Sequence Datalog: un prototipo per l'interrogazione di sequenze

Salvatore Labonia[1] and Giansalvatore Mecca *[2]

[1] Dipartimento di Discipline Scientifiche
Università di Roma Tre – 00146 Roma
tel: 06/55177053, fax: 06/5579295
labonia@inf.uniroma3.it
[2] D.I.F.A. – Università della Basilicata
via della Tecnica, 3 – 85100 Potenza
tel: 0971/474 638, fax: 0971/56537
mecca@dis.uniroma1.it

Abstract. *In questo articolo viene descritto il prototipo del linguaggio* Sequence Datalog. Sequence Datalog *è un linguaggio di interrogazione per basi di dati contenenti sequenze, ovvero stringhe di caratteri su un alfabeto finito; il modello di dati a cui si riferisce è una semplice estensione del modello relazionale, in cui le ennuple all'interno delle relazioni possono contenere, oltre che valori atomici come interi o caratteri, anche sequenze di caratteri, di lunghezza non fissata a priori. La sintassi e la semantica del linguaggio sono ispirate al Datalog, esteso per consentire la manipolazione delle sequenze. Il prototipo è stato implementato utilizzando il motore inferenziale di Coral ed arricchendolo con opportune primitive per la gestione delle sequenze.*

1 Introduzione

Le recenti applicazioni nel campo dell'informatica fanno delle basi di dati un uso sempre più avanzato ed in campi molto diversi tra di loro. Ciò ha fornito una spinta allo sviluppo di tecniche via via più sofisticate per la gestione delle basi di dati, portando ad introdurre nuovi paradigmi di interrogazione, come la programmazione logica e i linguaggi orientati ad oggetti, per consentire una maggiore flessibilità nella manipolazione dei dati.

Esistono però alcune applicazioni – come le *basi di dati testuali* [12] e le *basi di dati genetiche* [10] – in cui è necessario rappresentare e manipolare dati con struttura sequenziale, e per le quali il supporto fornito dalla tecnologia delle basi di dati è ancora insufficiente. Per esempio, nell'ambito delle applicazioni legate al *progetto Genoma* [10], la struttura molto ricca delle sequenze genetiche rende critica l'attività di interrogazione delle raccolte di dati di laboratorio. Da un lato, un linguaggio d'interrogazione che voglia dirsi efficace a questo scopo deve avere

* Ricerca parzialmente supportata dal MURST e dal Consiglio Nazionale delle Ricerche (CNR).

una buona capacità di riconoscimento di *pattern*, per confrontare la similarità tra diverse sequenze, dall'altro, anche la ristrutturazione di sequenze gioca un ruolo centrale, data la necessità di modellizzare trasformazioni di natura complessa. In sintesi, possiamo riassumere i requisiti di un buon linguaggio di interrogazione per dati sequenziali come segue:

- il linguaggio deve consentire di eseguire in modo efficiente tutte le operazioni necessarie per manipolare dati sequenziali, e cioè tanto le operazioni di riconoscimento di pattern quanto quelle di ristrutturazione delle sequenze;
- deve inoltre avere una sintassi semplice ed intuitiva ed una chiara semantica dichiarativa, in modo che possa essere utilizzato anche da utenti inesperti;
- deve infine essere efficace dal punto di vista computazionale, tanto in termini di complessità quanto di terminazione delle interrogazioni.

I sistemi di basi di dati che dispongono di un tipo di dato sequenza non sempre forniscono tutte queste caratteristiche: alcuni permettono di eseguire solo un certo numero di elaborazioni predefinite sulle sequenze, altri basano il loro funzionamento esclusivamente sul concetto di pattern matching, ed anche nei sistemi più evoluti risulta difficile trovare un buon compromesso tra espressività delle elaborazioni e terminazione della valutazione delle interrogazioni.

Sequence Datalog [16] è un linguaggio di interrogazione basato sulla logica che consente l'analisi e la ristrutturazione di sequenze. Il modello di dati cui fa riferimento è una estensione del modello relazionale tradizionale, nella quale si consente che i valori all'interno delle relazioni siano stringhe di simboli costruite a partire da un alfabeto fissato. La sintassi del Sequence Datalog (derivata da quella del *Datalog* [7]) fornisce due categorie di termini interpretati che permettono di eseguire manipolazioni sui dati di tipo sequenza: *termini indicizzati* come $X[I_1 : I_2]$, che consentono di estrarre una sottosequenza a partire da una sequenza data, e *termini costruttivi* come $X_1\$X_2\$...\$X_n$ permettono di ristrutturare le sequenze, ottenendo la sequenza che risulta dalla concatenazione delle sequenze X_1,X_2, ... ,X_n. Il linguaggio ha tanto una semantica dichiarativa, basata sulla nozione di modello di un programma, quanto una semantica operazionale, basata sulla teoria dei punti fissi.

In questo articolo, descriviamo il prototipo di Sequence Datalog, tuttora in corso di sviluppo presso il Dipartimento di Informatica dell'Università di Roma Tre. Il prototipo è basato sul motore inferenziale fornito da *Coral* [18], modificato opportunamente per rappresentare e manipolare dati sequenziali in modo efficiente. L'architettura è composta delle seguenti componenti fondamentali: *(i)* il *motore inferenziale*, basato sul sistema deduttivo Coral, ed esteso con le primitive per la rappresentazione e la manipolazione delle sequenze; *(ii)* il *traduttore*, che si incarica di effettuare i controlli sintattici sul codice Sequence Datalog e di tradurlo in una forma eseguibile dal motore inferenziale; *(iii)* l'*interfaccia utente*, che consente all'utente di interagire con il prototipo.

Nel seguito vengono descritti in maggior dettaglio il linguaggio ed il prototipo. In particolare, la sintassi e la semantica di Sequence Datalog sono brevemente descritti nella Sezione 2; il prototipo del linguaggio viene illustrato nella Sezione 3,

in cui vengono descritte le varie componenti dell'architettura; la Sezione 4 descrive le modalità secondo le quali viene effettuata la valutazione delle interrogazioni. Conclusioni e lavoro futuro sono contenute nella Sezione 5.

2 Il linguaggio Sequence Datalog

Il linguaggio *Sequence Datalog* [16] nasce come una estensione del linguaggio *Datalog* [7] che permette di rappresentare e di manipolare dati con struttura sequenziale, e cioè stringhe di caratteri appartenenti ad un alfabeto fissato. Per poter rappresentare il tipo di dati sequenza, il modello di dati su cui tale linguaggio si basa è una estensione del modello relazionale, che permetta la presenza di un dominio di tipo sequenza all'interno delle relazioni. in particolare, una relazione è un insieme di ennuple in cui possono comparire valori appartenenti a diversi domini, e cioè: *(i)* numeri interi; *(ii)* numeri reali; *(iii)* caratteri; *(iv)* sequenze di caratteri appartenenti ad un alfabeto Σ.

Riguardo alla possibilità di manipolare le sequenze ottenendone di nuove, vengono forniti due costrutti che consentono la concatenazione di due sequenze e l'estrazione della sottosequenza di una sequenza data. In particolare, se X e Y sono due sequenze, e I_1 e I_2 due numeri interi, abbiamo due categorie di termini complessi:

- *termini indicizzati*, della forma $X[I_1 : I_2]$; questi termini vengono utilizzati per estrarre la sottosequenza di X compresa tra il carattere in posizione I_1 ed il carattere in posizione I_2. I_1 ed I_2 sono detti *indici* della sequenza X;*end* è un indice particolare che sta ad indicare l'ultima posizione all'interno della sequenza. Una sottosequenza del tipo di $X[I : I]$ si può anche scrivere usando la notazione abbreviata $X[I]$. Se $I_1 = I_2 + 1$, la sottosequenza ottenuta viene convenzionalmente interpretata come la stringa vuota. I valori ammissibili per gli indici sono $I_1 \geq 1$, $I_2 \leq length(X)$, $I_1 \leq I_2 + 1$. Se gli indici non rispettano tali vincoli la sottosequenza risulta indefinita.
- *termini costruttivi*, della forma $X_1\$X_2\$...\$X_n$, utilizzati per concatenare le sequenze X_1, X_2, ... , X_n;

Qui di seguito riportiamo alcuni termini considerati validi nel linguaggio con la relativa interpretazione:

$abcd$	$abcdefghi$
$abcdefghi[1:4] = abcd$	$abcdefghi[2:4] = bcd$
$abcdefghi[3:4] = cd$	$abcdefghi[1:0] = \epsilon$
$abcdefghi[4:end] = defghi$	$abcdefghi[end:end] = i$
$ab\$cd = abcd$	$abcdefghi[3:4]\$defghi[end] = cdi$

L'idea su cui è basata la semantica del Sequence Datalog è la valutazione delle variabili nelle clausole rispetto ad una forma particolare di dominio attivo: il *Dominio Attivo Esteso*. Tale nome è dovuto al fatto che oltre a contenere tutte le sequenze appartenenti alla base di dati, contiene anche tutte le loro sottosequenze contigue, per consentire la ricorsione strutturale [16]. D'altra parte, tale dominio

cresce durante la computazione del modello, dato che si possono generare nuove sequenze per mezzo di termini costruttivi.

Si può quindi definire informalmente la semantica di un programma Sequence Datalog P su una base di dati DB come segue:

- si inizializza il *dominio attivo esteso* $\mathcal{D}^{ext}_{P,\mathrm{DB}}$ con l'unione dei tre seguenti insiemi:
  1. il *dominio attivo* $\mathcal{D}_{P,\mathrm{DB}}$ della istanza in input DB, cioè l'insieme di tutte le sequenze presenti in DB;
  2. tutte le sottosequenze contigue delle sequenze in $\mathcal{D}_{P,\mathrm{DB}}$;
  3. l'insieme degli interi $\{0,1,2,...,l_0+1\}$, dove l_0 è la massima lunghezza di una sequenza in $\mathcal{D}_{P,\mathrm{DB}}$;
- si calcola il più piccolo punto fisso della trasformazione T_P associata al programma P; ad ogni passo della computazione, si richiede che le sostituzioni varino sul *dominio attivo esteso* corrente.
- quando si deriva logicamente un nuovo fatto ground ed una nuova sequenza (cioè una sequenza non appartenente a $\mathcal{D}^{ext}_{P,\mathrm{DB}}$) risulta dall'uso di un termine costruttivo del tipo $s_1\$s_2\$...\$s_k$, allora questa sequenza viene aggiunta al *dominio attivo esteso* con tutte le sue sottosequenze contigue.

In [16] viene dimostrato che l'operatore $T_{P,\mathrm{DB}}$ risulta monotono e continuo e che per ogni programma P e istanza DB la trasformazione $T_{P,\mathrm{DB}}$ possiede il più piccolo punto fisso. A seconda che il più piccolo punto fisso risulti un insieme finito o infinito, la semantica del punto fisso di P su DB si dice finita o infinita. [1]

Vediamo ora una serie di esempi di utilizzo del linguaggio Sequence Datalog:

Example 1. **[Sequenze Palindrome]** Il seguente programma calcola tutte le sequenze palindrome contenute nella relazione $r1$:

$$\begin{aligned}
answer(X) &\leftarrow r1(X),\\
&\quad\ palindroma(X).\\
palindroma(\text{""}) &\leftarrow true.\\
palindroma(X) &\leftarrow length(X,1).\\
palindroma(X) &\leftarrow X[1] = X[end],\\
&\quad\ palindroma(X[2:end-1]).
\end{aligned}$$

$answer(X)$ risulta vero per ogni sequenza X contenuta nella relazione $r1$ per cui è vero $palindroma(X)$; $palindroma(X)$ risulta vero per ogni sequenza composta da un solo carattere, o per ogni sequenza in cui il primo carattere è uguale all'ultimo e la sottosequenza di X dal secondo al penultimo carattere è a sua volta palindroma.

Example 2. **[Riconoscimento di Pattern]** In questo esempio vengono riconosciute tutte le sequenze della relazione $r1$ del tipo $a^n b^n c^n$ con $n \geq 0$ (come *abc, aaabbbccc, ...*).

[1] Per valutare i programmi Sequence Datalog che contenengono nel corpo dei letterali negati (letterali preceduti dalla parola chiave *not*), viene utilizzata una *semantica stratificata*.

$$
\begin{array}{lll}
answer(X) & \leftarrow r1(X), \ \ length(X, L), & \\
 & \quad L \ mod \ 3 = 0, & \\
 & \quad abc(X[1:L/3], X[L/3+1:L*2/3], X[L*2/3+1:L]). & \\
abc(\text{""}, \text{""}, \text{""}) & \leftarrow true. & \\
abc(X, Y, Z) & \leftarrow X[1] = \ \texttt{'a'}, & \\
 & \quad Y[1] = \ \texttt{'b'}, & \\
 & \quad Z[1] = \ \texttt{'c'}, & \\
 & \quad abc(X[2:end], \ Y[2:end], \ Z[2:end]). &
\end{array}
$$

Il predicato *abc* è vero per ogni terna di sequenze nella forma (a^n, b^n, c^n) presente nel dominio attivo esteso; *answer(X)* risulta vero per ogni sequenza X in $r1$ di lunghezza multipla di tre, tale che divisa in tre parti uguali X_1, X_2, X_3 renda vero il predicato $abc(X_1, X_2, X_3)$.

Example 3. **[Complemento Inverso]** In questo esempio viene calcolata la sequenza complementare e inversa di tutte le sequenze contenute nella relazione *sequenza* usando la nozione di complementarietà memorizzata nella relazione *compl*.

$$
\begin{array}{lll}
reverse\_compl(\text{""}, \text{""}) & \leftarrow true. & \\
reverse\_compl(S[1:N+1], Y\$Z) & \leftarrow sequenza(S), & \\
 & \quad reverse\_compl(S[1:N], Z), & \\
 & \quad compl(S[N+1], Y). &
\end{array}
$$

Le regole stabiliscono che il complemento inverso dei primi $N+1$ caratteri della sequenza S, è dato dalla concatenazione del complemento inverso dei primi N caratteri della stessa sequenza con il complemento (secondo la relazione *compl*) del carattere in posizione $N+1$ all'interno della sequenza S.

3 Architettura del prototipo

Per valutare le interrogazioni, nel prototipo viene utilizzato un motore inferenziale capace di effettuare deduzioni su clausole logiche di Horn, modificato in modo di poter rappresentare e manipolare nel modo più efficiente possibile le sequenze ed il dominio attivo esteso.

Dal momento che la sintassi riconosciuta dal motore inferenziale è diversa dalla sintassi Sequence Datalog, il programma viene tradotto in una nuova versione che rispetta la sintassi del motore inferenziale. Durante la fase di traduzione si controlla anche la correttezza sintattica del programma dato in input.

Inoltre, al fine di rendere disponibile all'utente una interfaccia flessibile per interagire con il prototipo, vengono offerte differenti modalità di interazione: nella *modalità batch*, tutti i fatti derivabili a partire dal programma e dalla base di dati vengono inseriti in un nuovo file; nella *modalità interattiva*, invece, si consente all'utente di eseguire interattivamente le interrogazioni.

L'architettura del prototipo Sequence Datalog (mostrata in figura 1) si compone quindi delle seguenti tre componenti:

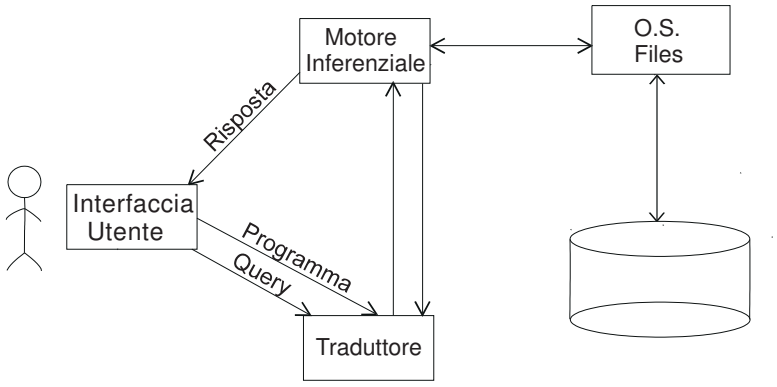

Fig. 1. Architettura del prototipo

- il *traduttore*, che si incarica di effettuare il controllo sulla correttezza sintattica del programma e di tradurlo in una forma comprensibile al motore inferenziale;
- *il motore inferenziale*, che valuta le regole derivando nuovi fatti a partire dalla base di dati;
- l'*interfaccia utente*, attraverso la quale l'utente interagisce col sistema, specificando le interrogazioni.

Esaminiamo di seguito più in dettaglio le varie componenti del sistema.

3.1 Motore Inferenziale

Il motore inferenziale è basato sul sistema deduttivo Coral, scelto per la sua versatilità e la sua affidabilità. Per consentire la manipolazione delle sequenze, sono state apportate alcune modifiche al sistema. In particolare, tutte le operazioni di manipolazione delle sequenze e di gestione del dominio attivo esteso sono state realizzate aggiungendo dei predicati *built-in* implementati direttamente in C. Per ottenere il massimo di efficenza nella valutazione delle interrogazioni, i predicati sono stati aggiunti al nucleo di Coral, in modo da implementare il tipo di dati sequenza come tipo primitivo.

Coral offre varie strategie per la valutazione delle regole. Coerentemente con la semantica di Sequence Datalog [16], il prototipo utilizza la modalità di valutazione *bottom-up*; la valutazione viene inoltre ottimizzata utilizzando tecniche di riscrittura di tipo *magic-set* [6], anche queste supportate da Coral.

3.2 Traduttore

Il traduttore è stato realizzato utilizzando i tools *Lex* e *Yacc* [4]. Compito di questo traduttore, che riceve in input un programma Sequence Datalog, è quello

di controllarne la sua correttezza dal punto di vista della sintassi e produrre in output un nuovo programma che sia rispondente alla sintassi del motore inferenziale. Qui di seguito riportiamo un esempio di traduzione del codice Sequence Datalog.

Example 4. **[Traduzione]** Consideriamo le regole riportate nell'Esempio 3, in cui veniva calcolata la sequenza complementare e inversa di tutte le sequenze contenute nella relazione *sequenza* usando la nozione di complementarietà memorizzata nella relazione *compl*:

$$\begin{aligned} reverse\_compl(\text{"","""}) &\leftarrow true. \\ reverse\_compl(S[1:N+1], Y\$Z) &\leftarrow sequenza(S), \\ &\quad reverse\_compl(S[1:N], Z), \\ &\quad compl(S[N+1], Y). \end{aligned}$$

A seguito della traduzione, viene prodotto il seguente codice. Si noti l'uso dei predicati *substring* e *concat* che implementano le operazioni sulle sequenze.

```
reverse_compl("","").
reverse_compl(S_1,CONCAT_reverse_compl1_1) :- sequenza(S),
                            substr(S,1,N,S_2),
                            reverse_compl(S_2,Z),
                            INDICE1_S_3 = N+1,
                            substr(S,INDICE1_S_3,INDICE1_S_3,S_3),
                            compl(S_3,Y),
                            INDICE2_S_1 = N+1,
                            substr(S,1,INDICE2_S_1,S_1),
                            concat([Y,Z],CONCAT_reverse_compl1_1).
```

3.3 Interfaccia

Il prototipo fornisce all'utente due modalità di interazione in modo che quest'ultimo possa scegliere quella a lui più adeguata:

- nella *modalità batch* il sistema genera integralmente il modello del programma sulla base di dati, e cioè tutti i fatti derivabili a partire dalla base di dati utilizzando le regole del programma; il modello viene salvato in un file, che può essere in seguito interrogato dall'utente oppure utilizzato come base di dati per ulteriori elaborazioni;
- la *modalità interattiva*, invece, consente all'utente di ottenere la risposta a specifiche interrogazioni inserite tramite un prompt fornito dall'interfaccia. Tali interrogazioni possono essere *goal* come "? $domesteso(X)$.", che consente di ottenere tutti gli elementi contenuti nel dominio attivo esteso, oppure delle regole che seguono la sintassi del Sequence Datalog come

  $$? \; suffisso(X[N:end]) \leftarrow r(X).$$

  per ottenere tutti i suffissi delle sequenze contenute nella relazione r. In questo modo, utilizzando regole della forma $p_0(..) \leftarrow p_1(...), ..., p_n(...).$, è possibile effettuare operazioni di join, selezione e proiezione utilizzando la sintassi Sequence Datalog.

4 Valutazione di un programma

Un programma P Sequence Datalog viene utilizzato per effettuare le interrogazioni su una base di dati. Il programma è formato da varie parti:

- un *file delle regole*, contenente un insieme S di clausole che rispettano la sintassi Sequence Datalog;
- un *file di schema*, contenente lo *schema* R della base di dati;
- un *file di istanza*, contenente l'*istanza* DB della base di dati, contenente i valori effettivamente in essa presenti;

$\langle schema \rangle \rightarrow$	$\langle predicato \rangle$ "."
	$\mid \langle schema \rangle \langle predicato \rangle$ "."
$\langle predicato \rangle \rightarrow$	$\langle predicato\_in \rangle$
	$\mid \langle predicato\_out \rangle$
$\langle predicato\_in \rangle \rightarrow$	$\langle nome\_predicato \rangle$ "(" $\langle lista\_argomenti \rangle$ ")"
$\langle lista\_argomenti \rangle \rightarrow$	$\langle argomento \rangle$
	$\mid \langle argomento \rangle$ "," $\langle lista\_argomenti \rangle$
$\langle argomento \rangle \rightarrow$	"$integer$"
	$\mid$ "$float$"
	$\mid$ "$sequence$"
$\langle predicato\_out \rangle \rightarrow$	$\langle predicato\_in \rangle$ "(" $\langle lista\_flussi \rangle$ ")"
$\langle lista\_flussi \rangle \rightarrow$	$\langle flusso \rangle$
	$\mid \langle lista\_flussi \rangle$ "," $\langle flusso \rangle$
$\langle flusso \rangle \rightarrow$	"f"
	$\mid$ "b"
	$\mid \langle flusso \rangle$ "f"
	$\mid \langle flusso \rangle$ "b"
$\langle nome\_predicato \rangle \rightarrow$	IDENTIFICATORE

Fig. 2. Sintassi dello Schema

All'interno del programma possono comparire predicati di tre tipi:

- i *predicati di input* sono i predicati contenuti all'interno della base di dati; all'interno del programma possono comparire solo nel corpo delle regole (in quanto premesse per poter derivare nuovi fatti);
- i *predicati di output* quelli visibili all'utente per le interrogazioni; all'interno del programma possono comparire solo nella testa delle regole;
- i *predicati di lavoro* sono tutti i predicati presenti all'interno del sorgente Sequence Datalog ma che non compaiono nell'istanza della base di dati e non sono destinati ad essere esportati dal programma.

Di questi, solo i predicati di input e di output vengono memorizzati all'interno del file di schema (la cui sintassi è mostrata nella figura 2). Per ognuno di questi

predicati viene specificata l'arità e il tipo di dati dei relativi argomenti, al fine di poter verificare la correttezza del programma e del file di istanza.

Per quanto riguarda i predicati di output, vengono differenziati da quelli di input, in quanto all'interno del file di schema vanno specificati i flussi di esportazione da ottimizzare. Un flusso di ottimizzazione è una sequenza di caratteri 'b' o 'f' [2] di lunghezza uguale alla arità del predicato. Per ogni predicato di output bisogna specificare almeno un flusso, ed eventualmente anche più di uno.

$\langle istanza \rangle \rightarrow$	$\langle predicato \rangle$ "."
	$\mid \langle istanza \rangle \langle predicato \rangle$ "."
$\langle predicato \rangle \rightarrow$	$\langle nome\_predicato \rangle$ "(" $\langle lista\_argomenti \rangle$ ")"
$\langle lista\_argomenti \rangle \rightarrow$	$\langle argomento \rangle$
	$\mid \langle lista\_argomenti \rangle$ "," $\langle argomento \rangle$
$\langle nome\_predicato \rangle \rightarrow$	IDENTIFICATORE
$\langle argomento \rangle \rightarrow$	INTEGER
	| FLOAT
	| SEQUENCE

Fig. 3. Sintassi della istanza

Il file di istanza (la cui sintassi è mostrata nella figura 3) contiene invece tutta la conoscenza disponibile nel programma al fine di effettuare delle deduzioni. I predicati contenuti all'interno del file di istanza devono essere stati precedentemente dichiarati nel file di schema con gli esatti tipi di dati.

Infine, il file delle regole contiene le regole del programma. Il file si compone di due parti:

- *istruzioni per il caricamento delle basi di dati utilizzate*: In tale sezione vanno specificati i files di schema e di istanza necessari nel programma; per includere nel programma il file di schema *nomefile.dbs* bisogna inserire nel programma una opportuna istruzione, della forma *schema(nomefile)*; in modo analogo, per includere il file di istanza *nomefile.dbi* bisogna inserire l'istruzione *instance(nomefile)*;
- *regole Sequence Datalog*: Tale sezione contiene l'insieme di clausole che costituisce il programma vero e proprio. Tali clausole hanno la forma:

$$p_0(t_1^0, ..., t_{n_0}^0) \ :- \ p_1(t_1^1, ..., t_{n_1}^1), \ ... \ , \ p_m(t_1^m, ..., t_{n_m}^m).$$

  in cui p_0 è un *predicato derivato* (predicato di output o di lavoro) e $p_1, ..., \ p_n$ sono predicati corrispondenti a predicati di base (predicati di input) o predicati derivati. I termini t_i^j possono essere variabili, numeri interi, numeri

[2] L'interrogazione dei predicati avviene per mezzo di un goal, costituito da un punto interrogativo, dal nome del predicato da interrogare e da una lista di termini. Tali termini sono detti bound (b) se sono o dei termini costanti o delle variabili il cui valore è legato ad un termine costante; sono invece detti free (f) se sono delle variabili libere (non legate a nessun termine costante).

⟨*programma*⟩	→	⟨*caricamento_bd*⟩⟨*regole*⟩
⟨*caricamento_bd*⟩	→	⟨*carica_schema_bd*⟩ | ⟨*carica_istanza_bd*⟩ | ⟨*caricamento_bd*⟩⟨*carica_schema_bd*⟩ | ⟨*caricamento_bd*⟩⟨*carica_istanza_bd*⟩
⟨*carica_schema_bd*⟩	→	"*schema*(" ⟨*nome_schema*⟩ ")."
⟨*carica_istanza_bd*⟩	→	"*instance*(" ⟨*nome_istanza*⟩ ")."
⟨*regole*⟩	→	⟨*regole*⟩⟨*regola*⟩ "." | ⟨*regola*⟩ "."
⟨*regola*⟩	→	⟨*head*⟩ " : −" ⟨*lista_tail*⟩ | ⟨*head*⟩
⟨*head*⟩	→	⟨*predicato_h*⟩
⟨*lista_tail*⟩	→	⟨*lista_tail*⟩ "," ⟨*tail*⟩ | ⟨*tail*⟩
⟨*tail*⟩	→	⟨*espr_logica*⟩ | ⟨*predicato_t*⟩
⟨*espr_arit*⟩	→	⟨*variabile*⟩ | ⟨*numero*⟩ | ⟨*variabile*⟩ "*mod*" ⟨*numero*⟩ | ⟨*espr_arit*⟩ " + " ⟨*espr_arit*⟩ | ⟨*espr_arit*⟩ " − " ⟨*espr_arit*⟩ | ⟨*espr_arit*⟩ " ∗ " ⟨*espr_arit*⟩ | ⟨*espr_arit*⟩ "/" ⟨*espr_arit*⟩ | "(" ⟨*espr_arit*⟩ ")"
⟨*espr_str*⟩	→	⟨*term_indic*1⟩ | SEQUENZA
⟨*espr_logica*⟩	→	"*true*" | "*false*" | ⟨*espr_arit*⟩⟨*op_relaz*⟩⟨*espr_arit*⟩ | ⟨*carattere*⟩⟨*op_relaz*⟩⟨*carattere*⟩ | ⟨*espr_str*⟩ ⟨*op_relaz*1⟩ ⟨*espr_str*⟩
⟨*op_relaz*⟩	→	⟨*op_relaz*1⟩ | ⟨*op_relaz*1⟩
⟨*op_relaz*1⟩	→	" = " | " <> "
⟨*op_relaz*2⟩	→	" < " | " <= " | " > " | " >= "
⟨*predicato_h*⟩	→	⟨*nome_predicato*⟩ | ⟨*nome_predicato*⟩ "(" ⟨*lista_argomenti_h*⟩ ")"
⟨*lista_argomenti_h*⟩	→	⟨*argomento_h*⟩ | ⟨*lista_argomenti_h*⟩ "," ⟨*argomento_h*⟩
⟨*argomento_h*⟩	→	VARIABILE | SEQUENZA | ⟨*numero*⟩ | ⟨*term_indic*⟩ | ⟨*term_costr*⟩
⟨*predicato_t*⟩	→	⟨*nome_predicato*⟩ | ⟨*nome_predicato*⟩ "(" ⟨*lista_argomenti_t*⟩ ")"
⟨*lista_argomenti_t*⟩	→	⟨*argomento_t*⟩ | ⟨*lista_argomenti_t*⟩ "," ⟨*argomento_t*⟩
⟨*argomento_t*⟩	→	VARIABILE | SEQUENZA | ⟨*numero*⟩ | ⟨*term_indic*⟩
⟨*term_indic*⟩	→	⟨*term_indic*1⟩ | ⟨*term_indic*2⟩
⟨*term_indic*1⟩	→	⟨*variabile*⟩ "[" ⟨*indice*⟩ " : " ⟨*indice*⟩"]"
⟨*term_indic*2⟩	→	⟨*variabile*⟩ "[" ⟨*indice*⟩ "]"
⟨*indice*⟩	→	VARIABILE | INTERO | "*end*" | ⟨*indice*⟩ " + " ⟨*indice*⟩ | ⟨*indice*⟩ " − " ⟨*indice*⟩ | ⟨*indice*⟩ " ∗ " ⟨*indice*⟩ | ⟨*indice*⟩ "/" ⟨*indice*⟩
⟨*term_costr*⟩	→	⟨*elem_costr*⟩ "$" ⟨*elem_costr*⟩ \| ⟨*term_costr*⟩ "$" ⟨*elem_costr*⟩
⟨*elem_costr*⟩	→	VARIABILE | SEQUENZA | ⟨*term_indic*⟩
⟨*carattere*⟩	→	⟨*term_indic*2⟩ | CARATTERE
⟨*nome_predicato*⟩	→	IDENTIFICATORE
⟨*nome_schema*⟩	→	SEQUENZA
⟨*nome_istanza*⟩	→	SEQUENZA
⟨*numero*⟩	→	NUM_INTERO | NUM_REALE

Fig. 4. Sintassi del programma

reali, sequenze e termini indicizzati o costruttivi (questi ultimi compaiono solo nella testa delle regole).

Diamo ora un esempio di un programma Sequence Datalog, riportandone tutte le sue componenti:

– **File di schema:** *'demo.dbs'*:

```
sequenza_car(sequence).
sequenza_dna(integer,sequence).
compl(sequence,sequence).

abc(sequence,integer)(ff,fb).
reverse_compl(sequence,sequence)(ff,bf).
```

– **File di istanza:** *'demo.dbi'*:

```
sequenza_car("aabbcc").
sequenza_car("aaabbbccc").
sequenza_car("abcc").

sequenza_dna(1,"accaggcacgat").
sequenza_dna(2,"tcgatcatcac").

compl("a","t").
compl("c","g").
compl("t","a").
compl("g","c").
```

– **File delle regole:** *'demo.sd'*:

```
schema("demo").
instance("demo").


abc(X) :-     sequenza_car(X),
              length(X,L),
              L mod 3=0,
              abcn(X[1:L/3],X[L/3+1:L*2/3],X[L*2/3+1:L]).

abcn("","","").
abcn(X,Y,Z) :- X[1]='a',
               Y[1]='b',
               Z[1]='c',
               abc_(X[2:end],Y[2:end],Z[2:end]).

reverse_compl("","").
reverse_compl(S[1:N+1],Y$Z) :-  sequenza_dna(M,S),
                                reverse_compl(S[1:N],Z),
                                compl(S[N+1],Y).
```

Riassumiamo ora i passi da eseguire per valutare un programma Sequence Datalog. Per cominciare, l'utente specifica un programma P, composto da uno schema di base di dati R, una istanza di base di dati DB ed un insieme di clausole S, e chiede al sistema di valutarlo. A questo punto:

1. il sistema verifica la correttezza sintattica del programma P. Se viene riscontrato un errore sintattico la valutazione viene interrotta e vengono restituiti gli opportuni messaggi di errore;
2. se non ci sono errori, l'insieme di clausole S viene tradotto in un nuovo insieme S^*, valutabile dal motore inferenziale;
3. a questo punto, il sistema si incarica di caricare la base di dati e di inizializzare le strutture necessarie alla valutazione delle regole, tra cui il dominio attivo;
4. le regole vengono poi valutate secondo la modalità di interazione richiesta;
5. infine, l'utente accede all'interfaccia, attraverso la quale può interrogare il risultato della valutazione.

5 Conclusioni

La ricerca su Sequence Datalog ha come obiettivo la definizione e la realizzazione di un linguaggio per l'interrogazione di sequenze all'interno di basi di dati. In questo articolo abbiamo descritto l'architettura utilizzata per l'implementazione del prototipo del linguaggio, basato sul motore inferenziale fornito da Coral. Nonostante il prototipo sia già stato utilizzato con successo nell'ambito di vari studi di caso su sequenze genetiche di laboratorio, ci sono aspetti che meritano ulteriore studio. In particolare, in questo contesto, sembra essenziale la cura dell'ottimizzazione del sistema, sia dal punto di vista dei metodi di accesso alle sequenze nella base di dati, sia dal punto di vista della valutazione delle interrogazioni. Entrambi questi aspetti rappresentano interessanti direzioni di ricerca futura.

References

1. S. Abiteboul, S. Cluet, and T. Milo. Querying and updating the file. In *International Conf. on Very Large Data Bases (VLDB'93), Dublin*, pages 73–84, 1993.
2. S. Abiteboul, S. Cluet, and T. Milo. A database interface for file update. In *ACM SIGMOD International Conf. on Management of Data (SIGMOD'95), San Jose*, pages 386–397, 1995.
3. S. Abiteboul, R. Hull, and V. Vianu. *Foundations of databases.* Addison Wesley Publ. Co., Reading, Massachussetts, 1994.
4. A. V. Aho, R. Sethi, and J. D. Ullman. *Compilers: Principles, Techniques and Tools.* Addison Wesley Publ. Co., Reading, Massachussetts, 1986.
5. P. Atzeni, editor. *LOGIDATA+: Deductive Databases with Complex Objects, Lecture Notes in Computer Science 701.* Springer-Verlag, 1993.
6. F. Bancilhon, D. Maier, Y. Sagiv, and J. D. Ullman. Magic sets and other strange ways to implement logic programs. In *Fifth ACM SIGACT SIGMOD Symp. on Principles of Database Systems (PODS'86)*, pages 1–15, 1986.

7. S. Ceri, G. Gottlob, and L. Tanca. *Logic Programming and Data Bases*. Springer-Verlag, 1989.
8. S. Ceri, G. Gottlob, and L. Tanca. What you always wanted to know about Datalog (and never dared to ask). *IEEE Transactions on Knowledge and Data Engineering*, 1(1):146–166, March 1989.
9. L. S. Colby, E. L. Robertson, L. V. Saxton, and D. Van Gucht. A query language for list-based complex objects. In *Thirteenth ACM SIGMOD Intern. Symposium on Principles of Database Systems (PODS'94)*, pages 179–189, 1994.
10. Communications of the ACM. Special issue on the Human Genome project. vol. 34(11), November 1991.
11. S. Ginsburg and X. Wang. Pattern matching by RS-operations: towards a unified approach to querying sequence data. In *Eleventh ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS'92)*, pages 293–300, 1992.
12. G. H. Gonnet. Text dominated databases: Theory, practice and experience. Tutorial presented at PODS, 1994.
13. S. Grumbach and T. Milo. An algebra for POMSETS. In *Fifth International Conference on Data Base Theory, (ICDT'95), Prague, Lecture Notes in Computer Science*, pages 191–207, 1995.
14. G. Mecca. Querying genome databases. In *Terza Conferenza Nazionale su Sistemi Evoluti per Basi di Dati (SEBD'95), Ravello (Costiera Amalfitana), 28–30 giugno*, 1995.
15. G. Mecca and A. J. Bonner. Finite query languages for sequence databases. In *Fifth Intern. Workshop on Database Programming Languages (DBPL'95), Gubbio, Italy*, 1995.
16. G. Mecca and A. J. Bonner. Sequences, Datalog and Transducers. In *Fourteenth ACM SIGMOD Intern. Symposium on Principles of Database Systems (PODS'95), San Jose, California*, pages 23–35, 1995.
17. R. Ramakrishnan, P. Seshadri, D. Srivastava, and S. Sudarshan. The CORAL user manual: A tutorial introduction to CORAL. Unpublished manuscript, 1993.
18. R. Ramakrishnan, D. Srivastava, and S. Sudarshan. CORAL – control, relations and logic. In *Eighteenth International Conference on Very Large Data Bases (VLDB'92), Vancouver, Canada*, pages 238–250, 1992.
19. J. Richardson. Supporting lists in a data model (a timely approach). In *Eighteenth International Conference on Very Large Data Bases (VLDB'92), Vancouver, Canada*, pages 127–138, 1992.
20. D. Stott Parker, E. Simon, and P. Valduriez. SVP – a model capturing sets, streams and parallelism. In *Eighteenth International Conference on Very Large Data Bases (VLDB'92), Vancouver, Canada*, pages 115–126, 1992.
21. X. Wang. *Pattern matching by RS-operations: Towards a unified approach to querying sequence data*. PhD thesis, University of Southern California, 1992.