Implementazione di un Interprete per Clausole Horn

Clara Pizzuti
ISI-CNR c/o DEIS
Univ. della Calabria
87030 Rende, Italy

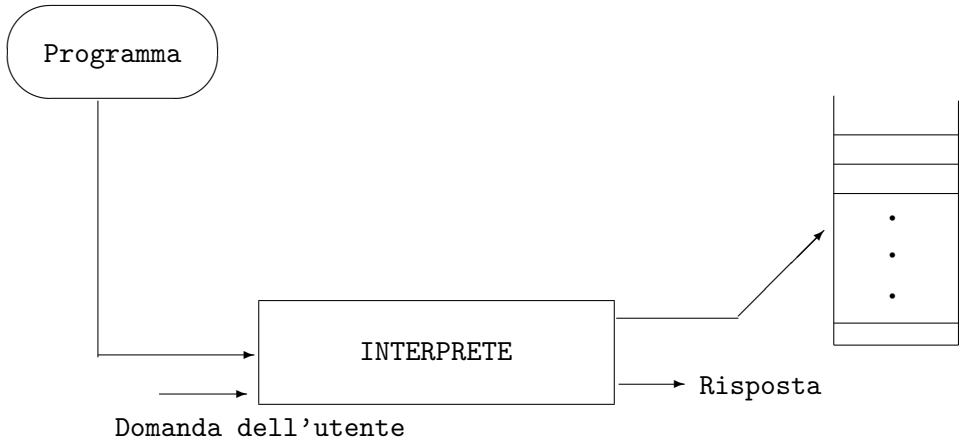

Figure 1: Interprete con le aree di memoria usate.

1 Introduzione

In qesto documento viene presentata l'implementazione di un interprete per le clausole di Horn. Il lavoro e' suddiviso in due parti. Nella prima si specifica il modo in cui viene gestito il controllo dell'interprete che esegue un programma logico, mentre nella seconda si specifica il modo in cui vengono rappresentati internamente i dati costruiti dall'interprete nel corso delle successive operazioni di unificazione.

2 Gestione del controllo

Nell'esecuzione di un programma logico, l'interprete fa uso di due principali aree di memoria come mostrato in figura 1. La prima area contiene una rappresentazione del programma logico, opportunamente codificato e compattato, e resta sempre la stessa durante tutta l'esecuzione, cioe' essa e' essenzialmente statica in quanto viene inizialmente generata dall'interprete e non viene piu' modificata. La seconda area, invece e' altamente dinamica e serve all'interprete per memorizzare la storia delle proprie azioni in modo da poter realizzare le regole di ricerca e di scelta ed il meccanismo di backtracking. Questa area e' organizzata secondo la modalita' *LIFO* (*Last in First Out*), cioe' a pila, e contiene sia le informazioni relative al controllo sia le strutture dati costruite per assegnare un valore alle variabili presenti nel goal. Nel prossimo paragrafo si richiameranno i concetti principali del meccanismo di esecuzione di un programma logico, nel successivo paragrafo, invece, vedremo il modo in cui le informazioni riguardanti il controllo sono gestite nella struttura dinamica a pila.

2.1 Meccanismo di esecuzione

L'esecuzione di un programma logico da parte dell'interprete puo' essere resa molto piu' intuitiva istituendo un parallelo tra le clausole di un programma logico e i sottoprogrammi di un linguaggio imperativo. Consideriamo infatti una generica clausola di un programma logico indicandola nella forma di regola:

$$R : -R_1, \ldots, R_k$$

dove con R abbiamo indicato la testa della regola e con R_i, con $0 \leq i \leq k$, i k goal che costituiscono la parte destra della regola (ovviamente se $k = 0$ la clausola e' un fatto). Questa clausola puo' essere letta come una dichiarazione di sottoprogramma, di cui la parte sinistra della regola e' l'intestazione e la parte destra e' il corpo. Tale corpo e' costituito da un insieme di chiamate di sottoprogrammi, ognuna rappresentata da uno dei sottogoal della parte destra della regola. La differenza piu' macroscopica con i linguaggi imperativi e' che in questo caso, ad ogni nome di sottoprogramma, cioe' ad ogni predicato, corrispondono in generale piu' corpi ognuno rappresentato da una delle clausole che hanno il predicato nella parte sinistra. L'esecuzione di una chiamata all'interno del corpo di una clausola puo' percio' avvenire in piu' modi diversi, tanti quante sono le clausole la cui parte sinistra puo' essere unificata con la chiamata. In questo senso si parla, per i linguaggi logici, di esecuzione nondetemministica, perche' alla stessa invocazione di sottoprogramma si puo' rispondere in piu' di un modo, al contrario che nei linguaggi imperativi, nei quali esiste un solo corpo di sottoprogramma e la chiamata avviene in modo univoco. L'intero processo di esecuzione di un programma logico puo', in conclusione, essere modellato come una successione di chiamate di sottoprogrammi, con entrate e uscite dalle relative clausole che ne rappresentano i corpi. La domanda stessa posta dall'utente all'interprete, cioe' il goal iniziale

$$G: \quad :-R_1, \ldots, R_k$$

da dimostrare, puo' essere pensata come il corpo di un sottoprogramma che e' stato invocato al momento dell'inizio della esecuzione. L'entrata e l'uscita dai sottoprogrammi rappresentati dalle clausole del programma logico fanno variare il **punto di controllo**, termine col quale si indica astrattamente un indice alla parte del programma correntemente eseguita. Una entrata in una clausola avviene, all'interno del corpo di una regola, nel tentativo di rispondere ad una chiamata costituita da uno dei suoi sottogoal. All'inizio di ogni chiamata il punto di controllo avra' raggiunto una certa chiamata all'interno di un corpo di sottoprogramma. Rappresentiamo un generico sottoprogramma con la clausola

$$P: \quad A :- P_1, P_2, \ldots, P_m$$

e supponiamo che, in una ipotetica esecuzione, il punto di controllo abbia raggiunto la chiamata P_i. Cio' significa che tutte le chiamate precedenti sono state risolte, anche se non in tutti i possibili modi. In generale, piu' di una clausola puo' essere usata nel tentativo di rispondere alla chiamata P_i dal momento che possono esistere piu' clausole per il predicato ad essa corrispondente. L'insieme delle clausole che possono essere usate nel tentativo di rispondere ad una chiamata e' detto **l'insieme delle candidate** per quella chiamata. In questo esempio, puo' darsi che alcune delle candidate per P_i siano gia' state usate in precedenti tentativi di soddisfare quella chiamata. Indichiamo con Q, Q', Q", quelle che rimangono ancora intentate, in ordine di priorita' decrescente. Supponiamo che Q abbia la forma:

$$Q: \quad B :- Q_1, Q_2, \ldots, Q_n$$

in cui B e' unificabile con P_i. Questa entrata nella clausola Q sposta il punto di controllo da P_i alla prima chiamata, Q_1, della procedura Q. All'atto della entrata nella nuova procedura alcuni parametri riguardanti questa invocazione devono essere registrati dall'interprete. L'entrata in Q in risposta alla chiamata P_i inizia il tentativo di rispondere a tale chiamata, risolvendola. Se l'interprete riesce a rispondere a tutte le chiamate Q_i del corpo di Q, esso esegue una **uscita con successo** da P_i, dopo la quale il punto di controllo viene spostato alla chiamata P_{i+1} che e' la prossima chiamata ad essere attivata. Se invece risolvere una

qualsiasi delle chiamate di Q si rivela impossibile, si ha una **uscita con fallimento** da P_i, ed il punto di controllo ritorna alla chiamata P_i, che viene riattivata con la prossima candidata Q'. Questo comportamento viene chiamato **backtracking dopo fallimento**. Se anche il tentativo di risolvere P_i attraverso Q' o una qualsiasi delle altre candidate di P_i si rivela inutile, allora avviene un'ulteriore mossa di backtrack e il punto di controllo viene arretrato alla chiamata P_{i_1} di P e via via ai suoi predecessori.

Se l'esecuzione risolve tutte le chiamate in G allora l'interprete prima restituisce la risposta e poi trasferisce il controllo indietro alla chiamata attivata piu' di recente, se esiste, per cui vi erano ancora candidate intentate e la riattiva per cercare nuove soluzioni. Questo comportamento in seguito ad una uscita con successo da G e' chiamato **backtracking dopo successo**. Se nessuna di queste chiamate esiste dopo un'uscita con successo da G allora tutte le chiamate sono state tentate in tutti i modi possibili e quindi l'esecuzione termina.

2.2 I record di attivazione

Come intuibile dalla precedente informale descrizione della gestione delle successive entrate e uscite dalle clausole di un programma logico, l'interprete deve registrare informazioni riguardanti l'evoluzione del punto di controllo per due principali ragioni. La prima e' la necessita', a seguito della uscita con successo da una chiamata interna al corpo di una clausola, di continuare l'esecuzione dalla chiamata successiva, per arrivare a dimostrare la procedura corrente e, alla fine, rispondere alla domanda posta dall'utente. La seconda ragione e' la necessita' di eseguire l'operazione di backtrack, sia a seguito del fallimento di una chiamata per intraprendere un nuovo tentativo con una candidata non ancora sperimentata, sia a seguito della scoperta di una soluzione, per trovare le rimanenti, se ne esistono. Le informazioni necessarie a questi scopi vengono immagazzinate in una struttura di dati chiamata, in analogia a quanto succede nei linguaggi convenzionali, **record di attivazione**. La regola della quale si stanno correntemente eseguendo le chiamate e' quella attiva, dal momento che il punto di controllo e' entrato in essa ma non ne e' ancora uscito. Ad ogni entrata in una regola viene creato un esemplare di questa struttura dati, che rimane in vita fino a che non e' stato portato a termine il tentativo di rispondere alla chiamata attraverso quella regola. Mostriamo quali informazioni vengono registrate nel record di attivazione considerando cio' di cui ha bisogno l'interprete per spostare il punto di controllo. Riprendendo l'esempio del paragrafo precedente supponiamo venga creato un record di attivazione R al momento in cui la chiamata P_i nel sottoprogramma

$$P: \quad A :- P_1, P_2, \ldots, P_i, \ldots, P_m$$

causa l'entrata nella regola

$$Q: \quad B :- Q_1, Q_2, \ldots, Q_n$$

Il record di attivazione R rappresenta percio' l'entrata in Q. Nel caso in cui l'esecuzione proceda con successo e tutti i Q_i possano essere soddisfatti, l'esecuzione dovra' procedere, dopo la chiamata di Q_n con la chiamata di P_{i+1}, che segue P_i all'interno di P. Perche' cio' sia possibile il record di attivazione deve contenere un valore che funga da **punto di ritorno** (PR) e indichi P_{i+1}. Concretamente, il punto di ritorno sara' l'indirizzo della chiamata che segue quella associata al record di attivazione, nel codice del programma logico che viene interpretato. Il punto di ritorno pero' da solo non e' sufficiente, perche' l'interprete deve essere in grado di individuare qual e' il record di attivazione del sottoprogramma chiamante. Per questo motivo ogni record di attivazione deve anche possedere un **puntatore al record**

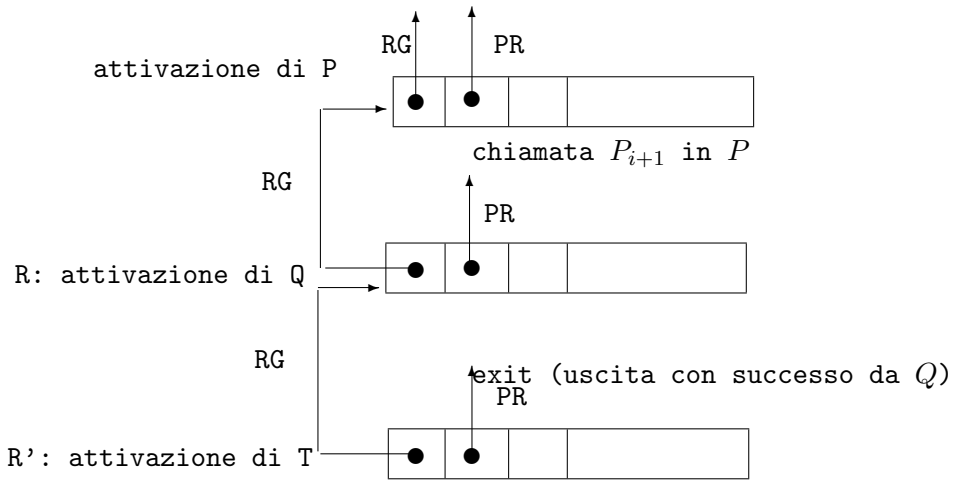

Figure 2: Informazioni nei record di attivazione per l'uscita con successo da P

.

genitore (RG), indicante il record che rappresenta l'attivazione della regola contenente la chiamata attualmente in esecuzione. Nel nostro esempio, il valore del puntatore RG puo' essere definito come segue: per tutti i record di attivazione creati attivando regole in risposta a chiamate contenute nella regola Q, il campo RG punta al record R, dal momento che R rappresenta l'entrata nella regola Q. Quando un record di attivazione corrisponde all'ultima chiamata del corpo di una regola, il suo campo PR viene posto ad un particolare valore convenzionale che indica l'uscita con successo dalla regola corrente.

Example 1 Supponiamo che dopo l'entrata in Q l'esecuzione proceda con successo per tutti i Q_i che si susseguono nel suo corpo, fino ad arrivare all'ultimo di essi, Q_n e che la chiamata di Q_n provochi l'invocazione di un fatto T. La situazione e' rappresentata in figura 2, dove sono riportati gli ultimi tre record di attivazione rimasti in uso: quelli di P, di Q e di T (nota che i record di attivazione mostrano di contenere altre informazioni: queste saranno introdotte nella prossima sezione).

Poiche' Q_n e' l'ultima chiamata di Q, il suo record di attivazione R' contiene, nel campo PR, il valore convenzionale, indicato con *exit*. Poiche' T e' un fatto e quindi non contiene alcuna chiamata, I'interprete compie un'uscita con successo da T. Consultando il campo PR di R' esso trova il valore *exit* che indica l'uscita con successo da Q, e percio' si sposta, facendo uso del puntatore RG di R', sul record genitore R, ed esegue una seconda operazione di uscita, spostando il punto di controllo alla chiamata P_{i+1}.

Si osservi che la figura 2 rappresenta tre passi (attivazione di P, Q e T) che fanno parte del seguente segmento di computazione:

$\ldots$

attivazione di P: $?P_1, \ldots, P_m, \ldots$

$\ldots$

$?P_i, P_{i+1} \ldots, P_m, \ldots$

attivazione di Q: $?Q_1, \ldots, Q_n, P_{i+1} \ldots, P_m, \ldots$

...

$?Q_n, P_{i+1} \ldots, P_m, \ldots$

attivazione di T:

uscita da T:

uscita da Q: $?P_{i+1} \ldots, P_m, \ldots$

...

I meccanismi appena introdotti sono sufficienti per rendere possibile l'entrata nelle regole e l'uscita con successo, in modo del tutto analogo a quanto avviene per le invocazioni di sottoprogrammi nei linguaggi procedurali. Tuttavia, informazioni aggiuntive si rendono necessarie per gestire le forme di nondeterminismo che derivano dalla presenza di piu' clausole per lo stesso predicato, combinate con l'operazione di backtracking.

2.3 Backtracking

Continuiamo a riferirci alle clausole P e Q del paragrafo precedente, supponendo ora che, al contrario di quanto avveniva prima, dopo che e' stata effettuata l'entrata in Q in risposta alla chiamata P_i, contenuta in P, una certa Q_j, contenuta in Q, si riveli irresolubile, rendendo quindi impossibile rispondere alla chiamata P, mediante Q. In questo caso deve avvenire il backtracking, e perche' cio' accada l'interprete deve essere in grado di compiere due operazioni: in primo luogo, deve ricordare l'ultima, tra le chiamate recentemente attivate, per la quale erano rimaste delle clausole candidate non ancora tentate, e in seguito deve attivare la prima (cioe' quella a maggiore priorita') tra di esse. Per soddisfare il primo di questi due requisiti, l'interprete necessita semplicemente di un registro, chiamato PRB, che viene mantenuto durante tutta l'esecuzione, e che punta sempre al piu' recente record di attivazione corrispondente ad una chiamata per la quale c'era almeno ancora una candidata non provata, oltre a quella attualmente invocata. Un record di attivazione che abbia queste caratteristiche viene chiamato **punto di backtrack**, e il nome del registro dell'interprete necessario per l'operazione di backtrack significa, appunto, **Piu' Recente punto di Backtrack (PRB)**. Occorre mettere in rilievo che, al contrario dei puntatori PR ed RG introdotti nel paragrafo precedente per gestire attivazione e disattivazione di regole, PRB non e' un campo del record di attivazione di qualche clausola, ma e' un registro dell'interprete, di cui esiste una sola copia nel corso di tutta l'esecuzione. Considerando l'interprete di un linguaggio come una macchina astratta per l'esecuzione di programmi di quel linguaggio, PRB e' un registro di tale macchina astratta, e non fa parte della porzione di memoria gestita dinamicamente nel corso dell'esecuzione. Per illustrare il modo in cui PRB viene gestito, consideriamo il momento appena precedente la creazione del record di attivazione che rappresenta l'entrata nella regola Q. A quel punto, PRB punta a qualche altro record di attivazione R che e' correntemente il piu' recente punto di backtrack, come mostrato in figura 3(a). Dal momento che la chiamata P_i, ha altre candidate Q' e Q", oltre a Q, il record R e' un nuovo punto di backtrack e quindi da ora in avanti PRB punta ad esso, come mostrato in figura 3(b).

Naturalmente il registro PRB non e' da solo sufficiente per realizzare il backtrack. Consideriamo cosa deve accadere quando Q_j si rivela irresolubile. Possiamo supporre, senza perdere in generalita', che nel frattempo non siano stati creati altri punti di backtrack e che quindi PRB punti ancora ad R. A questo punto l'interprete deve svolgere la seconda operazione connessa al backtrack, cioe' ricordarsi di Q', la prima candidata intentata per la chiamata P_i. Il modo piu' semplice di rendere cio' possibile e' quello di memorizzare un **puntatore alla prossima candidata (PC)** in un campo del record di attivazione R.

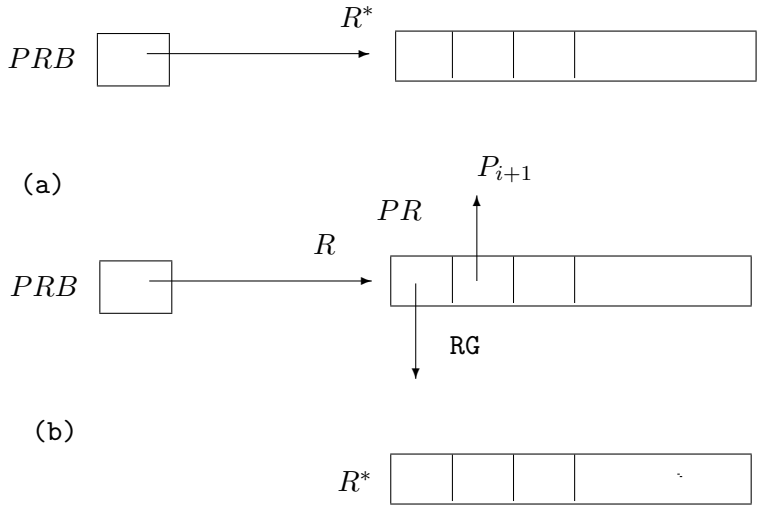

Figure 3: Creazione di un nuovo punto di backtrack R.

Quando Q_j fallisce, l'interprete consulta PRB per trovare R e poi trova nel suo campo PC la prima candidata Q'. La chiamata da riattivare con questa nuova candidata e' identificata dal puntatore PR nel record R, che punta a P_{i+1} successore di P_i, in P. Una volta che tutte queste informazioni sono state recuperate, il record R e tutti i successivi, rappresentanti il tentativo di risolvere P_i attraverso Q, possono essere scartati, per proseguire normalmente l'esecuzione attraverso l'attivazione di Q'. Con cio' si completa l'esecuzione della operazione di **backtrack dopo fallimento**; il **backtrack dopo successo** (che avviene dopo che e' stata trovata una soluzione ed ha lo scopo di trovare quelle successive, se esistono) avviene in maniera del tutto analoga, eccetto che prima le soluzioni ottenute vengono mostrate all'utente. Dal fatto che al momento del backtrack tutti i record di attivazione a partire dal piu' recente punto di backtrack possono essere scartati, e' possibile inferire che i record di attivazione vengono gestiti a pila: i primi record ad essere cancellati sono gli ultimi inseriti.

Abbiamo visto il modo in cui al registro PRB viene assegnato un nuovo valore all'atto della creazione di un record di attivazione che sia anche punto di backtrack. Notiamo che, in modo simmetrico, il registro PRB deve essere ripristinato al suo valore precedente, durante l'operazione di backtrack. Considera per esempio l'attimo in cui il record R sta per essere scartato durante il backtrack. In questo momento, in virtu' della nostra precedente assunzione, R e' il precedente punto piu' recente punto di backtrack, e PRB punta ad esso. Quando R viene scartato, PRB deve assumere nuovamente il suo valore precedente $R^{\sim}$, cosi che l'interprete non perda traccia delle strade non ancora percorse nella dimostrazione. Per poter identificare $R^{\sim}$ al momento opportuno si puo' fare in modo che il precedente valore di PRB, che puntava ad esso, sia copiato nel record di attivazione R di Q, appena prima di assegnare a PRB il suo nuovo valore che punta ad R. Il campo del record di attivazione in cui copiare il vecchio valore di PRB viene chiamato PB (**Precedente punto di Backtrack**); quando avviene il backtrack, il suo contenuto viene copiato nel registro PRB appena prima di scartare il record di attivazione, e in tal modo l'interprete puo' mantenere traccia dei progressi fatti nel corso della computazione. C'e' da notare che i campi PC (**prossima candidata**) e PB (**precedente punto di backtrack**) sono significativi, in un record di attivazione, solo se esso e' un punto di backtrack, cioe' esistono altre candidate per la chia-

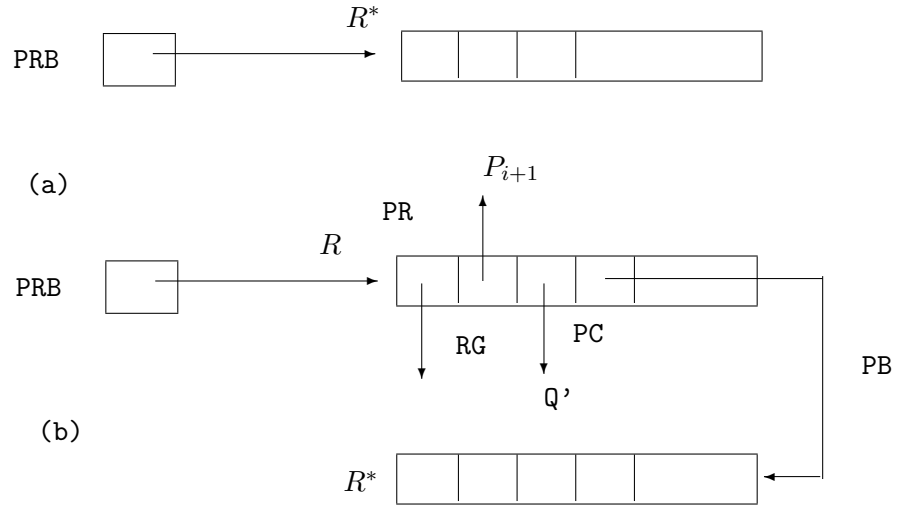

Figure 4: Informazioni complete immagazzinate nel punto di backtrack. (a) prima della creazione di R, (b) dopo la creazione di R.

PRB : registro Piu' Recente punto di Backtrack;
PR : punto di ritorno;
RG : puntatore al record genitore;
PC : puntatore alla prossima candidata;
PB : puntatore precedente punto di backtrack;
R^* : puntatore al record di backtrack precedente.

mata cui tale record corrisponde. La figura 4 mostra l'informazione completa mantenuta al momento della creazione del nuovo punto di backtrack R corrispondente alla attivazione di Q in risposta a P_i.

Il campo RG punta al record creato al momento dell'entrata in P, il campo PR indica la chiamata da eseguire dopo la risoluzione di P_i, cioe' P_{i+1}, la cella PC indica la prossima candidata Q' da tentare come risposta alternativa alla chiamata P_i, mentre PB punta al record $R^{\sim}$ che era precedentemcnle il piu' recente punto di backtrack. Poiche' R e' a sua volta un punto di backtrack, PRB e' stato aggiornato e punta ad esso.

2.4 L'invocazione di una regola da parte dell'interprete

Vediamo ora quali sono i passi eseguiti dall'interprete all'atto della invocazione di una regola. Oltre alle strutture dati dinamiche costituite dai record di attivazione gestiti a pila, l'interprete fa uso di un certo numero di registri. Per semplicita' di linguaggio, nel seguito useremo, ove cio' non crei confusione, il nome di un registro sia per indicare il registro stesso sia per denotare l'oggetto individuato dal contenuto del registro. Per esempio, diremo "il record di attivazione PRB" per indicare il record di attivazione individuato dal valore contenuto in PRB. All'inizio di una invocazione di regola, l'interprete avra' selezionato una chiamata all'interno della regola correntemente attiva. L'interprete ha un registro, chiamato CC (**Chiamata Corrente**) che contiene un puntatore a questa chiamata. Un altro registro, chiamato PCA (**Prossima CAndidata**) punta alla successiva candidata da tentare in risposta a questa chiamata (comunque non e' affatto detto che questa candidata esista ad ogni attivazione di regola: cio' accade solo se il record che sta per essere creato e' un

punto di backtrack). Il primo compito dell'interprete nel rispondere ad una chiamata e' di trovare una clausola che risponda alla chiamata CC. Possiamo immaginare che esista un qualche meccanismo che permetta all'interprete di scorrere sequenzialmente il programma e trovare le clausole che possono rispondere alla chiamata, realizzando cosi' la regola di ricerca. Scorrendo tra le candidate nel loro ordine testuale, l'inteprete cerca la prima che:

- possa rispondere alla chiamata CC, e inoltre
- sia posta (nell'ordine testuale del programma) in posizione PCA o in una posizione successiva.

L'effetto di questa ricerca e' duplice. Se viene trovata una clausola che e' una candidata alla chiamata CC, allora l'interprete immagazzina un puntatore ad essa in un apposito registro chiamato RC (**Regola Corrente**). Se nessuna candidata viene trovata il registro RC viene posto uguale al puntatore nullo $\perp$. L'altro effetto di questa ricerca della clausola e' di aggiornare il valore di PCA. Quando viene trovata una clausola, a PCA viene assegnato il puntatore alla candidata successiva, se ne esiste una; altrimenti gli viene assegnato il valore nullo $\perp$. Il valore assegnato a PCA assicura che in caso di futuro backtracking che porti a invocare di nuovo CC, la regola RC non venga di nuovo attivata, ma venga attivata una nuova candidata che si trovi ad un punto non precedente PCA. In conclusione, l'invocazione della chiamata corrente CC puo' sortire tre diversi effetti:

1. $RC = \perp$. Cio' significa che nessuna regola non precedente a PCA puo' essere usata per rispondere alla chiamata CC. Non e' quindi possibile rispondere alla chiamata CC e occorre operare un passo di backtracking, alla ricerca di una precedente chiamata che possa essere invocata, avendo delle candidate non ancora attivate. Se l'operazione di backtrack ha successo, l'esecuzione riprende con l'invocazione di questa chiamata, altrimenti l'esecuzione termina con un fallimento.

2. $RC \neq \perp$ e $PCA = \perp$. In questo caso si dice che viene eseguita una **attivazione deterministica**, in cui un nuovo record viene creato nella pila a rappresentare l'invocazione della regola RC. L'attivazione e' detta *deterministica* perche' la $PCA = \perp$ sta proprio ad indicare che non esistono candidate alternative a RC e quindi la chiamata CC puo' essere risolta in un solo modo, e cioe' attivando RC. Il record creato per l'attivazione di RC non sara' un punto di backtrack, e pertanto non necessitera' delle celle PC e PB, ma solo di PR e RG.

3. $RC \neq \perp$ e $PCA \neq \perp$. In questo caso ha luogo una **attivazione nondeterministica**, in cui viene creato un nuovo record per l'attivazione di RC, che sara' un punto di backtrack e necessitera' percio' di tutte e quattro le celle. L'attivazione e' detta "nondeterministica" perche' esistono piu' candidate alla soluzione della chiamata CC. In caso di backtracking la prima candidata ad essere tentata per risolvere CC sara' PCA.

2.5 L'algoritmo di controllo

Il processo di esecuzione dei programmi logici verra' ora descritto in modo formale dando l'algoritmo di controllo dell'interprete. Tale interprete realizza la strategia standard di Prolog con la regola di computazione, la regola di ricerca e il meccanismo di backtrack noti. La gestione di queste attivita' e' ottenuta mediante la creazione dei record di attivazione che rappresentano lo stato dell'esecuzione. I record di attivazione sono gestiti a pila: ne viene

creato uno nuovo ad ogni attivazione di una regola in risposta ad una chiamata, e ne vengono cancellati un certo numero, durante le operazioni di backtrack. Oltre all'informazione mantenuta nella pila, l'interprete mantiene in un certo numero di registri, informazioni importanti riguardanti le operazioni in corso di esecuzione. Molti di essi sono gia' stati introdotti. Immediatamente dopo la creazione di un nuovo record di attivazione il loro contenuto e' il seguente:

- il registro PRB (**Piu' Recente punto di Backtrack**) contiene un puntatore all'ultimo record di attivazione che sia un punto di backtrack, se ne esiste uno, altrimenti vale $\perp$;
- il registro RC (**Regola Corrente**) punta alla regola in cui si e' entrati attraverso la creazione del record di attivazione corrente;
- il registro CC (**Chiamata Corrente**) punta alla chiamata che ha causato l'attivazione della regola corrente;
- il registro PCA (**Prossima Candidata**) punta alla prima candidata ancora intentata, se ne esiste una, altrimenti vale $\perp$;
- un nuovo registro, chiamato PRG (**Piu Recente Genitore**), punta al record di attivazione genitore di quello corrente.
- Un altro nuovo registro, chiamato t (**top della pila**), individua la posizione nella pila del record di attivazione corrente. Senza fare alcuna ipotesi sul modo in cui la pila viene implementata, supporremo che le successive posizioni in essa siano individuate da numeri interi crescenti a partire da 1, e che l'accesso ai campi di ogni singolo record di attivazione avvenga mediante un operatore che ha lo stesso nome del campo e riceve come parametro la posizione del record sulla pila. Cosi', per esempio, la scrittura $PR(t)$ indica il campo **punto di ritorno** del record corrente.

L'algoritmo di controllo viene dato qui di seguito mediante cinque passi, ognuno dei quali scritto con un linguaggio simile al Pascal. Per denotare le parti del programma e della pila dei record di attivazione attraverso i registri useremo la seguente notazione: se r e' un registro, $r \rightarrow$ indica l'oggetto puntato dal registro e se x e' un oggetto, $\rightarrow x$ indica il suo indirizzo, cioe' il valore da mettere in un registro che si vuole punti ad esso.

Passo 1 (Inizializzazione):

t:=1;

$RC :=\rightarrow$ goal di ingresso;

$PRG :=\perp$;

$PRB :=\perp$;

$RG(1) :=\perp$;

$PR(1) := exit$;

Il primo passo crea il primo record di attivazione corrispondente al goal iniziale, che si presume contenga almeno una chiamata. Questo primo record di attivazione non e' un punto di backtrack.

Passo 2 (Selezione della chiamata):

if $RC \rightarrow$ e' un fatto

then

$CC := PR(t)$;

while CC= **exit and** $PRG \neq 1$
do
$CC := PR(PRG)$;
$PRG := RG(PRG)$;
od;
if CC = **exit**
then
scrivi in uscita la soluzione;
goto passo 5;
fi
else
$CC :=\rightarrow$ prima chiamata in RC;
$PRG := t$;
fi;
if non esistono candidate per $CC \rightarrow$
then goto passo 5;
else $PCA :=\rightarrow$ prossima candidata;
fi;

Nel secondo passo se RC e' un fatto allora la chiamata corrente e' soddisfatta e occorre cercare la prossima chiamata attraverso una o piu' operazioni di ritorno. Se non esiste alcuna chiamata significa che il goal iniziale e' stato risolto e si scrive in uscita la soluzione. Altrimenti CC e' posto uguale alla prima chiamata di RC. Dal momento che il record corrente t significa entrata in RC, esso diventera' il piu' recente genitore dopo il passo corrente, e quindi PRG viene posto uguale ad esso. Se CC non ha candidate allora si fa backtrack, altrimenti si pone PCA uguale alla prima delle candidate.

Passo 3 (Selezione della regola):
Usando i valori correnti di CC e PCA,
trova la regola con cui rispondere a CC
nel modo indicato nel paragrafo precedente,
aggiornando RC e (eventualmente) PCA.
if $RC = \perp$
then goto passo 5
fi;

Nel terzo passo si tenta l'unificazione di CC con la testa delle sue candidate, iniziando dalla posizione PCA. Il fallimento causa backtrack e il successo assegna a RC la prima candidata attivabile, e a PCA la prima intentata, se esiste.

Passo 4 (Creazione di un record di attivazione):
$t := t + 1$;
$RG(t) := PRG$;
if $CC \rightarrow$ e' l'ultima chiamata di $RC \rightarrow$
then $PR(t) :=$ **exit**;
else $PR(t) :=\rightarrow$ chiamata che segue $CC \rightarrow$

fi

if $PCA \neq \perp$

then

$PC(t) := PCA$;

$PB(t) := PRB$;

$PRB := t$;

fi;

goto passo 2;

Il quarto passo crea il record di attivazione di RC in risposta alla chiamata CC. Il record e' posto nella posizione $t+1$ della pila, e le sue celle RG e PR sono assegnate usando i registri PRG e CC. Se $PCA \neq \perp$ allora esistono altre candidate per CC e il record di attivazione e' anche punto di backtrack; in questo caso i campi PC e PB vengono assegnati usando PCA e PRB, e poi PRB viene aggiornato per puntare a questo nuovo record.

Passo 5 (Backtracking):

if $PRB = \perp$

then termina l'esecuzione

else

$PCA := PC(PRB)$;

$CC :=\rightarrow$ chiamata precedente a $PR(PRB)$;

$PRG := RG(PRB)$;

$t := PRB - 1$;

$PRB := PB(PRB)$;

fi;

goto passo 3;

Nel quinto passo se $PRB = \perp$ allora il backtrack e' impossibile e l'esecuzione termina. Altrimenti il piu' recente punto di backtrack PRB viene usato per identificare la chiamata da reinvocare e la prossima candidata da attivare. La riduzione di t a $PRB - 1$ elimina dalla pila tutti i record di attivazione a partire dal punto di backtrack incluso.

E' da notare come sia stato assunto che i puntatori alla rappresentazione statica del programma logico, quando puntano ad una chiamata, permettano implicitamente di individuare anche la clausola cui tale chiamata appartiene e quindi anche la chiamata precedente o quella successiva. Tale assunzione e' stata fatta, in particolare, ai passi 4 e 5 per il registro CC e il campo PR del record di attivazione.

Example 2 Consideriamo il seguente programma logico in cui le clausole sono state numerate per stabilire un ordinamento implicito:

1. capoDi(emilio,francesco).
2. capoDi(franca,cesare).
3. capoDi(franca,emilio).
4. maschio(emilio).
5. maschio(francesco).
6. maschio(cesare).
7. femmina(franca).
8. donnaAcapo(X,Y):-femmina(X),capoDi(X,Y).

Clausola attivata	Posizione del record	Contenuti del record			
		RG	PR	PC	PB
Goal iniziale	1	$\perp$	exit	-	-
8	2	1	exit	-	-
7	3	2	2^a chiamata di 8	-	-
2	4	2	exit	$\rightarrow 3$	$\perp$

Figure 5: Configurazione della pila corrispondente alla risposta A=franca, B=cesare

e la domanda *donnaAcapo(A,B)*. Al termine della allocazione del record di attivazione della clausola 2 che porta alla risposta $A = franca$, $B = cesare$, la pila dei record di attivazione ha la configurazione mostrata in figura 5. Per facilitarne la lettura, la pila e' rappresentata come crescente verso il basso, con i record piu' "giovani" che occupano le posizioni inferiori. I contenuti dei registri dell'interprete sono i seguenti:

$PRG = 2$ (il record 2 e' il piu' recente genitore)

$PRB = 4$ (il record 4 e' il piu' recente punto di backtrack)

$t = 4$ (sulla pila sono presenti quattro record di attivazione)

$RC = \rightarrow 2$ (la regola 2 e' l'ultima regola attivata)

$CC = \rightarrow$ seconda chiamata di 8 (che ha attivato la regola 2).

I primi tre record di attivazione non sono punti di backtrack, e quindi in essi i campi PC (**Prossima Chiamata**) e PB (**Precedente punto di Backtrack**) non sono stati inizializzati. Il campo PB del record 4 punta al precedente punto di backtrack, che e' $\perp$.

Supponiamo adesso che si voglia calcolare un'altra risposta. Questo e' possibile in quanto il piu' recente punto di backtrack non e' nullo ($PRB = 4$). Si entra allora nel *passo 5* e si eseguono le seguenti operazioni: $PCA = PC(PRB) = PC(4) = \rightarrow 3$, $CC = capoDi(franca, Y)$, $PRG = RG(4) = 1$, $t = 3$, $PRB = PB(4) = \perp$, quindi il record 4 viene distrutto e la computazione ricomincia cercando un'altra alternativa per $CC = capoDi(franca, Y)$.

3 La Rappresentazione dei Dati

3.1 Introduzione

Nella sezione precedente ci siamo occupati di specificare il modo in cui viene gestito il controllo dell'interprete che esegue i programmi di un linguaggio logico. Abbiamo messo in rilievo che l'interprete ha bisogno di un'area in cui mantenere una rappresentazione codificata e statica del programma e di una altamente dinamica in cui tenere traccia dello stato dell'esecuzione mediante una pila di record di attivazione, ognuno rappresentante l'entrata in una clausola in risposta ad una chiamata contenuta nel corpo di una regola. Siamo arrivati a dare l'algoritmo di controllo e l'insieme dei registri dell'interprete, cioe' a definire la macchina astratta che lo realizza. Nulla pero' e' stato ancora detto riguardo al modo in cui vengono rappresentati internamente i dati costruiti dall'interprete nel corso delle successive operazioni di unificazione, che al termine dell'elaborazione costituiscono la risposta alla domanda posta dall'utente. La questione della rappresentazione interna dei dati non ha una risposta univoca, nel senso che esistono numerose diverse tecniche, anche assai sofisticate,

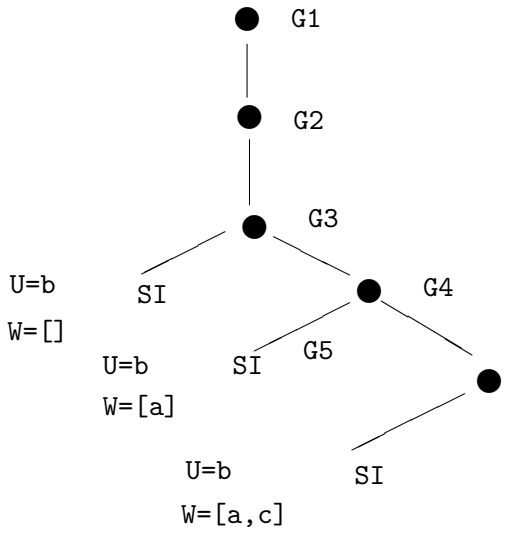

Figure 6: Albero di prova semplificato per la domanda: $?prefisso([U, a \mid W], [b, a, a, c])$

che permettono di minimizzare lo spazio occupato, o il tempo necessario per costruire le strutture di dati. L'efficacia di tali tecniche non vale in assoluto e indipendentemente dai programmi e dalle domande che occorre valutare, ma varia grandemente in funzione del tipo di computazione che e' necessario svolgere. Non esiste quindi un metodo di rappresentazione dei dati che sia assolutamente migliore di tutti gli altri, ma esiste un insieme di tecniche la cui relativa convenienza ed efficacia va valutata caso per caso.

3.2 Rappresentazione mediante pila singola

Qualsiasi decisione sul modo di rappresentare le strutture dati create nell'esecuzione di un programma logico, deve partire dalla considerazione che le variabili locali contenute in ogni clausola possono ricevere, ad ogni attivazione di tale clausola, dei valori differenti. Per questo motivo e' necessario, ad ogni attivazione di una clausola, creare nuove copie di queste variabili o, detto in altri termini, allocare una nuova porzione di spazio di memoria per contenere i loro valori nella nuova attivazione. Viene allora spontaneo pensare di allocare questo nuovo spazio di memoria, che deve venire allocato ad ogni attivazione di una clausola, nel record di attivazione che viene pure creato ogni volta che una candidata viene attivata per rispondere ad una chiamata. Mostriamo come questo metodo puo' essere realizzato su un esempio di programma logico in esecuzione su di una domanda. Consideriamo il programma logico costituito dalle due clausole che definiscono il predicato $prefisso(X, Y)$, che indica che la stringa X e' un prefisso della stringa Y. Per poterci poi riferire ad esse simbolicamente all'interno dei record di attivazione diamo alle due clausole del programma i nomi $P1$ e $P2$.

$P1 : prefisso([], Y)$

$P2 : prefisso([V \mid X], [V \mid Y]) : -prefisso(X, Y)$

In questo programma le stringhe sono rappresentate mediante liste, e i componenti di una stringa corrispondono agli elementi della lista. Si tratta di un tipico programma ricorsivo in cui la prima clausola costituisce la base della ricorsione e dice che la stringa vuota [] e' prefisso di qualsiasi stringa, e la seconda dice che una stringa e' prefisso di un'altra se

Clausola attivata	Posizione del record	Contenuti del record				
		RG	PR	PC	PB	Variabili
G1	1	$\perp$	exit	-	-	$U_1 = V_2$, $W_1 = X_3$
P2	2	1	exit	-	-	$V_2 = b$, $X_2 = [a \mid W_1]$,$Y_2 = [a, a, c]$
P2	3	2	exit	-	-	$V_3 = a$, $X_3 = [a \mid X_4]$,$Y_3 = [a, c]$
P2	4	3	exit	-	-	$V_4 = a$, $X_4 = []$, $Y_4 = [c]$
P1	5	4	exit	$\rightarrow P_2$	$\perp$	$Y_5 = [c]$

Figure 7: I record di attivazione corrispondenti alla computazione $G1$-$G5$

entrambe iniziano con la stessa variabile e la stringa ottenuta dalla prima togliendole la testa e' prefisso di quella ottenuta dalla seconda mediante la stessa operazione. La domanda posta all'interprete, indicata con G1, e' la seguente

$$G1 :?prefisso([U, a \mid W], [b, a, a, c])$$

L'albero di prova costruito applicando la solita strategia e' mostrato, in forma semplificata, in figura 6. Occupiamoci in particolare della successione di goal G1- G5, che porta alla seconda delle tre soluzioni possibili. I goal e gli assegnamenti a variabili locali sono i seguenti

$G1 :?prefisso([U, a \mid W], [b, a, a, c])$

$U = b$

$G2 :?prefisso([a \mid W], [a, a, c])$

$G3 :?prefisso(W, [a, c])$

$W = [a \mid X]$

$G4 :?prefisso(X, [c])$

$X = []$

$G5 : SI$ risposta e' $U = b, W = [a]$

I primi tre passi della computazione invocano ognuno la clausola P2, le cui variabili locali sono V, X, e Y; ogni record di attivazione della clausola $P2$ deve percio' contenere, oltre ai campi necessari per la gestione del controllo, anche spazio per una nuova copia di queste variabili. I contenuti della pila dei record di attivazione dopo l'entrata in $P1$ corrispondente al goal G5 e' mostrata in figura 7. Lo stato dei registri dell'interprete immediatamente dopo la creazione del quinto record di attivazione e' il seguente

$PRG = 4$ (record 4 e' il piu recente record genitore);

$PRB = 5$ (record 5 e' il piu' recente, oltre che unico, punto di backtrack);

$t = 5$ (sono stati creati cinque record di attivazione);

$RC =\rightarrow P1$ (la regola appena attivata);

$CC =\rightarrow$ prima chiamata in $P2$ (che ha attivato la regola RC).

Nella figura 7, ad ogni riga corrisponde un record di attivazione diverso. I record di attivazione, oltre ai campi necessari per la gestione del controllo introdotti nella sezione precedente, contengono delle celle di memoria, uno per ogni variabile presente nella regola attivata. Ad ogni istante della valutazione, queste celle possono contenere il valore della variabile che rappresentano, oppure un simbolo speciale, per esempio '*', che significa "valore non ancora calcolato". Se il valore e' gia' stato calcolato, esso puo' essere una semplice

costante, nel qual caso puo' venire scritto direttamente nella cella assegnata alla variabile. Se invece il valore della variabile e' una struttura ingombrante in termini di occupazione di spazio di memoria, allora esso deve essere immagazzinato a parte e la cella contiene un puntatore ad esso. Infine, il valore assegnato ad una variabile puo' contenere un riferimento ad un'altra variabile cio' succede, ad esempio, per la variabile X_3 che contiene un puntatore alla variabile W_1.

I nomi delle variabili sono mostrati nella rappresentazione della pila in figura 7, ma non sono realmente presenti nei record di attivazione delle clausole. Invece, l'identita' di una cella e' stabilita implicitamente dalla sua posizione all'interno del record di attivazione, in modo del tutto analogo a quanto succede nei record di attivazione dei linguaggi di programmazione convenzionali, in cui ad ogni variabile di un sottoprogramma e' assegnata una posizione all'interno del corrispondente record di attivazione. Quindi, nel nostro esempio, l'interprete puo' sempre assumere che le variabili V, X e Y abbiano nel record di attivazione della regola $P2$, la prima, seconda e terza posizione, rispettivamente. Il fatto che ogni esemplare di variabile sia allocato all'interno di un dato record di attivazione permette automaticamente di distinguere tra variabili della stessa clausola create in successive distinte attivazioni.

Quando un nuovo record di attivazione viene creato, alcune delle celle corrispondenti alle variabili locali alla clausola attivata, ma non necessariamente tutte, vengono inizializzate ad un valore ottenuto mediante l'operazione di unificazione col goal che ha causato l'attivazione della clausola; le altre vengono provvisoriamente settate a '*' e rimangono in attesa di assegnazione. Ad esempio, le variabili U_1, W_1 e X_4 sono tutte non assegnate, quando vengono creati i rispettivi record di attivazione, 1 e 4. Piu' tardi, U_1 riceve il valore b al momento della creazione del record 2; W_1 riceve il valore $[a \mid X_4]$ al momento della creazione del record 4; X_4 riceve il valore [] (che denota la lista vuota) al momento della creazione del record 5.

Cio' mostra anche che il processo di unificazione al momento della creazione di un nuovo record di attivazione puo' assegnare valori non solo alle variabili nel record in corso di creazione, ma anche a variabili appartenenti a record di attivazione gia' allocati in precedenza. Questo tipo di assegnamento viene chiamato **assegnamento di uscita**, perche' trasferisce valori dalla clausola attivata alla regola chiamante. Notiamo che la clausola attivata in corrispondenza della creazione del record 5 e' un fatto ($P1$). In virtu' dell'algoritmo di controllo, cio' provoca una sequenza di uscite da regole che risale fino all'uscita del goal iniziale, G_1, indicando che il goal e' stato risolto. A questo punto la soluzione deve essere ricostruita mediante il contenuto dei record di attivazione allocati fino ad ora, a partire dalle variabili contenute nel primo record e seguendo i puntatori ad altre variabili presenti in esse. Quindi, per presentare la soluzione $U_1 = b$ e $W_1 = [a]$, anche la cella X_4 deve essere consultata.

Dopo la presentazione della soluzione, l'interprete deve costruire le eventuali successive soluzioni, e per far cio' deve eseguire un'operazione di backtrack. Sappiamo dalla sezione precedente, che nel corso del backtrack tutti i record di attivazione, a partire dal piu' recente punto di backtrack incluso, vengono scartati. In questo caso il piu' recente (in realta' l'unico) punto di backtrack e' il record 5, che verra' percio' scartato. Questa operazione di pop sulla pila dei record di attivazione crea pero' un problema. Poiche' la rimozione di un record dalla pila viene effettuata nell'intento di annullare il contributo da esso portato al calcolo della soluzione, occorre fare in modo che siano eliminati anche gli effetti di eventuali assegnamenti di uscita eseguiti durante l'unificazione che ha portato alla creazione del record. In questo esempio, un assegnamento di uscita e' stato eseguito, quando e' stato assegnato a X_4 il valore [], unificandolo col primo argomento del fatto P_1. Questo assegnamento deve essere disattivato, riassegnando a X_4 il valore '*'. Perche' l'operazione di backtrack possa avvenire

in modo corretto, e' percio' necessario che l'interprete possa ritrovare le variabili, allocate in zone della pila dei record di attivazione "piu' profonde" del piu' recente punto di backtrack, che hanno ricevuto un valore mediante un assegnamento di uscita durante la creazione di uno dei record di attivazione che devono essere scartati. Nel prossimo paragrafo mostreremo un metodo per permettere all'interprete di ricordarsi di queste variabili.

3.3 La pila di ripristino

Il modo piu' immediato per permettere all'interprete di ricordarsi delle variabili che hanno subito un assegnamento di uscita e che devono essere ripristinate a '*', e' quello di mantenere una struttura dati, dinamicamente aggiornata ad ogni allocazione di record, che contenga gli indirizzi di tali variabili. Tale struttura dati, chiamata **pila di ripristino** (in inglese, *trail*) viene appunto gestita a pila, attraverso un insieme di operazioni che ora illustreremo e che possono essere facilmente incluse nell'algoritmo di controllo della sezione precedente. La macchina astratta che realizza l'interprete deve possedere un registro in piu', chiamato TT (**Top della Trail**), che punti alla cima della pila costituita dalla pila di ripristino. All'inizio dell'esecuzione la pila di ripristino e' vuota e deve contenere semplicemente il puntatore nullo $\perp$. Quando viene creato un nuovo record di attivazione, vengono eseguite un certo numero di operazioni.

Innanzitutto, se il record creato e' un punto di backtrack, in esso viene allocato un nuovo campo, chiamato TR (**puntatore al TRail**), in cui si immagazzina il valore corrente di TT; tale valore verra' usato durante l'operazione di backtrack nel modo che spiegheremo tra poco. Se le unificazioni effettuate comportano degli assegnamenti di uscita, l'indirizzo delle variabili che sono state assegnate (cioe' un puntatore ad esse) viene memorizzato in cima alla pila di ripristino, viene cioe' eseguita una *push* che incrementa il valore del registro TT. Quando viene eseguita un'operazione di backtrack, si considera il valore $TR(PRB)$ cioe' il puntatore contenuto nel campo TR del piu' recente punto di backtrack, e a partire dalla posizione (esclusa) da esso indicata sulla pila di ripristino gli indirizzi in essa contenuta vengono usati per cancellare gli assegnamenti in uscita eseguiti durante le unificazioni necessarie per la creazione dei record di attivazione che stanno per essere cancellati. Dopo questa operazione di ripristino al valore '*', il registro TT stesso viene aggiornato al valore $TR(PRB)$, cioe' la pila di ripristino viene riportata alla situazione che aveva immediatamente prima della creazione dell'ultimo punto di backtrack. Dopo di cio' l'operazione di backtrack continua nel solito modo, assegnando a PRB il valore $PB(PRB)$ e scartando i record di attivazione a partire dal piu' recente punto di backtrack incluso. Poi l'esecuzione procede normalmente. Il campo TR nei record di attivazione che sono anche punti di backtrack ha percio' una duplice funzione: serve per individuare, tra tutte le variabili contenute nella pila, quelle da ripristinare nella successiva operazione di backtrack, e inoltre conserva il valore da assegnare al registro TT per riportare la pila di ripristino allo stato precedente la creazione dell'ultimo punto di backtrack. Queste operazioni sono mostrate graficamente su un esempio generico, in figura 8.

In esso la pila dei record di attivazione contiene due punti di backtrack, ognuno dei quali possiede anche il campo TR, che punta ad una locazione della pila di ripristino. La pila dei record e' divisa dai due punti di backtrack in tre parti, che per comodita' indichiamo con A, B e C. Le parti della pila di ripristino che contengono gli indirizzi delle variabili cui e' stato assegnato un valore durante la creazione di record dei segmenti B e C sono state indicate con B' e C', rispettivamente.

I campi TR dei punti di backtrack puntano alla posizione appena precedente l'inizio delle

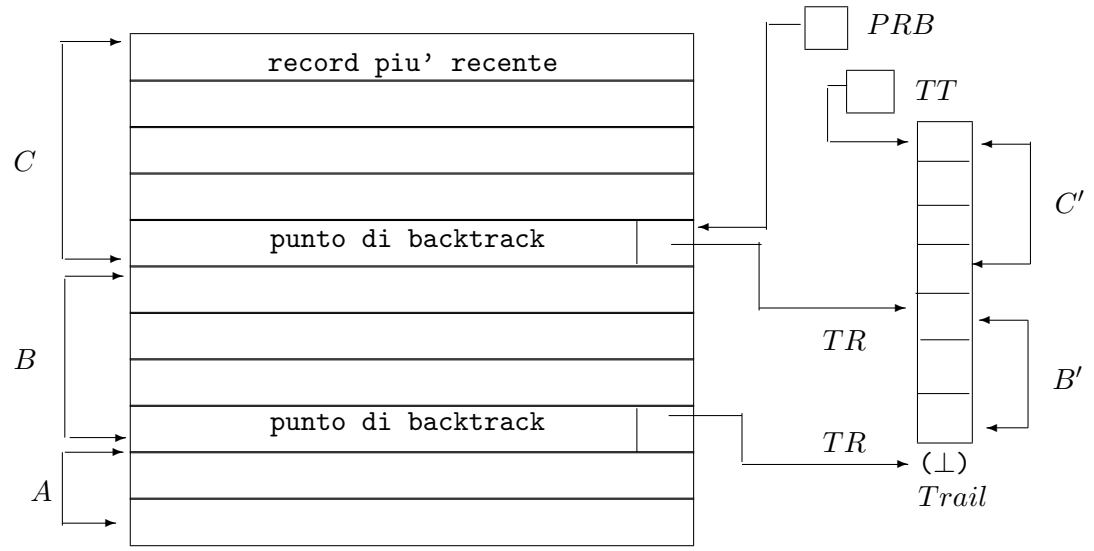

Figure 8: Pile dei record e di ripristino prima di una operazione di backtrack

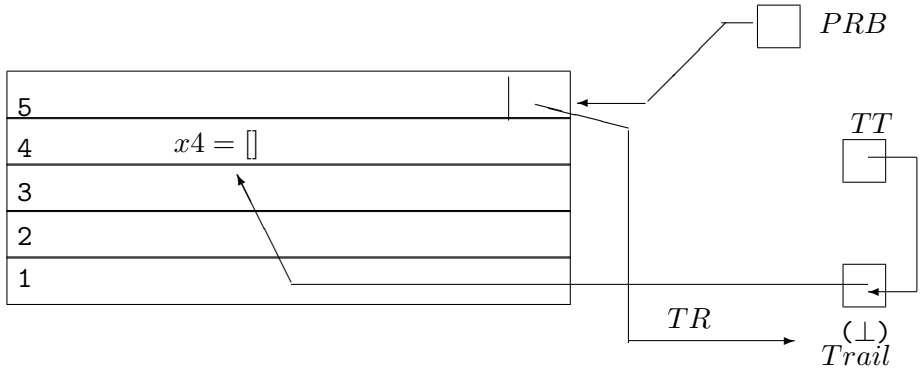

Figure 9: Pile dei record e di ripristino prima di una operazione di backtrack

parti B' e C', e permettono quindi di percorrere tali zone della pila di ripristino. Se a questo punto viene eseguita una operazione di backtrack, il segmento C della pila dei record deve essere scartato, ma prima occorre disattivare gli assegnamenti alle variabili il cui indirizzo e' contenuto nella porzione C' della pila di ripristino, ed assegnare a TT il valore del campo TR del record puntato da PRB.

Tornando all'esempio del programma prefisso nel paragrafo precedente, la situazione della pila dei record e della pila di ripristino sono presentate in figura 9. Il record 5 e' l'ultimo punto di backtrack. Il campo TR vale in esso $\perp$ perche' oltre ad essere l'ultimo, il record 5 e' anche il primo punto di backtrack. La pila di ripristino contiene un solo valore, l'indirizzo della cella di X_4, cui e' stato assegnato in uscita il valore [] e TT punta ad esso. PRB punta al record 5, il cui puntatore TR al trail vale $\perp$. Quando si fa backtrack, il segmento del trail che individua le celle delle variabili da annullare, consiste del solo valore X_4; a questa cella viene riassegnato il valore '*', TT e' resettato a $TR(PRB) = \perp$, PRB e' resettato a $PB(PRB) = \perp$, e il record 5 viene scartato.

Riferimenti Bibliografici

Christopher J. Hogger, *Introduction to Logic Programming*, Academic Press,Inc.(London), 1984.

A.Morzenti, *Corso di linguaggi e metodi di programmazione*, Consorzio per l'Universita' a distanza, 1990.