[4] Lloyd, J. W., *Foundations of Logic Programming*, Symbolic Computation Series, Springer, 1984.

[5] Clocksin, W. F. and Mellish, C. S., *Programming in Prolog*, Springer, 1981.

[6] Bancilhon, F., A Logic-Programming/Object-Oriented Cocktail *SIGMOD RECORD*, Vol. 15, No. 3, September 1986.

[7] Tsur, S. and Zaniolo, C., LDL: A logic-based data-language, *In Proc. Twelfth Intl. Conf. on Very Large Data Bases*, IEEE, pp. 33-41, Kyoto, Japan, 1986.

[8] Zaniolo, C., Object-Oriented Programming in Prolog, *Proceedings, International Symposium on Logic Programming, Atlantic City, IEEE, pp. 265-270, 1984.*

[9] Ait-Kaci, H., Nasr, R., LOGIN: A Logic Programming Language with Built-in Inheritance, *Journal Logic Programming*, 3(3), Oct. 1986.

[10] C. Beeri, S. Naqvi, R. Ramakrishnan, O. Shmueli and S. Tsur. Sets and Negation in a Logic Database Language (LDL). *Proc. 6th ACM SIGACT-SIGMODSIGART Symposium on Principles of Database Systems*, San Diego, Cal., 21-37 (1987).

[11] G. M. Kuper. Logic Programming with Sets. *Proc. 6th ACM SIGACT-SIGMODSIGART Symposium on Principles of Database Systems*, San Diego, Cal., 11-20 (1987).

[12] Shapiro, E. and A. Takeushi. Object Oriented Programming in Concurrent Prolog. *New Generation Computing*, 1, 1983.

[13] Shmueli, O., Tsur, S., Zaniolo, C., Compilation of Rules Containing Set Terms In A Logic Data Language (LDL), unpublished manuscript.

[14] Abiteboul, S. and S. Grumbach. COL: A Logic-Based Language for Complex Objects. *Proc. of the Workshop on Database Programming Languages*, Roscoff, France, Sept. 1987, pp. 253-276.

[15] D. Maier, A Logic for Objects, unpublished manuscript.

[16] Warren, D. H. D., Higher-Order Extensions to PROLOG: are they Needed?, in *Machine Intelligence*, Vol 10, Ellis Horwood Limited, 1982.

[17] Apt, K. R., Blair, H., Walker, A., Towards a Theory of Declarative Knowledge, *Preprints of Workshop on Foundations of Deductive Databases and Logic Programming (J. Minker, ed.)*, August 18-22, Washington, D.C., pp. 546-628, 1986.



# LOAR : un sistema di interrogazione logico-algebrico

*M. A. Dispinzeri*†
Ente Ferrovie dello Stato - Direzione Centrale Informatica, Roma

*L. Palopoli, C. Pizzuti*
CRAI, Rende (CS)

*SOMMARIO*

*LOAR e` un sistema di gestione di basi di dati relazionali con linguaggio di interrogazione di tipo logico. Sfruttando l'esistenza di una corrispondenza tra le clausole di Horn e l'algebra relazionale (SPJ) con gli operatori di selezione, proiezione e join, LOAR realizza una integrazione tra il paradigma algebrico e quello logico. Tale integrazione porta ad un sistema che offre sia la potenza espressiva dei linguaggi logici che una efficiente esecuzione delle query tramite gli operatori dell'algebra relazionale che, arricchiti con l'operatore di antijoin, consentono un trattamento efficace della negazione.*

## 1. Introduzione

La logica matematica costituisce un supporto formale comune sia alla Programmazione Logica che alla teoria delle Basi di Dati relazionali. Integrare le due tecnologie significa, da una parte espandere la potenza di un sistema di basi di dati dotandolo di meccanismi di inferenza per dedurre nuovi fatti da quelli preesistenti, dall'altra arricchire un sistema di programmazione logica con meccanismi efficienti per la gestione di grosse moli di dati [Ul], [BrJa], [Za], [Ll], [Ko], [GaMiNi].

Le tecniche fondamentali per la realizzazione di tale integrazione si basano essenzialmente o sull'espansione di uno dei due sistemi con capacita` proprie dell'altro [Ch], oppure su una integrazione dei due sistemi che puo` essere di tipo debole [ChWa], [ScWa] lasciando i due sistemi separati, oppure di tipo forte in cui entrambi i

† L' autrice ha svolto questo lavoro in qualita` di dipendente CRAI.

sistemi vengono modificati producendo un unico sistema ben integrato.

Con l'approccio dell'integrazione debole e` il sistema logico che si occupa del trattamento della parte deduttiva, lasciando al DBMS sottostante il compito di semplice esecutore. Nella realizzazione di LOAR si e` scelto l'approccio della integrazione forte, sfruttando in gran parte sia la potenza espressiva propria dei linguaggi logici che la manipolazione efficiente dei dati tipica dei DBMS. La scelta di trasferire la risoluzione di una query al DBMS, dopo una opportuna trasformazione in un insieme di espressioni dell'algebra relazionale, consente una implementazione piu` efficiente rispetto ai meccanismi di unificazione e backtracking propri del Prolog [ClMe].

L'assunzione semantica fatta nel modello relazionale sulla informazione negativa, cioe` che se una tupla non e` nell'istanza di una relazione allora essa sta nel complementare della relazione stessa, consente l'uso delle clausole non di Horn, la cui introduzione porta ad un sensibile arricchimento linguistico del sistema. L'insieme dei soli operatori di selezione, proiezione e join dell'algebra relazionale non e` sufficiente per interpretare algebricamente le clausole logiche generali. A tale scopo si e` introdotto l'operatore di antijoin [TsZa] che consente un mapping interpretativo opportuno tra occorrenze di letterali negativi nei corpi delle clausole logiche e algebra relazionale.

L' attuale versione di LOAR non ammette la ricorsione. Studi in corso mirano ad un ulteriore ampliamento dell'algebra SPJ con l'introduzione dell'operatore di minimo punto fisso per consentirne il trattamento.

Questo lavoro e` organizzato come segue. Nel secondo paragrafo vengono introdotti alcuni concetti di base utili nel resto del lavoro. Il terzo paragrafo descrive l'approccio usato per il trattamento della negazione. Nel quarto paragrafo viene dato un insieme di condizioni, dette di *safety*, che definiscono la classe dei programmi LOAR ammissibili. Nel quinto paragrafo, infine, sono state esposte sia le regole di traduzione delle clausole di Horn in termini di algebra relazionale, che alcune tecniche di ottimizzazione usate nella implementazione degli operatori algebrici.

## 2. Concetti di base

In questa sezione vengono introdotte alcune nozioni utilizzate in quanto segue.

*Definizione.* Una *clausola* e` una formula ben formata della forma

$$\forall x_1, x_2 \cdots x_s(L_1 \vee \cdots \vee L_n)$$

dove ogni $L_i$ e` un letterale, ed $x_1, ..., x_s$ sono tutte le variabili che occorrono in $L_1 \cdots L_n$. □ La definizione precedente fa riferimento ad una sintassi di tipo disgiuntivo. Una clausola puo` anche essere scritta in forma implicativa, notazione che sara` qui utilizzata, nel modo seguente:

$$\forall x_1, x_2 \cdots x_s(A{:-}L_1,...,L_n)$$



dove ogni $L_i$ e` un letterale, ed $x_1, ..., x_s$ sono tutte le variabili che occorrono in $L_1 \cdots L_n$.

In una clausola, la parte a sinistra del segno di implicazione si dice *testa* e quella a destra *corpo* della clausola.

Vengono supposti noti concetti riguardanti le basi di dati relazionali quali *relazione, tupla, proiezione, selezione, congiunzione (join), prodotto cartesiano.*

*Definizione.* Un programma logico è un insieme finito di clausole. □

*Definizione.* Una base di dati deduttiva è un programma logico in cui le clausole con corpo vuoto *(fatti)*, sono memorizzate come estensione di una relazione in una base di dati relazionale.

## 3. Trattamento della negazione

Nel modello relazionale l'informazione positiva e` direttamente memorizzata mentre quella negativa viene rappresentata in maniera implicita attraverso una metaregola di inferenza che stabilisce che una tupla $\overline{a}$ non appartiene ad una relazione P se $\overline{a}$ non fa parte dell'estensione di P nella base di dati. Cio` equivale a dire, in termini logici, che in una base di dati relazionale Db, $\neg P(\overline{a})$ e` conseguenza logica di Db se $\overline{a}$ non appartiene all'estensione di P in Db.

Tale metaregola, detta Assunzione del mondo chiuso (CWA) [RE78], si fonda sull' *assunzione di completa conoscenza,* che stabilisce che il dominio di conoscenza modellato dalla base di dati e` totalmente rappresentato dall'estensione della base di dati stessa.

Un'analoga assunzione puo` essere riportata nell'ambito delle basi di dati deduttive, estendendo quella per le basi di dati relazionali. Tale estensione tiene conto del fatto che la conoscenza rappresentata in una base di dati deduttiva consiste non solo di quella memorizzata nella sua componente estensionale, ma anche di quella implicitamente rappresentata attraverso le regole di deduzione della sua componente logica.

Data, quindi, una base di dati deduttiva DDb ed un atomo $p(\overline{a})$, allora:

$$DDb \vdash \neg p(\overline{a}) \quad se \quad \neg(DDb \vdash p(\overline{a}))$$

*Esempio.*

Si consideri la base di dati deduttiva seguente:

*Mortale*(x) :– *Uomo*(x).

con i seguenti fatti:

*Uomo*(*Socrate*).

*Uomo*(*Plinio*).

E` chiaro che Mortale(Platone) non e` una conseguenza logica della base di dati, pertanto per la CWA possiamo concludere che $\neg Mortale(Platone)$ e` conseguenza logica della base di dati. □

E` stato dimostrato [RE78] che la CWA produce sempre un insieme di fatti negativi consistenti con la base di dati deduttiva solo nel caso in cui la componente logica della base deduttiva stessa e` un programma logico di Horn, viceversa, se il programma logico contiene clausole non di Horn, allora la CWA puo` portare ad inconsistenze.

*Esempio.*

Sia DDb la seguente base di dati deduttiva:

$$p :- \neg q$$

E` chiaro che ne` p, ne` q sono conseguenze logiche di DDb e, pertanto, per la CWA possiamo dedurre che sia $\neg p$ che $\neg q$ sono conseguenze logiche di DDb. E` immediato, allora, concludere che $\neg p$ e $\neg q$ insieme con la clausola di DDb formano un insieme logicamente inconsistente. □

Dato che i programmi logici trattati dal nostro sistema non sono sempre programmi di Horn, la CWA non puo` essere utilizzata.

Tuttavia la classe dei programmi logici ammessi dal nostro sistema, che denoteremo con LOAR_P, e` tale per cui e` possibile definire una particolare forma di CWA [Pa], che consente di derivare comunque un insieme di fatti negativi completo e consistente rispetto al programma logico cui viene applicata.

Intuitivamente puo` essere considerata come una CWA *ristretta* applicata per passi successivi al programma logico che garantisce la consistenza tra gli insiemi di fatti negativi generati ed il programma logico da cui si parte in modo che tali fatti negativi si riferiscono a letterali che compaiono nel corpo di una clausola, piuttosto nella testa della clausola stessa.

*Esempio.*

Si consideri, ancora una volta, il programma seguente:

$$p :- \neg q$$

Esso ha due modelli minimali : $\{p\}$ e $\{q\}$, pertanto la sua semantica non e` univocamente caratterizzabile. Tuttavia viene naturale interpretare la regola non seguendo la logica del primo ordine piatta, ma una sorta di logica causale che impone che, se possibile, venga inferita la verita` di $p$ dalla falsita` di $q$ e che quindi venga scelto uno solo dei due modelli, cioe` $\{p\}$.

Si noti che, forzando la scelta del modello *causale*, la semantica del programma logico e` ben definita e non da` luogo a inconsistenze.

□



Si puo` dimostrare che tale modello *causale* e` unico per una ampia classe di programmi logici generali tra cui quelli non ricorsivi e quindi per LOAR_P. Inoltre esiste un algoritmo che costruisce il modello causale per *strati*, ma questo e` oltre gli scopi del presente lavoro. Definiamo ora la corrispondenza tra negazione logica e algebra relazionale.

La soluzione piu` opportuna ci sembra quella di interpretare la negazione logica mediante l'operazione di complementazione algebrica.

*Esempio.*

La clausola:

$$p(x) :- \neg q(x, y)$$

viene interpretata in termini algebrici come:

$$p = \Pi_x(q^c) \qquad \square$$

Si puo` dimostrare che tale interpretazione e` corretta per i programmi stratificati e, quindi, per i programmi in LOAR_P, nel senso che se si ha un'occorrenza di $\neg p(x)$ nel corpo di una regola, in $p^c$ ci sono tutte e sole le tuple $\overline{a}$ tali che $\neg p(\overline{a})$ e` inferibile per SCWA dal programma logico.

Nel prossimo paragrafo si vedra` come, in realta`, tale interpretazione, sebbene corretta, porti a dei problemi di *safety*. E` stata quindi proposta un'interpretazione della negazione diversa dalla complementazione algebrica e semanticamente equivalente a questa per i programmi in LOAR_P.

## 4. Controllo di consistenza

La classe dei programmi LOAR ammissibili, denotata con LOAR_P, e` tale da soddisfare un insieme di condizioni dette di *safety*. L' esigenza di introdurre tali condizioni nasce dalla considerazione che interpretando la negazione mediante l' operazione di complementazione algebrica, si introduce la possibilita` di generare relazioni infinite. A tale scopo definiamo la classe delle espressioni algebriche ammissibili, denotate con LOAR_E, come l'insieme delle interpretazioni algebriche di programmi logici in cui nessuna sottoespressione rappresenta una relazione infinita. In LOAR_E la negazione e` interpretata mediante l'operatore di antijoin, inoltre tutte le variabili che compaiono in predicati di confronto hanno un intervallo di ammissibilita` limitato.

*Esempio.*

L'espressione algebrica corrispondente alla seguente regola

$$p(x):-q(x),y>5.$$

non e` in LOAR_E poiche` l'intervallo di ammissibilita` della variabile y, rappresentato da tutti gli interi maggiori di 5, e` infinito.

Data la regola $p{:-}q,\neg t$ in LOAR_P, la sua corrispondente espressione algebrica in LOAR_E e` espressa usando l'operatore di antijoin come $q \dotplus t$, dove $\dotplus$ e` definito come combinazione degli usuali operatori dell'algebra relazionale, cioe`

$$q \dotplus t \equiv q \; - \; \Pi_q \, ( q \rhd\!\lhd t)$$

E` possibile dimostrare che ogni espressione di LOAR_E e` equivalente ad una espressione algebrica (rispetto ad uno stesso programma in LOAR_P), in cui la negazione e` interpretata mediante l'operazione di complementazione.

Le condizioni di *safety* che daremo in seguito, esprimono delle restrizioni sulle variabili che compaiono in un programma logico allo scopo di non far comparire relazioni infinite nell' espressione algebrica equivalente.

Una variabile e` *safe* se

a) appare in un predicato positivo non di confronto, oppure se

b) appare in un predicato di uguaglianza in cui il secondo argomento sia una costante oppure una variabile safe, oppure se

c) appare in un unico predicato negato non di confronto.

Una regola e` *safe* se tutte le variabili che compaiono in essa sono *safe*.

*Esempio.*

Si consideri il programma seguente:

$$p(x,y){:-}q(y,z),\neg r(x,t).$$

per esso la condizione di *safety* non e` verificata in quanto la variabile $x$ appare in un predicato negato nel corpo della regola e nel predicato di testa della stessa, ma in nessun predicato positivo del corpo della regola.

## 5. Risoluzione di una query

Una query LOAR e' composta da una congiunzione di predicati di base e/o derivati del tipo:

$$G_1,..., G_n$$

La sua risoluzione e' composta da due fasi distinte. La prima, detta di compilazione, consiste nell' individuazione delle regole coinvolte, nella traduzione di ognuna di queste in una espressione dell' algebra relazionale e nella costruzione di un corrispondente albero di rappresentazione. La seconda implementa questo insieme di espressioni costruendo un unico albero *algebrico* di rappresentazione.

L' applicazione di algoritmi di ottimizzazione all' albero algebrico costituisce la fase conclusiva della risoluzione del goal. L' uso di tali algoritmi consente, da una parte di riconfigurare l'albero spostando opportunamente alcuni nodi, dall' altra di



individuare gruppi di operazioni da eseguire contemporaneamente sulla base di dati riducendo l' albero bottom-up, fino ad ottenere un unico nodo corrispondente alla soluzione del goal in esame.

### 5.1. Compilazione

Un programma in LOAR_P consiste di un insieme di clausole del tipo

$$R :- \; L, N, E, D, K$$

dove $L$ e` una congiunzione di predicati non di confronto positivi, $N$ e` una congiunzione di predicati non di confronto negati, $K$ e` la congiunzione di tutti i predicati di confronto senza variabili e di tutti i predicati di diseguaglianza della forma $t \; \theta \; t$, $E$ e` una congiunzione di predicati di uguaglianza non negati con almeno una variabile che non compaiono in $K$, e $D$ e` la congiunzione di tutti gli altri predicati di confronto (anche negati). L' insieme di regole con lo stesso simbolo di predicato nella testa definisce una relazione derivata. Non e` ammessa la definizione di relazioni parzialmente derivate, cioe` si assume che non ci siano regole con simbolo di predicato nella testa uguale al nome di una relazione di base. Per ragioni di semplicita`, d' ora in avanti ci si riferira` sia ad un predicato che alla relazione corrispondente usando lo stesso simbolo, ove questo non generi ambiguita`.

Durante la fase di controllo di consistenza, tutti i predicati di confronto di verifica immediata vengono eliminati cioe`: tutti i predicati di uguaglianza del tipo $t = t$ vengono rimossi dalla regola in quanto non portano alcun contributo nella soluzione della stessa; quelli del tipo $t = u$, con almeno una variabile, vengono eliminati dopo aver sostituito ad ogni occorrenza di una delle due variabili l' altra, se entrambe $t$ ed $u$ sono variabili, oppure sostituendo la costante al posto della variabile; infine, quelli di tipo $t \; \theta \; u$, dove $\theta$ e' un predicato di disuguaglianza e $t$ e $u$ sono costanti, vengono valutati e, se sono insoddisfacibili, viene rimossa l' intera regola, altrimenti vengono rimossi essi stessi. Dopo tale elaborazione, tutte le regole di riducono alla forma:

$$R :- \; L_1,..,L_n,D_1,..,D_m,N_1,..,N_k$$

dove $L_1,..,L_n$ sono i predicati in $L$, $D_1,..,D_m$ sono i predicati in $D$ e $N_1,..,N_k$ sono i predicati in $N$ a cui si e' tolta la negazione.

Dato un goal $G$, per ogni predicato nella congiunzione, il sistema individua tutte la regole che contribuiscono alla sua soluzione. Ognuna di esse viene trasformata nella seguente espressione dell' algebra relazionale:

$$R = \; \pi_{I_R}(b \times \sigma_{F_L}(\pi_{I_{L_1}}(\sigma_{F_{L_1}}(L_1)) \times , \; . \; . \; . \; , \times \pi_{I_{L_n}}(\sigma_{F_{L_n}}(L_n)))$$
$$\dotplus \; \pi_{I_{N_1}}(\sigma_{F_{N_1}}(N_1)) \dotplus, \; . \; . \; . \; , \dotplus \pi_{I_{N_m}}(\sigma_{F_{N_m}}(N_m)))$$

dove

- $\pi$ e` l'operazione di proiezione, $\sigma$ di selezione, $\times$ di prodotto cartesiano e $\div$ di antijoin che, ricordiamo, corrispondente a sottrazione piu` semijoin. Cioe` $A \div B$ corrisponde ad effettuare prima la join naturale di A con B e poi proiettare il risultato sugli attributi di $A$ e sottrarlo ad $A$. Si noti che nelle proiezioni e` ammesso proiettare su un insieme di attributi vuoto. In tal caso, se la relazione iniziale e` vuota, il risultato e` una relazione con zero attributi vuota, altrimenti e` una relazione con zero attributi contenente un'unica tupla fittizia, corrispondente a una tupla vuota. La tupla vuota puo` essere congiunta a qualsiasi tupla durante una operazione di join. Essa puo` essere considerata un elemento neutro per la join e prodotto cartesiano.
- $F_{L_i}$, $1 \le i \le n$, e` una congiunzione di condizioni $j\ \theta\ k$, dove $j$ e` un indice che denota un argomento di $L_i$, $k$ e` una costante o denota un argomento di $L_i$, e $\theta$ e` un operatore di confronto. Le condizioni in $F_{L_i}$ sono derivate dalla presenza della stessa variabile in piu` argomenti di $L_i$, o dalla presenza di una costante in qualche argomento di $L_i$, o da un predicato in $D$ che lega variabili tutte in $L_i$.
- $I_{L_i}$, $1 \le i \le n$, e` una lista di indici di argomenti di $L_i$ (eventualmente vuota), denotante tutte le variabili che appaiono in qualche altro predicato in $L$ o in un predicato in $D$ in cui l'altra variabile appare in un qualche altro predicato di $L$. Se una variabile appare in piu` di un argomento di $L_i$, allora e` sufficiente includere nella lista solo uno degli indici corrispondenti alla variabile.
- $F_{N_i}$, $1 \le i \le m$, e` una congiunzione di condizioni $j\ \theta\ k$, dove $j$ e` un indice che denota un argomento di $N_i$, $k$ e` una costante o denota un argomento di $N_i$, e $\theta$ e` un operatore di confronto. Le condizioni in $F_{N_i}$ sono derivate dalla presenza della stessa variabile in piu` argomenti di $N_i$, o dalla presenza di una costante in qualche argomento di $N_i$, o da un predicato in $D$ che lega variabili tutte in $N_i$.
- $I_{N_i}$, $1 \le i \le m$, e` una lista di indici di argomenti di $N_i$ (eventualmente vuota), denotante tutte le variabili che appaiono in qualche predicato in $L$. Se una variabile appare in piu` di un argomento di $N_i$, allora e` sufficiente includere nella lista solo uno degli indici corispondenti alla variabile.
- $F_L$, e` una congiunzione di condizioni $j\ \theta\ k$, dove $j$ e $k$ sono indici che denotano argomenti del prodotto cartesiano di tutte le relazioni corrispondenti a predicati in $L$, e $\theta$ e` un operatore di confronto. Le condizioni in $F_L$ sono derivate dalla presenza della stessa variabile in piu` predicati di $L$, o da un predicato in $D$ che lega variabili che appaiono in piu` predicati di $L$.
- $I_R$, e` una lista di indici di argomenti del prodotto cartesiano della tupla $b$ con il risultato del prodotto cartesiano delle relazioni corrispondenti a $L$.



*Esempio*:

Supponiamo che per risolvere la query r(x,y), siano state selezionate le regole:

r(x,y) :- a(x,z),b(z,y),c(x,'e'), z<3.

a(x,y) :- f(x,y),¬ c(x,y), y > 6

a(x,y) :- c(x,w),d(w,y).

con b, d, c ed f predicati di base. La compilazione di tali regole produce le seguenti espressioni algebriche:

$$r = \pi_{1,4}(\sigma_{1=5,2=3}(\sigma_{2<3}(a)\times(\sigma_{1<3}(b))\times\pi_1(\sigma_{2='e'}(c)))$$

$$a = (\sigma_{2>6}(f \div c))$$

$$a = \pi_{1,4}(\sigma_{2=3}(c\times d))$$

## 5.2. Esecuzione

Il processo di compilazione produce un albero per ogni regola del programma logico utilizzabile per la soluzione del goal. Nella fase successiva, tali alberi vengono espansi sostituendo ad ogni occorrenza di predicato derivato il sottoalbero corrispondente alla sua espressione algebrica. Tale espansione e' fatta a partire dal goal e termina quando tutti i nodi foglia rappresentano relazioni di base.

*Esempio*:

Le due espressioni algebriche dell'esempio precedente vengono sostituite dalla seguente espressione :

$$r = \pi_{1,4}(\sigma_{1=5,2=3}(\sigma_{2<3}(\sigma_{2>6}(f \div c)\cup\pi_{1,4}(\sigma_{2=3}(c\times d)))\times(\sigma_{1<3}(b))\times\pi_1(\sigma_{2='e'}(c))))$$

Il passo successivo consiste in una ottimizzazione puramente algebrica allo scopo di ridurre il tempo di esecuzione e di avere dei risultati intermedi di dimensioni ragionevoli. Tale ottimizzazione sfrutta le proprieta' associativa e commutativa degli operatori algebrici, nel tentativo di spostare i nodi selezione e proiezione piu' in basso possibile nell' albero di rappresentazione.

Dall' albero ottimizzato si cerca di raggruppare operazioni unarie con le operazioni n-arie immediatamente precedenti, in modo che le operazioni n-arie abbiano operandi di dimensioni il piu' possibile limitate. I vari raggruppamenti vengono eseguiti con una strategia bottom-up e i risultati parziali sostituiti nell'albero, riducendolo via via fino ad avere solo il nodo radice che e' rappresentativo della soluzione del

goal.

## 6. Conclusioni

In questo lavoro e' stato presentato il prototipo di un sistema di gestione di basi di dati relazionale con linguaggio di interrogazione di tipo logico. Tale sistema sfrutta le relazioni esistenti tra la logica clausale e l'algebra relazionale, ampliando l'insieme degli operatori algebrici con l'introduzione dell'operatore di antijoin per un trattamento efficace della negazione. Dopo aver descritto l' approccio seguito per il trattamento della negazione e le regole di consistenza che un programma logico deve soddisfare sono state schematizzate le regole di traduzione da clausole logiche ad espressioni dell'algebra relazionale. Una versione funzionante del sistema e' stata realizzata in ambiente VAX/VMS in Pascal.

## Ringraziamenti

Si ringraziano i borsisti del corso di formazione tenuto presso il CRAI, A.Arcuri, S.Cozza, V.De Cicco, G.Federici, N.Leone, T. Pontieri, G.Sangiovanni, G.Scalzo, P.Tortorici, che hanno implementato il prototipo del sistema.

## 7. Bibliografia

[ApBlWa]
K. R. Apt, H. Blair, A. Walker, Towards a Theory of declarative Knowledge, Technical report, IBM T. J. Watson RC, NY, 1986

[BrJa]M. L. Brodie e M. Jarke, On Integrating Logic and Databases, Proc of I Int. Workshop on Expert Databases Systems, South Carolina, 1984

[CW]C. L. Chang, A. Walker, PROSQL : a Prolog Programming Interface with SQL/DS. Proc. of I Int. Workshop on Expert Database Systems, South Carolina, Oct. 1984.

[ClMe]
W. F. Clocsin, C. S. Mellish, "Programming in Prolog", Springer-Verlag 1981

[Ch] C. L. Chang, DEDUCE - A Deductive Query Language for Relational Data Bases, in Pattern Recognition and Artificial Intelligence (C. H. Chen ed.), Accademic Press, Inc., New York, 1976.

[GaMiNi]
H. Gallaire, J. Minker, J. Nicolas, Logic and Databases: a Deductive Approach, Joint report CERT/CGE/University of Meryland, Oct. 1982.

[Ll] J.W. Lloyd, "Foundations of Logic Programming", Springer-Verlag 1984.

[Ko] R. Kowalski, "Logic for Problem Solving", North-Holland Publishing Co. New York, 1979.



[Pa] L. Palopoli, Accessi a database relazionali con programmi logici generali, Tesi di laurea, Universita` di Pisa, Corso di Laurea in Scienze dell' Informazione, 1986

[Re77]
R. Reiter, An approach to Deductive Question-Answering, BBN Tecn. Report No 3649, 1977.

[Re78]
R. Reiter, On Closed World Data Bases, in Logic and Databases, H. Gallaire e J. Minker (eds), Plenum Press, New York, 1978.

[ScWa]
E. Sciore, D. S. Warren, Towards an Integrated Database-Prolog System, Proc. of I Int. Workshop on Expert Databases Systems, South Carolina, 1984

[Ul] J.D. Ullman, "Principles of Database Systems", Computer Science Press, Rockville, Md., 1982.

[TsZa]
S. Tsur, C. Zaniolo, Logic Data Language, MCC Technical Report, 1985.

[Za] C. Zaniolo, Prolog: a Database Query Language for All Seasons, Proc. of I Int. Workshop on Expert Databases Systems, South Carolina, 1984