\langle I, S, A, G\rangle$, il problema è trovare un piano per PS di lunghezza n a partire dallo stato I.
La complessità di tali problemi dipende, chiaramente, dal modo in cui la specifica di planning è codificata. Solitamente, è richiesto che la lunghezza dei piani sia limitata da una codifica di dimensione polinomiale. Sotto questa assunzione il planning classico è NP-Completo [12,6].

I problemi di planning classico possono essere codificati in PLD con la semantica answer set. Tale codifica fu introdotta da Vladimir Lifschitz [13] e denominata *Answer Set Planning* (ASP). In Answer Set Planning, una specifica di planning PS=$\langle I, S, A, G\rangle$ (con stato iniziale I completo) è codificata tramite un programma P in PLD, tale che ogni answer set di P corrisponda ad un piano per PS da I.

Come esempio, riportiamo il più conosciuto problema di planning: *Blocks World* (mondo dei blocchi).

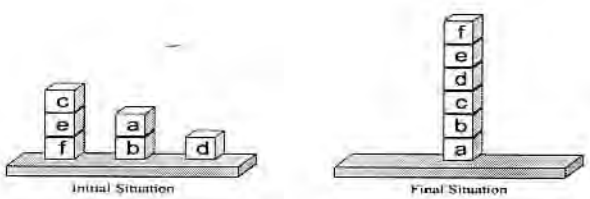

Figura 1. Blocks World

Esempio. Nel problema del Blocks World, è necessario convertire una configurazione

iniziale in una finale, differente (vedere figura 1). La configurazione iniziale dei blocchi rappresenta lo stato iniziale, mentre il goal è rappresentato dalla configurazione finale desiderata. Lo spostamento di un blocco B in una posizione L (per esempio su un altro blocco o sul piano) al passo T è un'azione, che rappresentiamo con un atomo $move(B, L, T)$. Non è permesso lo spostamento di un blocco che di trova in mezzo alla torre, vale a dire, che ha altri blocchi su si esso. Analogamente, su di un blocco che si trova in mezzo alla torre non si può collocare un altro blocco.
La sequenza di azioni: $move(a, table, 1)$, $move(b, a, 2)$, $move(c, b, 3)$, $move(d, c, 4)$, $move(e, d, 5)$, $move(f, e, 6)$ è un piano per il problema di planning della figura 1.

3 Il sistema di Controllo di Reazione dello Space Shuttle (RCS)

3.1 Cos'è l' RCS

L'*RCS*, come già menzionato, è un complesso sistema [15], che ha il compito di gestire il controllo delle manovre di atterraggio, di decollo dello Space Shuttle e di supportare gli austronauti durante tutte le manovre da effettuare in orbita. Esso è dotato, essenzialmente, di cisterne contenenti combustibile o ossidante, di valvole e di altri impianti idraulici, tutti atti ad indirizzare nella giusta direzione il propellente affinchè i getti generati permettano l'esecuzione della manovra desiderata. Sostanzialmente, i getti, raggiunta la posizione opportuna, vengono incendiati e le spinte risultanti permettono l'esecuzione della manovra. Il sistema è dotato di un circuito elettrico che è di ausilio al controllo delle valvole presenti lungo il percorso del combustibile.

Le operazioni di atterraggio e di decollo sono controllate dal computer; mentre la navicella è in orbita, tuttavia, gli astronauti ne hanno il controllo principale. Queste azioni richiedono, generalmente, di azionare interruttori, usati per aprire o chiudere valvole, o dare energia al circuito. In talune circostanze può succedere che esista qualche interruttore malfunzionante, in tal caso, gli astronauti comunicano il problema ai controllori di volo da terra, i quali suggeriscono degli opportuni comandi digitali alternativi per risolvere il problema e istruiscono il computer dello Shuttle ad eseguirli[15].

Durante le normali operazioni della navicella, esistono dei piani stabiliti che gli astronauti fanno eseguire per raggiungere alcuni obiettivi. La situazione va tenuta sotto controllo nel momento in cui si verificano degli errori nel sistema. Il numero dei possibili malfunzionamenti è così grande che stabilire dei piani a priori, per ogni situazione, risulterebbe molto complesso. Tuttavia si rende necessario, per la sicurezza dell'equipaggio, oltre che per portare a buon fine la missione, garantire l'esecuzione delle giuste manovre a prescindere dai possibili malfunzionamenti.

Un sistema intelligente, che verifichi e generi dei piani, potrebbe essere una soluzione. È con questa prospettiva che abbiamo realizzato tale lavoro, agendo sul sistema di controllo di *RCS*.

3.2 Cos'è il Sistema di controllo di RCS

Il Sistema di Controllo di *RCS* è un sistema di supporto alle decisioni per i controllori di volo dello Shuttle. Sostanzialmente, si tratta di una tecnica di pianificazione del comportamento di *RCS*, implementata nel sistema di programmazione logica disgiuntiva

DLV. Tale tecnica è ispirata ad una già esistente, realizzata dal gruppo di ricerca KRLab del Texas Tech University[7,8,9] e implementata in Smodels[16].

Tale sistema intelligente può essere utilizzato praticamente. Per meglio comprenderne l'applicazione [8], immaginiamo che i controllori di volo debbano preparare la navicella per una manovra e che si siano accorti di una serie di difetti presenti nell'*RCS*, per esempio: interruttori staccati, bloccati in una posizione (aperti o chiusi), valvole con dispersioni di propellente, getti danneggiati, circuito elettrico malfunzionante in qualche punto, ecc. In questa situazione i controllori hanno bisogno di trovare una sequenza di azioni, un piano, che renda lo shuttle pronto per la manovra stabilita.

Il nostro sistema può essere loro di aiuto. Esso genera automaticamente la sequenza delle operazioni da compiere per l'esecuzione di una manovra. I controllori possono, dunque, utilizzare direttamente tale sequenza o, semplicemente, impiegarla per testare un piano in attuazione.

3.3 Struttura del Sistema

Abbiamo suddiviso il problema in diversi moduli, ognuno dei quali mira alla descrizione di una corrispondente parte di *RCS*.

Ogni modulo identifica una sottoparte del problema. La composizione di tutti i sottoproblemi consente di realizzare il goal prefisso. Il piano finale da realizzare dovrà essere conforme alle richieste riguardanti le manovre da eseguire ed inoltre, essere in grado di ovviare ai difetti del sistema. Scomporre il problema in moduli è il primo passo necessario per poter applicare tecniche di planning, come si è visto nella precedente sezione.

Forniamo nel seguito una descrizione dettagliata delle caratteristiche dei principali moduli.

Plumbing Module

Per realizzare correttamente una manovra, è necessario[15] indirizzare carburante e ossidante nelle posizioni opportune per affinché il carburante sia incendiato.

Il modulo di *Plumbing* modella il sistema idraulico di RCS. Esso è suddiviso in tre sottosistemi: *FORWARD, LEFT, RIGHT*. Una manovra è la risultante dei contributi dei 3 sottosistemi, che lavorano, dunque, in parallelo. Ognuna delle 3 parti è a sua volta, ripartita in due ulteriori sezioni, relative al percorso coperto dal carburante e dall'ossidante.

Il sistema idraulico è rappresentato dal grafo Gr, di cui riportiamo una parte in figura 2. In Gr i nodi rappresentano cisterne, giunzioni o getti e gli archi rappresentano le valvole.

Per effettuare una manovra è necessario pressurizzare nelle direzioni opportune carburante e di ossidante. Bisogna, dunque, stabilire i percorsi per entrambi, dai nodi iniziali (cisterne all'elio) a quelli finali, rappresentanti i getti richiesti per l'esecuzione della data manovra.

Il sistema idraulico potrebbe presentare dei malfunzionamenti, come è stato già accennato in 3.2. Se una valvola permette al liquido di fuoriuscire, per esempio, è opportuno chiuderla, in modo da non permettere al carburante o all'ossidante di utilizzarla nel rispettivo percorso. Se una valvola è staccata, e quindi bloccata in una certo stato

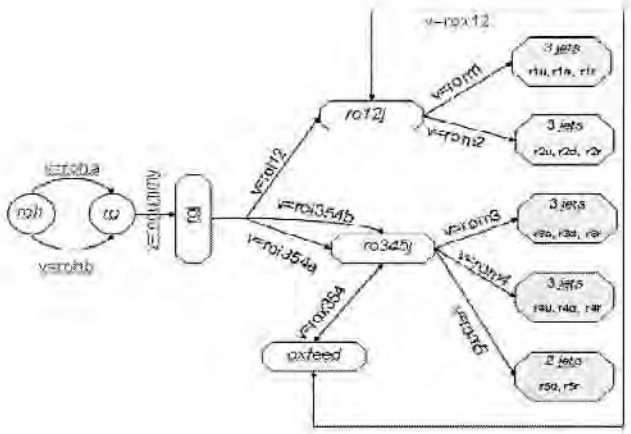

Figura 2. Sottosistema Right(Ossidante)

("closed" o "open"), è preferibile non utilizzarla perché poco efficiente, a meno che non si tratti dell'unica strada possibile; ovviamente se essa è staccata nella pozione "closed" è impossibile utilizzarla.

Affinché un nodo riceva pressione, bisogna assicurarsi che il nodo immediatamente precedente, lungo il percorso, sia già pressurizzato, in caso contrario non è opportuno che la valvola che connette i due nodi sia aperta. In caso di malfunzionamento potrebbe verificarsi che una valvola V che connette la giunzione $N1$ a $N2$ sia staccata, nella posizione "open"; in questo caso bisognerebbe evitare di pressurizzare il nodo $N1$, e preferire una direzione alternativa per il fluido.

In definitiva, per la realizzazione di un piano che porti all'esecuzione di una manovra, è necessario: 1) stabilire un cammino che carburante e ossidante possano seguire per effettuare la data manovra, tenendo in considerazione le componenti danneggiate e il tipo di malfunzionamento; 2) restituire la sequenza di azioni necessarie sulle valvole che permattano al fluido di seguire il cammino stabilito.

Valve control module (VCM)

A questo modulo è affidato il compito di supervisionare lo stato del circuito meccanico ed elettrico e di segnalarne gli eventuali malfunzionamenti.

I difetti che possiamo riscontrare sono relativi ad interruttori o a valvole malfunzionanti oppure a circuiti elettrici di connessione danneggiati.

Il *VCM* si suddivide in due sottomoduli: *Basic*, *Extended*

Al sottomodulo Basic è affidata essenzialmente una descrizione delle condizioni di valvole ed interruttori. A questo livello vengono rilevati gli stati delle componenti ed eventuali conflitti di stato. In quest'ultimo caso il Basic suggerisce differenti posizioni per le valvole o per gli interruttori.

Lo stato delle valvole può essere monitorato anche dal circuito elettrico di cui è dotato RCS.

Il sottomodulo Extended ha un ruolo di controllo sul circuito elettrico.

Circuit theory module(CTM)
Il *CTM* descrive le componenti del circuito elettrico. Come per il circuito idraulico, anche per quello elettrico, esiste un grafo descrittivo (che non riportiamo per brevità). In esso i nodi rappresentano le porte logiche e gli archi i fili di connessione.

Al CTM è affidato il monitoraggio di eventuali ritardi di propagazione del segnale, tramite i valori logici $(0,\ 1,\ u)$: 0 se non si riscontra alcun ritardo, 1 se esso viene rilevato, u quando l'informazione è sconosciuta.

4 Descrizione della codifica logica in DLV

Abbiamo implementato il pianificatore per la generazione automatica di manovre per lo Space Shuttle in DLV. Abbiamo analizzato una codifica in Smodels[16], realizzata dal gruppo KRLab della Texas Tech University[15], suggerendo sostanziali miglioramenti in termini di efficienza.

L'utente decide il numero massimo di azioni necessario per effettuare la manovra. I modelli stabili restituiti dalla nostra codifica logica rappresentano i possibili piani per il raggiungimento del goal (esecuzione della manovra desiderata), supportando, dunque, le decisioni dei controllori di volo dello Space Shuttle.

4.1 Struttura della Codifica

Descriviamo ora, in maniera più dettagliata, il codice del programma. Esso è diviso in moduli logici, ognuno dei quali è relativo ad un particolare sottoproblema del sistema di pianificazione.

rcs1
Il modulo logico *rcs1* contiene una codifica del grafo Gr, indicativo dei percorsi che il carburante segue per giungere nei punti in cui viene incendiato. Ricordiamo che i nodi di tale grafo rappresentano cisterne, giunzioni e getti, mentre gli archi sono rappresentano le valvole.

I fatti di tipo $link(N1, N2, V)$ indicano che i nodi $N1$, $N2$ sono connessi tramite valvola V; $tank_of(X, R)$ che X è una cisterna appartenente al sottosistema R; $jet_of(J, R)$ che J è un getto del sottosistema R; $direction(J, D)$ indica che il getto J ha direzione D.

Molte regole contenute in rcs1 permettono di determinare la giusta distribuzione di pressione, lungo i percorsi di carburante e ossidante, relativamente ai 3 sottosistemi, necessaria per realizzare la manovra desiderata.

La descrizione di tutte le possibili manovre è affidata ad un elenco di fatti appartenenti a questo modulo. Non tutte le manovre necessitano della collaborazione dei 3 sottosistemi. Riportiamo, in questo lavoro, la codifica di qualche manovra, tra le possibili, rappresentando lo spazio di movimento della navicella con un sistema cartesiano ortogonale in R^3

- $Plus_X$: spostamento lungo l'asse positivo delle X. Per la realizzazione di questa manovra occorre l'apporto del sottosistema $rigth$ e del $left$, ognuno dei quali ha bisogno di un getto nella direzione aft;

- $Plus_Roll$: manovra di rotazione. Sono coinvolti i sottosistemi $left$ e $right$: il $left$ genera un getto nella direzione $down$, mentre il $right$ un getto nella direzione up;
- $Minus_Yaw$: manovra di rotazione. È necessario l'apporto del sottosistema $left$ e del $forward$: il $left$ hanno bisogno di un getto nella direzione $left$, mentre il $forward$ di un getto nella direzione $right$.

Immaginiamo, ad esempio, di voler eseguire la manovra $plus_x$
La regola:

$$maneuver_of(plus_x, left_rcs, T) \leftarrow time(T),\ jet_of(J, left_rcs),\ direction(J, aft),\ ready_to_fire(J, T).$$

esprime che al tempo T desideriamo realizzare la manovra $plus_x$. Essa coinvolge il sottosistema $left$ e ha bisogno di un getto nella direzione aft.
Selezionato il getto, affinchè esso sia pronto per essere incendiato, deve verificarsi l'evento:

$$ready_to_fire(J, T) \leftarrow time(T),\ jet_of(J, R),\ tank_of(TK1, R),\ tank_of(TK2, R),\ TK1! = TK2,\ pressurized_by(J, TK1, T),\ pressurized_by(J, TK2, T),\ notdamaged(J).$$

La seconda parte di rcs1 contiene la codifica del modulo VCM, ossia una descrizione di interruttori e comandi digitali che controllano le valvole. I fatti $controls(SW, V), basic_command(CC, V, S), commands(CC1, CC2, V, S)$ rappresentano i possibili dispositivi di controllo.

Riportiamo una regola relativa allo stato di un generico interruttore:

$$in_state(Sw, S, T1) \leftarrow next(T, T1),\ type_of(Sw, _, v_switch),\ state_of(S, v_switch),\ flip(Sw, S, T),\ not\ stuck(Sw).$$

Questa specifica che all'istante $T1$ l'interruttore Sw si trova nella posizione S (open o closed), se, all'istante precedente T, è stata compiuta un'azione con l'effetto di portare l'interruttore su tale posizione.

Abbiamo introdotto le seguenti regole per fissare la sequenzialità delle azioni:

$$time(N) \leftarrow \#int(N). \quad next(N, N1) \leftarrow \#succ(N, N1).$$

Il valore della variabile N deve essere assegnato dall'utente e indica il numero massimo di passi desiderato per il raggiungimento del goal.

rcs2

rcs2 rappresenta la codifica logica del modulo VCM Extended; modella la struttura del circuito elettrico, rilevando eventuali malfunzionamenti dei fili di connessione. I fatti:

$$input(Wi, D). \quad output(Wo, D). \quad connects(S, D, Wi, Wo).$$

specificano che Wi e Wo sono, rispettivamente, fili di input e output. Essi sono connessi all'interruttore D, che si trova nello stato S.

circuit

circuit contiene la dettagliata descrizione di tutte le componenti digitali del sistema, porte logiche, fili di connessione, ecc.

Ogni porta logica contiene i fili di input e output. L'output di una porta è direttamente connesso all'input di un'altra.

Sono presenti delle regole logiche per computare i ritardi di una porta e le possibilità di adattamento nel caso di funzionamento scorretto.

Una porta funziona scorrettamente se uno o più dei suoi fili di input o output risultano permanentemente staccati e quindi bloccati in certo stato(1, 0, u).

Come esempio consideriamo il codice che definisce il valore sull'uscita di una porta AND quando tutti i suoi ingressi sono pari ad 1:

$value(W, 1, T1) \leftarrow of_type(G, and_gate),\ delay(G, D),\ time(T),\ time(T1),$
$T1 = T + D,\ output(W, G),\ all(G, 1, T),\ not\ is_stuck(W, G).$

fhca, fhcb, fmc1, fmc2, fmc3, fmc4

I moduli logici $fhca$, $fhcb$, $fmc1$, $fmc2$, $fmc3$, $fmc4$ contengono ulteriori descrizioni di dispositivi di connessione. Sono costituiti da una raccolta di fatti descriventi dettagli del circuito elettrico.

plan

Una regola disgiuntiva, in plan, consente di rappresentare in modo semplice e naturale le possibili azioni da compiere per la generazione dei piani.

$flip(Sw, S, T) \vee -flip(Sw, S, T) \vee issue(A, T) \vee -issue(A, T) \leftarrow time(T),$
$action_of(A, R),\ type_of(Sw, R, v_switch),\ state_of(S, v_switch),$
$not\ goal(T, R),\ not\ done(R).$

Per portare a buon fine l'esecuzione di una manovra, viene compiuta, ad ogni passo T, un'azione, manuale o digitale($flip$ o $issue$), per ogni sottosistema coinvolto nella realizzazione della manovra data.

Vengono eseguite azioni opportune finché ciascun sottosistema non raggiunge il goal parziale, ovviamente, non superando il numero di passi stabilito dall'utente.

$fgoal \leftarrow time(T1),\ time(T2),\ time(T3),\ goal(T1, left_rcs),$
$goal(T2, right_rcs),\ goal(T3, fwd_rcs).$

$\leftarrow not\ fgoal.$

$goal(T, S) \leftarrow time(T),\ done(S).$

È possibile che i goals parziali siano raggiunti in tempi diversi dai tre sottosistemi ($T1$, $T2$, $T3$); l'obbiettivo finale ($fgoal$) si ottiene quando i tre sottosistemi hanno raggiunto il goal parziale.

4.2 Eliminazione di azioni ridondanti

Per impedire azioni inutili o ridondanti, abbiamo inserito una serie di vincoli che hanno l'effetto si snellire i piani. Ne riportiamo qualcuno.

- Impedire che la stessa azione sia eseguita in due successivi istanti di tempo:
  $\leftarrow type_of(Sw, v_switch),\ state_of(S, v_switch),\ time(T1),\ time(T2),$
  $T1 <> T2,\ flip(Sw, S, T1),\ flip(Sw, S, T2).$
  $\leftarrow action_of(A, _),\ time(T1),\ time(T2),\ T1 <> T2,$
  $issue(A, T1),\ issue(A, T2).$
- Non compiere due azioni che abbiano lo stesso effetto:
  $\leftarrow time(T),\ time(T1),\ flip(Sw, S, T),\ controls(Sw, S, _)$
  $basic_command(CC, V, S),\ issue(CC, T1).$
- Non effettuare un'azione digitale con l'effetto di portare una valvola in uno stato in cui già si trova:
  $\leftarrow time(T),\ in_state(V, S, T),\ basic_command(CC, V, S),\ issue(CC, T).$

- Non compiere un'azione digitale, con l'effetto di portare una valvola in uno stato S, se ad un passo precedente un'azione manuale ha portato la valvola nello stato $S1$ opposto ad S.
  $\leftarrow$ $time(T1)$, $time(T2)$, $T2 <> T1$, $basic_command(CC, V, S1)$, $issue(CC, T1)$, $opposite_state(S1, S2)$, $controls(Sw, S, _)$, $flip(Sw, S2, T)$.

4.3 Miglioramento della qualità dei piani

Osservando i modelli stabili, restituiti dal programma implementato in Smodels, abbiamo notato come alcuni piani risultino essere poco convenienti, in quanto presentano una serie di azioni superflue e suggeriscono percorsi non del tutto efficienti.

Introducendo nuove euristiche, grazie all'uso dei "weak constraints" di DLV, abbiamo suggerito un metodo idoneo a superare molti limiti di efficienza.

Quando un dispositivo risulta poco efficiente, a causa del fatto che esso è bloccato su una certa posizione, è consigliabile non utilizzarlo a meno che non sia l'unica strada attraverso la quale realizzare la manovra.

I comandi digitali CC controllano coppie di valvole, appartenenti, rispettivamente, ai percorsi coperti dall'ossidante e dal carburante. Supponiamo, per esempio, di avere una valvola malfunzionante e bloccata sullo stato *closed*; in questo caso cegliere di eseguire un comando digitale non è consigliabile, perchè si agirebbe, inutilmente anche sulla valvola malfunzionante.

Utilizzando un "weak constraint" possiamo limitare l'uso di un CC che controlla una coppia di valvole, di cui una staccata.

$:\sim$ $time(T)$, $stuck_in(V, closed)$, $system(R)$, $type_of(V, R, valve)$, $type_of(V1, R, valve)$, $V <> V1$, $not\ stuck(V1)$, $basic_command(CC, V, _)$, $basic_command(CC, V1, _)$, $issue(CC, T)$.

Considerazioni analoghe valgono per le azioni manuali su interruttori.

$:\sim$ $time(T)$, $stuck(V, closed)$, $system(R)$, $type_of(V, R, valve)$, $type_of(V1, R, valve)$, $V <> V1$, $not\ stuck(V1)$, $type_of(Sw, R, v_switch)$, $controls(Sw, V, _)$, $controls(Sw, V1, _)$, $flip(Sw, S, T)$.

Analizziamo la situazione in cui non funzioni un interruttore, bloccato nello stato "open" Immaginiamo che lo stato dell'interruttore non è discordante da quello delle valvole che esso controlla. Potremmo scegliere un percorso che coinvolga queste valvole, in quanto aperte, ma non è preferibile poichè non possiamo averne un controllo completo. Scegliamo, dunque, di limitarne l'uso. Possiamo agire solo su percorsi intermedi poiché se la valvola aperta si trova sulla parte iniziale del percorso, essa è automaticamente pressurizzata dalle cisterne all'elio, che sono perennemente sotto pressione. In caso contrario possiamo agire scegliendo un percorso alternativo:

$:\sim$ $time(T)$, $system(R)$, $type_of(Sw, R, v_switch)$, $type_of(Sw1, R, v_switch)$, $stuck_in(Sw, open)$, $link(N1, N2, V1)$, $link(N2, N, V)$, $not\ stuck(V1)$, $controls(Sw, V, _)$, $controls(Sw1, V1, _)$, $flip(Sw1, _, T)$.

$:\sim$ $time(T)$, $system(R)$, $type_of(Sw, R, v_switch)$, $type_of(Sw1, R, v_switch)$, $stuck_in(Sw, open)$, $link(N1, N2, V1)$, $link(N2, N, V)$, $not\ stuck(V1)$, $controls(Sw, V, _)$, $basic_command(CC, V1, _)$, $issue(CC, T)$.

Come è stato già osservato, l'utente ha la possibilità di stabilire il numero massimo di passi da effettuare per il raggiungimento del goal; è opportuno, comunque, per una

maggiore efficienza, limitare il numero di azioni. Questo tipo di controllo si ottiene con il weak constraint $:\sim action_occurs(A, T)$, che limita le operazioni superflue.

5 Conclusioni

In questo lavoro abbiamo descritto una tecnica, basata su answer set planning, per la generazione automatica di manovre per lo Space Shuttle, anche in caso di malfunzionamenti del sistema. Abbiamo migliorato la tecnica proposta dal KRLab della Texas Tech University[15], riuscendo ad ottenere piani decisamente più efficienti, privi di azioni superflue e, ove possibile, privi di percorsi, per carburante e ossidante, poco sicuri.

Abbiamo implementato la nostra tecnica nel sistema di programmazione logica disgiuntiva, DLV, che ci ha consentito di esprimere, grazie all'uso dei vincoli deboli (weak contraints), dei criteri di preferenza sui piani, che consentano di eliminare piani ridondanti o insicuri.

Abbiamo confrontato sperimentalmente i modelli restituiti dalla codifica originaria in Smodels con quelli restituiti dalla nuova codifica in DLV.

Di seguito riportiamo un esempio in cui la nuova codifica produce risultati differenti, grazie alla presenza dei weak constraints.

Vogliamo che sia eseguita la manovra $plus_x$, che coinvolge i sottosistemi $left$ e $right$; è necessario generare getti di carburante e ossidante di tipo aft per entrambi i sottositemi. Immaginiamo che il sistema idraulico presenti i seguenti malfunzionamenti: interruttori $lfdummy$, rha, $rm1$ non funzionanti e bloccati nello stato $open$. Vogliamo generare un piano che ci consenta di raggiungere il goal in non più di 4 passi.

Ricordiamo che per raggiungere il goal è necessario che la manovra venga eseguita, anche in presenza di eventuali malfunzionamenti. Affinché ciò avvenga, carburante e ossidante devono raggiungere la posizione necessaria perché vengano incendiati. Per realizzare la manovra presa in considerazione in questo esempio, i fluidi devono raggiungere la posizione di tipo aft (figura 3).

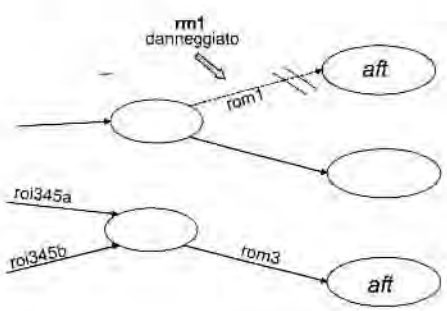

Figura 3. Parte del grafo Gr - Right(Ossidante)

Il programma implementato in Smodels restituisce il seguente piano[1]:

passo 1 $\{flip(ri345b, open, 0),\ issue(close_lfi345a, 0)\}$
passo 2 $\{flip(ri12, open, 1),\ issue(open_lfha, 1)\}$
passo 3 $\{flip(li345b, open, 2)\}$
passo 4 $\{flip(lm3, open, 3)\}$

Il programma realizzato in DLV restituisce:

passo 1 $\{flip(ri345a, open, 0),\ issue(open_lfha, 0)\}$
passo 2 $\{issue(open_rfm3, 1),\ flip(li345b, open, 1)\}$
passo 3 $\{flip(lm3, open, 2)\}$

Possiamo notare che le azioni del piano generato da DLV sono di numero inferiore rispetto a quelle contenute nel piano restituito da Smodels; DLV suggerisce un piano che usa un numero di passi inferiore rispetto al massimo consentito, raggiunge il goal in 3 step; risulta, quindi, decisamente più efficiente.

Possiamo migliorare il piano generato da Smodels riducendo a 3 il numero passi consentiti.

In questo caso il piano restituito dal programma in Smodels è :

passo 1 $\{issue(open_rfhb, 0),\ issue(open_lfha, 0)\}$
passo 2 $\{issue(open_roi345b, 1),\ flip(li345a, open, 1)\}$
passo 3 $\{issue(open_ri12, 2),\ flip(lm3, open, 2)\}$

Il piano generato da Smodels contiene ancora un'azione in più rispetto a quello restituito da DLV. Il piano suggerito dalla nostra tecnica è, inoltre, decisamente più efficiente perché non contiene azioni "inutili" ed evita strade poco sicure. Notiamo infatti che le azioni $issue(open_roi345b, 1)$ e $issue(open_ri12, 2)$ proposte dal pianificatore realizzato dal KRLab sarebbero da evitare (figura 3).

La prima, che suggerisce di aprire le valvole $roi345$ e $rfi345$, è inutile, sarebbe opportuna solo se al passo successivo venisse suggerita l'apertura delle valvole $rom3$ e $rfm3$, che porterebbero facilmente ossidante e carburante a getti di tipologia aft, come richiesto dalla manovra che vogliamo eseguire.

La seconda consente l'esecuzione della manovra, pressurizzando i fluidi e indirizzandoli verso i getti opportuni, ma viene suggerita una strada sconsigliabile: vengono usate le valvole $rom1$ e $rfm1$ giá aperte, ma poco sicure, in quanto l'interruttore che dovrebbe controllarne lo stato, $rm1$, non è funzionante.

Ringraziamenti

Questo lavoro non sarebbe mai stato realizzato senza il prezioso sostegno professionale e, spesso, morale offertomi dal prof. Nicola Leone. Ringrazio l'ing. G. Ianni, per la sua infinita disponibilità e pazienza. Sono, infine, particolarmente grata all'ing. Stefania Nevone Blasi, che ha collaborato, con il suo lavoro di tesi, alla realizzazione di questo articolo.

[1] È importante notare che sia la codifica in Smodels che quella in DLV producono, in generale, diversi piani di azioni. In questo esempio consideriamo il primo piano generato sia da DLV che da Smodels. Considerazioni analoghe, tuttavia, si applicano alla maggior parte dei piani generati dai due sistemi. In particolare, DLV genera **solo** piani di lunghezza 3.

Riferimenti bibliografici

1. F. Buccafurri, N. Leone, and P. Rullo. Enhancing Disjunctive Datalog by Constraints. 12(5):845–860, 2000.
2. T. Dell'Armi, W. Faber, G. Ielpa, C. Koch, N. Leone, S. Perri, and G. Pfeifer. System Description: DLV. In Thomas Eiter, Wolfgang Faber, and Mirosław Truszczyński, editors, *Logic Programming and Nonmonotonic Reasoning — 6th International Conference, LPNMR'01, Vienna, Austria, September 2001, Proceedings*, number 2173, pages 409–412. Springer Verlag, September 2001.
3. T. Eiter, W. Faber, N. Leone, and G. Pfeifer. Declarative Problem-Solving Using the DLV System. In Jack Minker, editor, *Logic-Based Artificial Intelligence*, pages 79–103. Kluwer Academic Publishers, 2000.
4. T. Eiter, G. Gottlob, and H. Mannila. Disjunctive datalog. *ACM Trans. on Database Systems*, 22(3):315–363, September 1997.
5. Thomas Eiter, Wolfgang Faber, Nicola Leone, Gerald Pfeifer, and Axel Polleres. System Description: The DLV$^{\mathcal{K}}$ Planning System. In Thomas Eiter, Wolfgang Faber, and Mirosław Truszczyński, editors, *Logic Programming and Nonmonotonic Reasoning — 6th International Conference, LPNMR'01, Vienna, Austria, September 2001, Proceedings*, number 2173, pages 413–416. Springer Verlag, September 2001.
6. Thomas Eiter, Wolfgang Faber, Nicola Leone, Gerald Pfeifer, and Axel Polleres. A Logic Programming Approach to Knowledge-State Planning: Semantics and Complexity. Technical Report INFSYS RR-1843-01-11, *Institut für Informationssysteme, Technische Universität Wien, December 2001*.
7. M. Gelfond, M. Balduccini, and J. Galloway. Diagnosing physical systems in a-prolog. In *LPNMR 2001*, pages 213–225, 2001.
8. M. Gelfond, M. Balduccini, and J. Galloway. The usa-advisor: A case study in answer set programming. In *JELIA 2002*, pages 566–568, Cosenza, Italy, 2002. Springer.
9. M. Gelfond and N. Leone. Logic programming and knowledge representation - the a-prolog perspective. *Artificial Intelligence*, 138:3–38, 2002.
10. M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In *In Logic Programming: Proceedings Fifth Intl Conference and Symposium*, pages 1070–1080, Cambridge, Mass., 1988. MIT Press.
11. M. Gelfond and V. Lifschitz. Classical Negation in Logic Programs and Disjunctive Databases. *New Generation Computing*, 9:365–385, 1991.
12. N. Leone, R. Rosati, and F. Scarcello. Enhancing answer set planning. In *Proceedings of the IJCAI'01*, pages 33–42, 2001.
13. V. Lifschitz. Answer set planning. In *Proceedings of the ICLP'99*, pages 23–37, 1999.
14. J. Minker. On indefinite data bases and the closed world assumption. In *Conference on Automated Deduction (CADE-82)*, pages 292–308, 1982.
15. M. Nogueira, M. Balduccini, M. Gelfond, R. Watson, and M. Barry. An a-prolog decision support system for the space shuttle. In *PADL '01*, pages 169–183, 2001.
16. Tommi Syrjänen, Lparse 1.0 User's Manual. *http://www.tcs.hut.fi/Software/smodels/lparse.ps.gz*, 2002.