UNIVERSITÀ DEGLI STUDI DI MODENA
Facoltà di Ingegneria
Corso di Laurea in Ingegneria Informatica

---

---

# ODB-QOptimizer: un sistema per l'Ottimizzazione delle interrogazioni nelle Basi di Dati ad Oggetti Complessi

Tesi di Laurea di
Maurizio Vincini

Relatore
Chiar.ma Prof. Sonia Bergamaschi

Correlatori
Dott. Jean Paul Ballerini
Dott. Ing. Domenico Beneventano

Anno Accademico 1993 - 94

*Ai miei genitori*

Parole chiave:
Basi di dati
Oggetti complessi
Sussunzione Incoerenza
Interrogazione di basi di dati
Ottimizzazione semantica di interrogazioni

# RINGRAZIAMENTI

Ringrazio la Prof.ssa Sonia Bergamaschi, il Dott. Jean Paul Ballerini e l'Ing. Domenico Beneventano per l'aiuto fornito alla realizzazione della presente tesi.

# Indice

# Elenco delle figure

# Elenco delle tabelle

# Capitolo 1

# Introduzione

Le Basi di Dati Orientate ad Oggetti, OODB (*Object Oriented Database*), sono attualmente il soggetto di intensi sforzi di ricerca e di sviluppo. La motivazione principale per questo interesse è che il paradigma orientato ad oggetti offre un grande insieme di strutture dati e di facilità di manipolazione che lo rendono adatto per il supporto sia delle tradizionali che delle nuove applicazioni.

L'aspetto fondamentale di un modello di dati orientato ad oggetti, OODM (*Object Oriented Data Model*), proposti per OODB in [A+89] e [KFL89] è che sono basati sulla definizione di oggetti, classi, attributi, inferenza e metodi, dove le classi e gli attributi sono usati per descrivere gli aspetti strutturali (la conoscenza in un dominio di applicazione) mentre i metodi sono usati per rappresentare gli aspetti comportamentali (come ad esempio l'incapsulazione).

Il problema di rappresentare la conoscenza è stato trattato anche dalla comunità della Intelligenza Artificiale. In questo settore la ricerca ha prodotto soprattutto formalismi di rappresentazione della conoscenza basati su una logica ristretta in modo da ottenere efficienti tecniche di inferenza.

In particolare in questa tesi viene utilizzata la definizione formale di un modello di dati orientato ad oggetti presentato in [BN94] e il suo accoppiamento con tecniche di inferenza sviluppate nei modelli di rappresentazione della conoscenza nell'area dell'Intelligenza Artificiale. Le tecniche di inferenza sono basate sulla relazione di sussunzione tra due classi di oggetti, cioè sulla relazione esistente tra due classi quando l'appartenenza di un oggetto ad una classe implica necessariamente l'appartenenza dell'oggetto all'altra classe.





Queste tecniche di inferenza possono essere utilizzate in attività centrali delle basi di dati, come l'acquisizione di schemi consistenti e minimali, e, come verrà discusso nella tesi, l'ottimizzazione delle interrogazioni, più precisamente, l'ottimizzazione semantica delle interrogazioni.

L'obiettivo di questa tesi è quello di progettare e sviluppare il componente ODB-QOptimizer (Object Data Base Query Optimizer) capace, a runtime, di ottenere l'ottimizzazione semantica delle interrogazioni utilizzando il componente ODL-Designer presentato in [Bal92].

## 1.1 I modelli di dati orientati ad oggetti

Nel seguito sono brevemente riportate le principali caratteristiche strutturali di un OODM.

Ogni elemento componente la realtà da modellare viene rappresentato tramite un unico concetto di base: l'*oggetto*. Ogni oggetto è unicamente identificato da un identificatore di oggetto, *oid* (object identifier) [KC86], ed ha associato uno stato che è costituito dal valore delle sue *proprietà* o *attributi*. Nei classici linguaggi orientati ad oggetti, come ad esempio Smalltalk [GR83], il valore associato ad un oggetto è sempre atomico oppure una tupla di altri oggetti. Questa caratteristica è stata in parte ereditata da alcuni sistemi OODB, dove questo valore è una tupla oppure un insieme di altri oggetti, cioè è sempre un valore piatto che può solo contenere identificatori di altri oggetti e non direttamente altri valori complessi.

Per superare questa limitazione sono stati sviluppati diversi modelli ad oggetti con valori complessi [AK89, Atz93, LR89a, Bee90]. In questi modelli si trattano in modo uniforme sia oggetti con identità sia valori complessi senza identità. Un *valore complesso* o *valore strutturato* è un valore definito a partire sia da valori atomici che da identificatori di oggetti mediante l'uso ricorsivo di costruttori, quali ad esempio il costruttore di *tupla*, di *insieme*, e di *sequenza*. Uno *schema* contiene le informazioni sulla struttura dei dati. Nei citati lavori sono presenti entrambe le nozioni di *classe* e di *tipo*. I tipi denotano una struttura e una *estensione*, intesa come insieme di valori. Anche le classi denotano un'estensione, intesa come insieme di oggetti. Comunque, mentre l'estensione denotata da un tipo è definita dalla sua struttura, l'estensione associata ad una classe è definita dall'utente. Ad ogni classe è normalmente associato un tipo che descrive la struttura degli oggetti che possono essere *istanziati* nella classe. Quindi le istanze della classe sono

oggetti il cui valore associato è, a sua volta, istanza del tipo che descrive la classe.

L'*ereditarietà*, stabilita tramite la relazione *isa*, è un'importante caratteristica degli OODB. Uno dei principali vantaggi dell'ereditarietà è che essa costituisce un potente mezzo di modellazione, essendo in grado di dare una precisa e coincisa descrizione del dominio di applicazione [A+89]. Da un punto di vista intensionale, la dichiarazione di ereditarietà tra due classi $A$ e $B$, cioè $A$ *isa* $B$, consente la definizione della *sottoclasse* $A$ come specializzazione della *superclasse* $B$. Una sottoclasse eredita le proprietà della superclasse, può avere proprietà aggiuntive e può ridefinire alcune proprietà della superclasse. Da un punto di vista estensionale, la dichiarazione $A$ *isa* $B$, stabilisce che ogni oggetto di $A$ sia anche un oggetto di $B$. Nel caso in cui una classe può avere più di una superclasse si parla di *ereditarietà multipla*.

Il modello introdotto nella presente tesi è un modello per basi di dati orientate ad oggetti con ereditarietà multipla che permette la modellazione di valori complessi tramite la nozione di tipo e di classe.

## 1.2 Sussunzione nei modelli orientati ad oggetti

Nei classici modelli orientati ad oggetti la descrizione di una classe è intesa solo a rappresentare una struttura dati e la classe deve essere esplicitamente riempita con oggetti, cioè l'*estensione* della classe è soggetta solo a condizioni necessarie. Nel nostro modello questa semantica viene rappresentata tramite i cosidetti concetti *primitivi*. In aggiunta un concetto può essere *definito*, nel qual caso la struttura esprime condizioni sia necessarie che sufficienti.

Quindi, la fondamentale differenza del modello proposto nella tesi rispetto ai precedenti OODM, è la nozione, accanto a quella originaria di classequi denominata come classe base, di *definizione di classe* tramite la cosiddetta *classe virtuale*. Si è preferito adottare il termine *virtuale*, anziché *definita*, perchè nella terminologia delle basi di dati tradizionalmente i tipi la cui estensione è determinata sulla base della loro espressione vengono denominati in questo modo. Inoltre, la nozione di *definizione di classe* è simile alla nozione di *vista* delle basi di dati, recentemente detta anche *classe virtuale* [AB91] nell'ambito dei sistemi ad oggetti.

Altra differenza è quella che nel nostro modello viene utilizzata la relazione di *sussunzione*, o generalizzazione, tra due classi, ovvero la relazione esistente tra due classi quando ogni elemento istanziato in una classe deve essere necessariamente, in virtú delle descrizioni, istanziato anche nell'altra classe. L'idea di sussunzione è quindi molto intuitiva:



$classe_1$ sussume $classe_2$ se e solo se
ogni oggetto che è $classe_2$ è anche $classe_1$.

La presenza di classi definite (virtuali) comporta che le relazioni di specializzazioni stabilite nella tassonomia non sono solo quelle dichiarate esplicitamente dall'utente ma ve ne sono delle ulteriori *implicite* nelle descrizioni delle classi. Il progettista della base di conoscenza descrive una classe in termini di ereditarietà da altre classi e di proprietà *locali*, e il sistema *classifica* automaticamente la classe, cioè determina il suo posto "giusto" nella tassonomia esistente, tra le sue più specifiche generalizzazioni e le sue più generali specializzazioni. La classificazione è effettuata tramite il cosidetto *ragionatore tassonomico* che trova tutte le relazioni di sussunzione tra la classe in questione e le classsi nella tassonomia già esistente. Il ragionatore tassonomico è quindi un servizio *inferenziale*, o *deduttivo*, fornito all'utente dal sistema di rappresentazione.

Altro concetto utilizzato nella tesi è quello di *incoerenza*: intuitivamente un tipo si dice incoerente sse non è possibile trovare una istanza che soddisfi le condizioni di appartenenza alla descrizione del tipo.

L'intera base di dati si dirá incoerente se esiste almeno un tipo incoerente.

## 1.3 L'ottimizzazione semantica delle interrogazioni

L'applicazione del ragionamento tassonomico ai tradizionali modelli semantici dei dati porta a promettenti risultati per il progetto di schemi per basi di dati [FS86, DD89, BS92] e per altri rilevanti aspetti quali l'elaborazione di interrogazioni e il riconoscimento dei dati [BGN89, BBMR89]. In particolare, nella nostra tesi prenderemo in considerazione una nuova area applicativa, relativa all'elaborazione di interrogazioni: l'ottimizzazione semantica delle interrogazioni [BBS94].

L'obiettivo dell'ottimizzazione delle interrogazioni è quella di trasformare un'interrogazione in una *equivalente*, la quale restituisce lo stesso risultato di

quella originale, con un minor costo di esecuzione. A differenza dell'ottimizzazione convenzionale [JK84], basata sulla conoscenza sintattica dell'interrogazione e sull'organizzazione fisica della base di dati, con la tecnica dell'ottimizzazione semantica le trasformazioni avvengono utilizzando la conoscenza semantica relativa alla base di dati [Kin81b, HZ80]. L'idea di base è che i *vincoli di integrità*, espressi per forzare la consistenza di una base di dati, possono essere utilizzati anche per ottimizzare le interrogazioni fatte dall'utente. Quindi il modello ad oggetti viene arricchito con vincoli di integrità espressi come *regole* [Ben94]. Intuitivamente il processo di espansione semantica è semplice: a partire da uno schema che definisce l'ODBMS di partenza supponiamo di aggiungere, a run-time, una interrogazione. L'ottimizzazione consiste nel controllare dapprima che l'interrogazione sia coerente (cioè che non esprima una classe di oggetti vuota) e poi cercare di incorporare ogni possibile restrizione che non sia presente nell'interrogazione di partenza ma che è *logicamente implicata* dal tipo della query e dallo schema (in particolare dai vincoli di integrità). L'espansione semantica è basata sull'iterazione di questa semplice trasformazione: se l'interrogazione implica l'antecedente di un vincolo di integrità allora posso aggiungere il conseguente di quel particolare vincolo di integrità. Si noti che una query esprime la semantica di una classe virtuale, cioè la sua descrizione esprime un insieme di condizioni necessarie e sufficienti affinchè un oggetto del dominio applicativo appartenga all'insieme degli oggetti che soddisfano la query.

La relazione di implicazione logica viene ottenuta attraverso l'uso della relazione di sussunzione: ciò è motivato soprattutto dalla considerazione che la definizione formale di sussunzione come relazione semantica permette di dimostrare la correttezza delle trasformazioni, cioè di dimostrare che le interrogazioni trasformate siano equivalenti a quella originale [BM92]. Un'altra considerazione è che le trasformazioni riguardano esclusivamente quella parte dell'interrogazione che è esprimibile come una descrizione del modello, quindi un'espressione per la quale, in generale, il calcolo della sussunzione è efficiente.

Il risultato finale del metodo da noi proposto è quello di riuscire a riclassificare l'interrogazione dopo l'espansione semantica secondo la tassonomia delle classi dello schema rispetto alle nuove relazione di ereditarietà (*isa*) trovate e quindi, in generale di ottenere uno spostamento verso il basso della query nella tassonomia, restringendo l'insieme degli oggetti candidati a soddisfare la query.



## 1.4 Esempio: il dominio Magazzino

Volendo illustrare il nostro metodo, andiamo a considerare il seguente esempio che riguarda la struttura di un società che si occupa della gestione di un magazzino (vedi figura 1.1, dove le classi sono rappresentate da ellissi, le relazioni di specializzazione esplicite da frecce in neretto, le relazioni di sussunzione calcolate da frecce tratteggiate e gli attributi da archi orientati.

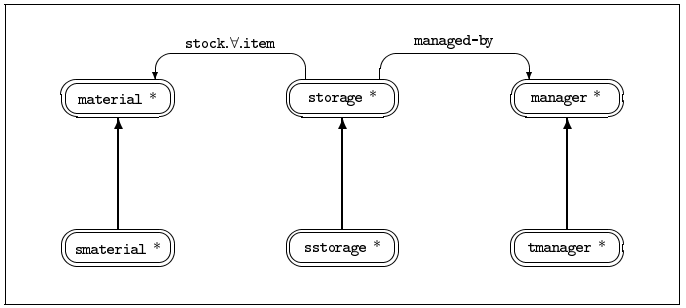



Figura 1.1: Il dominio Magazzino

I materiali (`material`) sono descritti da un nome (`name`) dato da una stringa di caratteri, un rischio (`risk`) dato da un intero e da una caratteristica (`feature`) data da un insieme di stringhe. I magazzini (`storage`) sono identificati da una categoria (`category`) data da una stringa, sono guidati (`managed-by`) da un dirigente (`manager`) e contengono (`stock`) un insieme di articoli (`item`) che sono dei materiali (`material`) per ciascuno dei quali è indicata con un intero la quantità presente (`qty`). I dirigenti (`manager`) hanno un nome (`name`) dato da una stringa, un salario (`salary`) superiore ai 40K dollari e un livello (`level`) compreso tra 1 e 15. I massimi dirigenti (`tmanager`) sono quei dirigenti (`manager`) che hanno un livello compreso tra 8 e 12. Infine abbiamo dei magazzini marcati come "speciali" (`sstorage`) che sono magazzini (`storage`) e materiali "speciali" (`smaterial`) che sono materiali (`material`). Le classi che abbiamo descritto sono tutte di tipo base, avendo fissato solo condizioni necessarie per l'appartenenza, quindi, per ora, le relazioni di sussunzione sono soltanto quelle esplicite mostrate dalle frecce in neretto in figura 1.1.

A queste classi vogliamo ora aggiungere i vincoli di integrità, i quali rappresentano condizioni necessarie e sufficienti per la legalità delle istanze dello schema. Questi vincoli possono essere descritti in linguaggio naturale nel seguente modo:

" **per tutti** i dirigenti (`manager`),
**se** il livello (`level`) è compreso tra 5 e 10,
**allora** il salario (`salary`) deve essere compreso tra 40K e 60K. "

" **per tutti** i materiali (`material`),
**se** il rischio (`risk`) è maggiore di 10,
**allora** devono essere dei materiali speciali (`smaterial`). "

" **per tutti** i magazzini (`storage`),
**se** la categoria (`category`) è "B4",
**allora** devono essere guidati da un massimo dirigente (`tmanager`). "

" **per tutti** i magazzini (`storage`),
**se** ciascun articolo (`item`) immagazzinato (`stock`) è di un materiale speciale (`smaterial`),
**allora** devono essere dei magazzini speciali (`sstorage`). "

" **per tutti** i magazzini (`storage`),
**se** ciascuna quantità (`qty`) immagazzinata (`stock`) è compresa tra 10 e 50,
**allora** la categoria (`category`) deve essere "A2". "

Usando un linguaggio di tipo OODB-like, lo schema del Magazzino comprensivo dei vincoli di integrità è riportato nella tabella 1.1.

A partire dallo schema con regole vogliamo ora mostrare come opera il metodo di ottimizzazione semantica delle interrogazioni sviluppato in questa tesi ed introdotto in [BBS94]; il primo esempio riguarda la rilevazione di una interrogazione incoerente, la quale permette di fornire una risposta senza accedere al database.
Vediamo quindi la seguente interrogazione:



| | | | |
|---|---|---|---|
| **class** | material | = | [name: string, risk: integer, feature : {string}] |
| **class** | smaterial | = | **isa** material |
| **class** | storage | = | [managed-by : manager, category : string, stock : {item : material, qty : $10 \div 300$]}] |
| **class** | sstorage | = | **isa** storage |
| **class** | manager | = | [name: string, salary: $40K \div \infty$, level : $1 \div 15$] |
| **class** | tmanager | = | **isa** manager **and** [level : $8 \div 12$] |

Vincoli di integrità:

| | | | |
|---|---|---|---|
| **if** manager | **and** (level : $5 \div 10$) | **then** | (salary : $40K \div 60K$) |
| **if** material | **and** (risk $> 10$) | **then** | smaterial |
| **if** storage | **and** (category $= "B4"$) | **then** | (managed-by : tmanager) |
| **if** storage | **and** (stock.$\forall$.item = smaterial) | **then** | sstorage |
| **if** storage | **and** (stock.$\forall$.qty : $10 \div 50$) | **then** | (category $= "B4"$) |

Tabella 1.1: Lo schema del Magazzino in una sintassi OODB–like

$Q_1$ : "Seleziona i dirigenti che hanno un livello superiore a 20".

La definizione di una interrogazione è esplicitamente una classe virtuale, poichè sono richieste condizioni necessarie e sufficienti di appartenenza, quindi, nella sintassi OODB-like introdotta, la $Q_1$ può essere espressa nel seguente modo:

**virtual class** $Q_1$ = **isa** manager **and** [level : $20 \div \infty$]

Poichè l'intervallo per il livello definito nell'interrogazione (level : $20 \div \infty$) è disgiunto da quello definito nella classe manager (level : $1 \div 15$) risulta che $Q_1$ è incoerente quindi nessun dominio di oggetti può popolare la classe $Q_1$.
Se ora applichiamo il nostro metodo di ottimizzazione semantica alla $Q_1$ viene riconosciuta l'incoerenza dell'interrogazione e quindi possiamo subito affermare che la ricerca non fornirá alcun risultato senza dover accedere fisicamente al database.

Il successivo esempio illustra l'effettiva ottimizzazione calcolata tramite la tecnica dell'espansione semantica che riclassifica l'interrogazione introducendo la conoscenza contenuta nei vincoli di integrità:

$Q_2$ : "Seleziona i magazzini che hanno categoria "B4", sono guidati da un funzionario con livello inferiore a 10 e contengono materiali ciascuno con rischio superiore a 10".

La corrispondente rappresentazione in sintassi OODB-like diventa:

**virtual class** $Q_2$ = **isa** storage
**and** [managed-by : **isa** manager
**and** (level $< 10$),
stock.∀.item : **isa** material
**and** (risk $> 10$),
category = "$B4''$ ]

Per l'interrogazione $Q_2$ possiamo immediatamente calcolare la classificazione nella gerarchia delle classi che è rappresentata in figura 1.2.a.
Vediamo il comportamento dell'ottimizzatore semantico: poichè l'interrogazione è coerente cerca di applicare i vincoli di integrità: a $Q_2$ possiamo applicare il secondo e il terzo vincolo di integrità (essendo verificate le condizioni) dando origine ad una nuova interrogazione trasformata:

**virtual class** $Q_2'$ = **isa** storage
**and** [managed-by : **isa** tmanager
**and** (level $< 10$),
stock.∀.item : **isa** smaterial
**and** (risk $> 10$),
category = "$B4''$ ]

Iterando il procedimento di controllo sull'applicabilità dei vincoli di integrità alla $Q_2'$ l'ottimizzatore trova che possiamo applicare il primo e il quarto vincolo dando origine alla:



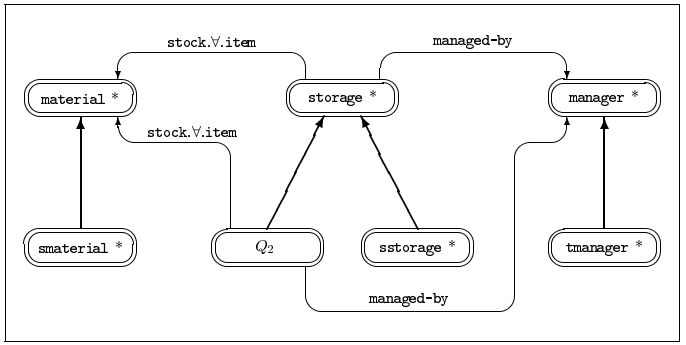



Figura 1.2.a: La query $Q_2$ prima dell'espansione semantica

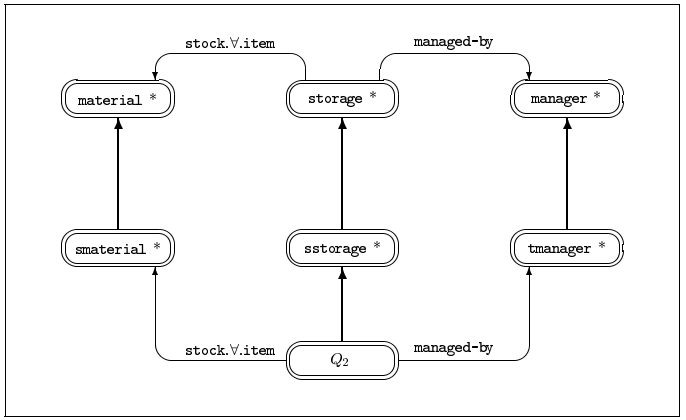



Figura 1.2.b: La query $Q_2$ dopo l'espansione semantica

Figura 1.2: Classificazione dell'interrogazione nella tassonomia delle classi

**virtual class** $Q_2''$ = **isa** sstorage
**and** [managed-by : **isa** tmanager
**and** (level $< 10$),
**and** (salary $: 40K \div 60K$),
stock.$\forall$.item : **isa** smaterial
**and** (risk $> 10$),
category = "$B4''$ ]

Alla classe $Q_2''$ non è possibile applicare ulteriori vincoli di integrità; ció significa che abbiamo trovato l'espansione semantica dell'interrogazione che rappresenta la classe più specializzata tra quelle equivalenti all'interrogazione di partenza.

La trasformazione ottenuta per la $Q_2$ porta alla nuova rappresentazione dell'interrogazione nella gerarchia delle classi mostrata in figura 1.2.b. Si può notare come l'interrogazione risulti ottimizzata in quanto essa è ora una specializzazione di `sstorage` (invece della sola classe `storage`) ed inoltre contenga l'attributo `managed-by` il cui dominio ha valori in `tmanager` (anzichè `manager`) e l'attributo `stock` con valori in `smaterial` (anzichè `material`).

## 1.5 Contenuto della tesi

Il secondo capitolo presenta il modello di dati ad oggetti complessi ODL, con particolare attenzione alla definizione di oggetti complessi, di schema di base di dati, dell'operatore di intersezione, tramite il quale viene descritta l'ereditarietà, e di istanze (possibili e legali) di uno schema. Successivamente viene definita la relazione di sussunzione, e una conseguente relazione di incoerenza. L'introduzione della sussunzione negli OODM è stata presentata nelle seguenti pubblicazioni [BBS91a, BBS91b, BBS91c, BBS93, BN91]; in particolare, il lavoro [BN91] è la base del modello presentato nella tesi.

Il terzo capitolo contiene la definizione di schema canonico, il cui principale obiettivo è quello di rendere relativamente semplici i problemi del controllo di incoerenza e del calcolo di sussunzione, che invece sono complicati in uno schema generale. In questo capitolo vengono infatti presentati gli algoritmi di sussunzione e di incoerenza basati sullo schema canonico.



Il contenuto di questi due capitoli riprende i lavori [BN91, BB93, Ben94].

Il quarto capitolo propone la relazione di sussunzione come strumento per effettuare l'ottimizzazione semantica delle interrogazioni. Prima di affrontare il problema dell'ottimizzazione, allo scopo di aumentare i vincoli di integrità esprimibili in forma dichiarativa in un modello di dati orientato ad oggetti, viene introdotta la nozione di *regola di integrità*. Le regole di integrità sono aggiunte alla nozione di schema definita in precedenza e stabiliscono *relazioni di inclusione* tra generici tipi; per esse verrà data una precisa semantica, investigando formalmente il loro effetto sull'istanza di uno schema di dati e sulla relazione di sussunzione tra tipi. Nella seconda parte del capitolo si considera un'interrogazione, o parte di essa, esprimibile come tipo del sistema, e precisamente come classe virtuale. Per queste interrogazioni il metodo proposto per l'ottimizzazione semantica viene prima illustrato con alcuni esempi e successivamente formalizzato. Viene infine presentato un algoritmo di calcolo dell'espansione semantica di un tipo.

Il quinto capitolo contiene la descrizione architetturale e funzionale del software ODB-QOpti-mizer per l'ottimizzazione semantica di interrogazioni implementato in questa tesi: vengono dapprima presentati il componente ODL-Designer [Bal92] che diventerà parte integrante dell'ottimizzatore, ed il componente GES (Generatore di Espansione Semantica) realizzato durante il lavoro di tesi, per poi definire l'architettura funzionale dell'intero sistema ODB-QOptimizer e le tecniche di interfaccia tra i diversi ambienti di lavoro ODL-Designer e GES. Infine il componte GES viene dettagliatamente descritto mediante la rappresentazione PHOS (Programming Hierarchically from Output Structure).

Il sesto capitolo descrive una sessione di lavoro con ODB-QOptimizer illustrando due esempi completi di espansione semantica di interrogazione.

Infine, nell' Appendice A viene riportato il codice sorgente del componente GES, che è stato sviluppato in linguaggio di programmazione C, versione standard ANSI C, su piattaforma hardware SUN x-sparc 10, sistema operativo sunOS, release 4.1.3.

# Capitolo 2

# ODL - Modello di Dati ad Oggetti Complessi

Come è stato detto nell'Introduzione, il modello ODL è un modello per basi di dati orientate ad oggetti che permette la modellazione di valori complessi e dell'ereditarietà multipla.

Gli aspetti principali del modello sono riportati nel seguito. Si assume una ricca struttura per il *sistema di tipi atomici* o *sistema di tipi base*: oltre ai tipi atomici *integer*, *boolean*, *string*, *real* e *tipi mono–valore*, consideriamo anche la possibilità che siano usati dei sottoinsiemi di tipi atomici, come ad esempio intervalli di interi.

Basandosi su questi tipi atomici possono essere create *tuple*, *sequenze*, *insiemi* e, in particolare, *tipi oggetto*. I tipi oggetto sono definiti tramite il costruttore $\triangle$: tale costruttore applicato ad un generico tipo $S$ definisce un insieme di oggetti con un valore associato di tipo $S$.

L'ereditarietà, sia semplice che multipla, è espressa direttamente nella descrizione di una classe, cioè nel tipo associato alla classe, tramite l'operatore *intersezione* ($\sqcap$).

Infine, ai tipi può essere dato un nome: avremo quindi nomi di tipo–valore e nomi di tipo–classe. I nomi di tipo–classe (alcune volte verrà usato semplicemente il termine nome di classe) può essere *base* (denotato anche come primitivo) o *virtuale*, per riflettere la differente semantica discussa nell'introduzione.





## 2.1 Sistema dei tipi atomici

Sia $\mathbf{D}$ l'insieme infinito numerabile dei valori atomici (che saranno indicati con $d_1, d_2, \ldots$), e.g., l'unione dell'insieme degli interi, delle stringhe e dei booleani. Non distingueremo fra i valori atomici e la loro codifica.

Sia $\mathbf{B}$ un insieme numerabile di designatori di tipi atomici (denotati da $B, B', \ldots$) che contiene $\mathbf{D}$ (i.e., tutti i tipi mono–valore), e sia $\mathcal{I}_\mathbf{B}$ la funzione di interpretazione standard (fissata) da $\mathbf{B}$ a $2^\mathbf{D}$ tale che per ogni $d \in \mathbf{D}$: $\mathcal{I}_\mathbf{B}[d] = \{d\}$. Sia "$\sqcap$" un'operazione di congiunzione su $\mathbf{B}$ definita da:

$$B' \sqcap B'' = B \text{ sse } \mathcal{I}_\mathbf{B}[B'] \cap \mathcal{I}_\mathbf{B}[B''] = \mathcal{I}_\mathbf{B}[B].$$

Diciamo che $\mathbf{B}$ è un *sistema di tipi atomici* sse $\mathbf{B}$ è completo rispetto a $\sqcap$. Il tipo speciale che ha un'interpretazione vuota è detto *tipo vuoto* ed è indicato con $\bot$.[1] Un sistema di tipi atomici $\mathbf{B}$ è detto *PTIME* sse $B' \sqcap B'' = B$ può essere deciso in tempo polinomiale. In seguito assumiamo che un sistema di tipi atomici abbia questa proprietà.

A volte parleremo anche di sistemi di tipi atomici con una particolare semplice struttura, ovvero, sistemi tali che per ogni sottoinsieme $\mathbf{X} \subseteq \mathbf{B}$ con $\bigsqcap\mathbf{X} = B$, ci sono due elementi $B', B'' \in \mathbf{X}$ tali che $B' \sqcap B'' = B$. Tale sistema di tipi atomici è detto *compatto binario*.

Consideriamo il seguente insieme di designatori di tipi atomici, che useremo in tutti gli esempi:

$$\mathbf{B} = \{\texttt{integer}, \texttt{string}, \texttt{bool}, \texttt{real}, i_1\text{–}j_1, i_2\text{–}j_2, \ldots, d_1, d_2, \ldots\},$$

dove gli $i_k$–$j_k$ indicano tutti i possibili intervalli di interi e i $d_k$ indicano tutti gli elementi di $\texttt{integer} \cup \texttt{string} \cup \texttt{bool}$ ($i_k$ può essere $-\infty$ per denotare il minimo elemento di `integer` e $j_k$ può essere $+\infty$ per denotare il massimo elemento di `integer`). Assumendo l'interpretazione standard dei designatori di tipi atomici, $\mathbf{B}$ è ovviamente un sistema di tipi atomici compatto binario.

## 2.2 Oggetti complessi, tipi e classi

Sia $\mathbf{A}$ un insieme numerabile di *attributi* (denotati da $a_1, a_2, \ldots$) e $\mathbf{N}$ un insieme numerabile di *nomi di tipi* (denotati da $N, N', \ldots$) tali che $\mathbf{A}$, $\mathbf{B}$, e

[1] Questo tipo deve appartenere a $\mathbf{B}$ perchè la congiunzione di differenti tipi mono–valore è vuota.

$\mathbf{N}$ siano a due a due disgiunti. $\mathbf{N}$ è partizionato in tre insiemi $\mathbf{C}$, $\mathbf{V}$ e $\mathbf{T}$, dove $\mathbf{C}$ consiste di nomi per *tipi-classe basi o primitive* $(C, C' \ldots)$, $\mathbf{V}$ consiste di nomi per *tipi-classe virtuali* $(V, V' \ldots)$, e $\mathbf{T}$ consiste di nomi per *tipi-valori* $(t, t', \ldots)$.

**Definizione 1 (Tipi)** *Dati gli insiemi* $\mathbf{A}$, $\mathbf{B}$ *e* $\mathbf{N}$, *il* sistema di tipi $\mathbf{S}(\mathbf{A}, \mathbf{B}, \mathbf{N})$ *denota l'insieme di tutte le* descrizioni dei tipi *(*$S$, $S'$, *...), detti anche brevemente* tipi, *su* $\mathbf{A}, \mathbf{B}, \mathbf{N}$, *che sono costruiti rispettando la seguente regola sintattica astratta (assumendo* $a_i \neq a_j$ *per* $i \neq j$*):*

| | | | |
|---|---|---|---|
| $S$ | $\rightarrow$ | $\top$ | |
| | | $B$ | *tipo atomico* |
| | | $N$ | *nome di tipo* |
| | | $\{S\}$ | *tipo insieme* |
| | | $\langle S \rangle$ | *tipo sequenza* |
| | | $[a_1 \colon S_1, \ldots, a_k \colon S_k]$ | *tipo tupla* |
| | | $S \sqcap S'$ | *intersezione* |
| | | $\triangle S$ | *tipo oggetto* |

Sia $\mathcal{O}$ un insieme numerabile di *identificatori di oggetti*, (detti anche brevemente *oid*, e denotati da $o, o', \ldots$) disgiunto da $\mathbf{D}$.

**Definizione 2 (Valori)** *Dati gli insiemi* $\mathcal{O}$ *e* $\mathbf{D}$, *si definisce l'insieme* $\mathcal{V}(\mathcal{O})$ *dei* valori su $\mathcal{O}$ *(denotati da* $v, v'$*) come segue (assumendo* $p \geq 0$ *e* $a_i \neq a_j$ *per* $i \neq j$*):*

| | | | |
|---|---|---|---|
| $v$ | $\rightarrow$ | $d$ | *valore atomico* |
| | | $o$ | *identificatore di oggetto* |
| | | $\{v_1, \ldots, v_p\}$ | *valore insieme* |
| | | $\langle v_1, \ldots, v_p \rangle$ | *valore sequenza* |
| | | $[a_1 \colon v_1, \ldots, a_p \colon v_p]$ | *valore tupla* |

**Definizione 3 (Dominio)** *Dato un insieme di identificatori di oggetti* $\mathcal{O}$, *un* dominio $\delta$ *su* $\mathcal{O}$ *è una funzione totale da* $\mathcal{O}$ *a* $\mathcal{V}(\mathcal{O})$.

Un dominio $\delta$ associa agli identificatori di oggetti un valore. In genere si dice che il valore $\delta(o)$ è lo *stato* dell'oggetto identificato dall'oid $o$. Un dominio verrà detto *finito* se l'insieme $\mathcal{O}$ è finito.



## 2.3 Schema di base di dati

**Definizione 4 (Schema)** *Dato un sistema di tipi* $\mathbf{S}(\mathbf{A}, \mathbf{B}, \mathbf{N})$, *uno* schema $\sigma$ *su* $\mathbf{S}(\mathbf{A}, \mathbf{B}, \mathbf{N})$ *è una funzione totale da* $\mathbf{N}$ *a* $\mathbf{S}(\mathbf{A}, \mathbf{B}, \mathbf{N})$.

Uno schema $\sigma$ associa ai nomi di tipi–classe e di tipi–valore la loro descrizione: introduciamo la notazione $\sigma_{\mathtt{P}}$, $\sigma_{\mathtt{V}}$, $\sigma_{\mathtt{T}}$ per rappresentare rispettivamente lo schema di una classe primitiva, di una classe virtuale e di un tipo-valore.

La possibilità di utilizzare un nome di tipo nella descrizione di un altro nome può far sorgere nello schema *descrizioni circolari*, cioè descrizioni di nome che fanno riferimento, direttamente o indirettamente tramite altri nomi, al nome stesso. Formalmente i cicli sono riconosciuti attraverso la nozione di *dipendenza*:

**Definizione 5 (Dipendenza)** *Dati* $N$ *e* $N' \in \mathbf{N}$, *diciamo che* $N$ *dipende direttamente da* $N'$ *sse* $N'$ *è contenuto nella descrizione di* $N, \sigma(N)$.

La prima condizione che poniamo sulle proprietà degli schemi considerati è quella di schema aciclico, cioè che il nome di un tipo definito attraverso altri nomi di tipi deve descrivere valori con annidamento finito.

**Definizione 6 (Schema aciclico)** *Uno schema* $\sigma$ *su* $\mathbf{S}(\mathbf{A}, \mathbf{B}, \mathbf{N})$ *si dice aciclico sse la chiusura transitiva della funzione "dipende direttamente da" è un ordine parziale stretto.*

Un'importante caratteristica del modello proposto è il modo con il quale viene dichiarata la relazione *isa* tra le classi. L'*ereditarietà*, semplice e multipla, è espressa nella descrizione del nome della classe tramite l'operatore di intersezione. Per generalità, si estende la definizione a tutti i nomi di tipo.

**Definizione 7 (Ereditarietà)** *Dato uno schema* $\sigma$ *su* $\mathbf{S}(\mathbf{A}, \mathbf{B}, \mathbf{N})$, *diremo che* $N$ eredita da $N'$, *denotato con* $N \prec_\sigma N'$, *sse* $\sigma(N) = S_1 \sqcap \cdots \sqcap S_n$, $n > 0$, *e* $N' = S_i$, *per alcuni* $i$, $1 \leq i \leq n$.

La chiusura riflessiva di $\prec_\sigma$ sarà denotata con $\preceq_\sigma$.

La seconda condizione richiesta è quella che la relazione *isa* non contenga cicli, che va sotto il nome di schema *ben–formato sull'ereditarietà*. Uno schema $\sigma$ *ben–formato sull'ereditarietà* sse $\prec_\sigma$ è un ordine parziale stretto.

Nel seguito si assume sempre che lo schema sia ben–formato sull'ereditarietà e che non contenga cicli.

$$
\begin{array}{rcl}
\mathbf{T} & = & \{\} \\
\mathbf{C} & = & \{\texttt{manager}, \texttt{tmanager}, \texttt{material}, \\
 & & \texttt{smaterial}, \texttt{storage}, \texttt{sstorage}\} \\
\mathbf{V} & = & \{\}
\end{array}
$$

$$
\begin{array}{rcl}
\sigma_{\mathrm{P}}(\texttt{material}) & = & \triangle\Big[\texttt{name: string}, \texttt{risk : integer}, \texttt{feature : \{string\}}\Big] \\
\sigma_{\mathrm{P}}(\texttt{smaterial}) & = & \texttt{material} \\
\sigma_{\mathrm{P}}(\texttt{storage}) & = & \triangle\Big[\texttt{managed-by: manager}, \texttt{category: string} \\
 & & \texttt{stock : \{[item : material}, \texttt{qty} : 10 \div 300\Big] \\
\sigma_{\mathrm{P}}(\texttt{sstorage}) & = & \texttt{storage} \\
\sigma_{\mathrm{P}}(\texttt{manager}) & = & \triangle\Big[\texttt{name: string}, \texttt{salary} : 40000 \div \infty, \texttt{level} : 1 \div 15\Big] \\
\sigma_{\mathrm{P}}(\texttt{tmanager}) & = & \texttt{manager} \sqcap \triangle\Big[\texttt{level:} 8 \div 12\Big]
\end{array}
$$

Tabella 2.1: Schema del domino Magazzino

### 2.3.1 Schema del dominio Magazzino

L'esempio descrive la base di dati riguardante la gestione del magazzino di una società, presentata in sintassi OODB-like in sezione 1.4.

Uno schema corrispondente a tale descrizione è mostrato in tabella 2.1, dove si è assunto che tutte le classi siano classi primitive (per il momento non introduciamo i vincoli di integrità). Si può facilmente verificare che questo schema è aciclico e ben–formato sull'ereditarietà.

La relativa gerarchia delle classi è quella che abbiamo già mostrato in figura 1.1.

## 2.4 Istanza legale di uno Schema

In questa sezione viene data una formale semantica al sistema dei tipi. L'approccio seguito è simile alla formale semantica *set–theoretic* per i tipi introdotta da Cardelli [Car84] e adottata nel modello dei dati $O_2$ [LR89a, LR89b]. Successivamente verranno specificati gli *stati* legali della base di dati tramite



la nozione di *istanza legale* di uno schema. A tale scopo, si definisce il concetto di *interpretazione* come funzione che associa ad ogni tipo un insieme di valori.

**Definizione 8 (Interpretazione)** *Dato un sistema di tipi* $\mathbf{S}$ *e un dominio* $\delta$, *l'*interpretazione $\mathcal{I}$ *di* $\mathbf{S}$ *su* $\delta$, *è una funzione da* $\mathbf{S}$ *a* $2^{\mathcal{V}(\mathcal{O})}$ *tale che:*

$$
\begin{array}{rcl}
\mathcal{I}[\top] & = & \mathcal{V}(\mathcal{O}) \\
\mathcal{I}[B] & = & \mathcal{I}_{\mathbf{B}}[B] \\
\mathcal{I}[C] & \subseteq & \mathcal{O} \\
\mathcal{I}[V] & \subseteq & \mathcal{O} \\
\mathcal{I}[t] & \subseteq & \mathcal{V}(\mathcal{O}) - \mathcal{O} \\
\mathcal{I}[\{S\}] & = & \Big\{\{v_1, \ldots, v_p\} \Big| v_i \in \mathcal{I}[S], 0 \leq i \leq p\Big\} \\
\mathcal{I}[\langle S\rangle] & = & \Big\{\langle v_1, \ldots, v_p\rangle \Big| v_i \in \mathcal{I}[S], 0 \leq i \leq p\Big\} \\
\mathcal{I}\Big[[a_1\colon S_1, \ldots, a_p\colon S_p]\Big] & = & \Big\{[a_1\colon v_1, \ldots, a_q\colon v_q] \Big|\ p \leq q, v_i \in \mathcal{I}[S_i], 0 \leq i \leq p, \\
 & & \qquad\qquad v_j \in \mathcal{V}(\mathcal{O}), p+1 \leq j \leq q\Big\} \\
\mathcal{I}[S \sqcap S'] & = & \mathcal{I}[S] \cap \mathcal{I}[S'] \\
\mathcal{I}[\triangle S] & = & \Big\{o \in \mathcal{O} \Big| \delta(o) \in \mathcal{I}[S]\Big\}.
\end{array}
$$

Prima di tutto si noti che per i tipi tupla si adotta una semantica di mondo aperto simile a quella in [Car84]. Ad esempio:

$$
\begin{array}{rcl}
\Big[\texttt{name: } \textit{“Mark”}, \texttt{salary:} 8000, \texttt{level:} 3\Big] & \in & \mathcal{I}\Big[\texttt{[name: string]}\Big] \\
\Big[\texttt{name: [bname: } \textit{“Research”}, \texttt{sname: } \textit{“DB”}\texttt{]}\Big] & \in & \mathcal{I}\Big[\texttt{[name: [bname: string]}\Big]
\end{array}
$$

Tramite la nozione di interpretazione sopra definita non si impone che un valore istanziato in un nome di tipo abbia una descrizione corrispondente a quella del nome di tipo stesso. Per i nomi di tipo, la funzione interpretazione si limita a vincolare i nomi delle classi ad un insieme di oid e i nomi di tipo–valore ad un insieme di valori che non siano oid.

**Definizione 9 (Istanza Legale)** *Dato uno schema* $\sigma$ *su* $\mathbf{S}$ *e un dominio* $\delta$, *un interpretazione* $\mathcal{I}$ *di* $\mathbf{S}$ *su* $\delta$, *è detta* istanza legale *di* $\sigma$ *su* $\delta$ *sse* $\delta$ *è* finito

*e per ogni* $C \in \mathbf{C}, V \in \mathbf{V}, t \in \mathbf{T}$*:*

$$\begin{aligned} \mathcal{I}[C] &\subseteq \mathcal{I}[\sigma(C)] \\ \mathcal{I}[V] &= \mathcal{I}[\sigma(V)] \\ \mathcal{I}[t] &= \mathcal{I}[\sigma(t)]. \end{aligned}$$

Con la nozione di istanza possibile di uno schema $\sigma$ si impone che gli oggetti nell'interpretazione di una classe base $C$ siano un sottoinsieme degli oggetti nell'interpretazione della descrizione della classe $\sigma(C)$. Questo significa che gli oggetti *istanziati* in una classe base sono esplicitamente forniti dall'utente e soddisfano la descrizione della classe stessa. Invece, per i nomi di classi virtuali e per i nomi di tipo–valore l'interpretazione è uguale all'interpretazione della descrizione del nome, cioè per tali nomi $N$ l'interpretazione è calcolata sulla base della loro descrizione $\sigma(N)$.

In altri termini, la descrizione $\sigma(N)$ costituisce un insieme di condizioni *necessarie e sufficienti* per i nomi di tipo $N \in \mathbf{V} \cup \mathbf{T}$, mentre costituisce un insieme di condizioni *solamente necessarie* per i nomi di tipo $N \in \mathbf{C}$.

La nostra assunzione di aciclicità dello schema nasce dalla seguente considerazione: nella maggior parte dei sistemi per la rappresentazione della conoscenza derivati dal modello KL-ONE le descrizioni circolari sono proibite. Questo per un duplice motivo: non è ovvio come definire la semantica di un nome ciclico e, anche fissata una certa semantica, non è facile ottenere i corrispondenti algoritmi di inferenza.

## 2.5 Sussunzione e coerenza

In questa sezione definiamo una relazione di inclusione semantica, detta *relazione di sussunzione*, tra i tipi in uno schema, indicata con il simbolo $\sqsubseteq$.

**Definizione 10 (Sussunzione)** *Dato uno schema* $\sigma$ *su* $\mathbf{S}$ *e due tipi* $S, S' \in \mathbf{S}$*, si definisce la relazione di sussunzione* $\sqsubseteq_\sigma$ *come segue:*

$$S \sqsubseteq_\sigma S' \quad \textit{sse} \quad \mathcal{I}[S] \subseteq \mathcal{I}[S'] \quad \textit{per ogni istanza } \mathcal{I} \textit{ di } \sigma;$$

La relazione $\sqsubseteq_\sigma$ è un preordine (cioè transitivo e riflessivo ma antisimmetrico) che induce una relazione di *equivalenza* $\simeq_\sigma$ sui tipi: $S \simeq_\sigma S'$ sse $S \sqsubseteq_\sigma S'$ e



$S' \sqsubseteq_\sigma S$. La relazione di equivalenza $\simeq_\sigma$ permette di definire i tipi $S$ che hanno, nello schema $\sigma$, un'interpretazione sempre vuota.

**Definizione 11 (Incoerenza)** *Dato uno schema* $\sigma$ *su* $\mathbf{S}$*, un tipo* $S \in \mathbf{S}$ *è detto* incoerente *nello schema* $\sigma$ *sse* $S \simeq_\sigma \bot$.

Uno schema $\sigma$ è detto *coerente* sse per ogni $N \in \mathbf{N}$, $N \not\simeq_\sigma \bot$. Si noti che in uno schema coerente vi possono essere dei tipi incoerenti usati nei tipi set e sequence: infatti i tipi $\{\bot\}$ e $\langle\bot\rangle$ sono coerenti e denotano rispettivamente l'insieme vuoto e la sequenza vuota.

La relazione intuitiva tra ereditarietà e sussunzione è espressa dalla seguente proposizione:

**Proposizione 1** *Dato uno schema* $\sigma$*, siano* $N, N' \in \mathbf{N}$*; se* $N \preceq_\sigma N'$ *allora* $N \sqsubseteq_\sigma N'$.

In generale il contrario non vale. La principale ragione è la semantica data ai nomi $N \in \mathbf{V} \cup \mathbf{T}$. Un'altra ragione è che un nome di tipo può essere incoerente. Per schemi coerenti possiamo dare un viceversa parziale alla precedente proposizione.

**Proposizione 2** *Dato uno schema coerente* $\sigma$*, sia* $N \in \mathbf{C} \cup \mathbf{V}$ *e* $N' \in \mathbf{C}$*; allora* $N \sqsubseteq_\sigma N'$ *sse* $N \preceq_\sigma N'$.

In altri termini, in uno schema coerente le relazioni di sussunzione in cui la *superclasse* è una classe base sono solo quelle stabilite esplicitamente tramite la relazione di ereditarietà.

Mediante la relazione di sussunzione si individuano, per ogni nome $N$, tutte le sue *generalizzazioni* rispetto all'intera tassonomia, dalle quali è possibile selezionarne le *più specifiche*. Formalmente, dato uno schema $\sigma$, per un nome $N \in \mathbf{N}$ si definisce l'insieme $GS(N)$ (*Generalizations Set*):

$$GS(N) = \{N' \in \mathbf{N} \mid N' \neq N \wedge N \sqsubseteq_\sigma N'\}$$

e l'insieme $MSGS(N)$ (*Most Specialized Generalizations Set*)

$$MSGS(N) = \{N' \in GS(N) \mid \nexists N'' \in GS(N) \colon N'' \sqsubseteq_\sigma N'\}$$

Infine, in uno schema si può introdurre la *classe universale*, denotata con $\top_C$ ($top-class$), che rappresenta la classe *più generale* dello schema e che quindi sussume tutte le altre classi, come classe virtuale con descrizione $\sigma_{\mathbf{V}}(\top_C) = \triangle\top$. Infatti è immediato verificare che, in qualsiasi istanza possibile $\mathcal{I}$, si ha sempre $\mathcal{I}[\top_C] = \mathcal{O}$.

# Capitolo 3

# Schemi Canonici ed Algoritmi di Incoerenza e Sussunzione

In uno schema l'ereditarietà è espressa nella descrizione di un nome di tipo tramite l'operatore di intersezione. Tramite la nozione di *schema canonico* si vuole definire uno schema nel quale la descrizione di un nome di tipo è espressa senza congiunzioni e contiene tutte le proprietà del nome, sia quelle definite localmente sia quelle ereditate. Lo schema canonico deve preservare la semantica dello schema originale, nel senso che le relazioni semantiche valide nello schema originale devono essere conservate nello schema canonico.

Il principale obiettivo dello schema canonico è quello di rendere relativamente semplici i problemi del controllo di incoerenza e del calcolo di sussunzione, che invece sono complicati in uno schema generale.

Prima di dare le definizioni formali, facciamo alcune considerazioni intuitive. Lo schema canonico si ottiene sostituendo ricorsivamente, in una descrizione di un tipo, i nomi dei tipi dal quale esso eredita con la rispettiva descrizione (questo è possibile in quanto la relazione di ereditarietà è un ordine parziale stretto). Nel tipo risultante vengono quindi risolte le intersezioni. La sostituzione di un nome di tipo preserva la semantica dello schema originale ad eccezione che si tratti di un nome di classe base: infatti l'estensione di un nome di classe base è un sottoinsieme dell'estensione della rispettiva descrizione. Allora, prima di effettuare le sostituzioni, ad una classe base $C$ nello schema $\sigma$ si fa corrispondere nello schema canonico una classe virtuale espressa come l'intersezione della descrizione di $C$, $\sigma(C)$, e di un nuovo nome di classe base $\overline{C}$, detto *atomo fittizio*, la cui descrizione è $\triangle\top$.





## 3.1 Schema canonico

Si introduce prima di tutto la nozione di tipo in forma canonica.

$\overline{\mathbf{S}}(\mathbf{A},\mathbf{B},\overline{\mathbf{N}})$ denota l'insieme delle *descrizioni canoniche di tipo* $(\overline{S}, \overline{S}', \ldots)$ su $\mathbf{A},\mathbf{B},\overline{\mathbf{N}}$, con $\overline{\mathbf{N}} = \overline{\mathbf{C}} \cup \overline{\mathbf{V}} \cup \mathbf{T}$ e $\overline{\mathbf{V}} = \mathbf{C} \cup \mathbf{V}$, che sono costruite rispettando la seguente regola sintattica astratta: [1]

$$\overline{S} \;\rightarrow\; \overline{S} \mid \{\overline{S}\} \mid \langle\overline{S}\rangle \mid [a_1{:}\,\overline{S}_1, \ldots, a_k{:}\,\overline{S}_k] \mid \textstyle\bigsqcap_i \overline{C}_i \sqcap \triangle\overline{S}.$$

con $k \geq 0$, $i \geq 0$ e $\overline{S} \in \{\top\} \cup \mathbf{B} \cup \overline{\mathbf{V}} \cup \overline{\mathbf{T}}$ e $\overline{C}_i \in \overline{\mathbf{C}}$[2].

**Definizione 12 (Schema Canonico equivalente)** *Uno* schema canonico $\nu$ *è uno schema su* $\overline{\mathbf{S}}(\mathbf{A},\mathbf{B},\overline{\mathbf{N}})$ *tale che per ogni* $\overline{C}_i \in \overline{\mathbf{C}}$, $\nu(\overline{C}_i) = \triangle\top$. *Dato uno schema* $\sigma$ *su* $\mathbf{S}(\mathbf{A},\mathbf{B},\mathbf{N})$ *si dice che uno schema canonico* $\nu$ *su* $\mathbf{S}$ *con* $\mathbf{N} \subseteq \overline{\mathbf{N}}$ *è* equivalente *a* $\sigma$ *sse per ogni istanza possibile* $\mathcal{I}$ *di* $\sigma$ *esiste un'istanza possibile* $\mathcal{I}'$ *di* $\nu$ *e* viceversa *tale che* $\mathcal{I}[N] = \mathcal{I}'[N], \forall N \in \mathbf{N}$.

Da questa definizione segue immediatamente che le relazioni semantiche in due schemi equivalenti sono le stesse:

**Proposizione 3** *Dato uno schema* $\sigma$ *su* $\mathbf{S}(\mathbf{A},\mathbf{B},\mathbf{N})$, *per ogni* $N, N' \in N$, $N \sqsubseteq_\sigma N'$ *sse* $N \sqsubseteq_\nu N'$, *con* $\nu$ *schema canonico su* $\overline{\mathbf{S}}(\mathbf{A},\mathbf{B},\overline{\mathbf{N}})$ equivalente *a* $\sigma$.

Di conseguenza, il calcolo della sussunzione e il controllo di coerenza di uno schema arbitrario possono essere effettuati considerando lo schema canonico equivalente.

Per quanto riguarda la possibilità di trasformare effettivamente uno schema in una forma canonica equivalente, vale quanto segue:

**Proposizione 4** *Ogni schema* $\sigma$ *su* $\mathbf{S}(\mathbf{A},\mathbf{B},\mathbf{N})$ *aciclico può essere effettivamente trasformato in uno schema canonico* $\nu$ *su* $\overline{\mathbf{S}}(\mathbf{A},\mathbf{B},\overline{\mathbf{N}})$ *che è* equivalente *a* $\sigma$.

[1] Si assume che per $i = 0$ la regola $\bigsqcap_i \overline{C}_i \sqcap \triangle\overline{S}$ produca il tipo $\triangle\overline{S}$.

[2] Si assume che nel dominio di un attributo non compaiono intervalli di valori ed enumerazioni ma i tipi che li descrivono.

## 3.2 Generazione dello schema canonico

Un generico tipo di uno schema $\sigma$ può essere trasformato in un tipo equivalente nella quale le intersezioni riguardano *esclusivamente* nomi di tipo, cioè siano esprimibili come $\bigsqcap_k N_k \sqcap S'$; una tale espressione di tipo è detta *ridotta all'intersezione*. Questa trasformazione è naturalmente indispensabile per ottenere uno schema canonico.

**Proposizione 5** *Dato uno schema $\sigma$ su $\mathbf{S}$, per ogni tipo $S \in \mathbf{S}$, esiste un'espressione di tipo di $\mathbf{S}$ ridotta all'intersezione, denotata con $\tilde{\mu}(S)$, equivalente a $S$: $\tilde{\mu}(S) \simeq_\sigma S$.*

La dimostrazione della proposizione è data dal fatto che $\tilde{\mu}(S)$ è ottenibile tramite delle semplici manipolazioni algebriche basate sulle equivalenze tra tipi mostrate in tabella 3.1. (Per ulteriori chiarimenti vedere [Bal92]).

Sulle espressioni di tipo $\tilde{\mu}(S)$ si considera una relazione di *uguaglianza sintattica*, indicata con $\approx$, definita a meno di permutazione sugli attributi dei tipi tupla sui fattori nelle congiunzioni $\bigsqcap_k N_k \sqcap S'$. Ovviamente, se $S \approx S'$ allora $S \simeq_\sigma S'$ mentre il viceversa non è valido.

In seguito sono descritte le varie fasi di trasformazione di uno schema $\sigma$ in uno schema canonico $\nu$. Prima di tutto, ad uno schema $\sigma$ viene associato uno schema $\tau$ detto *schema equazionale* di $\sigma$: ogni classe base $C$ di $\sigma$ viene sostituita in $\tau$ con una classe virtuale espressa come la congiunzione di una *classe atomica fittizia* $\overline{C}$ e della sua descrizione $\sigma(C)$. Una classe atomica fittizia è una classe base di $\tau$ la cui descrizione è $\triangle\top$.

Formalmente, dato uno schema $\sigma$ su $\mathbf{S}(\mathbf{A}, \mathbf{B}, \mathbf{N})$, sia $\overline{\mathbf{C}}$ una copia disgiunta di $\mathbf{C}$, $\pi$ una biiezione, $\pi\colon \mathbf{C} \to \overline{\mathbf{C}}$, e $\overline{\mathbf{N}} = \overline{\mathbf{C}} \cup \overline{\mathbf{V}} \cup \mathbf{T}$ con $\overline{\mathbf{V}} = \mathbf{C} \cup \mathbf{V}$. Lo *schema equazionale* di $\sigma$ è uno schema $\tau$ su $\mathbf{S}(\mathbf{A}, \mathbf{B}, \overline{\mathbf{N}})$ definito come segue:

$$\tau(N) \;=\; \begin{cases} \triangle\top & \text{se } N \in \overline{\mathbf{C}} \\ \pi(N) \sqcap \sigma(N) & \text{se } N \in \mathbf{C} \\ \sigma(N) & \text{altrimenti} \end{cases}$$

È immediato verificare che per ogni istanza possibile $\mathcal{I}$ di $\sigma$ esiste un'istanza possibile $\mathcal{I}'$ di $\tau$, e viceversa, tale che

$$\mathcal{I}[N] = \mathcal{I}'[N], \qquad \text{per ogni } N \in \mathbf{N}$$

La funzione che segue fornisce, per un nome di tipo $N \in \overline{\mathbf{V}} \cup \overline{\mathbf{T}}$, la descrizione di tipo che si ottiene sostituendo ogni nome di tipo $N'$ da cui $N$ eredita con



$$\begin{array}{rcl} S \sqcap S & \simeq_\sigma & S \\ S \sqcap S' & \simeq_\sigma & S' \sqcap S \\ S \sqcap (S' \sqcap S'') & \simeq_\sigma & (S \sqcap S') \sqcap S'' \end{array}$$

$$[a_1\colon S_1, \ldots, a_p\colon S_p] \;\sqcap\; [a'_1\colon S'_1, \ldots, a'_q\colon S'_q] \;\simeq_\sigma\; \begin{cases} [a_1\colon S_1 \sqcap S'_1, a_2\colon S_2, \ldots, a_p\colon S_p,] \sqcap \\ [a'_2\colon S'_2, \ldots, a'_q\colon S'_q,] \\ \text{se } a'_1 = a_1 \\ \\ [a'_1\colon S'_1, a_1\colon S_1, \ldots, a_p\colon S_p,] \sqcap \\ [a'_2\colon S'_2, \ldots, a'_q\colon S'_q,] \\ \text{se } a'_1 \neq a_i \text{ per } 1 \leq i \leq p \end{cases}$$

$$\begin{array}{rcl} \{S\} \sqcap \{S'\} & \simeq_\sigma & \{S \sqcap S'\} \\ \langle S \rangle \sqcap \langle S' \rangle & \simeq_\sigma & \langle S \sqcap S' \rangle \\ \triangle S \sqcap \triangle S' & \simeq_\sigma & \triangle(S \sqcap S') \end{array}$$

$$N \sqcap N' \;\simeq_\sigma\; N \quad \text{se } N \prec_\sigma N'$$

$$S \sqcap S' \;\simeq_\sigma\; \bot \quad \text{se } (S, S') \in \mathbf{I}$$

$$B \sqcap B' \;\simeq_\sigma\; B'' \quad \text{dove } B'' = B \sqcap B'$$

Tabella 3.1: Equivalenze tra tipi

la rispettiva descrizione:

$$\iota(S) \;=\; \begin{cases} \tau(S) & \text{se } S \in \overline{\mathbf{V}} \cup \overline{\mathbf{T}} \\ \iota(S') \sqcap \iota(S'') & \text{se } S = S' \sqcap S'' \\ S & \text{altrimenti} \end{cases}$$

Si definisce $\tilde{\iota} = \iota^n$, dove $n$ è il più piccolo numero naturale tale che $\iota^n = \iota^{n+1}$; tale numero esiste sempre per schemi ben–formati sull'ereditarietà. Il tipo che si ottiene con $\tilde{\iota}$ è equivalente (per tutte e tre le semantiche) in $\tau$ a quello originale: $\tilde{\iota}(S) \simeq_\tau S$.

La composizione della funzione $\tilde{\mu}$ con la funzione $\tilde{\iota}$ verrà indicata con $\kappa$: per ogni tipo $S$

$$\kappa(S) = \tilde{\mu}(\tilde{\iota}(S))$$

Si noti che $\kappa(S)$ è un'espressione di tipo ridotta all'intersezione particolare: le eventuali intersezioni a livello più esterno riguardano esclusivamente nomi di classi atomiche fittizie. Per quanto detto sulle trasformazioni $\tilde{\tau}$ e $\tilde{\mu}$ segue immediatamente che $\kappa(S) \simeq_\tau S$.

A questo punto può essere data la definizione operazionale di schema canonico $\nu$. Informalmente, partendo da $\tau$, si costruiscono una sequenza di schemi $f$ su $\overline{\mathbf{N}}^0, \overline{\mathbf{N}}^1, \overline{\mathbf{N}}^2 \ldots$; si passa da $\overline{\mathbf{N}}^i$ a $\overline{\mathbf{N}}^{i+1}$ sostituendo ogni tipo $S$ di $f$ che non è in $\{\top\} \cup \mathbf{B} \cup \overline{\mathbf{N}}^i$, con un nome $N$ dove $N$ è un nome di $\overline{\mathbf{N}}^i$ se $f(N)$ è uguale a $S$ oppure $N$ è un nuovo nome introdotto in $\overline{\mathbf{N}}^{i+1}$, con $f(N) = S$. Il processo termina quando tutti i nomi dello schema hanno una descrizione in forma canonica. L'algoritmo è riportato in tabella 3.2.

L'introduzione di nomi nuovi, e quindi l'algoritmo, si conclude dopo un numero finito di passi, in quanto nella determinazione di $\tilde{\mu}$, compresa in $\kappa(S)$, si utilizza la seguente equivalenza tra nomi di tipo: $N \sqcap N' \simeq_\sigma N$ se $N \prec_\sigma N'$.

Per lo schema di riferimento di tabella 2.1 l'algoritmo produce lo schema canonico mostrato in tabella 3.3.

## 3.3 Algoritmi di incoerenza e sussunzione

In questa sezione presentiamo gli algoritmi per il controllo di coerenza e il calcolo della sussunzione in uno schema canonico. Per ciascun algoritmo sarà dimostrata la sua correttezza e completezza e ne verrà discussa la sua complessità.

Siccome per ogni tipo $S$ è stata fissata una precisa semantica estensionale, la relazione di sussunzione tra due tipi $S$ e $S'$, definita per inclusione semantica, è determinabile tramite un algoritmo che effettua il confronto strutturale delle espressioni $S$ e $S'$. Il confronto strutturale è definito nello schema canonico, cioè in uno schema dove sono state risolte le intersezioni (le intersezioni riguardano solo le classi atomiche fittizie). Di conseguenza, i confronti per il calcolo della sussunzione sono simili a quelli che definiscono sintatticamente la relazione strutturale di *sottotipo* (*subtyping* o *refinement*) generalmente adottata negli OODB.

Nello stesso modo la coerenza di un tipo $S$ è determinabile tramite un'analisi strutturale. Da un punto di vista teorico, l'incoerenza è definita sulla base della relazione di sussunzione e quindi la rilevazione dell'incoerenza è riconducibile al calcolo della sussunzione. In pratica si osserva che la rilevazione di incoerenza è più efficiente se implementata come algoritmo separato.



**Algoritmo (Schema Canonico)**

- **Inizializzazione**: $f = \tau$
- **Iterazione**: dato $f$ su $\overline{\mathbf{N}}^i$ calcola $f$ su $\overline{\mathbf{N}}^{i+1}$, con $\overline{\mathbf{N}}^i \subseteq \overline{\mathbf{N}}^{i+1}$, nel seguente modo:

  per ogni $N \in \overline{\mathbf{N}}^i$:

  $$f(N) = \mathbf{GC}(\kappa(N))$$

  dove

  $$\begin{aligned}
  \mathbf{GC}(S) &= S \text{ se } S \in \{\top\} \cup \mathbf{B} \cup \overline{\mathbf{V}} \cup \mathbf{T} \\
  \mathbf{GC}(\{S\}) &= \{\mathbf{NT}(S)\} \\
  \mathbf{GC}(\langle S \rangle) &= \langle \mathbf{NT}(S) \rangle \\
  \mathbf{GC}([a_1\!:\!S_1, \ldots, a_p\!:\!S_p]) &= [a_1\!:\!\mathbf{NT}(S_1), \ldots, a_p\!:\!\mathbf{NT}(S_p)] \\
  \mathbf{GC}(\bigsqcap_i \overline{C}_i \sqcap \triangle S) &= \bigsqcap_i \overline{C}_i \sqcap \triangle \mathbf{NT}(S)
  \end{aligned}$$

  e

  $$\mathbf{NT}(S) = \begin{cases} S & \text{se } S \in \{\top\} \cup \mathbf{B} \cup \overline{\mathbf{V}} \cup \mathbf{T} \\ N & \text{se } \exists N \in \overline{\mathbf{N}}^{i+1}\!: \kappa(N) \approx \kappa(S) \\ N' & \text{nuovo in } \overline{\mathbf{N}}^{i+1} \text{ con } f(N') = \kappa(S) \\ & \text{aggiungi } N' \text{ a } \overline{\mathbf{N}}^{i+1} \end{cases}$$

- **Stop**: $\overline{\mathbf{N}}^{i+1} = \overline{\mathbf{N}}^i$; definisci $\tilde{\nu} = f$.

Tabella 3.2: Generazione dello schema canonico

$$
\begin{aligned}
\overline{\mathbf{T}} &= \{b_1 \dots b_4\} \cup \{t_1 \dots t_7\} \\
\overline{\mathbf{C}} &= \{\overline{\texttt{material}}, \overline{\texttt{smaterial}}, \overline{\texttt{storage}}, \overline{\texttt{sstorage}}, \\
&\quad \overline{\texttt{manager}}, \overline{\texttt{tmanager}}\} \\
\overline{\mathbf{V}} &= \{\texttt{material}, \texttt{smaterial}, \texttt{storage}, \texttt{sstorage}, \\
&\quad \texttt{manager}, \texttt{tmanager}\} \\
\nu(\texttt{material}) &= \overline{\texttt{material}} \sqcap \triangle t_1 \\
\nu(\texttt{smaterial}) &= \overline{\texttt{smaterial}} \sqcap \triangle t_1 \\
\nu(\texttt{storage}) &= \overline{\texttt{storage}} \sqcap \triangle t_3 \\
\nu(\texttt{sstorage}) &= \overline{\texttt{storage}} \sqcap \overline{\texttt{sstorage}} \sqcap \triangle t_3 \\
\nu(\texttt{manager}) &= \overline{\texttt{manager}} \sqcap \triangle t_6 \\
\nu(\texttt{tmanager}) &= \overline{\texttt{manager}} \sqcap \overline{\texttt{tmanager}} \sqcap \triangle t_6 \\
\nu(t_1) &= [\texttt{feature} : t_2, \texttt{name} : \texttt{string}, \texttt{risk} : \texttt{integer}] \\
\nu(t_2) &= \{\texttt{string}\} \\
\nu(t_3) &= [\texttt{category} : \texttt{string}, \texttt{stock}: t_4, \texttt{managed-by} : \texttt{manager}] \\
\nu(t_4) &= \{t_5\} \\
\nu(t_5) &= [\texttt{item} : \texttt{material}, \texttt{qty} : b_1] \\
\nu(t_6) &= [\texttt{level} : b_2, \texttt{name} : \texttt{string}, \texttt{salary} : b_3] \\
\nu(t_7) &= [\texttt{level} : b_4, \texttt{name} : \texttt{string}, \texttt{salary} : b_3] \\
\nu(b_1) &= 10 \div 300 \\
\nu(b_2) &= 1 \div 15 \\
\nu(b_3) &= 40000 \div \infty \\
\nu(b_4) &= 8 \div 12
\end{aligned}
$$

Tabella 3.3: Schema canonico di un Magazzino



Nel seguito, per semplicità, un tipo in forma canonica verrà indicato con $S$.

### 3.3.1 Controllo di incoerenza

L' algoritmo di incoerenza è basato sul calcolo iterativo di un insieme $\Phi$ di tipi. Fissato l'insieme dei nomi incoerenti, i restanti tipi canonici incoerenti sono ricavabili in maniera immediata. A tale scopo, vengono imposte delle condizioni sui tipi presenti in $\Phi$ ad un generico passo dell'algoritmo.

Dato uno schema canonico $\nu$ su $\overline{\mathbf{S}}(\mathbf{A}, \mathbf{B}, \overline{\mathbf{N}})$, sia $\Phi$ un sottoinsieme di $\overline{\mathbf{S}}(\mathbf{A}, \mathbf{B}, \overline{\mathbf{N}})$ tale che[3]:

$$
\begin{aligned}
&\bot \in \Phi \\
[a_1 \colon S_1, \dots, a_n \colon S_n] \in \Phi \quad &\text{sse} \quad \exists k, 1 \leq k \leq n \colon S_k \in \Phi \\
\bigsqcap \overline{C}_i \sqcap \triangle S \in \Phi \quad &\text{sse} \quad S \in \Phi
\end{aligned}
$$

Fissato una relazione di incoerenza per i nomi di tipo, le condizioni imposte su $\Phi$ permettono di estenderla agli altri tipi in forma canonica. Ora si deve appunto completare il calcolo dei tipi incoerenti considerando i nomi di tipo $N \in \overline{\mathbf{N}}$. Si definisce l'algoritmo di incoerenza nel quale, dato un insieme di partenza $\Phi^0$, si calcolano iterativamente l'insieme dei nomi di tipo incoerenti sulla base della loro descrizione.

**Algoritmo** (**Incoerenza**)

- **Inizializzazione**: $N \notin \Phi^0$, per ogni $N \in \overline{\mathbf{N}}$;
- **Iterazione**: $N \in \Phi^{i+1}$ sse $\nu(N) \in \Phi^i$;
- **Stop**: $\Phi^{i+1} = \Phi^i$; definisci $\widetilde{\Phi}^g_\nu = \Phi^i$.

Tabella 3.4: Algoritmo di Incoerenza

La funzione $\Phi^i$ incrementa in modo monotono all'aumentare di $i$; di conseguenza, poichè $\overline{\mathbf{N}}$ è finito, esiste un numero naturale $n$ tale che $\Phi^n = \Phi^{n+1}$,

[3] Si ricordi che $\{\bot\}$ e $\langle\bot\rangle$ sono tipi coerenti che indicano rispettivamente l'insieme vuoto e la sequenza vuota.

cioè l'algoritmo termina. Inoltre il massimo $n$ è $|\overline{\mathbf{N}}|$, cioè l'algoritmo è polinomiale in $\nu$. L'algoritmo calcola l'insieme di nomi di tipi incoerenti; le condizioni imposte su $\Phi$ permettono di ricavare l'insieme totale dei tipi incoerenti.

L'insieme $\widetilde{\Phi}_\nu$ determinato dagli algoritmi ha le seguenti proprietà:

1. $N \simeq_\nu \bot$ **se** $N \in \widetilde{\Phi}_\nu$ (**correttezza** dell'algoritmo)
2. $N \simeq_\nu \bot$ **solo se** $N \in \widetilde{\Phi}_\nu$ (**completezza** dell'algoritmo)

Formalmente, ciò è espresso dal seguente teorema dimostrato in [Ben94].

**Teorema 1** *Sia $\nu$ uno schema canonico su $\overline{\mathbf{S}}(\mathbf{A},\mathbf{B},\overline{\mathbf{N}})$. Allora*

$$N \simeq_\nu \bot \quad sse \quad N \in \widetilde{\Phi}_\nu$$

### 3.3.2 Calcolo della sussunzione

L'algoritmo di sussunzione è basato sul calcolo iterativo di una relazione $\leq$ tra i tipi in forma canonica. Fissata una relazione di sussunzione tra i nomi di tipo è facilmente estendibile a tutti i restanti tipi canonici. Pertanto, si definisce prima di tutto la relazione $\leq$ per una coppia di tipi diversa da una coppia di nomi.

Dato uno schema canonico $\nu$ su $\overline{\mathbf{S}}(\mathbf{A},\mathbf{B},\overline{\mathbf{N}})$, sia $\leq$ una relazione tra $S, S' \in \overline{\mathbf{S}}(\mathbf{A},\mathbf{B},\overline{\mathbf{N}})$ tale che:

- per ogni tipo canonico $S \in \overline{\mathbf{S}}(\mathbf{A},\mathbf{B},\overline{\mathbf{N}})$

$$S \leq \top$$

- per ogni coppia di tipi atomici $B, B' \in \mathbf{B}$

$$B \leq B' \text{ sse } B \sqcap B' = B$$

- per ogni $S \in \{\top\} \cup \mathbf{B}$, $N \in \overline{\mathbf{N}}$:

$$\begin{array}{rl} S \leq N & \text{sse } S \leq \nu(N) \\ N \leq S & \text{sse } \nu(N) \leq S \end{array}$$



- per ogni $S, S' \in \{\top\} \cup \mathbf{B} \cup \overline{\mathbf{N}}$:

$$\begin{array}{rl} \{S\} \leq \{S'\} & \text{sse } S \leq S' \\ \langle S \rangle \leq \langle S' \rangle & \text{sse } S \leq S' \\ [\ldots, a_i{:}\, S_i, \ldots] \leq [\ldots, a'_j{:}\, S'_j, \ldots] & \text{sse } \forall j \exists i{:}\ (a_i = a'_j \wedge S_i \leq S'_j) \\ \bigsqcap_k \overline{C}_k \sqcap \triangle S \leq \bigsqcap_j \overline{C}'_j \sqcap \triangle S' & \text{sse } S \leq S' \wedge \forall j \exists k{:} \overline{C}_k = \overline{C}'_j. \end{array}$$

Intuitivamente, fissato un ordinamento tra i nomi dei tipi, le condizioni imposte su $\leq$ permettono di estenderlo a tutti i tipi in forma canonica. Ora si deve appunto completare la definizione della relazione $\leq$ considerando i confronti tra due nomi di tipo $N \in \overline{\mathbf{N}}$.

Si definisce l'algoritmo di sussunzione che, fissata una relazione di partenza $\leq^0$, calcola iterativamente la relazione $\leq$ tra i nomi di tipo basandosi sulle loro descrizione e sull'insieme dei nomi di tipo incoerenti.

**Algoritmo (Sussunzione)**

- **Inizializzazione**: $N \leq^0 N'$ sse $(N,N') \in \overline{\mathbf{V}}^2 \cup \overline{\mathbf{T}}^2 \cup \left\{(\overline{C},\overline{C}) \,\middle|\, \overline{C} \in \overline{\mathbf{C}}\right\}$
- **Iterazione**: $N \leq^{i+1} N'$ sse $\nu(N) \leq^i \nu(N') \ \vee \ (N \in \tilde{\Phi}^g_\nu)$
- **Stop**: $\leq^{i+1} = \leq^i$; definisci $\lesssim^g_\nu = \leq^i$.

Tabella 3.5: Algoritmo di Sussunzione

Nell'algoritmo la relazione $\leq^i$ incrementa in modo monotono all'aumentare di $i$. Di conseguenza, come nel caso dell'algoritmo di incoerenza, poichè $\overline{\mathbf{N}}$ è finito, esiste un numero naturale $n$ tale che $\leq^n = \leq^{n+1}$. Il massimo $n$ è $|\overline{\mathbf{N}}|^2$, cioè anche l'algoritmo di sussunzione è polinomiale in $\nu$.

La relazione $\lesssim_\nu$ determinata dall'algoritmo ha le seguenti proprietà:

1. $S \sqsubseteq_\sigma S'$ **se** $S \lesssim_\nu S'$ (**correttezza** dell'algoritmo)
2. $S \sqsubseteq_\sigma S'$ **solo se** $S \lesssim_\nu S'$ (**completezza** dell'algoritmo)

Formalmente, ciò è espresso dal seguente teorema dimostrato in [Ben94].

**Teorema 2** *Sia $\nu$ uno schema canonico su $\overline{\mathbf{S}}(\mathbf{A},\mathbf{B},\overline{\mathbf{N}})$. Allora per ogni $S,S' \in \overline{\mathbf{S}}(\mathbf{A},\mathbf{B},\overline{\mathbf{N}})$:*

$$
\begin{array}{lll}
S \sqsubseteq^{g}_{\nu} S' & sse & S \lesssim^{g}_{\nu} S' \\
S \sqsubseteq^{d}_{\nu} S' & sse & S \lesssim^{d}_{\nu} S' \\
S \sqsubseteq^{l}_{\nu} S' & sse & S \lesssim^{l}_{\nu} S'
\end{array}
$$

# Capitolo 4

# Ottimizzazione Semantica delle Interrogazioni

L'obiettivo dell'ottimizzazione semantica delle interrogazioni è quello di trasformare un'interrogazione in una *equivalente* con un minor costo di esecuzione. L'ottimizzazione è detta semantica in quanto le trasformazioni avvengono utilizzando la conoscenza semantica relativa alla base di dati.

La nozione di ottimizzazione semantica di interrogazioni è stata introdotta per le basi di dati relazionali da King [Kin81a, Kin81b] e da Hammer e Zdonik [HZ80]. L'idea di base di queste proposte è che i vincoli di integrità, espressi per forzare la consistenza di una base di dati, possono essere utilizzati anche per ottimizzare le interrogazioni fatte dall'utente.

Nel presente capitolo viene presentato il metodo di ottimizzazione semantica basato sulla relazione di sussunzione in uno schema, ampiamente descritto in [Ben94]. Il metodo si propone come strumento di ottimizzazione semantica delle interrogazione in ambito OODB e costituisce un'integrazione tra le tecniche sviluppate nel settore delle basi di dati, in particolare nei sistemi relazionali, e quelle sviluppate per alcuni sistemi di rappresentazione della conoscenza basati sulle logiche descrittive.

L'approccio proposto è caratterizzato dai seguenti punti:

- *Base di dati e i vincoli di integrità considerati*:

  L'ottimizzazione è riferita al modello ad oggetti complessi introdotto nei precedenti capitoli al quale viene aggiunta una classe ristretta di regole di integrità, quelle esprimibili come implicazioni logiche tra i





  tipi del sistema. Con questa restrizione, le regole di integrità possono essere trattate efficientemente tramite gli algoritmi di sussunzione.

- *Correttezza ed equivalenza semantica*:

  La precisa semantica sia del modello dei dati che della relazione di sussunzione permettono di garantire formalmente la correttezza delle trasformazioni semantiche.

- *Processo di generazione di interrogazioni equivalenti*:

  La fase principale del processo di generazione, chiamata *espansione semantica*, permette di incorporare ogni possibile restrizione che non è presente nell'interrogazione originale ma che è *logicamente implicata* dall'interrogazione e dallo schema con le regole di integrità. In questo modo viene determinata l'*interrogazione più specializzata* tra tutte quelle semanticamente equivalenti.

- *integrazione con ottimizzazioni convenzionali*:

  La generalità e l'indipendenza da ogni specifico modello di costo e da ogni dettaglio di memorizzazione fisica rendono il metodo proposto adatto per l'integrazione con un ottimizzatore di interrogazione tradizionale.

Nella presente tesi non viene discusso il problema delle *contraddizioni* e delle *ridondanze* che possono sorgere in un insieme di regole di integrità. Questo problema è stato trattato in [BBLS94a] dove, attraverso l'estensione del modello proposto in questa tesi, viene controllata la consistenza dei tipi (compresi i vincoli di integrità) e generata una classificazione tassonomica delle interrogazioni, che porta all'individuazione dei tipi più specifici che generalizzano tali interrogazioni.

L'organizzazione del presente capitolo è la seguente.

Nella prima parte, allo scopo di aumentare i vincoli di integrità esprimibili in forma dichiarativa in un modello di dati orientato ad oggetti, viene introdotta la nozione di *regola di integrità*. Le regole di integrità esprimono dei vincoli nella forma di *if then rule* universalmente quantificate su tutti i valori del dominio i cui antecedenti e conseguenti sono tipi del formalismo.

Viene quindi definita la nozione di espansione semantica di un tipo che costituisce lo strumento fondamentale sia per analizzare che per calcolare l'ottimizzazione semantica delle interrogazioni.

Nella seconda parte vengono analizzate le interrogazioni esprimendole come tipi del sistema, e precisamente come classi virtuali. Per queste interrogazioni, il metodo proposto per l'ottimizzazione semantica viene prima illustrato con alcuni esempi e successivamente formalizzato.

Nell'ultima parte viene proposto un algoritmo efficiente per l'ottimizzazione semantica basato sul calcolo dello schema canonico e sulla sussunzione: questo algoritmo è alla base del programma di ottimizzazione GES presentato nei successivi capitoli della tesi.

## 4.1 Schemi e regole di integrità

I vincoli di integrità sono asserzioni che devono essere vere per ogni oggetto di una base di dati e hanno lo scopo di imporre la consistenza di una base di dati.

Nel modello presentato, come nella maggior parte dei modelli ad oggetti, alcuni vincoli di integrità sono già esprimibili nello schema e vengono imposti agli oggetti tramite la nozione di istanza legale;li possiamo così suddividere:

- *vincoli di dominio*: per ogni attributo è specificato l'insieme dei valori ammissibili;
- *vincoli di integrità referenziale*: un oggetto che è referenziato da un altro oggetto deve esistere;
- *vincoli di ereditarietà*: se un oggetto appartiene ad una classe $C$ allora deve appartenere anche alle superclassi di $C$.

Inoltre nel nostro modello possiamo esprimere vincoli di specializzazione [CW91] su una sequenza di attributi senza essere costretti a definire ulteriori sottoclassi.
Ad esempio, con riferimento allo schema di tabella 2.1, la sottoclasse `DStorage` di `storage` in cui il `name` è ristretto a "*DS-Company*" e nella quale gli oggetti che ricoprono il ruolo di `managed-by` hanno l'attributo `level` uguale a 3, può essere descritta nel seguente modo:

$$\sigma_{\mathrm{V}}(\texttt{DStorage}) \;=\; \texttt{storage} \sqcap \triangle\Big[\texttt{name}\colon \text{"}\textit{DS-Company}\text{"},\ \texttt{typing}\colon \triangle[\texttt{level}\colon 3]\Big]$$

senza dover introdurre la sottoclasse di `manager` che ha `level` uguale a 3.



La facilità di navigare attraverso la gerarchia di aggregazione mediante sequenze di attributi è un aspetto fondamentale anche nei linguaggi di interrogazione orientati ad oggetti. Nel seguito verrà pertanto introdotta una *notazione abbreviata* per le espressioni di tipo basata essenzialmente sulle sequenze di attributi.

Definiamo *cammino* $p$ su $\mathbf{S}(\mathbf{A},\mathbf{B},\mathbf{N})$ una sequenza di elementi $p = e_1 . e_2 . \ldots . e_n$, con $e_i \in \mathbf{A} \cup \{\{\}, \Diamond, \triangle\}$.
Si denota con $\epsilon$ il *cammino vuoto*.

La notazione abbreviata utilizzata, detta *tipo–cammino*, $(p\colon S)$ è definita come segue:

$$\begin{array}{lll}
(\epsilon\colon S) & \text{denota} & S \\
(a\colon S) & \text{denota} & [a\colon S] \\
(\{\}\colon S) & \text{denota} & \{S\} \\
(\Diamond\colon S) & \text{denota} & \langle S\rangle \\
(\triangle\colon S) & \text{denota} & \triangle S
\end{array}$$

$$(p\colon S) \quad \text{denota} \quad (e\colon (p'\colon S)) \quad \text{se } p = e.p'$$

Il tipo–cammino è (una notazione per) un tipo e quindi la sua estensione su un certo dominio è individuata dalla funzione interpretazione $\mathcal{I}$. Ad esempio, il tipo–cammino ($\triangle$.`name`: *"Silvano"*) individua tutti gli oggetti (il primo elemento del cammino è $\triangle$) che hanno un'attributo `name` con un valore *"Silvano"*; in particolare questi oggetti possono appartenere ad una generica classe che ha l'attributo `name` definito come una stringa. Per considerare una determinata classe, ad esempio `Employee`, il tipo–cammino deve essere congiunto con il nome della relativa classe: `Employee` $\sqcap$ ($\triangle$.`name`: *"Silvano"*). Nello stesso modo, il tipo–cammino $S = (\triangle.\texttt{managed-by}.\triangle.\texttt{salary}\colon 40 \div 60)$ non impone restrizioni sul dominio dell'attributo `managed-by`; se si considera la sua congiunzione con la classe `storage`, cioè `storage` $\sqcap\, S$, allora $\sigma$(`storage`) impone implicitamente che gli oggetti del dominio di `managed-by` appartengano alla classe `manager`. Inoltre, è possibile imporre esplicitamente una classe come dominio di un attributo nel seguente modo: $S' = (\triangle.\texttt{managed-by}\colon \texttt{tmanager} \sqcap (\triangle.\texttt{salary}\colon 40 \div 60))$.

Definiamo formalmente la nozione di *regola di integrità*.

**Definizione 13 (Regola di Integrità)** *Dato un sistema di tipi* **S**, *una* re-

gola di integrità, *o più semplicemente* regola, *R su* $\mathbf{S}$ *è un elemento* $(S^a, S^c)$ *del prodotto cartesiano* $\mathbf{S} \times \mathbf{S}$.

Informalmente, una regola di integrità $R = (S^a, S^c)$ ha lo scopo di vincolare ulteriormente l'istanza legale di uno schema, stabilendo una *relazione inclusione* tra il tipo $S^a$ e il tipo $S^c$: per ogni valore $v$, se $v$ è di tipo $S^a$ ($v \in \mathcal{I}[S^a]$) allora $v$ deve essere di tipo $S^c$ ($v \in \mathcal{I}[S^c]$). Una regola di integrità $R$ è quindi universalmente quantificata su tutti i valori $\mathcal{V}(\mathcal{O})$.

Nella regola di integrità $R = (S^a, S^c)$ i tipi $S^a$ e $S^c$ vengono chiamati rispettivamente *antecedente* e *conseguente* della regola e la regola verrà scritta anche nella usuale forma $R = S^a \rightarrow S^c$.

Nella nostra tesi consideriamo una restrizione delle regole di integrità sopra definite, e cioè solo quelle i cui antecedenti e conseguenti sono *classi virtuali* o *nomi di tipi valore* ($S^a, S^c \in \mathbf{V} \cup \mathbf{T}$) escludendo quindi la possibilità che antecedente e conseguente siano *classi base* e *tipi–valore base* ($S^a, S^c \in \mathbf{C} \cup \mathbf{B}$). Andiamo infatti a considerare il seguente vincolo di integrità $R_1 = (S^a, S^c)$ definito come classe base:

$$\begin{aligned} \sigma_{\mathtt{P}}(S^a) &= \mathtt{manager} \\ \sigma_{\mathtt{P}}(S^c) &= \mathtt{person} \end{aligned}$$

Dalla *relazione di inclusione* definita sulle regole risulta che

$$\mathcal{I}[\mathtt{manager}] \subseteq \mathcal{I}[\mathtt{person}],$$

pur mantenendo per ciascuna classe la semantica di *classe-base*. Quindi la regola $R_1$ definisce una relazione di ereditarietà `isa` tra `manager` e `person`, che nel formalismo ODL può essere espressa direttamente nella definizione di classe come:

$$\sigma_{\mathtt{P}}(\mathtt{manager}) = \mathtt{person} \sqcap \triangle[\cdots]$$

Tale considerazione suggerisce quindi di escludere le regole definite sulle *classi-base* poichè non aggiungono espressività al formalismo ODL da noi usato.
Inoltre imporre una regola su un *tipo valore base* significa (al piú) restringere un campo di valori (ad esempio passare da $\sigma(S^a) = \mathtt{integer}$ a $\sigma(S^c) =$ $10 \div 100$) che puó sempre essere applicata in sede di definizione dello schema restringendo direttamente i domini che sono contenuti in $S^a$ e quindi riteniamo di escludere tali regole poichè hanno poca rilevanza.



**Definizione 14 (Schema con Regole di Integrità)** *Dato un sistema di tipi* $\mathbf{S}$, *uno* schema con regole *su* $\mathbf{S}$ *è una coppia* $(\sigma, \mathbf{R})$, *dove* $\sigma$ *è uno schema su* $\mathbf{S}$ *e* $\mathbf{R}$ *è un insieme di regole su* $\mathbf{S}$.

Ad esempio, lo schema con regole mostrato in tabella 4.1 descrive una base di dati relativa a quella parte della struttura di una società che si occupa di materiali. Lo schema rappresenta, nel formalismo ODL per i tipi e in quello dei tipi-cammino per i vincoli di integrità, la stessa struttura presentata nell'introduzione nel formalismo OODB-like.

Commentiamo il significato di alcune regole. La regola $R_1$ dice che i manager con un livello compreso tra 5 e 10 devono percepire un salario maggiore di 40k e minore di 60k dollari. La regola $R_2$ impone che un materiale con rischio superiore o uguale a 10 sia anche nella classe dei materiali speciali (`smaterial`). In altri termini, *una condizione sufficiente* per essere un `smaterial` è quella di essere un `material` con rischio superiore o uguale a 10; pertanto per la classe base `smaterial` viene data una condizione necessaria (espressa in $\sigma(\mathtt{smaterial})$) e una condizione sufficiente (espressa tramite la regola) *differente* dalla condizione necessaria. Di conseguenza, si ottiene un diverso effetto da quello che si otterrebbe introducendo la classe `smaterial` come virtuale con la seguente descrizione

$$\sigma_{\mathtt{V}}(\mathtt{smaterial}) = \mathtt{material} \sqcap (\triangle.\mathtt{risk}\colon 10 \div \infty)$$

Le regole $R_3, R_4$ e $R_5$ sono regole che riguardano la classe `storage`.

Come abbiamo detto, una regola di integrità $R = S^a \rightarrow S^c$ stabilisce che ogni valore nell'interpretazione di $S^a$ deve essere anche contenuto nell'interpretazione di $S^c$. Quindi un'istanza dello schema con regole $(\sigma, \mathbf{R})$ è definita come un'istanza dello schema $\sigma$ che soddisfa tutte le inclusioni tra le interpretazioni dei tipi stabilite tramite le regole.

**Definizione 15 (Istanza Legale con Regole)** *Dato uno schema con regole* $(\sigma, \mathbf{R})$ *su* $\mathbf{S}$, *un*'istanza–gfp $\mathcal{I}$ *di* $\sigma$ *su* $\delta$ *è detta* istanza legale *di* $(\sigma, \mathbf{R})$ *sse per ogni* $R \in \mathbf{R}$, $\mathcal{I}[S^a] \subseteq \mathcal{I}[S^c]$.

La relazione di sussunzione rispetto ad uno schema con regole è quindi definita rispetto alle istanze legali dello schema con regole come segue:

$$
\left.
\begin{array}{ll}
\sigma & \left\{
\begin{array}{lcl}
\sigma_{\mathrm{P}}(\texttt{material}) & = & \triangle[\texttt{name: string, risk: integer, feature: \{string\}}] \\
\sigma_{\mathrm{P}}(\texttt{smaterial}) & = & \texttt{material} \\
\sigma_{\mathrm{P}}(\texttt{storage}) & = & \triangle[\texttt{managed-by: manager, category: string,} \\
 & & \quad \texttt{stock: \{[item: material, qty: } 10 \div 300\texttt{]\}]} \\
\sigma_{\mathrm{P}}(\texttt{sstorage}) & = & \texttt{storage} \\
\sigma_{\mathrm{P}}(\texttt{manager}) & = & \triangle[\texttt{name: string, salary: } 40000 \div \infty\texttt{, level: } 1 \div 15] \\
\sigma_{\mathrm{P}}(\texttt{tmanager}) & = & \texttt{manager} \sqcap \triangle[\texttt{level: } 8 \div 12]
\end{array}
\right. \\
\mathbf{R} & \left\{
\begin{array}{ll}
R_1: & \texttt{manager} \sqcap (\triangle.\texttt{level: } 5 \div 10) \rightarrow (\triangle.\texttt{salary: } 40000 \div 60000) \\
R_2: & \texttt{material} \sqcap (\triangle.\texttt{risk: } 10 \div \infty) \rightarrow \texttt{smaterial} \\
R_3: & \texttt{storage} \sqcap (\triangle.\texttt{category: ``B4''}) \rightarrow \triangle[\texttt{managed-by: tmanager}] \\
R_4: & \texttt{storage} \sqcap (\triangle.\texttt{stock.}\forall.\texttt{item: smaterial}) \rightarrow \texttt{sstorage} \\
R_5: & \texttt{storage} \sqcap (\triangle.\texttt{stock.}\forall.\texttt{qty: } 10 \div 50) \rightarrow (\triangle.\texttt{category: ``A2''})
\end{array}
\right.
\end{array}
\right\} (\sigma, \mathbf{R})
$$

Tabella 4.1: Esempio di Schema con Regole di Integrità

**Definizione 16 (Sussunzione con Regole)** *Dato uno schema con regole* $(\sigma, \mathbf{R})$ *su* $\mathbf{S}$*, siano* $S, S' \in \mathbf{S}$*; definiamo la relazione di sussunzione rispetto a* $(\sigma, \mathbf{R})$ *come segue:*

$$S \sqsubseteq_{\mathrm{R}} S' \quad \textit{sse} \quad \mathcal{I}[S] \subseteq \mathcal{I}[S'] \quad \textit{per ogni istanza } \mathcal{I} \textit{ di } (\sigma, \mathbf{R}).$$

La relazione di equivalenza sui tipi indotta da $\sqsubseteq_{\mathrm{R}}$ viene indicata con $\simeq_{\mathrm{R}}$.

È immediato verificare che, per ogni $S, S' \in \mathbf{S}$, se $S \sqsubseteq_\sigma S'$ allora $S \sqsubseteq_{\mathrm{R}} S'$. Il contrario, in generale, non vale in quanto le inclusioni tra le interpretazione dei tipi stabilite tramite le regole fanno sorgere *nuove* relazioni di sussunzione. Intuitivamente, come mostrato dall'esempio di figura 4.1, se $S \sqsubseteq_\sigma S^a$ e $S^c \sqsubseteq_\sigma S'$ allora $S \sqsubseteq_{\mathrm{R}} S'$. In tal caso diciamo che la regola $R$ è applicabile a $S$.

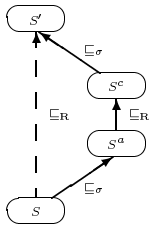
Più precisamente, dato uno schema con regole $(\sigma, \mathbf{R})$ su $\mathbf{S}$, sia $S \in \mathbf{S}$, $R \in \mathbf{R}$ e $p$ un cammino; allora diremo che

$R$ è *applicabile* a $S$ con cammino $p$ se $S \sqsubseteq_\sigma (p : S^a)$.

Dalla semantica delle regole segue immediatamente che dato un tipo $S$ e una regola $R$ applicabile a $S$, intersecando il tipo $S$ con il conseguente $S^c$ di $R$ si ottiene un tipo che è equivalente a quello originale rispetto allo schema con regole.

**Proposizione 6** *Per ogni* $S \in \mathbf{S}$ *e per ogni* $R \in \mathbf{R}$*, se* $R$ *è applicabile a* $S$ *con il cammino* $p$*, allora* $S \sqcap (p\colon S^c) \simeq_{\mathrm{R}} S$.






$$
\begin{array}{l}
S = \ \texttt{manager} \sqcap (\triangle.\texttt{level: } 5 \div 8) \\
S' = \ (\triangle.\texttt{salary: } 30 \div 70) \\
R_1 = \texttt{manager} \sqcap (\triangle.\texttt{level: } 5 \div 10) \\
\qquad\qquad \rightarrow (\triangle.\texttt{salary: } 40 \div 60) \\
S \sqsubseteq_{\mathrm{R}} S', \quad S \not\sqsubseteq_\sigma S'
\end{array}
$$

Figura 4.1: Relazione di sussunzione forzata mediante una regola.

Naturalmente, al tipo $S \sqcap (p\colon S^c)$ ottenuto tramite l'applicazione della regola $R$ possono essere applicate altre regole che non erano applicabili al tipo originale. Ad esempio, la regola $R_3$ è applicabile con cammino $\epsilon$ al tipo

$$\texttt{storage} \sqcap \triangle[\texttt{category: ``B4''}, \texttt{managed-by: } \triangle[\texttt{level: } 4 \div 10]]$$

ottenendo il tipo

$$\texttt{storage} \sqcap \triangle[\texttt{category: ``B4''}, \texttt{managed-by: tmanager} \sqcap \triangle[\texttt{level: } 4 \div 10]]$$

al quale è applicabile la regola $R_1$ con il cammino $\triangle$.`managed-by` ottenendo

$$\texttt{storage} \sqcap \triangle[\texttt{category: ``B4''}, \texttt{managed-by: tmanager} \sqcap \triangle[\texttt{level: } 4 \div 10, \texttt{salary: } 40 \div 60]]$$

Il tipo che si ottiene applicando *tutte* le trasformazioni possibili è detto *espansione semantica* ed è definito nella prossima sezione.

## 4.2 Espansione semantica di un tipo

L'espansione semantica di un tipo permette di incorporare ogni possibile restrizione che non è presente nel tipo originale ma che è *logicamente implicata* dal tipo e dallo schema. Formalmente questo viene espresso tramite la seguente definizione di espansione semantica:

**Definizione 17 (Espansione Semantica)** *Dato uno schema con regole* $(\sigma,\mathbf{R})$ *su* $\mathbf{S}$*, e un tipo* $S \in \mathbf{S}$*; l'*espansione semantica *di* $S$ *rispetto a* $(\sigma,\mathbf{R})$*,* $EXP(S)$*, è un tipo di* $\mathbf{S}$ *tale che:*

1. $EXP(S) \simeq_{\mathrm{R}} S$*;*
2. *per ogni* $S' \in \mathbf{S}$ *tale che* $S' \simeq_{\mathrm{R}} S$ *si ha che* $EXP(S) \sqsubseteq_\sigma S'$*.*

$EXP(S)$ è il tipo più specializzato tra tutti i tipi $\simeq_{\mathrm{R}}$–equivalenti al tipo $S$ in quanto include tutte le possibili restrizioni implicate dalle regole $\mathbf{R}$. In questo modo è il tipo più piccolo rispetto alla relazione $\sqsubseteq_\sigma$ tra tutti i tipi $\simeq_{\mathrm{R}}$–equivalenti a $S$. Si noti che $EXP(S)$ individua una classe di tipi $\simeq$–equivalenti, nella quale ogni elemento è un tipo $\simeq_{\mathrm{R}}$–equivalente al tipo $S$.

Il metodo proposto per determinare l'espansione semantica è caratterizzato dai seguenti punti:

- iterazione della trasformazione "se un tipo implica l'antecedente di una regola allora il conseguente di tale regola può essere ad esso congiunto";
- valutazione delle implicazioni logiche tramite il calcolo della sussunzione tra tipi.

Allo scopo di individuare tutte le trasformazioni che uno schema con regole $(\sigma,\mathbf{R})$ induce su un tipo, si introduce la funzione totale $, : \mathbf{S} \longrightarrow \mathbf{S}$ tale che

$$, (S) \;=\; \begin{cases} S \sqcap \bigsqcap_k (p_k : S_k^c) & \forall R_k, p_k :\ S \sqsubseteq_\sigma (p_k\colon S_k^a),\ S \not\sqsubseteq_\sigma (p_k\colon S_k^c) \\ S & \text{altrimenti} \end{cases}$$

e si definisce $\tilde{,} = ,^{\bar{i}}$, dove $\bar{i}$ è il più piccolo intero tale che $,^{\bar{i}} = ,^{\bar{i}+1}$. L'esistenza di $\bar{i}$ è garantita dal fatto che il numero di regole è finito e una regola non può essere applicata più di una volta con lo stesso cammino grazie al controllo $S \not\sqsubseteq_\sigma (p\colon S^c)$.

Il seguente teorema, dimostrato in [Ben94] , assicura la possibilità di utilizzare la funzione $\tilde{,}(S)$ in luogo di $EXP(S)$:

**Teorema 3** *Dato uno schema con regole* $(\sigma,\mathbf{R})$ *su* $\mathbf{S}$*, per ogni* $S \in \mathbf{S}$*,* $EXP(S)$ *può essere effettivamente calcolata tramite* $\tilde{,}(S)$*.*

Tramite $\tilde{,}(S)$ possiamo dunque formalizzare il legame esistente tra le relazioni di sussunzione $\sqsubseteq_{\mathrm{R}}$ e $\sqsubseteq_\sigma$.



**Corollario 1** *Dato uno schema con regole* $(\sigma,\mathbf{R})$ *su* $\mathbf{S}$*, per ogni* $S, S' \in \mathbf{S}$*, si ha che* $S \sqsubseteq_{\mathrm{R}} S'$ *se e solo se* $\tilde{,}(S) \sqsubseteq_\sigma S'$*.*

Pertanto, il calcolo della sussunzione in uno schema con regole $(\sigma,\mathbf{R})$ può essere effettuato prima determinando l'espansione semantica di un tipo e quindi calcolando la sussunzione in $\sigma$.

## 4.3 Esempi di ottimizzazione semantica delle interrogazioni

Un'interrogazione della base di dati corrisponde alla semantica di una classe virtuale, in quanto essa contiene le condizioni necessarie e sufficienti per definire gli oggetti da selezionare. Il risultato dell'interrogazione è l'istanza della classe virtuale, cioè l'insieme di oggetti (i loro oid) *esistenti* nella base di dati che soddisfano la descrizione della classe virtuale.

Descriviamo ora il metodo usato per l'ottimizzazione semantica di un'interrogazione $Q$, essenzialmente basato sul calcolo della sua espansione semantica $\tilde{,}(Q)$ e delle relazioni di sussunzione tra i fattori dell'interrogazione nello schema con regole. Prima della sua descrizione formale, esso viene introdotto tramite alcuni esempi.

Consideriamo la seguente interrogazione:

$[Q_1]$ "Seleziona gli `storage` in cui tutti i materiali conservati hanno un rischio più grande o uguale a 15":

$$\sigma_{\mathtt{V}}(Q_1) \;=\; \mathtt{storage} \sqcap (\triangle.\mathtt{stock}.\forall.\mathtt{item}.\ \triangle.\mathtt{risk}\colon 15 \div \infty)$$

Questa interrogazione può essere riscritta in modo da mettere in evidenza la sotto–interrogazione relativa alla classe `material`:

$$\begin{aligned} \sigma_{\mathtt{V}}(Q_1) &= \mathtt{storage} \sqcap (\triangle.\mathtt{stock}.\forall.\mathtt{item}\colon Q_1') \\ \sigma_{\mathtt{V}}(Q_1') &= \mathtt{material} \sqcap (\triangle.\mathtt{risk}\colon 15 \div \infty) \end{aligned}$$

All'interrogazione $Q_1$ sono applicabili le regole $R_2$ e $R_4$, ottenendo la seguente espansione semantica

$$\begin{aligned} \sigma_{\mathtt{V}}(\tilde{,}(Q_1)) &= \mathtt{sstorage} \sqcap (\triangle.\mathtt{stock}.\forall.\mathtt{item}\colon Q_1'') \\ \sigma_{\mathtt{V}}(Q_1'') &= \mathtt{smaterial} \sqcap (\triangle.\mathtt{risk}\colon 15 \div \infty) \end{aligned}$$

Il risultato di questa trasformazione è l'individuazione di nuove classi obiettivo per l'interrogazione e per la sua sotto–interrogazione, che sono una specializzazione di quelle dichiarate esplicitamente. Più precisamente, viene individuata la classe base più specifica che generalizza l'interrogazione $Q_1$ (classe `sstorage`) e la classe base più specifica che generalizza la sotto–interrogazione $Q_1'$ (classe `smaterial`). In altri termini $Q_1$ e $Q_1'$ vengono classificate rispetto alla tassonomia delle classi base. Questo è illustrato in figura 4.2: in figura 4.2.a è riportata la posizione di $Q_1$ e di $Q_1'$ rispetto alla gerarchia delle classi derivante dalla dichiarazione esplicita, posizione indicata con una freccia vuota; in figura 4.2.b è riportata la posizione di $Q_1$ e di $Q_1'$ dopo l'espansione semantica che ha permesso di individuare le relazioni di sussunzione, indicate con una freccia piena.

Consideriamo ora la seguente interrogazione:

$Q_2$: "Seleziona gli `storage` di categoria *''B4''* e diretti da un manager con livello minore o uguale a 10":

$$\sigma_{\mathtt{V}}(Q_2) \;=\; \mathtt{storage} \sqcap (\triangle.\mathtt{category}\colon \textit{''B4''}) \sqcap (\triangle.\mathtt{managed\text{-}by}\colon \mathtt{manager} \sqcap (\triangle.\mathtt{level}\colon \infty \div 10))$$

Riscriviamo questa interrogazione evidenziando la sotto–interrogazione relativa alla classe `manager`:

$$\begin{aligned} \sigma_{\mathtt{V}}(Q_2) &= \mathtt{storage} \sqcap (\triangle.\mathtt{category}\colon \textit{''B4''}) \sqcap (\triangle.\mathtt{managed\text{-}by}\colon Q_3') \\ \sigma_{\mathtt{V}}(Q_2') &= \mathtt{manager} \sqcap (\triangle.\mathtt{level}\colon \infty \div 10)) \end{aligned}$$

All'interrogazione $Q_2$ sono applicabili le regole $R_3$ e $R_1$ e la sua espansione semantica è la seguente:

$$\begin{aligned} \sigma_{\mathtt{V}}(\tilde{(},(Q_2)) &= \mathtt{storage} \sqcap (\triangle.\mathtt{category}\colon \textit{''B4''}) \sqcap (\triangle.\mathtt{managed\text{-}by}\colon Q_3'') \\ \sigma_{\mathtt{V}}(Q_2'') &= \mathtt{tmanager} \sqcap (\triangle.\mathtt{level}\colon 8 \div 10) \sqcap (\triangle.\mathtt{salary}\colon 40 \div 60) \end{aligned}$$

Si noti che considerando singolarmente la sotto–interrogazione $Q_2''$, cioè

$Q_3$: "Seleziona i manager con livello minore o uguale a 10":

$$\sigma_{\mathtt{V}}(Q_3) \;=\; \mathtt{manager} \sqcap (\triangle.\mathtt{level}\colon \infty \div 10))$$



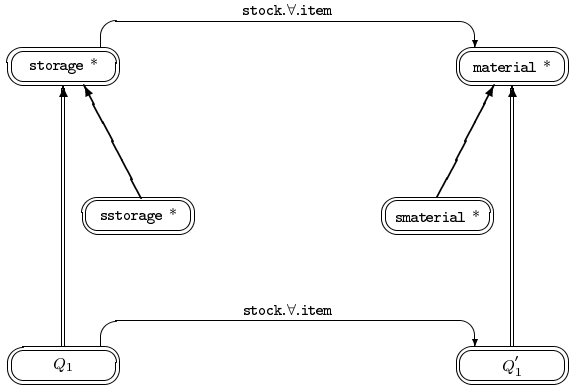



figura 4.2.a

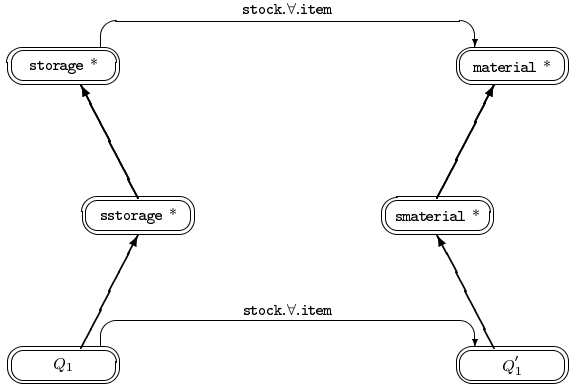



figura 4.2.b

Figura 4.2: Classificazione dell'interrogazione nella tassonomia delle classi

nessuna regola è applicabile a $Q_3$; di conseguenza $\tilde{,}(Q_3)$ coincide con la descrizione di $Q_3$ e quindi $\tilde{,}(Q_3)$ è diversa da $\sigma(Q_3'')$, più precisamente $\sigma(Q_2'') \sqsubseteq_\sigma \tilde{,}(Q_3)$. Pertanto, una sotto–interrogazione può subire delle trasformazioni dovute al fatto che è all'interno di un'altra interrogazione.

Il prossimo esempio mostra come è possibile ottenere tramite le regole di integrità la cosiddetta *introduzione di indice* (*index introduction*), cioè l'introduzione di un fattore che include un attributo con indice, che può essere utile per l'ottimizzazione dell'interrogazione.

$Q_4$ "Seleziona gli `storage` in cui tutti i materiali disponibili sono in una quantità compresa tra 40 e 50":

$$\sigma_{\mathtt{V}}(Q_4) \;=\; \mathtt{storage} \sqcap (\triangle.\mathtt{stock}.\forall.\mathtt{qty}\colon 40 \div 50)$$

All'interrogazione $Q_4$ è applicabile la regola $R_5$ ottenendo

$$\sigma_{\mathtt{V}}(\tilde{,}(Q_4)) \;=\; \mathtt{storage} \sqcap (\triangle.\mathtt{stock}.\forall.\mathtt{qty}\colon 40 \div 50) \sqcap (\triangle.\mathtt{category}\colon \text{``}\mathtt{A2}\text{''})$$

Se c'è un indice sull'attributo `category` allora il nuovo fattore può essere utilizzato nell'elaborazione dell'interrogazione riducendo gli accessi in memoria ed i tempi di risposta del sistema.

Naturalmente, il vantaggio fornito dall'ottimizzazione in termini di tempi di risposta del sistema all'interrogazione, può essere stimato in rapporto al numero di vincoli di integrità, alla specificità degli antecedenti (che misura la probabilità di applicazione del vincolo alla generica interrogazione) e dei conseguenti (che misura la specializzazione introdotta dall'espansione semantica).
Infatti il caso migliore si presenta quando abbiamo un buon numero di vincoli di integrità (almeno 3-4 per ciascuna classe base), antecedenti di facile applicazione (tipicamente con restrizioni sui domini degli attributi ad intervalli di valore) e conseguenti che introducono una forte specializzazione (tipicamente la restrizione a valori singoli dei domini degli attributi). Nelle condizioni descritte di caso migliore la probabilità di applicare i vincoli di integrità e di individuare nuovi fattori per l'interrogazione che facilitano la ricerca in memoria è alta, mentre questa probabilità diviene bassa nelle condizioni opposte di caso peggiore (pochi vincoli di integrità, antecedenti molto specifici e conseguenti molto generici).



Il metodo da noi proposto risulta efficiente per gli schemi con regole che soddisfano le condizioni che abbiamo definito di caso migliore: potrebbe sembrare una limitazione molto forte, ma in realtà i vincoli di integrità sono significativi quando rispettano tali condizioni e quindi in generale ci si aspetta che il nostro metodo di ottimizzazione sia efficace.

## 4.4 Algoritmo di Espansione Semantica di un tipo

Il punto di partenza per definire un algoritmo di espansione semantica è uno schema con regole $(\sigma, \mathbf{R})$, nel quale sia stata controllata la coerenza delle classi e (eventualmente) le contraddizioni e le ridondanze dell'insieme delle regole di integrità [BBLS94a].

L'algoritmo è basato sul calcolo della funzione di espansione semantica $\tilde{,}(Q)$ definita nella sezione precedente, la quale viene trasformata in una nuova funzione che permette un algoritmo di calcolo più efficiente. Gli strumenti necessari per l'algoritmo sono lo schema canonico (vedi algoritmo di tabella 3.2), l'incoerenza (vedi algoritmo di tabella 3.4) e la sussunzione (vedi algoritmo di tabella 3.5) già ampiamente descritti nei capitoli precedenti.
In particolare volendo calcolare l'espansione semantica del tipo $S$ occorre sempre calcolarne la coerenza poichè nel caso in cui $S$ sia effettivamente incoerente significa che è il tipo più specifico che si possa ottenere ($S \simeq_{\mathrm{R}} \bot$) e quindi non ha più significato il calcolo dell' espansione semantica.
Prima di descrivere l'algoritmo, riprendiamo il concetto di *dipendenza* di un tipo che lega il tipo stesso ai tipi contenuti nella propria descrizione: esso permette di definire l'insieme $\mathbf{C}^+(S)$ dei tipi (compreso S) che compaiono nella descrizione di S a qualsiasi livello di innestamento.

A partite dallo schema canonico $\overline{\mathbf{S}}(\mathbf{A}, \mathbf{B}, \overline{\mathbf{N}})$ diciamo che:

- $\mathbf{C}^0(S) = \{S\} \cup \{S_1 \in \overline{\mathbf{S}}(\mathbf{A}, \mathbf{B}, \overline{\mathbf{N}}) \setminus (\mathbf{B} \cup \overline{\mathbf{C}}) \mid S \textit{ dipende da } S_1\}$
- $\mathbf{C}^{i+1}(S) = \mathbf{C}^i(S) \cup \{S_1 \in \overline{\mathbf{S}}(\mathbf{A}, \mathbf{B}, \overline{\mathbf{N}}) \setminus \mathbf{B} \cup \overline{\mathbf{C}} \mid S_2 \in \mathbf{C}^i(S) \wedge S_2 \textit{ dipende da } S_1\}$
- $\mathbf{C}^+(S) = \mathbf{C}^{\bar{i}}(S)$

dove $\bar{i}$ è il piú piccolo intero tale che $\mathbf{C}^{\bar{i}}(S) = \mathbf{C}^{\bar{i}+1}$.

Quindi $\mathbf{C}^+(S)$ contiene tutti i tipi che compaiono nella descrizione di S a qualsiasi livello di innestamento ad esclusione degli *atomi fittizi* ed i $tipi - valore\ base$; il motivo di tale esclusione apparirá piú chiaro in seguito, per ora basti ricordare che le regole non sono definite sui $tipi - valore\ base$ e che gli *atomi fittizi* hanno descrizione $\triangle\top$.

Consideriamo ora come esempio l'interrogazione $Q_1$ introdotta a pagina 44, ed esprimiamola nella sua forma canonica:

$$
\begin{array}{lcl}
\nu(Q_1) & = & \overline{\texttt{storage}} \sqcap \triangle t_1 \\
\nu(t_1) & = & [\texttt{managed-by}\colon \texttt{manager}, \texttt{category}\colon \texttt{string}, \\
 & & \texttt{stock}\colon t_2] \\
\nu(t_2) & = & \{t_3\} \\
\nu(t_3) & = & [\texttt{item}\colon Q_1', \texttt{qty}\colon b_1] \\
\nu(Q_1') & = & \overline{\texttt{material}} \sqcap \triangle t_4 \\
\nu(t_4) & = & [\texttt{name}\colon \texttt{string}, \texttt{risk}\colon b_2, \texttt{feature}\colon t5] \\
\nu(t_5) & = & \{\texttt{string}\} \\
\nu(b_1) & = & 10 \div 300 \\
\nu(b_2) & = & 15 \div \infty \\
\nu(\texttt{manager}) & = & \overline{\texttt{manager}} \sqcap \triangle t_6 \\
\nu(t_6) & = & [\texttt{name}\colon \texttt{string}, \texttt{salary}\colon b_3, \texttt{level}\colon b_4] \\
\nu(b_3) & = & 40 \div \infty \\
\nu(b_4) & = & 1 \div 15
\end{array}
$$

Avendo a disposizione la forma canonica di $Q_1$, l'insieme $\mathbf{C}^+(Q_1)$ si calcola molto facilmente ottenendo:

$$\mathbf{C}^+(Q_1) = \{Q_1, t_1, \texttt{manager}, t_6, t_2, t_3, Q_1', t_4, t_5\}$$

L'insieme $\mathbf{C}^+(S)$ permette di introdurre la seguente proposizione di trasformazione della funzione di espansione semantica:

**Proposizione 7** *Dato lo schema canonico* $\overline{\mathbf{S}}(\mathbf{A}, \mathbf{B}, \overline{\mathbf{N}})$*, il tipo S e l'insieme* $\mathbf{C}^+(S) \subseteq \overline{\mathbf{S}}(\mathbf{A}, \mathbf{B}, \overline{\mathbf{N}})$*,*
*la funzione di Espansione Semantica* $, (S)$*:*



$$
, (S) = \begin{cases} S \sqcap \bigsqcap_k (p_k : S_k^c) & \forall R_k, p_k : \ S \sqsubseteq_\sigma (p_k\colon S_k^a), \\ & \qquad\qquad S \not\sqsubseteq_\sigma (p_k\colon S_k^c) \\ S & \textit{altrimenti} \end{cases} \tag{4.1}
$$

*è equivalente alla seguente funzione:*

$\forall\ S_1 \in \mathbf{C}^+(S)$

$$
, (S_1) = \begin{cases} S_1 \sqcap \bigsqcap_k (\epsilon : S_k^c) & \forall R_k : \ S_1 \sqsubseteq_\sigma (\epsilon\colon S_k^a), \\ & \qquad\qquad S_1 \not\sqsubseteq_\sigma (\epsilon\colon S_k^c) \\ S_1 & \textit{altrimenti} \end{cases} \tag{4.2}
$$

*che può anche essere scritta come:*

$\forall\ S_1 \in \mathbf{C}^+(S)$

$$
, (S_1) = \begin{cases} S_1 \sqcap \bigsqcap_k S_k^c & \forall R_k : \ S_k^a \in GS(S_1), \\ & \qquad\qquad S_k^c \notin GS(S_1) \\ S_1 & \textit{altrimenti} \end{cases} \tag{4.3}
$$

Illustriamo la proposizione applicandola all'esempio dell'interrogazione $Q_1$: nella funzione 4.1 i cammini significativi per il controllo della sussunzione sono soltanto quelli in cui il generico cammino $p_k$ identifica un *tipo cammino* $(p_k : S_k^c)$ confrontabile con S (ad esempio sono due classi oppure due tipi-valore). Inoltre, poichè le regole non sono definite sui $tipi - valore\ base$, per i cammini che mappano su di un $tipo - valore\ base$ possiamo evitare di controllarne la sussunzione in quanto a priori si puó affermare che le regole non sono applicabili con tali cammini.

Quindi i cammini significativi per $Q_1$, seguendo una visita depth first, sono i seguenti:
$\epsilon$
$\triangle$
$\triangle$.`managed-by`

△.managed-by.△
△.stock
△.stock.∀
△.stock.∀.item
△.stock.∀.item.△
△.stock.∀.item.△.feature

Mostriamo ora quali sono i cammini $p_k$ che vengono implicitamente considerati dalla funzione 4.2 al variare di $S_1 \in \mathbf{C}^+(Q_1)$ attraverso il controllo della condizione $S_1 \sqsubseteq_\sigma (\epsilon{:}\, S_k^a)$:

| | | |
|---|---|---|
| $S_1 = Q_1$ | cammino | $\epsilon$ |
| $= t_1$ | | △ |
| $=$ manager | | △.managed-by |
| $= t_6$ | | △.managed-by.△ |
| $= t_2$ | | △.stock |
| $= t_3$ | | △.stock.∀ |
| $= Q_1'$ | | △.stock.∀item |
| $= t_4$ | | △.stock.∀item.△ |
| $= t_5$ | | △.stock.∀item.△.feature |

Questi cammini coincidono esattamente con quelli considerati dalla funzione 4.1, quindi abbiamo verificato intuitivamente l'equivalenza tra 4.1 e 4.2 nell'esempio presentato.

Ricordando poi la definizione di $GS(S)$:

$$GS(S) = \{S_1 \in \mathbf{S} \mid S \sqsubseteq_\sigma S_1\}$$

risulta ovvia l'equivalenza tra $S \sqsubseteq_\sigma (\epsilon{:}\, S_k^a)$ e $S_k^a \in GS(S)$ da cui segue immediatamente che la 4.3 può essere applicata in luogo della 4.2 (e della 4.1).

Questa considerazione è molto importante poichè permette di ridurre il numero di $tipi - cammino$ da verificare nel calcolo dell'espansione semantica e porta alla definizione di un algoritmo efficiente di ottimizzazione semantica basato appunto sulla funzione 4.3.
L'algoritmo è riportato in tabella 4.2.



**Algoritmo (Espansione semantica di un tipo S)**

- **Inizializzazione**:
  Acquisizione del tipo $S$
- **Iterazione**:
  $\forall\, S_1 \in \mathbf{C}^+(S)$

$$\begin{aligned} \mathit{Forma\ canonica} &: \nu(S_1) \\ \mathit{Incoerenza} &: S_1 \in \widetilde{\Phi}_\nu \\ \mathit{Sussunzione} &: GS(S_1) \end{aligned}$$

$$\text{se } S_1 \notin \widetilde{\Phi}_\nu : \quad ,^i(S_1) = \begin{cases} S_1 \sqcap \bigsqcap_k S_k^c & \forall R_k : \ S_k^a \in GS(S_1), \\ & \quad S_k^a \notin GS(S_1) \\ S_1 & \text{altrimenti} \end{cases}$$

- **Stop**:
  $\forall\, S_1 \in \mathbf{C}^+(S), ,^{i+1}(S_1) = ,^i(S_1)$,
  or
  $\exists\, S_1 \in \mathbf{C}^+(S), ,^i(S_1) = \bot$.

L'espansione semantica è $\tilde{,}(S) = ,^i(S)$

Tabella 4.2: Algoritmo di espansione semantica

Il processo termina quando non abbiamo piú regole applicabili al tipo $S$ (ed ai tipi che compaiono nella sua descrizione ad ogni livello di innestamento): l'esistenza della condizione di terminazione è garantita dal fatto che il numero di regole è finito e che una regola non può essere applicata piú di una volta ad tipo $S_1 \in \mathbf{C}^+(S)$ per il controllo $S_k^c \notin GS(S_1)$.

Come risultato l'algoritmo fornisce il tipo S in forma canonica descritto dalla propria espansione semantica; per semplicità, rappresentiamo questa trasformazione con un nuovo schema canonico $\tilde{\nu}$, cioè $\tilde{\nu}(S) = \tilde{,}(S)$.

Vediamo l'espansione semantica che si ottiene per la $Q_1$ applicando l'algoritmo. Come abbiamo detto, un'interrogazione $Q$ viene considerata una classe virtuale dello schema $\sigma$; si considera quindi lo schema canonico $\nu$ equivalente a $\sigma$ e, in particolare, $\nu(Q_1)$. L'espressione $\nu(Q_1)$ permette tra l'altro il controllo di coerenza dell'interrogazione rispetto allo schema $\sigma$. Viene quindi determinata l'espansione semantica di $Q_1$ tramite $\tilde{,}$, cioè il tipo $\tilde{,}(Q_1)$, e viene considerata la sua forma canonica $\tilde{\nu}(Q_1)$ che è la seguente:

$$
\begin{array}{lcl}
\tilde{\nu}(Q_1) & = & \overline{\texttt{storage}} \sqcap \overline{\texttt{sstorage}} \sqcap \triangle t_1 \\
\tilde{\nu}(t_1) & = & [\texttt{managed-by}\colon \texttt{manager}, \texttt{category}\colon \texttt{string}, \\
 & & \texttt{stock}\colon t_2] \\
\tilde{\nu}(t_2) & = & \{t_3\} \\
\tilde{\nu}(t_3) & = & [\texttt{item}\colon Q_1', \texttt{qty}\colon b_1] \\
\tilde{\nu}(b_1) & = & 10 \div 300 \\
\tilde{\nu}(Q_1') & = & \overline{\texttt{material}} \sqcap \overline{\texttt{smaterial}} \sqcap \triangle t_4 \\
\tilde{\nu}(t_4) & = & [\texttt{name}\colon \texttt{string}, \texttt{risk}\colon b_2, \texttt{feature}\colon t_5\{\texttt{string}\}] \\
\tilde{\nu}(t_5) & = & \{\texttt{string}\} \\
\tilde{\nu}(b_2) & = & 15 \div \infty \\
\tilde{\nu}(\texttt{manager}) & = & \overline{\texttt{manager}} \sqcap \triangle t_6 \\
\tilde{\nu}(t_6) & = & [\texttt{name}\colon \texttt{string}, \texttt{salary}\colon b_3, \texttt{level}\colon b_4] \\
\tilde{\nu}(b_3) & = & 40 \div \infty \\
\tilde{\nu}(b_4) & = & 1 \div 15
\end{array}
$$

L'espressione $\tilde{\nu}(Q)$ consente di ottenere in modo immediato la classificazione dell'interrogazione rispetto alla tassonomia delle classi base. Più precisamente, è possibile ottenere sia le classi base più specifiche che generalizzano l'interrogazione sia le classi base più specifiche che generalizzano le sotto–interrogazioni. Riportiamo le definizioni:

$$GS(Q) = \{C \in \mathbf{C} \mid Q \sqsubseteq_{\mathrm{R}} C\}$$



Da $GS(Q)$ è ricavabile in maniera immediata

$$MSGS(Q) = \{C \in GS(Q) \mid \nexists C' \in GS(Q)\colon C' \sqsubseteq_{\mathrm{R}} C\}$$

$MSGS(Q)$ individua l'insieme delle classi base più specifiche che generalizzano l'interrogazione $Q$.

L'aspetto fondamentale è che l'insieme $GS(Q)$, e quindi $MSGS(Q)$, è ottenibile direttamente dalla forma canonica dell'espansione semantica e non si devono fare ulteriori calcoli di sussunzione.

**Proposizione 8** *Dato $\tilde{\nu}(Q) = \bigsqcap_i \overline{C}_i \sqcap \triangle N$, si ha che:*

$$Q \sqsubseteq_{\mathrm{R}} C \text{ sse esiste } i,\ 1 \leq i \leq n, \text{ tale che } \overline{C}_i = C.$$

Ad esempio, $GS(Q_1) = \{\texttt{storage}, \texttt{sstorage}\}$ e pertanto $MSGS(Q_1) = \{\texttt{sstorage}\}$ essendo $\texttt{sstorage} \sqsubseteq_{\mathrm{R}} \texttt{storage}$.

Lo stesso discorso è valido per le eventuali sotto–interrogazioni che devono essere valutate. Esse vengono indicate con $Q_k$; la posizione di $Q_k$ nella struttura di $Q$ è individuata tramite il cammino $p_k$; si noti che per tale cammino si ha che $Q \sqsubseteq_\sigma (p_k\colon Q_k)$.

Come è stato evidenziato con l'esempio relativo all'interrogazione $Q_3$, una sotto–interrogazione può subire delle trasformazioni dovute al fatto che è all'interno di un'altra interrogazione. Per questo motivo l'insieme delle generalizzazioni di una sotto–interrogazione $Q_k$ vanno riferite all'intera interrogazione $Q$. Si definisce l'insieme delle generalizzazioni di $Q_k$ rispetto a $Q$

$$GS_Q(Q_k) = \{C' \in \mathbf{C} \mid Q \sqsubseteq_{\mathrm{R}} (p_k\colon C')\}$$

Questo insieme è un sovrainsieme di $GS(Q_k)$. Da $GS_Q(Q_k)$ si selezionano quelle più specializzate:

$$MSGS_Q(Q_k) = \{C' \in GS_Q(Q_k) \mid \nexists C'' \in GS_Q(Q_k)\colon C'' \sqsubseteq_{\mathrm{R}} C'\}$$

**Proposizione 9** *Data una sotto–interrogazione $Q_k$ di $Q$, individuata dal cammino $p_k$, con $\tilde{\nu}(Q_k) = \bigsqcap_{j=1}^{k_n} \overline{C}_j \sqcap \triangle N_k$, si ha che:*

$$Q \sqsubseteq_{\mathrm{R}} (p_k\colon C) \text{ sse esiste } j,\ 1 \leq j \leq k_n, \text{ tale che } \overline{C}_j = C.$$

*Le sotto-interrogazioni $Q_k$ di $Q$ sono tutti e soli i tipi contenuti in $\mathbf{C}^+(Q) \backslash Q$.*

Ad esempio, $GS_{Q_1}(Q_1') = \{\texttt{material}, \texttt{smaterial}\}$ e poichè $\texttt{smaterial} \sqsubseteq_{\mathrm{R}} \texttt{material}$, $MSGS_{Q_1}(Q_1') = \{\texttt{smaterial}\}$. In definitiva, per $Q_1$ si ha che $MSGS(Q_1) = \{\texttt{sstorage}\}$ e $MSGS_{Q_1}(Q_1') = \{\texttt{smaterial}\}$; questo risultato è illustrato in figura 4.2.

## 4.5 Complessità computazionale

La complessità computazionale dell'algoritmo di Espansione Semantica di un tipo dipende da quella relativa al calcolo di incoerenza e di sussunzione.

Considerando per primo il controllo della coerenza, l'ipotesi della compattezza binaria del sistema dei tipi base è una condizione cruciale per l'efficienza.

**Teorema 4** *Il controllo della coerenza di uno schema può essere effettuato in tempo polinomiale a patto che il sistema dei tipi base sia compatto binario, ed è NP-hard nel caso generale.*

Il risultato NP-hard è conseguenza dell'aver riportato il problema di controllare se l'intersezione di automi finiti è vuota, che è NP-hard, alla coerenza di uno schema. Il calcolo della sussunzione è anche più difficile. Anche una restrizione al sistema di tipi base compatto binario non aiuta perché il problema dell'inclusione del linguaggio per automi a stati finiti può essere riportato alla sussunzione.

**Teorema 5** *La sussunzione è PSPACE-hard.*

Nonostante questi risultati suggeriscano che il calcolo della sussunzione abbia un costo proibitivo, in [Neb90] viene mostrato come in generale uno schema è formulato in modo da essere "quasi" canonico e che quindi passando attraverso la trasformazione in forma canonica il calcolo dell'incoerenza e della sussunzione viene riportato ad un costo di tipo polinomiale (come visto nel Capitolo 3). Sotto queste condizioni, anche la parte dell'algoritmo di Espansione Semantica di un tipo relativo al calcolo di , $^{i}(S)$ risulta polinomiale: infatti poichè $|\overline{\mathbf{C}}^{+}(S)| \leq |\overline{\mathbf{N}}|$ e $|GS(S_1)| \leq |\overline{\mathbf{N}}|$ vengono compiute al piú $|\overline{\mathbf{N}}|^2$ ispezioni.
Se invece ci poniamo nelle condizioni più generali, si può presentare la seguente condizione di caso peggiore [Neb90]:



$$
\begin{array}{lcl}
\mathtt{C_0} & = & [\mathtt{r_1} : \mathtt{C_1}, \mathtt{r_2} : \mathtt{C_1} \sqcap \mathtt{C_2}] \\
\mathtt{C_1} & = & [\mathtt{r_1} : \mathtt{C_2}, \mathtt{r_2} : \mathtt{C_2} \sqcap \mathtt{C_4}] \\
 & \vdots & \\
\mathtt{C_i} & = & \begin{cases} [\mathtt{r_1} : \mathtt{C_{i+1}}, \mathtt{r_2} : \mathtt{C_{i+1}} \sqcap \mathtt{C_{2(i+1)}}] & if\ 2i \leq n, \\ [\mathtt{r_1} : \mathtt{C_{i+1}}, \mathtt{r_2} : \mathtt{C_{i+1}}] & \text{otherwise.} \end{cases} \\
 & \vdots & \\
\mathtt{C_n} & = & \mathtt{Primitive}
\end{array}
$$

L'algoritmo di trasformazione in forma canonica applicato all'esempio genera un numero esponenziale di tipi e quindi nelle condizioni più generali il problema della classificazione di uno schema è di tipo PSPACE-*hard*.

# Capitolo 5

# ODB-QOptimizer: un ottimizzatore semantico di interrogazioni

Nel presente capitolo viene presentato il software ODB-QOptimizer per l'ottimizzazione delle interrogazioni di basi di dati ad oggetti rappresentate secondo il formalismo ODL, basato sulla tecnica dell' espansione semantica di un tipo.

Vengono illustrati i due componenti ODL-Designer (introdotto in [Bal92]) e GES (realizzato durante l'attivitá di tesi) che rappresentano le parti costitutive dell'ottimizzatore.

## 5.1 Architettura e funzionalitá di ODL-Designer

ODL-Designer (Object Description Language Designer) è un ambiente formale per l'acquisizione e la modifica di uno schema descritto con il linguaggio ODL che consente di:

- verificare che lo schema sia:
  - *coerente*: esiste almeno uno stato del DB tale che ogni classe e tipo ha un'estensione non vuota;
  - *ben formato*: non esistono cicli `isa` relativi alla relazione di ereditarietá e non esistono cicli nelle descrizioni dei tipi;





- ottenere la *minimalità* dello schema rispetto alla relazione `isa`, cioè per ogni tipo (classe) viene calcolata la giusta posizione nella tassonomia dei tipi (classe):
  - il tipo (classe) viene inserito sotto tutti i tipi (classi) che specializza;
  - il tipo (classe) viene inserito sopra tutti i tipi (classi) che lo specializzano.

ODL-Designer è stato realizzato in ambiente SICStus Prolog, versione 2.1, su piattaforma hardware SUN x-sparc 10, sistema operativo sunOS, release 4.1.3.

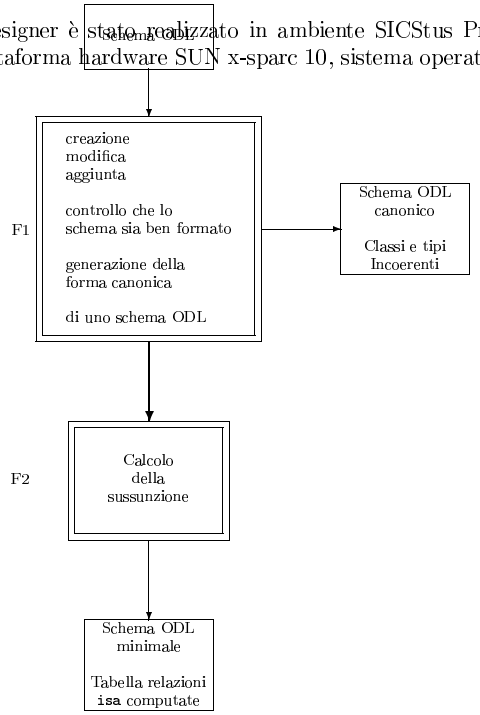



Figura 5.1: Architettura funzionale di ODL-Designer

In figura 5.1 riportiamo l'architettura funzionale di ODL-Designer. Come si può vedere il programma è diviso in due sottocomponenti funzionali principali che corrispondono a due fasi distinte che possono essere attivate separatamente: il primo, denotato con F1, è quello che permette, attraverso l'interfaccia utente, la creazione e la trasformazione di uno schema ODL, controlla la correttezza dello schema da un punto di vista sintattico e semantico, controlla l'aciclicità delle relazioni `isa` e dei tipi e, nel caso in cui lo schema risulta corretto, genera la forma canonica dello schema ODL.
Il secondo componente, denotato con F2, è quello che, partendo dallo schema ODL canonico, calcola le relazioni `isa` che intercorrono fra i tipi (classi) e, successivamente, determina qual' è lo schema ODL minimale.

La figura 5.2 rappresenta il procedimento di creazione di uno schema ODL, dettagliando maggiormente le attivitá dei sottocomponenti F1 ed F2. Per ottenere la creazione di uno schema l'utente deve, attraverso l'interfaccia utente, inserire direttamente la descrizione e/o indicare il file nel quale questa descrizione è presente. Lo schema in input viene quindi analizzato da un punto di vista sintattico e semantico. Se supera positivamente questo controllo viene verificata l'assenza di cicli nel grafo delle relazioni `isa` generato durante l'analisi dello schema inserito e l'assenza di tipi valore ciclici. Se dopo tale controllo lo schema risulta essere ben formato allora si procede alla generazione dello schema canonico per rilevare le eventuali classi o tipi incoerenti. La fase F1 risulta a questo punto terminata. In questa fase è prevista una interazione con l'utente per ogni eventuale errore rilevato.
Se viene richiesta anche l'attivazione della fase F2 lo schema canonico diventa l'input per il ragionatore tassonomico che genera lo schema minimale.

In modo analogo in figura 5.3 è riportata la procedura per aggiungere classi e tipi a uno schema ODL preesistente: essa coincide con quella di creazione fino all'analisi sintattica e semantica. A questo punto, infatti, il programma, per verificare l'assenza di cicli `isa` e di tipi ciclici, deve considerare sia i grafi generati dall'esecuzione corrente sia i grafi generati durante la creazione dello schema cui si stanno aggiungendo classi e tipi. Successivamente, se lo schema globale risulta essere ben formato il programma procede alla generazione della forma canonica delle classi e dei tipi di nuovo inserimento prendendo in input anche lo schema canonico preesistente. Vengono cosí determinate le classi e i tipi incoerenti nonché lo schema canonico globale.
Ancora, se scegliamo di eseguire anche la fase F2, lo schema canonico globale diventa l'input per il ragionatore tassonomico, come nella fase di creazione. Questo tipo di procedimento è consentito dal fatto che si presume una cresci-



Figura 5.2: ODL-Designer: Funzione CREA

Figura 5.3: ODL-Designer: Funzione AGGIUNGI



ta strettamente monotona della conoscenza che implica, quindi, il mantenimento della coerenza dello schema preesistente. Si ottengono ugualmente lo schema ODL minimale e il computo delle relazioni ISA a partire da schemi in cui si aggiunge ex-novo una o più classi.

## 5.2 Architettura e funzionalitá di GES

GES è un componente software che realizza l'espansione semnatica di un tipo. L'ambiente di sviluppo scelto per GES è il linguaggio di programmazione C, versione standard ANSI C, grazie alla portabilità del codice assicurata dal linguaggio e alla diffusione del C tra i DBMS (*Data Base Management System*) commerciali. La piattaforma hardware utilizzata è una SUN x-sparc 10, sistema operativo sunOS, release 4.1.3. L'editor utilizzato è EMACS e il compilatore utilizzato è GCC (GNU C Compiler), che sono prodotti free software della GNU (Free Software Foundation, Inc.).

La scelta per GES di un linguaggio (il C) differente da quello utilizzato da ODL-Designer (il prolog) esige la definizione di interfacce di comunicazione tra i due programmi per garantire loro la compatibilità: abbiamo perciò definito due file tipo testo di interfaccia (`ges_in.txt` e `ges_out.pl`) per lo scambio di dati validi e consistenti tra GES e ODL-Designer. La descrizione delle specifiche di tali file viene rimandata alla sezione 5.5.

Gli ostacoli tecnici per l'esecuzione sequenziale di programmi scritti nei linguaggi C e Prolog sono stati superati tecnicamente come verrá descritto in sezione 5.3.

Il programma GES si compone essenzialmente delle due funzioni denotate in figura 5.4 come F3 ed F4:
F3 legge il file `ges_in.txt`, generato da ODL-Designer, che contiene la descrizione della query e dei tipi che la compongono ad ogni livello di innestamento, gli insiemi GS relativi a tali tipi (classi) ed i tipi che definiscono i conseguenti di ciascuna regola (ad ogni livello di innestamento); controlla la correttezza sintattica di ciascuna descrizione e la trasforma in strutture a lista collegata che sono le strutture dati piú idonee al linguaggio C di implementazione del componente GES.
Se l'interrogazione non è incoerente, si prosegue eseguendo F4, che ne calcola l'espansione semantica in base all'algoritmo di Espansione Semantica di un Tipo ( tabella 4.2): per ciascun tipo $S$ che compare nella descrizione dell'interrogazione ad ogni livello di innestamento viene controllata la condizione di

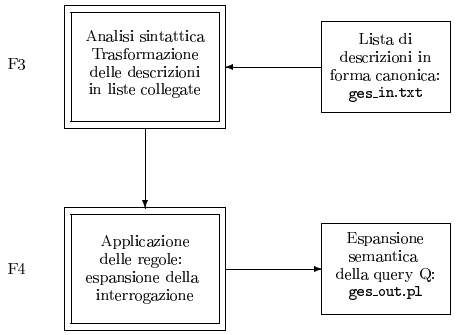



Figura 5.4: Architettura funzionale di GES

applicabilità delle regole ($S_k^a \in GS(S), S_k^c \notin GS(S)$) e, in tal caso, la regola $R_k$ viene applicata al tipo $S$ attraverso la congiunzione con il conseguente $S_k^c$ ($S \sqcap S_k^c$).

Il risultato di F4 viene memorizzato nel file `ges_out.pl`, il quale realizza l'interfaccia verso l'ambiente ODL-Designer comunicando il risultato dell'espansione semantica. Il file `ges_out.pl` contiene la descrizione, nel linguaggio ODL-Designer, dell'interrogazione comprensiva delle eventuali regole applicate.

Infine, GES termina restituendo un parametro di tipo intero che puó assumere uno dei seguenti valori:

**0 :** quando l'esecuzione è avvenuta correttamente ed è stata applicata almeno una regola.

**1 :** quando l'esecuzione è avvenuta correttamente ma non è stata applicata nessuna regola.

**2 :** quando l'esecuzione non è avvenuta correttamente a causa di errori di sintassi nelle descrizioni del file `ges_in.txt`



**3 :** quando l'esecuzione non è avvenuta correttamente a causa di errori semantici nelle descrizioni del file `ges_in.txt`(ad esempio l'assenza di una descrizione richiesta)

**4 :** quando l'esecuzione è avvenuta correttamente e l'interrogazione è stata rilevata incoerente.

Quando GES si interfaccia con ODL-Designer la sintassi e la semantica delle descrizioni contenute nel file `ges_in.txt` sono sicuramente corrette (essendo già state controllate da ODL-Designer) e quindi non avremo mai il caso di parametro di ritorno uguale a 2 o 3. Questi controlli sintattici e semantici sono stati introdotti in GES per garantire la robustezza del componente rispetto ad eventuali interfacce con componenti che non prevedono tali controlli e che quindi possono generare dati non corretti.

## 5.3 Architettura e funzionalitá di ODB-QOptimizer

ODB-QOptimizer è un software complesso per l'ottimizzazione semantica delle interrogazioni che, partendo da una generica interrogazione espressa in linguaggio ODL, ne calcola l'espansione semantica in forma canonica. Tale descrizione è la piú specifica tra quelle semanticamente equivalenti, contenendo le restrizioni fornite dai vincoli di integrità specificati nello schema ODL.

Naturalmente ha senso parlare di interrogazioni di una base di dati solamente quando esiste lo schema che descrive la realtà da rappresentare e l'istanza della base di dati. D'altra parte poichè l'ottimizzazione semantica di interrogazioni viene effettuata a livello intensionale, prerequisito indispensabile è l'esistenza dello schema con regole. Questa operazione viene realizzata da ODL-Designer in funzione CREA, opportunamente modificato in modo da riuscire a gestire anche le regole (come verrá descritto in sezione 5.4): l'esecuzione di ODL-Designer permette inoltre il controllo preliminare sulla coerenza dello schema (comprese le regole) e la generazione dello schema canonico.

Ora possiamo schematizzare l'ottimizzazione semantica di una interrogazione come illustrato in figura 5.5. L'interfaccia utente permette di aggiungere una interrogazione nello schema ODL preesistente: questa operazione viene

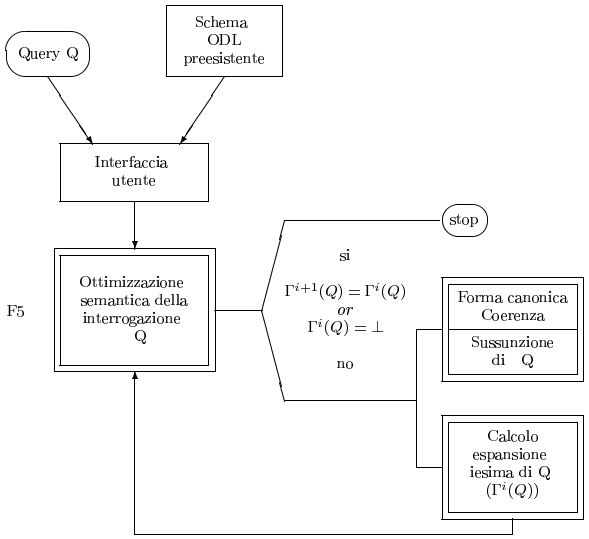



Figura 5.5: Architettura funzionale di ODB-QOptimizer



compiuta da ODL-Designer in modalità AGGIUNGI La funzione F5 calcola l'ottimizzazione semantica dell'interrogazione attraverso l'iterazione delle operazioni necessarie fino al raggiungimento della condizione di terminazione: essa coincide con il punto fisso della funzione di espansione semantica in cui il risultato dell'ottimizzazione è quello definitivo oppure con il rilevamento di incoerenza dell'interrogazione via via trasformata.

Il primo blocco di operazioni specificate dalla F5 (cioè il calcolo dello schema canonico, di incoerenza e sussunzione di una query Q) viene eseguito dal programma ODL-Designer in modalità AGGIUNGI attivando l'esecuzione di entrambe la fasi F1 ed F2, mentre la parte riguardante l'espansione semantica viene realizzata dal componente GES.

La funzione F5 si arresta quando si raggiunge il punto fisso di , $(Q)$ (quando cioè non si possono più applicare regole), oppure quando si rileva l'incoerenza dell'interrogazione .

Il controllo sulle condizioni di terminazione della funzione F5 è ottenuto controllando l'intero di ritorno di GES:

- Se l'intero vale 0 significa che non abbiamo ancora terminato e che è nuovamente necessario iterare la funzione F5, leggendo il file `ges_in.txt` in funzione AGGIUNGI, calcolando il nuovo schema canonico, la classificazione e chiamando nuovamente GES.

- Se l'intero vale 1 significa che abbiamo terminato l'ottimizzazione correttamente e l'interrogazione ottimizzata è presente in forma canonica.

- Se l'intero vale 4 significa che abbiamo terminato l'ottimizzazione avendo riscontrato l'incoerenza dell'interrogazione.

- Altrimenti significa che l'ottimizzazione viene interrotta a causa di errori.

Poichè sono da escludere errori sintattici, al termine dell'ottimizzazione abbiamo quindi disponibile in memoria l'espansione semantica dell'interrogazione in forma canonica, oppure sappiamo che l'interrogazione è risultata incoerente.

Rimane da definire l'ambiente di sviluppo del software che realizza la funzione F5 di chiamata sequenziale di ODL-Designer e GES e di controllo di terminazione: tra gli ambienti di programmazione SICStus Prolog e C

esiste la possibilità di interfacciamento che consente l'inclusione di funzioni descritte in un linguaggio all'interno di programmi scritti nell'altro. Dopo un'attenta analisi del problema e del software esistente abbiamo scelto di realizzare la funzione F5 in ambiente Prolog in modo tale che lo schema canonico e le relazioni `isa` computati da ODL-Designer nei vari passi dell'espansione semantica siano mantenuti in memoria evitando in tal modo, il costo di caricamento di tutta la base di dati ad ogni iterazione.

## 5.4 Nuova versione di ODL-Designer

Nella stesura e nella realizzazione di ODL-Designer l'obiettivo era quello di descrivere uno schema $\sigma$ su $\mathbf{S}(\mathbf{A}, \mathbf{B}, \mathbf{N})$ nel quale sono presenti nome di tipi-classe e di tipi-valore. Per tener conto della semantica differente delle classi (distinte in primitive e virtuali) ed dei tipi-valore, si introducono differenziazioni sintattiche miranti a distinguere i vari casi. Vediamo un esempio per ciascuna descrizione di tipo nella sintassi ODL-Designer:

classe primitiva:
s([prim,manager,=,/,\,'[',name,:,string,',', salary,:,integer,',',
level,:,integer,']'])

classe virtuale:
s([virt,tmanager,=,manager,&, /,\,'[',level,:,8,-,12,']']),

tipo-valore:
s([type,material,=,'[',name,:,string,',', risk,:, integer,']'])
s([type,device,=,'{',feature,:,string,'}'])
s([type,boat_contents,=,'<',container,:,string,'>'])

valore di tipo base:
s([b_type,b1,=,integer]);
s([b_type,level,=,5,-,15]);
s([b_type,b2,=,30,-]);



Nel progetto di ODB-QOptimizer le categorie sintattiche di ODL-Designer vengono integrate per permettere l'inserimento dello schema con vincoli di integritá e dell'interrogazione.

### 5.4.1 Inserimento dello schema con regole

Volendo descrivere uno schema con regole $(\sigma, \mathbf{R})$ in ambiente ODL-Designer introduciamo nuove strutture sintattiche che garantiscono l'appropriata semantica delle regole.

Nella sezione 4.1 abbiamo denotato una regola come una coppia antecedente e conseguente $R = (S^a, S^c)$ in cui sia $S^a$ che $S^c$ sono tipi di $\mathbf{S}(\mathbf{A}, \mathbf{B}, \mathbf{N})$: in modo piú specifico nello sviluppo dell'ottimizzatore considereremo che i tipi $S^a$ e $S^c$ possano essere solo classi virtuali e tipi-valore, in quanto possiamo escludere le classi primitive (poichè non possiedono la semantica delle regole) ed i tipi-base (poichè di nessuna rilevanza concreta) senza perdere in generalità.

Introduciamo cosí quattro nuove tipologie nella sintassi ODL-Designer che descrivono le regole di integrità:

`antev:` antecedente di una regola di tipo classe virtuale.

`antet:` antecedente di una regola di tipo valore.

`consv:` conseguente di una regola di tipo classe virtuale.

`const:` conseguente di una regola di tipo valore.

ODL-Designer interpreta i tipi che descrivono una regola come classi virtuali quando la tipologia è `antev` o `consv` mentre li interpreta come tipi-valore quando la tipologia è `antet` o `const`. Inoltre, per le regole di tipo virtuale che hanno l'antecedente associato ad una classe della base di dati, questa associazione è vera anche per il conseguente, sebbene nella notazione introdotta essa non sia direttamente esplicitata: per mantenere questa semantica, quando introduciamo in ODL-Designer conseguenti di tipo virtuale (tipo consv) occorre inserire esplicitamente tutte le eventuali classi associate all'antecedente.

Per meglio illustrare le nuove strutture sintattiche introdotte in ODL-Designer riportiamo la descrizione delle regole dello schema di tabella 4.1:

dapprima occorre tradurre le regole espresse secondo la notazione dei tipi-cammino nella tradizionale notazione ODL per poi scrivere nel file di input di ODL-Designer la descrizione secondo l'appropriata sintassi.

```
aggiungi(s([antev,1̊a,=,manager,&,
                    /,\,'[',level,:,5,-,10,']']),FLAG),
aggiungi(s([consv,1̊c,=,manager,&,
                    /,\,'[',salary,:,40000,-,60000,']']),FLAG),
aggiungi(s([antev,2̊a,=,material,&,
                    /,\,'[',risk,:,10,-,']']),FLAG),
aggiungi(s([consv,2̊c,=,material,&,smaterial]),FLAG),
aggiungi(s([antev,3̊a,=,storage,&,
                    /,\,'[',category,:,"B4",']']),FLAG),
aggiungi(s([consv,3̊c,=,storage,&,
                    /,\,'[',managed-by,:,tmanager,']']),FLAG),
aggiungi(s([antev,4̊a,=,storage,&,
                    /,\,'[',stock,:,'{','[',item,:,smaterial,']','}',
                        ']']), FLAG),
aggiungi(s([consv,4̊c,=,storage,&,sstorage]),FLAG),
aggiungi(s([antev,5̊a,=,storage,&,
                    /,\,'[',stock,:,'{','[',qty,:,10,-,50,']','}',
                        ']']), FLAG),
aggiungi(s([consv,5̊c,=,storage,&,
                    /,\,'[',category,:,"A2",']']),FLAG),
```

### 5.4.2 Inserimento di una interrogazione Q

Infine (vedi figura 5.5) commentiamo l'operazione di inserimento di un'interrogazione nello schema preesistente: come abbiamo spiegato nella sezione 4.3 la semantica di una interrogazione corrisponde a quella di una classe virtuale, quindi l'inserimento in ambiente ODL-Designer consiste nella trasformazione della query dalla notazione dei tipi-cammino alla tradizionale sintassi di una descrizione $\sigma_{\mathtt{v}}$ in ODL, di denotarla come classe virtuale e di eseguire ODL-Designer in modalità AGGIUNGI.

Ad esempio l'interrogazione $Q_1$ (vista a pag. 44):

$$\sigma_{\mathtt{v}}(Q_1) \quad = \quad \mathtt{storage} \sqcap (\triangle.\mathtt{stock}.\forall.\mathtt{item}.\ \triangle.\mathtt{risk}{:}\, 15 \div \infty)$$



viene trasformata nella classe virtuale

$$\sigma_{\mathtt{v}}(Q_1) \quad = \quad \mathtt{storage} \sqcap (\triangle[\mathtt{stock} : \{[\mathtt{item} : \triangle[\mathtt{risk}{:}\, 15 \div \infty)]]\}]$$

che viene poi inserita attraverso l'interfaccia utente secondo la sintassi dell'ambiente ODL-Designer:

```
s([virt,query,=,storage,&,
                    /,\,'[',stock,:,'{','[',item,:,/,\,'[', risk,:,15,-,']',']','}',
                        ']']),
```

## 5.5 Interazione tra i componenti ODL-Designer e GES

Come anticipato, abbiamo definito due diversi file di comunicazione tra gli ambienti di lavoro ODL-Designer e GES: essi contengono, secondo una predefinita sintassi, le descrizioni dei tipi dello schema necessari per compiere l'espansione semantica.

### 5.5.1 Interfaccia ODL-Designer → GES

Il primo file di comunicazione che descriviamo è il file `ges_in.txt` che viene generato da ODL-Designer ed è assunto da GES come il file contenente i dati di ingresso. Come abbiamo descritto nell'algoritmo di pag. 52, GES, per il calcolo dell'espansione semantica, necessita della descrizione in forma canonica dei tipi $S \in \mathbf{C}^+(Q)$ e, per ciascuno di questi tipi S, l'insieme delle generalizzazioni $GS(S)$; inoltre occorrono le descrizioni dei tipi $S \in \mathbf{C}^+(S_k^c)$ di ciascuna regola $R_k$ inserita nello schema $(\sigma, \mathbf{R})$ per poterle applicare all'interrogazione $Q$.

Il linguaggio utilizzato per il file `ges_in.txt` è un sottoinsieme di quello definito in [Bal92], poichè sono considerate solo le descrizioni della forma canonica, al quale viene aggiunta la sintassi dell'insieme $GS(S)$.

#### Sintassi dell'interfaccia ODL-Designer→ GES

Il risultato è la seguente grammatica:

$$
\begin{array}{rcl}
< \text{linguaggio} > & ::= & < \text{def-term}_1 > \cdots < \text{def-term}_n > \\
< \text{def-term} > & ::= & < \text{def-tipovalore} > \ | \\
 & & < \text{def-classe} > \ | \\
 & & < \text{def-gs-set} > \\
< \text{def-tipovalore} > & ::= & < \text{def-tipobase} > \ | \\
 & & < \text{def-tipo} > \\
< \text{def-classe} > & ::= & < \text{def-classe-prim} > \ | \\
 & & < \text{def-classe-vir} > \\
< \text{def-tipo} > & ::= & < \text{flag-tipo} > < \textit{nome tipovalore} > \\
 & & = < \text{tipo} > \\
< \text{flag-tipo} > & ::= & \textit{type} \ | \\
 & & \textit{antet} \ | \\
 & & \textit{const} \\
< \text{def-classe-prim} > & ::= & < \text{flag-classe-prim} > < \textit{nome classe} > \\
 & & \leq < \text{classe} > \\
< \text{flag-classe-prim} > & ::= & \textit{prim} \\
< \text{def-classe-vir} > & ::= & < \text{flag-classe-virt} > < \textit{nome classe} > \\
 & & = < \text{classe} > \\
< \text{flag-classe-virt} > & ::= & \textit{virt} \ | \\
 & & \textit{antev} \ | \\
 & & \textit{consv} \\
< \text{tipo} > & ::= & < \text{insiemi-di-tipi} > \ | \\
 & & < \text{sequenze-di-tipi} > \ | \\
 & & < \text{ennuple} > \\
< \text{classe} > & ::= & \triangle < \text{tipo} > \ | \\
 & & < \text{nomi-bar} > \sqcap \triangle < \text{tipo} > \\
 & & \\
< \text{insiemi-di-tipi} > & ::= & \left\{ < \text{tipo} > \right\} \\
< \text{sequenze-di-tipi} > & ::= & \langle < \text{tipo} > \rangle \\
< \text{ennuple} > & ::= & \left[ < \text{attributi} > \right]
\end{array}
$$



$$
\begin{array}{rcl}
< \text{attributi} > & ::= & < \textit{nome attributo} > : < \text{nome} > \ | \\
 & & < \textit{nome attributo} > : < \text{nome} > , < \text{attributi} > \\
< \text{nome} > & ::= & < \text{nome-di-tipi} > \\
 & & \textit{real} \ | \\
 & & \textit{integer} \ | \\
 & & \textit{string} \ | \\
 & & \textit{boolean} \\
< \text{nomi-di-tipi} > & ::= & < \textit{nome classe} > \ | \\
 & & < \textit{nome tipovalore} > \ | \\
 & & < \textit{nome tipobase} > \\
< \text{nomi-bar} > & ::= & - < \text{nome classe} > \ | \\
 & & - < \text{nome classe} > \sqcap < \textit{nomi-bar} > \\
 & & \\
< \text{def-tipobase} > & ::= & < \text{flag-tipobase} > < \textit{nome tipobase} > \\
 & & = < \text{tipobase} > \\
< \text{flag-tipobase} > & ::= & \textit{b\_type} \\
< \text{tipobase} > & ::= & \textit{real} \ | \\
 & & \textit{integer} \ | \\
 & & \textit{string} \ | \\
 & & \textit{boolean} \ | \\
 & & < \text{range-intero} > \ | \\
 & & < \textit{valore reale} > \ | \\
 & & < \textit{valore intero} > \ | \\
 & & < \textit{valore string} > \ | \\
 & & < \textit{valore boolean} > \\
< \text{range-intero} >^{1} & ::= & < \textit{valore intero} > \ - \textit{max} \ | \\
 & & \textit{min} - < \textit{valore intero} > \ | \\
 & & < \textit{valore intero} > \ - \ < \textit{valore intero} > \\
 & & \\
< \text{def-gs-set} > & ::= & < \text{flag-gs} > < \textit{nome-di-tipi} > < \text{gs-set} >
\end{array}
$$

$$
\begin{array}{rcl}
< \text{flag-gs-set} > & ::= & gs \\
< \text{gs-set} > & ::= & (< \text{nomi-di-tipi} > < \text{nome-flag} >) \ | \\
 & & (< \text{nomi-di-tipi} > < \text{nome-flag} >) \ < \text{gs-set} > \\
< \text{nome-flag} > & ::= & < \text{flag-classe-prim} > \ | \\
 & & < \text{flag-classe-virt} > \ | \\
 & & < \text{flag-tipo} >
\end{array}
$$

Alla grammatica ora definita occorre aggiungere alcune convenzioni per permettere una completa decodifica delle descrizioni di tipo nella definizione del file `ges_in.txt`:

- ciascuna descrizione è delimitata dai caratteri '[' e ']',
- ogni elemento terminale è separato dal successivo dalla virgola (','),
- l'elemento terminale $\sqcap$ viene sostituito dal carattere &,
- l'elemento terminale $\triangle$ viene sostituito dai caratteri /\,
- la prima descrizione del file è quella della interrogazione da ottimizzare,
- il file termina con il punto fermo ('.').

Come esempio riportiamo il file `ges_in.txt` generato da ODL-Designer per ottimizzare la $Q_1$ di pag. 44, a partire dallo schema canonico:

**ges_in.txt**

```
[virt,query,=,-storage,&,/,\,t23]
\*
      Le descrizioni che seguono sono quelle dei tipi che
      sono contenuti ricorsivamente nella query.
*\
[type,t23,=,[,category,:,string,,,stock,:,t22,,,managed-by,:,manager,]]
[type,t22,=,{,t21,}]
```

[1] Quando si definisce un intervallo è possibile definire, nell'ordine, solo il limite inferiore, quello superiore o entrambi i limiti.



```
[type,t21,=,[,item,:,n2,,,qty,:,b4,]]
[virt,n2,=,-material,&,/,\,t20]
[type,t20,=,[,feature,:,t1,,,name,:,string,,,risk,:,b11,]]
[type,t1,=,{,string,}]
[virt,manager,=,-manager,&,/,\,t3]
[type,t3,=,[,level,:,b1,,,name,:,string,,,salary,:, b2,]]
\*
      Riportiamo ora le descrizioni degli insiemi GS dei tipi
      precedentemente descritti.
*\
[gs,query,(,storage,prim,)]
[gs,t23,(,t7,type,)]
[gs,t22,(,t6,type,)]
[gs,t21,(,t5,type,)]
[gs,n2,(,material,prim,),(,2a,antev,)]
[gs,t20,(,t2,type,),(,t10,type,)]
[gs,t1]
[gs,manager]
[gs,t3]
\*
    Riportiamo infine le descrizioni dei conseguenti di tutte
    le regole (compresi i tipi che compaiono nelle loro descrizioni),
    necessari poich      e potenzialmente si possono applicare tutte le regole.
*\
[consv,1c,=,-manager,&,/,\,t9]
[type,t9,=,[,level,:,b1,,,name,:,string,,,salary,:,b2,]]
[consv,2c,=,-material,&,-smaterial,&,/,\,t2]
[type,t2,=,[,feature,:,t1,,,name,:,string,,,risk,:,integer,]]
[consv,3c,=,-storage,&,/,\,t12]
[type,t12,=,[,category,:,string,,,stock,:,t6,,,managed-by,:,n1,]]
[type,t6,=,{,t5,}]
[type,t5,=,[,item,:,material,,,qty,:,b4,]]
[prim,material,=,-material,&,/,\,t2]
[virt,n1,=,-manager,&,-tmanager,&,/,\,t4]
[type,t4,=,[,level,:,b3,,,name,:,string,,,salary,:,b2,]]
[consv,4c,=,-storage,&,-sstorage,&,/,\,t7]
[type,t7,=,[,category,:,string,,,stock,:,t6,,,managed-by,:,manager,]]
```

[consv,$\mathring{5}$c,=,-storage,&,/,\,t19]
[type,t19,=,[,category,:,b10,,,name,:,stock,:,t6,,,managed-by,:,manager,]]
.

### 5.5.2 Interfaccia GES → ODL-Designer

Il file di interfaccia tra GES ed ODL-Designer è denominato `ges_out.pl` e viene generato da GES al termine del calcolo dell'espansione semantica dell'interrogazione. Esso contiene la nuova descrizione dell' interrogazione comprensiva delle eventuali congiunzioni con i conseguenti delle regole che sono state applicate: la sintassi è quella di ODL-Designer in modalità di funzionamento AGGIUNGI.

**ges_out.pl**

In particolare avremo la seguente sintassi:

leggiinput(FLAG) :-
    aggiungi(s("Descrizione dell'interrogazione"),FLAG),
    aggiungi(fine,FLAG).

dove:

**"Descrizione dell'interrogazione":** segue le stesse regole della grammatica presentata in sezione 5.5.1, con l'unica differenza che gli elementi terminali [, ], {, }, <, > e , sono racchiusi tra singoli apici (cioè $'[', ']', '\{', '\}', '<', '>'$ e $','$).

Nel caso in cui GES non applichi nessuna regola viene ugualmente generato il file `ges_out.pl` privo di descrizioni che consiste di:

leggiinput(FLAG) :-
    aggiungi(fine,FLAG).

Infine se abbiamo rilevato errori sintattici o di coerenza il file `ges_out.pl` non contiene alcun carattere.

Ad ogni iterazione del calcolo di , $^i(Q)$ si genera una nuova interrogazione che è logicamente implicata dalla precedente ma rappresenta una classe



virtuale distinta da quest'ultima. Il componente GES genera allora una nuova classe virtuale che rappresenta l'espansione semantica dell'interrogazione che ha la radice del nome identica a quella dell'interrogazione presente in `ges_in.txt` ed apice incrementato di una unità. In questo modo ODL-Designer riconosce l'interrogazione come una nuova classe virtuale da inserire nello schema e classificare secondo l'opportuna tassonomia.
Ad esempio se l'interrogazione ha nome `query` allora inizialmente in `ges_in.txt` abbiamo, tra le altre, la descrizione della classe `query` mentre in `ges_out.pl` riportiamo la descrizione della classe $\texttt{query}_1$; nelle successive iterazioni vengono generate le descrizioni delle classi $\texttt{query}_2$, $\texttt{query}_3 \cdots \texttt{query}_n$.

La descrizione dell'interrogazione riportata nel file `ges_out.pl` non contiene nomi di classi o di tipi-valore creati da ODL-Designer durante la generazione dello schema canonico: infatti la descrizione dell'interrogazione da inserire in `ges_out.pl` è ottenuta partendo dalla nuova descrizione (comprensiva delle eventuali regole applicate) e sostituendo a ciascuna classe e ciascun tipo-virtuale la propria descrizione. La ragione di questo comportamento è dovuta al fatto che quando si aggiunge una nuova descrizione in ODL-Designer essa non può avere riferimenti a nomi di classi o tipi-valore creati precedentemente da ODL-Designer (che ora possono essere mutati in ragione dell'applicazione delle regole in modo da non risultare più nella descrizione canonica). Infatti nel calcolo del nuovo schema canonico, ODL-Designer, per le classi che referenziano tali tipi, non può riconoscere quali referenziano la descrizione dopo la congiunzione della regola e quali invece la descrizione originale, con conseguente perdita di consistenza dello schema.

Ad esempio se l'interrogazione prima dell'espansione semantica era descritta dal seguente schema:

$$\begin{aligned} \nu(Q) &= \overline{\texttt{manager}} \sqcap \triangle t_1 \\ \nu(t_1) &= [\texttt{name} : \texttt{string}, \texttt{level} : \texttt{integer}] \end{aligned}$$

e per effetto dell'applicazione di una regola il tipo $t_1$ ha l'attributo `level` ristretto all'intervallo 10-30 allora ODL-Designer avrebbe in ingresso il tipo $t_1$ (mutato per applicazione di una regola) senza però sapere quali delle classi che referenziavano quel tipo ora effettivamente vogliono la restrizione di $t_1$ oppure la descrizione iniziale.

Scrivendo invece nel file `ges_out.pl` l'intera descrizione dell'interrogazione senza riferirsi a classi o tipi-valore ma inserendo direttamente lo schema di tali tipi non vi è nessuna ambiguità nel calcolo del nuovo schema canonico.

## 5.6 Esecuzione di ODB-QOptimizer

Siamo ora in grado di descrivere in maniera completa il procedimento eseguito per ottenere l'ottimizzazione dell'interrogazione: l'operazione preliminare di immissione dello schema con regole è del tutto analoga all'esecuzione di ODL-Designer in modalità CREA (descritta in figura 5.2), con la convenzione che ora ODL-Designer è stato modificato secondo quanto visto in sezione 5.4. Quindi in figura 5.6 si parte da uno schema con regole al quale viene aggiunta l'interrogazione da ottimizzare attraverso l'interfaccia utente, inserendo direttamente la descrizione e/o indicando il file nel quale questa descrizione è presente. ODB-QOptimizer esegue l'operazione attivando ODL-Designer in modalità AGGIUNGI (con funzionalità F1 ed F2) che poi controlla la sintassi e la semantica del nuovo schema, rilevando eventuali cicli, generando lo schema canonico, la classificazione e il file di interfaccia `ges_in.txt`. Per compiere queste operazioni ODL-Designer accede allo schema e alla classificazione delle `isa` create in precedenza.
Se queste operazioni hanno fornito esito positivo ODB-QOptimizer fa partire l'esecuzione di GES il quale analizza la sintassi delle descrizioni del file di input `ges_in.txt`, controlla la coerenza della query e verifica che siano effettivamente contenute le descrizioni necessarie e sufficienti per l'espansione semantica. Ancora, se questi controlli sono corretti si procede con il calcolo dell'espansione semantica dell'interrogazione che viene memorizzata nel file `ges_out.pl`.
Infine ODB-QOptimizer controlla la terminazione dell'ottimizzazione: se abbiamo applicato almeno una regola significa che non è stato raggiunto il punto fisso e quindi si procede iterativamente con l'esecuzione di ODL-Designer in modalità AGGIUNGI che legge il contenuto del file `ges_out.pl` come nuove classi da inserire nello schema. Se invece GES ha trovato classi incoerenti ci fermiamo sapendo che l'interrogazione fornisce esito negativo, mentre se non è possibile applicare ulteriori regole significa che l'ottimizzazione è terminata avendo reso disponibile la descrizione in forma canonica dell'interrogazione ottimizzata .



Figura 5.6: Funzionalitá di ODB-QOptimizer

## 5.7 GES: Generatore di Espansione Semantica di un tipo

Nella sezione viene descritto il componente GES nelle sue varie parti secondo la metodologia PHOS (Programming Hierarchically from Output Structure).

Brevemente, riassumiamo le principali regole PHOS per la rappresentazione dei programmi e delle procedure:

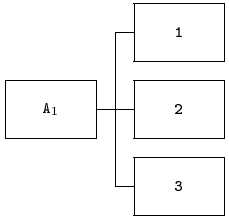



Figura 5.7: Esempio di chiamate in sequenza

- La figura 5.7 indica che la componente $A_1$ (ovvero la `procedura A1, indicata con questo tipo di carattere`, o il Modulo $A_1$) chiama in successione le componenti `1, 2, 3`

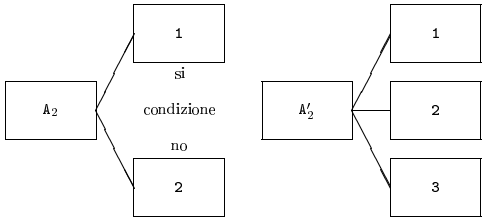



Figura 5.8: Esempio di chiamate in alternativa



- – nel primo caso della figura 5.8 la componente $A_2$ chiama la componente `1` se si verifica la condizione, altrimenti chiama la componente `2`
  – nel secondo caso della figura 5.8, invece, la componente $A'_2$ effettua chiamate in alternativa senza che la condizione venga indicata perché è implicita nel nome delle componenti chiamate

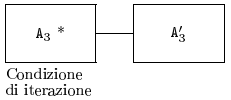



Figura 5.9: Esempio di iterazione

- La figura 5.9 indica che la componente $A_3$ itera su se stessa e ad ogni iterazione chiama la componente $A'_3$ (ad esempio, riceve in input una lista di elementi da passare singolarmente alla procedura chiamata e quindi dopo aver estratto il primo elemento, itera sul resto della lista finché questa non è vuota). A volte, per chiarezza, viene indicata anche la condizione di iterazione.

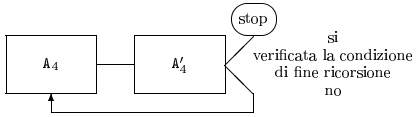



Figura 5.10: Esempio di ricorsione

- La figura 5.10 indica la ricorsione della componente $A'_4$ sulla componente $A_4$ nel momento in cui si sta eseguendo una procedura ricorsiva e la condizione di fine ricorsione non si è verificata. Non è indicativo, qui, il fatto che la ricorsione sia fra 2 elementi: come potremo vedere, infatti, in molti casi l'applicazione della procedura ricorsiva e' attuata da varie componenti e la ricorsione viene controllata da una componente a parte.

### 5.7.1 Programma principale: GES

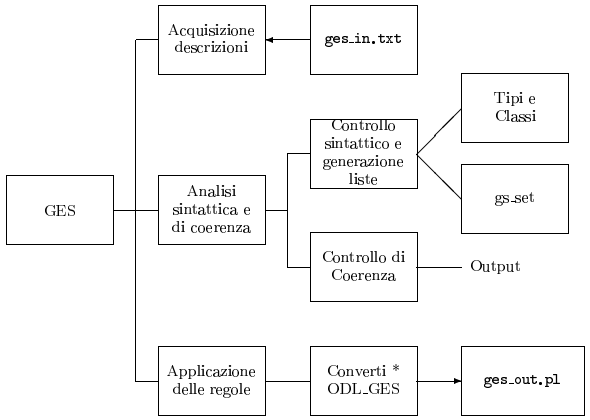



Figura 5.11: Struttura del programma GES

Il programma, la cui struttura è illustrata in figura 5.11, acquisisce la descrizione di tipi espressi in sintassi ODL da file di input (`ges_in.txt`), esegue l'analisi sintattica degli schemi, compie una trasformazione delle descrizioni in liste concatenate ed opera il controllo di coerenza dell'interrogazione.
L'analisi sintattica si pone come obiettivo la verifica della rappresentazione dei tipi nella corretta grammatica definita alla sottosezione 5.5.1 e il riconoscimento della categoria di appartenenza: da una parte i tipi e le classi e dall'altra i gs_set che verranno trasformati in due differenti tipologie di liste.

Se l'analisi sintattica e di coerenza hanno dato esito positivo, si passa all'applicazione delle regole: il procedimento consiste nel determinare le regole applicabili ed, iterativamente, congiungere tali regole alla descrizione dell'interrogazione da ottimizzare. Infine questa descrizione, che rappresenta l'espansione semantica dell'interrogazione, viene memorizzata nel file `ges_out.pl` utilizzato da ODL-Designer come input dei dati per l'esecuzione successiva.

Come detto, il programma viene chiamato senza parametri, poichè i dati di input sono memorizzati su file, mentre ritorna con valore intero che vale:



**0 :** quando l'esecuzione è avvenuta correttamente ed è stata applicata almeno una regola.

**1 :** quando l'esecuzione è avvenuta correttamente ma non è stata applicata nessuna regola.

**2 :** quando l'esecuzione non è avvenuta correttamente a causa di errori di sintassi nelle descrizioni del file `ges_in.txt`.

**3 :** quando l'esecuzione non è avvenuta correttamente poichè mancano delle descrizioni necessarie all'espansione nel file `ges_in.txt`.

**4 :** quando l'esecuzione è avvenuta correttamente ed è stata rilevata una classe incoerente.

Se l'esecuzione è terminata senza problemi viene visualizzata la lista di regole applicate, mentre nel caso in cui siano state riscontrate anomalie viene visualizzato un messaggio che ne specifica l'origine:

```
WARNING: esiste almeno un tipo incoerente,
non si procede con l'applicazione delle regole.

ERRORE : La sintassi della stringa n.' i ' e' scorretta:
'visualizzazione stringa errata i'
Non si procede con l'applicazione delle regole.

ERRORE: Nel file di input manca la descrizione
del tipo 'nome'.
L'esecuzione non e' stata completata correttamente.

ERRORE: Nel file di input manca la descrizione del
GS('nome').
L'esecuzione non e' stata completata correttamente.
```

I moduli del primo livello sono i seguenti:

Acquisizione descrizioni
Analisi sintattica e di coerenza
Applicazione delle regole

### 5.7.2 Modulo "Acquisizione descrizioni"

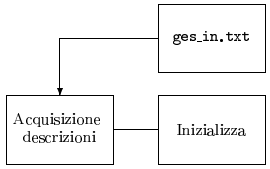



Figura 5.12: Modulo "Acquisizione descrizioni"

Il modulo "Acquisizione descrizioni" apre il file `ges_in.txt` e lo memorizza, riga dopo riga, in un array di stringhe per averlo a disposizione nelle successive operazioni.

#### Sottomodulo "Inizializza"

Il sottomodulo "Inizializza" è costituito dalla procedura `svuota_liste (void)`, la quale inizializza come vuote tutte le liste che rappresentano le descrizioni dei tipi e dei gs_set: queste liste sono puntate rispettivamente dagli array di puntatori *first_classe[i] e *first_gs[i], ciascuno dei quali punta ad una singola descrizione. Naturalmente il sottomodulo è chiamato senza parametri e non ne ha di ritorno.
Le procedure coinvolte nell'esecuzione sono:

`void togli_first_classe(int i)`: toglie il primo elemento della lista puntata da *first_classe[i]
`void togli_first_gs(int i)`: toglie il primo elemento della lista puntata da *first_gs[i]



### 5.7.3 Modulo "Analisi sintattica e di coerenza"

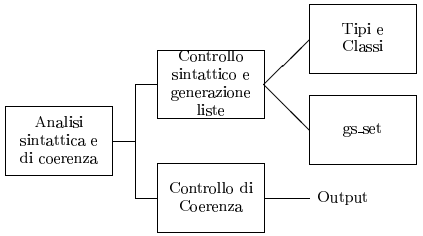



Figura 5.13: Modulo "Analisi sintattica e di coerenza"

#### Sottomodulo "Controllo sintattico e generazione liste"

Il sottomodulo "Controllo sintattico generazione liste" (figura 5.14) ha il compito di ispezionare tutte le descrizioni contenute in `ges_in.txt`, fare l'analisi sintattica e generare le corrispondenti liste.

Il sottomodulo è costituito dalla procedura `int analisi(void)`, la quale è chiamata senza parametri e ritorna con valore intero che vale:

**0 :** se la sintassi è corretta e quindi la conversione in liste ha dato risultato positivo.

**1 :** se è stato riscontrato un errore sintattico.

La procedura `analisi( )` chiama la procedura `int converti_stringa_ODL_GES(int i, int j)` per ciascuna descrizione di tipo e di gs_set contenuta nel file di input.

#### Sintassi delle LISTE create da GES

La sintassi corretta per le descrizioni dei tipi descritti in forma canonica e per i gs_set contenute nel file `ges_in.txt` è quella riportata nella sottosezione 5.5.1. A partire da quella grammatica definiamo una struttura dati a lista che ne mantenga la semantica; tali liste sono costituite da una sequenza di blocchi contenenti ciascuno i seguenti campi:

**NOME[LEN_WORD] :** array di caratteri (di lunghezza definita LEN_WORD) che descrive il nome dell'elemento considerato.

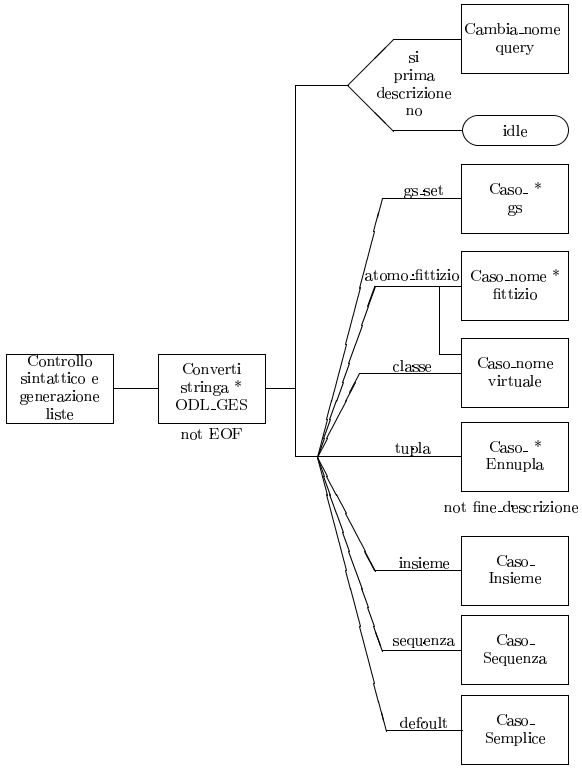



Figura 5.14: Sottomodulo "Controllo sintattico e generazione liste"



**TIPO[LEN_WORD] :** array di caratteri che indica il tipo considerato; puó essere: `N_Classe`, `Nome_Virtuale`, `Atomo_Fittizio`, `Attributo`, `Sequenza`, `Insieme` o `Semplice`.

**C_TIPO[LEN_WORD] :** array di caratteri che indica la tipologia nella sintassi ODL-Designer: `prim`, `virt`, `antev`, `consv`, `type`, `antet`, `const`, `b_type` e `gs`.

**D_ATTRIBUTO[LEN_WORD] :** array di caratteri che specifica il dominio di un attributo (quando viene convertita una tupla).

**NEXT :** puntatore al successivo elemento della lista.

Descriviamo ora (anche attraverso esempi) come si rappresentano con le liste collegate le possibili descrizioni di ODL-Designer.

### NOME del tipo o del gs_set

L'elemento in testa alla lista specifica sempre il nome (e il tipo) della descrizione che andiamo a trasformare. Il blocco costitutivo della struttura dati a lista collegata lo possiamo rappresentare nel seguente modo:

| | |
|---|---|
| NOME | char[ ] |
| TIPO | N_Classe |
| C_TIPO | char[ ] |
| D_ATTRIBUTO | |
| NEXT | $\longrightarrow$ |

dove

| | | |
|---|---|---|
| NOME | = | nome del tipo o del gs_set |
| TIPO | = | N_Classe |
| C_TIPO | $\in$ | {prim, virt, antev, consv, type, antet, const, b_type , gs} |
| D_ATTRIBUTO | | non rilevante |
| NEXT | | puntatore al prossimo elemento |

A partire dal secondo elemento si possono presentare le seguenti tipologie:

### ATOMO FITTIZIO

[prim,manager,=,-manager,$\cdots$ ]

| NOME | char[ ] |
|---|---|
| TIPO | Atomo_Fittizio |
| C_TIPO | |
| D_ATTRIBUTO | |
| NEXT | $\longrightarrow$ |

dove

NOME = Nome dell'atomo fittizio (nell'esempio -manager)

TIPO = Atomo_Fittizio

C_TIPO non rilevante

D_ATTRIBUTO non rilevante

NEXT puntatore al prossimo elemento

### CLASSE VIRTUALE

[virt,department,=,/,\,t1,$\cdots$ ]

| NOME | char[ ] |
|---|---|
| TIPO | Nome_Virtuale |
| C_TIPO | |
| D_ATTRIBUTO | |
| NEXT | $\longrightarrow$ |

dove

NOME = Nome della classe (nell'esempio t1)

TIPO = Nome_Virtuale

C_TIPO non rilevante

D_ATTRIBUTO non rilevante

NEXT puntatore al prossimo elemento



### TUPLA

[type,t1,=,'[',name,:,string,$\cdots$,']' ]

| NOME | char[ ] |
|---|---|
| TIPO | Attributo |
| C_TIPO | |
| D_ATTRIBUTO | char[ ] |
| NEXT | $\longrightarrow$ |

dove

NOME = Nome dell'attributo (nell'esempio name)

TIPO = Attributo

C_TIPO non rilevante

D_ATTRIBUTO Dominio dell'attributo (nell'esempio string)

NEXT puntatore al prossimo elemento

### INSIEME

[type,t2,=,'{',material,'}']

| NOME | char[ ] |
|---|---|
| TIPO | Insieme |
| C_TIPO | |
| D_ATTRIBUTO | |
| NEXT | $\longrightarrow$ |

dove

NOME = Nome del tipo degli elementi dell'insieme (nell'esempio material)

TIPO = Insieme

C_TIPO non rilevante

D_ATTRIBUTO non rilevante

NEXT puntatore al prossimo elemento

**SEQUENZA**

[type,t3,=,'<',string,'>']

| | |
|---|---|
| NOME | char[ ] |
| TIPO | Sequenza |
| C_TIPO | |
| D_ATTRIBUTO | |
| NEXT | $\longrightarrow$ |

dove

NOME = Nome del tipo degli elementi della sequenza (nell'esempio string)

TIPO = Sequenza

C_TIPO non rilevante

D_ATTRIBUTO non rilevante

NEXT puntatore al prossimo elemento

**SEMPLICE**

[b_type,t4,=,11,-,20]
[prim,sstorage,=,storage]

| | |
|---|---|
| NOME | char[ ] |
| TIPO | Semplice |
| C_TIPO | |
| D_ATTRIBUTO | |
| NEXT | $\longrightarrow$ |

dove

NOME = Nome del valore a destra del segno = (RVALUE) (nell'esempio 11,-,20 e storage rispettivamente)

TIPO = Semplice

C_TIPO non rilevante



D_ATTRIBUTO non rilevante

NEXT puntatore al prossimo elemento

**Sottomodulo "Converti_stringa_ODL_GES"**

Il sottomodulo "Converti_stringa_ODL_GES" svolge la funzione di verificare la sintassi di una singola descrizione e di costruire la la lista corrispondente secondo le regole definite alla sottosezione precedente.

La procedura è la `int converti_stringa_ODL_GES(int i, int j)` che ha due parametri di ingresso (`i` e `j`) ed uno di ritorno che sono degli interi.

`int i` : se viene convertito un tipo allora viene aggiornata la lista puntata da `*first_classe[i]`.

`int j` : se viene convertito un gs_set allora viene aggiornata la lista puntata da `*first_gs[j]`.

L'intero di ritorno vale:

**0 :** se la sintassi è corretta ed è stato convertito un tipo.

**1 :** se la sintassi è corretta ed è stato convertito un gs_set.

**2 :** se è stato riscontrato un errore sintattico.

La procedura `converti_stringa_ODL_GES(int i, int j)` controlla se sta analizzando la prima descrizione del file di input `ges_in.txt`: se questo è il caso allora assume di compiere l'analisi dell'interrogazione che comporta la trasformazione del nome. Questa trasformazione è quella prevista dall'interfaccia tra GES ed ODL-Designer: la radice del nome rimane invariata mentre il pedice viene incrementato di una unitá (ad esempio se il nome dell'interrogazione è $Q$ allora il nuovo nome sará $Q_1$, poi $Q_2$ e così via).

Dopo il controllo sulla descrizione dell'interrogazione, il sottomodulo inizia l'analisi sintattica e, a seconda della tipologia incontrata, vengono chiamate le appropriate procedure di riconoscimento sintattico e generazione di liste.
Le possibili procedure eseguite in alternativa sono:

`int caso_gs(int j)` : converte l'intera descrizione dei gs_set memorizzandola nella lista punatata da `*first_gs[j]`.
Ritorna:
1 se la sintassi è corretta,
2 altrimenti.

`int caso_nome_fittizio(int i)` : converte la descrizione degli atomi fittizi memorizzandoli nella lista puntata da `*first_classe[i]`.
Ritorna:
0 se la sintassi è corretta,
2 altrimenti.

`int caso_nome_virtuale(int i)` : converte la descrizione di una singola classe memorizzandola nella lista puntata da `*first_classe[i]`.
Ritorna:
0 se la sintassi è corretta,
2 altrimenti.

`int caso_ennupla(int i)` : converte la descrizione di una tupla memorizzandola nella lista puntata da `*first_classe[i]`.
Ritorna:
0 se la sintassi è corretta,
2 altrimenti.

`int caso_insieme(int i)` : converte la descrizione di un insieme memorizzandolo nella lista puntata da `*first_classe[i]`.
Ritorna:
0 se la sintassi è corretta,
2 altrimenti.

`int caso_sequenza(int i)` : converte la descrizione di una sequenza memorizzandola nella lista puntata da `*first_classe[i]`.
Ritorna:
0 se la sintassi è corretta,
2 altrimenti.

`int caso_tipo_semplice(int i)` : converte la descrizione di un singolo nome memorizzandolo nella lista puntata da `*first_classe[i]`.
Ritorna:



0 se la sintassi è corretta,
2 altrimenti.

Le procedure di supporto chiamate per eseguire le operazioni sono le seguenti:

`void aggiungi_lista_classe_gs (char NOME1[ ], char NOME2[ ], char NOME3[ ], char NOME4[ ], int i, int j)` :
esegue l'inserimento di un elemento in coda ad una delle liste specificate dai parametri di input secondo le condizioni:
se `j = 0` allora aggiungi nella lista puntata da `first_classe[i]`,
se `j = 1` allora aggiungi nella lista puntata da `first_gs[i]`
La modalità di definizione della struttura dati da inserire è la seguente:
`NOME1[ ]` definisce il `NOME`,
`NOME2[ ]` definisce il `D_ATTRIBUTO`,
`NOME3[ ]` definisce il `C_TIPO`,
`NOME4[ ]` definisce il `TIPO`.
Ovviamente non ha parametri di ritorno.

`void leggi_nome(char name[ ], int modo)` : legge il prossimo token (cioè una stringa di caratteri alfanumerici terminanti con una virgola) dal buffer di input e lo memorizza nella variabile indicata da `name[ ]`.
Vi sono due modalità di lettura:
con `modo = 0` scandisce dal prossimo carattere,
con `modo = 1` scandisce dal presente carattere
Ovviamente non ha parametri di ritorno.

`void leggi_stringa_ascii(char name[ ])` : legge una stringa di caratteri rappresentati in codice ASCII e la memorizza nella variabile indicata da `name[ ]`.

`void unisci_nome(char DATO1[ ], DATO2[ ])` : unisce le due stringhe contenute in `DATO1[ ]` e `DATO2[ ]` interponendo la sequenza di caratteri ",-,". Il risultato è memorizzato nella variabile globale `dato.NOME[ ]`.

`int confronta_nomi(char DATO1[ ], DATO2[ ])` : confronta le due stringhe contenute in `DATO1[ ]` e `DATO2[ ]` e ritorna:

0 se le stringhe coincidono,
1 altrimenti.

`void errore(void)` : scrive sullo standard output un messaggio di errore.

**Sottomodulo "Controllo coerenza"**

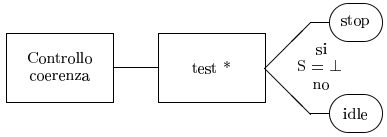



Figura 5.15: Sottomodulo "Controllo coerenza"

Per ciascuna descrizione viene controllata la coerenza: se il tipo è equivalente a $\perp$ (bottom) allora abbiamo incoerenza che comporta la fine del programma senza cercare di applicare le regole (figura 5.15).
Si utilizza la procedura `int converti_stringa_ODL_GES(int i, int j)` che ritorna con valore 4 se esiste incoerenza.

### 5.7.4 Modulo "Applicazione delle regole"

Il modulo "Applicazione delle regole" (figura 5.16) è realizzato dalla procedura `int applica_regole(void)` la quale viene chiamata senza parametri e genera l'espansione semantica dell'interrogazione ritornando un valore intero che vale:

**0 :** se viene applicata almeno una regola.

**1 :** se se non è possibile applicare ulteriori regole.

**3 :** se non sono disponibili (da file di input) tutte le descrizioni necessarie all'applicazione delle regole.

L'espansione semantica generata viene memorizzata nel file `ges_out.pl` che è scritto chiamando la procedura ricorsiva `int converti_ges_odl_tipo`



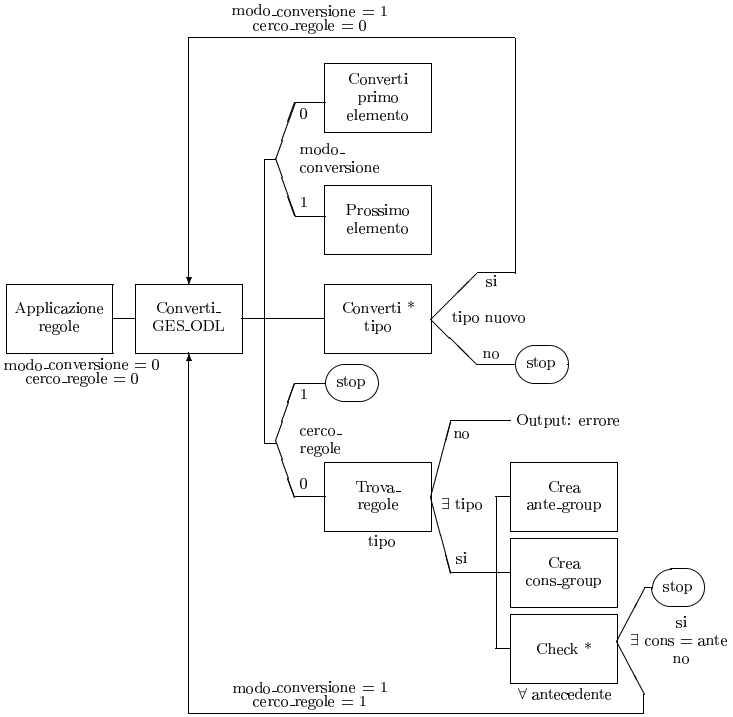



Figura 5.16: Modulo "Applicazione delle regole"

`(int modo_conversione, ELEMENTO *punta_classe, int cerco_regole)`
con i parametri:
`modo_conversione = 0`
`cerco_regole = 0`
`*punta_classe = interrogazione`

### Sottomodulo "Converti_GES_ODL"

Il sottomodulo "Converti_GES_ODL" è costituito dalla procedura ricorsiva `int converti_ges_odl_tipo (int modo_conversione, ELEMENTO *punta_classe, int cerco_regole)`, la quale inizialmente controlla il valore del parametro `modo_conversione`:

se vale 0 allora viene convertito nella sintassi ODL-Designer l'elemento in testa alla lista puntata dal parametro `*punta_classe` (la conversione è scritta sul file di output dalla procedura `scrivi_nome()`). Infine viene puntato il successivo elemento della lista.

se vale 1 allora non viene convertito l'elemento in testa alla lista e viene puntato il secondo elemento.

Successivamente inizia un procedimento iterativo su ciascun elemento della lista che possiamo così riassumere:

1. conversione (e scrittura su file) in sintassi ODL-Designer dell'attuale elemento puntato nella lista (viene chiamata la procedura `scrivinome()`).
2. controllo del nome del tipo di cui ci stiamo occupando:
   (a) se è un nome di un tipo la cui descrizione è contenuta nel file di input `ges_in.txt` allora richiama se stesso con parametri:
   `modo_conversione = 1`
   `cerco_regole = 0`
   `*punta_classe =` attuale tipo puntato.
   (b) altrimenti termina.



Terminata tale procedura viene fatto un controllo sul valore del parametro `cerco_regole`:

1. se vale 1 allora termina.
2. se vale 0 allora chiama la procedura `int trova_regole (ELEMENTO *tipo)` specificando l'attuale tipo analizzato. Questa procedura esegue le seguenti operazioni:
   (a) seleziona l'insieme GS(S) dove S è l'attuale tipo puntato.
      i. se il tipo S (o il GS(S)) è inesistente allora termina con un messaggio di errore.
      ii. altrimenti :
         A. crea una lista (`ante_group` [2] ) degli antecedenti delle regole ∈ GS(S).
         B. crea una lista (`cons_group` [2]) dei conseguenti delle regole ∈ GS(S).
         C. per ciascun nome di antecedente nella lista `ante_group`:
            - se esiste il conseguente con lo stesso nome termina.
            - altrimenti significa che posso applicare la regola esaminata che viene congiunta chiamando nuovamente `converti_ges_odl_`
            con parametri:
            `modo_conversione = 1`
            `cerco_regole = 1`
            `*punta_classe =` conseguente della regola applicabile.

Al termine della procedura ricorsiva abbiamo nel file `ges_out.pl` i dati di uscita corretti.
Le procedure di supporto sono le seguenti:

`int trova_regole(ELEMENTO *tipo)` Cerca le regole applicabili a `*tipo` e ritorna:
0 se non ci sono regole applicabili,

[2]Le liste vengono preventivamente vuotate chiamando la procedura `azzera_list_group(void)`

2 se manca la descrizione del GS(`*tipo`).
Se esistono regole applicabili, le applica (chiamando `converti_ges_odl_tipo ( )`) e ritorna con valore 1.

`void scrivi_nome(char DATO[ ], FILE *stream):` scrive il contenuto dell'array `DATO[ ]` sul file puntato da `*stream`.

`void aggiungi_regola(char VAR[ ], int i) :` viene aggiunto il nome della regola di nome `VAR[ ]` in coda alla lista:
`ante_group` se `i` = 0,
`cons_group` se `i` = 1.

`int seleziona_classe(char VAR[ ]) :` cerca il tipo il cui nome vale `VAR[ ]` e ritorna:
0 se la ricerca ha successo,
1 altrimenti.

`void azzera_list_group(void) :` inizializza a voute le liste `ante_group` e `cons_group`.

`void togli_F_ante(void) :` elimina il primo elemento della lista `ante_group`.

`void togli_F_cons(void) :` elimina il primo elemento della lista `cons_group`.

# Capitolo 6

# Sessione di lavoro con ODB-QOptimizer

In questo capitolo presentiamo due esempi di ottimizzazione ottenuti da una sessione di lavoro di ODB-QOptimizer. Le operazioni da compiere sull'elaboratore SUN da noi utilizzato (SUN x-sparc 10, sistema operativo sunOS, release 4.1.3) sono le seguenti:

- Operazione preliminare:
  scrittura del file di testo di nome `input.pl` contente la descrizione dello schema con regole in sintassi ODL-Designer.

- In ambiente SICStus Prolog lanciare l'esecuzione di ODB-QOptimizer.
  Comandi:
  `[prepara].`
  `p.` carica il modulo oggetto di GES, compila ODL-Designer e ODB-QOptimizer,
  `odb_qopt.` manda in esecuzione il componente.

  L'esecuzione del sistema attiva ODL-Designer in modalità CREA (non essendo ancora stato caricato lo schema) con funzionalità F1 ed F2 e termina dopo aver calcolato lo schema canonico e la sussunzione.

- Inserire la query da ottimizzare:





  1. scrivere il file di testo di nome `input.pl` contenente la descrizione dell'interrogazione in sintassi ODL-Designer.

  2. mandare in esecuzione il componente con il comando `odb_qopt.`
     L'ottimizzatore legge il file `input.pl` e calcola l'espansione semantica dell'interrogazione:

     (a) se l'ottimizzazione è avvenuta correttamente allora la query ottimizzata viene visualizzata su video.
     (b) in caso di errore appare un messaggio sullo schermo che specifica il tipo di anomalia.

- Volendo inserire nuove interrogazioni eseguo nuovamente ODB-QOptimizer (comando `odb_qopt.`) dopo averla scritta nel file di input `input.pl`

Le sezioni del capitolo contengono la documentazione dettagliata delle varie fasi della sessione, mostrando il contenuto dei file di comunicazione tra i componenti ODL-Designer e GES.

## 6.1 Schema ODL con regole

Oggetto del nostro esempio è lo schema con regole presentato in [BBLS94b], al quale abbiamo aggiunto una regola sui tipi-valore, in modo da verificare la corretta esecuzione di ODB-QOptimizer per tutti i tipi di regole ammesse dal nostro formalismo.

Lo schema con regole utilizzato nell'esempio è riportato secondo il formalismo ODL in tabella 6.1 ; la prima operazione da compiere è a carico dell'utente il quale genera il seguente file `input.pl`:

**File** `input.pl`

```
leggiinput(FLAG) :-
aggiungi(s( [prim,manager,=,/,\,'[',name,:,string,',',
                                  salary,:,integer,',',
                                  level,:,integer,']']),FLAG),
aggiungi(s([prim,department,=,/,\,'[',name,:,string,',',
                                    managed-by,:,manager,']',&,
```

$$
\left.
\begin{array}{l}
\sigma \left\{
\begin{array}{lcl}
\sigma_{\mathrm{P}}(\texttt{manager}) & = & \triangle[\texttt{name: string, salary: integer, level: integer}] \\
\sigma_{\mathrm{P}}(\texttt{department}) & = & \triangle[\texttt{name: string, managed-by: manager}] \sqcap \\
 & & \triangle[\texttt{managed-by:} \triangle[\texttt{salary:} 30000 \div \infty]] \\
\sigma_{\mathrm{P}}(\texttt{tmanager}) & = & \texttt{manager} \sqcap \triangle[\texttt{level:} 8 \div 12] \\
\sigma_{\mathrm{P}}(\texttt{material}) & = & \triangle[\texttt{name: string, risk: integer}] \\
\sigma_{\mathrm{P}}(\texttt{smaterial}) & = & \texttt{material} \sqcap \triangle[\texttt{feature: string}] \\
\sigma_{\mathrm{P}}(\texttt{storage}) & = & \texttt{department} \sqcap \triangle[\texttt{container: string, stock: material, qty:} 10 \div 300]\}] \\
\sigma_{\mathrm{P}}(\texttt{sstorage}) & = & \texttt{storage} \sqcap \triangle[\texttt{stock: smaterial}]
\end{array}
\right. \\
\mathbf{R} \left\{
\begin{array}{ll}
R_1: & \texttt{manager} \sqcap (\triangle.\texttt{level:} 5 \div 10) \rightarrow (\triangle.\texttt{salary:} 40000 \div 60000) \\
R_2: & \texttt{manager} \sqcap (\triangle.\texttt{level:} 11 \div 15) \rightarrow (\triangle.\texttt{salary:} 50000 \div 70000) \\
R_3: & \texttt{manager} \sqcap (\triangle.\texttt{level:} 16 \div \infty) \rightarrow (\triangle.\texttt{salary:} 70000 \div \infty) \\
R_4: & \texttt{storage} \sqcap (\triangle.\texttt{stock.} \triangle.\texttt{risk:} 10 \div \infty) \rightarrow (\triangle.\texttt{managed-by: tmanager}) \\
R_5: & \texttt{storage} \sqcap (\triangle.\texttt{qty:} 100 \div \infty) \rightarrow (\triangle.\texttt{container: } \textit{``A2''}) \\
R_6: & \texttt{storage} \sqcap (\triangle.\texttt{stock.} \triangle.\texttt{risk:} 20 \div \infty) \rightarrow \texttt{sstorage} \\
R_7: & [\texttt{level:} 11 \div 20] \rightarrow [\texttt{name: } \textit{``Silvano''}, \texttt{salary:} 50000 \div 75000]
\end{array}
\right.
\end{array}
\right\} (\sigma, \mathbf{R})
$$

Tabella 6.1: Esempio di schema per ODB-QOptimizer

```
                                        /,\,'[',managed-by,:,
                                            /,\,'[', salary,:,30000,-,']',
                                            ']']),FLAG),
aggiungi(s([prim,tmanager,=,manager,&,
                                    /,\,'[',level,:,8,-,12,']']),FLAG),
aggiungi(s([prim,material,=,/,\,'[',name,:,string,',',
                                      risk,:, integer,']']),FLAG),
aggiungi(s([prim,smaterial,=,material,&,
                                     /,\,'[',feature,:,string,']']),FLAG),
aggiungi(s([prim,storage,=,department,&,
                                   /,\,'[',stock,:,material,',',
                                       qty,:,10,-,300,',',
                                       container,:,string,']']),FLAG),
aggiungi(s([prim,sstorage,=,storage,&,
                                    /,\,'[',stock,:,smaterial,']']),FLAG),
```



```
aggiungi(s([antev,1a,=,manager,&,
                        /,\,'[',level,:,5,-,10,']']),FLAG),
aggiungi(s([consv,1c,=,manager,&,
                        /,\,'[',salary,:,40000,-,60000,']']),FLAG),
aggiungi(s([antev,2a,=,manager,&,
                        /,\,'[',level,:,11,-,15,']']),FLAG),
aggiungi(s([consv,2c,=,manager,&,
                        /,\,'[',salary,:,50000,-,70000,']']),FLAG),
aggiungi(s([antev,3a,=,manager,&,
                        /,\,'[',level,:,16,-,']']),FLAG),
aggiungi(s([consv,3c,=,manager,&,
                        /,\,'[',salary,:,70000,-,']']),FLAG),
aggiungi(s([antev,4a,=,storage,&,
                        /,\,'[',stock,:,/,\,'[', risk,:,10,-,']',']']),FLAG),
aggiungi(s([consv,4c,=,storage,&,
                        /,\,'[',managed-by,:,tmanager,']']),FLAG),
aggiungi(s([antev,5a,=,storage,&,
                        /,\,'[',qty,:,100,-,']']),FLAG),
aggiungi(s([consv,5c,=,storage,&,
                        /,\,'[',container,:,"A2",']']),FLAG),
aggiungi(s([antev,6a,=,storage,&,
                        /,\,'[',stock,:,/,\,'[', risk,:,20,-,']',']']),FLAG),
aggiungi(s([consv,6c,=,storage,&,sstorage]),FLAG),
aggiungi(s([antet,7a,=,'[',level,:,11,-,20,']']),FLAG),
aggiungi(s([const,7c,=,'[',name,:,"Silvano",',',
                    salary,:,50000,-,75000,']']),FLAG),
aggiungi(fine,FLAG).
```

Ora lanciando ODB-QOptimizer (comando `odb_qopt` il file viene letto da ODL-Designer in funzione CREA che genera la base dei dati richiesta controllando la sintassi, la presenza di cicli e calcolando lo schema canonico e la sussunzione.

**SCHEMA CANONICO**

$$\overline{\mathbf{T}} \;=\; \{\mathtt{r}_7^a, \mathtt{r}_7^c\} \cup \{b_1 \ldots b_{16}\} \cup \{t_1 \ldots t_{23}\}$$

$$
\begin{aligned}
\overline{\mathbf{C}} &= \{\overline{\texttt{manager}}, \overline{\texttt{department}}, \overline{\texttt{tmanager}}, \overline{\texttt{material}}, \\
&\quad \overline{\texttt{smaterial}}, \overline{\texttt{storage}}, \overline{\texttt{sstorage}}\} \\
\overline{\mathbf{V}} &= \{\texttt{manager}, \texttt{department}, \texttt{tmanager}, \texttt{material}, \\
&\quad \texttt{smaterial}, \texttt{storage}, \texttt{sstorage}\} \cup \\
&\quad \{\texttt{r}_1^a \ldots \texttt{r}_6^a\} \cup \{\texttt{r}_1^c \ldots \texttt{r}_6^c\} \cup \{n_1 \ldots n_6\} \\
\nu(\texttt{manager}) &= \overline{\texttt{manager}} \sqcap \triangle t_1 \\
\nu(\texttt{department}) &= \overline{\texttt{department}} \sqcap \triangle t_3 \\
\nu(\texttt{tmanager}) &= \overline{\texttt{manager}} \sqcap \overline{\texttt{tmanager}} \sqcap \triangle t_4 \\
\nu(\texttt{material}) &= \overline{\texttt{material}} \sqcap \triangle t_5 \\
\nu(\texttt{smaterial}) &= \overline{\texttt{material}} \sqcap \overline{\texttt{smaterial}} \sqcap \triangle t_6 \\
\nu(\texttt{storage}) &= \overline{\texttt{department}} \sqcap \overline{\texttt{storage}} \sqcap \triangle t_7 \\
\nu(\texttt{sstorage}) &= \overline{\texttt{department}} \sqcap \overline{\texttt{storage}} \sqcap \overline{\texttt{sstorage}} \sqcap \triangle t_8 \\
\nu(\texttt{r}_1^a) &= \overline{\texttt{manager}} \sqcap \triangle t_9 \\
\nu(\texttt{r}_1^c) &= \overline{\texttt{manager}} \sqcap \triangle t_{10} \\
\nu(\texttt{r}_2^a) &= \overline{\texttt{manager}} \sqcap \triangle t_{11} \\
\nu(\texttt{r}_2^c) &= \overline{\texttt{manager}} \sqcap \triangle t_{12} \\
\nu(\texttt{r}_3^a) &= \overline{\texttt{manager}} \sqcap \triangle t_{13} \\
\nu(\texttt{r}_3^c) &= \overline{\texttt{manager}} \sqcap \triangle t_{14} \\
\nu(\texttt{r}_4^a) &= \overline{\texttt{department}} \sqcap \overline{\texttt{storage}} \sqcap \triangle t_{16} \\
\nu(\texttt{r}_4^c) &= \overline{\texttt{department}} \sqcap \overline{\texttt{storage}} \sqcap \triangle t_{18} \\
\nu(\texttt{r}_5^a) &= \overline{\texttt{department}} \sqcap \overline{\texttt{storage}} \sqcap \triangle t_{19} \\
\nu(\texttt{r}_5^c) &= \overline{\texttt{department}} \sqcap \overline{\texttt{storage}} \sqcap \triangle t_{20} \\
\nu(\texttt{r}_6^a) &= \overline{\texttt{department}} \sqcap \overline{\texttt{storage}} \sqcap \triangle t_{22} \\
\nu(\texttt{r}_6^c) &= \overline{\texttt{department}} \sqcap \overline{\texttt{storage}} \sqcap \overline{\texttt{sstorage}} \sqcap \triangle t_{23} \\
\nu(\texttt{r}_7^a) &= [\texttt{level}\colon b_1] \\
\nu(\texttt{r}_7^c) &= [\texttt{name}\colon b_2, \texttt{salary}\colon b_3] \\
\nu(b_1) &= 11\text{–}20 \\
\nu(b_2) &= [83, 105, 108, 118, 97, 110, 111] \\
\nu(b_3) &= 50000\text{–}75000 \\
\nu(b_4) &= 30000 \div \infty \\
\nu(b_5) &= 8\text{–}12 \\
\nu(b_6) &= 10\text{–}300
\end{aligned}
$$



$$
\begin{aligned}
\nu(b_7) &= 5\text{–}10 \\
\nu(b_8) &= 40000\text{–}60000 \\
\nu(b_9) &= 11\text{–}15 \\
\nu(b_{10}) &= 50000\text{–}70000 \\
\nu(b_{11}) &= 16 \div \infty \\
\nu(b_{12}) &= 70000 \div \infty \\
\nu(b_{13}) &= 10 \div \infty \\
\nu(b_{14}) &= 100\text{–}300 \\
\nu(b_{15}) &= [65, 50] \\
\nu(b_{16}) &= 20 \div \infty \\
\nu(t_1) &= [\texttt{level}\colon \texttt{integer}, \texttt{name} : \texttt{string}, \texttt{salary}\colon \texttt{integer}] \\
\nu(t_2) &= [\texttt{level}\colon \texttt{integer}, \texttt{name} : \texttt{string}, \texttt{salary}\colon b_4] \\
\nu(t_3) &= [\texttt{name}\colon \texttt{string}, \texttt{managed-by}\colon n_1] \\
\nu(t_4) &= [\texttt{level}\colon b_5, \texttt{name}\colon \texttt{string}, \texttt{salary}\colon \texttt{integer}] \\
\nu(t_5) &= [\texttt{name}\colon \texttt{string}, \texttt{risk}\colon \texttt{integer}] \\
\nu(t_6) &= [\texttt{feature}\colon \texttt{string}, \texttt{name}\colon \texttt{string}, \texttt{risk}\colon \texttt{integer}] \\
\nu(t_7) &= [\texttt{container}\colon \texttt{string}, \texttt{name}\colon \texttt{string}, \texttt{qty}\colon b_6, \\
&\quad \texttt{stock}\colon \texttt{material}, \texttt{managed-by}\colon n_1] \\
\nu(t_8) &= [\texttt{container}\colon \texttt{string}, \texttt{name}\colon \texttt{string}, \texttt{qty}\colon b_6, \\
&\quad \texttt{stock}\colon n_2, \texttt{managed-by}\colon n_1] \\
\nu(t_9) &= [\texttt{level}\colon b_7, \texttt{name}\colon \texttt{string}, \texttt{salary}\colon \texttt{integer}] \\
\nu(t_{10}) &= [\texttt{level}\colon \texttt{integer}, \texttt{name} : \texttt{string}, \texttt{salary}\colon b_8] \\
\nu(t_{11}) &= [\texttt{level}\colon b_9, \texttt{name}\colon \texttt{string}, \texttt{salary}\colon \texttt{integer}] \\
\nu(t_{12}) &= [\texttt{level}\colon \texttt{integer}, \texttt{name} : \texttt{string}, \texttt{salary}\colon b_{10}] \\
\nu(t_{13}) &= [\texttt{level}\colon b_{11}, \texttt{name}\colon \texttt{string}, \texttt{salary}\colon \texttt{integer}] \\
\nu(t_{14}) &= [\texttt{level}\colon \texttt{integer}, \texttt{name} : \texttt{string}, \texttt{salary}\colon b_{12}] \\
\nu(t_{15}) &= [\texttt{name}\colon \texttt{string}, \texttt{risk}\colon b_{13}] \\
\nu(t_{16}) &= [\texttt{container}\colon \texttt{string}, \texttt{name}\colon \texttt{string}, \texttt{qty}\colon b_6, \\
&\quad \texttt{stock}\colon n_3, \texttt{managed-by}\colon n_1] \\
\nu(t_{17}) &= [\texttt{level}\colon b_5, \texttt{name} : \texttt{string}, \texttt{salary}\colon b_4] \\
\nu(t_{18}) &= [\texttt{container}\colon \texttt{string}, \texttt{name}\colon \texttt{string}, \texttt{qty}\colon b_6, \\
&\quad \texttt{stock}\colon \texttt{material}, \texttt{managed-by}\colon n_4]
\end{aligned}
$$

$$
\begin{aligned}
\nu(t_{19}) &= [\texttt{container}\colon \texttt{string}, \texttt{name}\colon \texttt{string}, \texttt{qty}\colon b_{14}, \\
&\quad\ \texttt{stock}\colon \texttt{material}, \texttt{managed-by}\colon n_1] \\
\nu(t_{20}) &= [\texttt{container}\colon b_{15}, \texttt{name}\colon \texttt{string}, \texttt{qty}\colon b_6, \\
&\quad\ \texttt{stock}\colon \texttt{material}, \texttt{managed-by}\colon n_1] \\
\nu(t_{21}) &= [\texttt{name}\colon \texttt{string}, \texttt{risk}\colon b_{16}] \\
\nu(t_{22}) &= [\texttt{container}\colon \texttt{string}, \texttt{name}\colon \texttt{string}, \texttt{qty}\colon b_6, \\
&\quad\ \texttt{stock}\colon n_5, \texttt{managed-by}\colon n_1] \\
\nu(t_{23}) &= [\texttt{container}\colon \texttt{string}, \texttt{name}\colon \texttt{string}, \texttt{qty}\colon b_6, \\
&\quad\ \texttt{stock}\colon n_2, \texttt{managed-by}\colon n_6] \\
\nu(n_1) &= \overline{\texttt{manager}} \sqcap \triangle t_2 \\
\nu(n_2) &= \overline{\texttt{material}} \sqcap \overline{\texttt{smaterial}} \sqcap \triangle t_6 \\
\nu(n_3) &= \overline{\texttt{material}} \sqcap \triangle t_{15} \\
\nu(n_4) &= \overline{\texttt{manager}} \sqcap \overline{\texttt{tmanager}} \sqcap \triangle t_{17} \\
\nu(n_5) &= \overline{\texttt{material}} \sqcap \triangle t_{21} \\
\nu(n_6) &= \overline{\texttt{manager}} \sqcap \triangle t_2
\end{aligned}
$$

Nuovamente sottolineamo che rispetto ai precedenti lavori di [Bal92] e [BN94] lo schema canonico presentato contiene, oltre alla descrizione dei tipi (classe e valore) che definiscono il mondo reale da rappresentare anche le regole di integrità e la query da ottimizzare; ció permette una classificazione globale dello schema con regole e (successivamente) dell'interrogazione secondo la sussunzione, che è alla base di tutte le operazioni successive di ottimizzazione semantica.

## 6.2 Primo esempio di interrogazione

Si voglia ora associare la seguente interrogazione alla base di dati presente in memoria:

`query` : "Seleziona i magazzini in cui la quantitá di merce disponibile è compresa tra 200 e 400 , i materiali conservati hanno rischio maggiore o uguale a 30 , sono gestiti da manager il cui salario è compreso tra 50000 e 80000 ed hanno un livello tra 11 e 20".



L'interrogazione viene trasformata nel formalismo ODL fornendo il seguente risultato:

$$
\begin{aligned}
\sigma_{\texttt{V}}(\texttt{query}) \;=\; & \texttt{storage} \sqcap (\triangle.\texttt{stock}.\ \triangle.\texttt{risk}\colon 30 \div \infty) \\
& \sqcap\ (\triangle.\texttt{qty}\colon 200 \div 400) \\
& \sqcap \triangle[\texttt{managed-by}\colon \texttt{manager} \sqcap \\
& \qquad\qquad (\triangle.\texttt{level}\colon 11 \div 20) \sqcap \\
& \qquad\qquad (\triangle.\texttt{salary}\colon 50000 \div 80000)]
\end{aligned}
$$

Come operazione preliminare l'utente scrive il file `input.pl` che contiene l'interrogazione in sintassi ODL-Designer:

**File** `input.pl`

```
leggiinput(FLAG) :-
aggiungi(s([virt,query,=,storage,&,
                    /,\,'[',stock,:,/,\,'[', risk,:,30,-,']',']',&,
                    /,\,'[',qty,:,200,:,400,']',&,
                    /,\,'[',managed-by,:,manager,&,
                                /,\,'[',salary,:,50000,-,
                                                80000,']',&,
                                /,\,'[',level,:,11,-,20,']',
                        ']'])FLAG),
aggiungi(fine,FLAG).
```

Mandando nuovamente in esecuzione ODB-QOptimizer viene attivato ODL-Designer in funzione AGGIUNGI (con funzionalità F1 ed F2) il quale legge il file di ingresso ed inserisce l'interrogazione nella base di dati già presente.

### 6.2.1 Prima esecuzione dell'espansione semantica

Ora ODB-QOptimizer puó iniziare le operazioni che portano all'ottimizzazione della query: ODL-Designer prosegue la propria esecuzione ottenendo come risultato lo schema canonico, la classificazione tassonomica delle classi e il file di interfaccia con GES, poi attiva GES che applica effettivamente le regole.
Il primo dei passi che iterativamete portano al calcolo della funzione di espansione semantica si conclude con il controllo da parte di ODB-QOptimizer sulla

condizione di terminazione: se abbiamo raggiunto il punto fisso della funzione , (query) allora abbiamo terminato, altrimenti viene iterato il procedimento ora descritto.

### Esecuzione di ODL-Designer

### SCHEMA CANONICO

$$
\begin{aligned}
\overline{\mathbf{T}} &= \{\mathtt{r}_7^a, \mathtt{r}_7^c\} \cup \{b_1 \ldots b_{19}\} \cup \{t_1 \ldots t_{26}\} \\
\overline{\mathbf{C}} &= \{\overline{\mathtt{manager}}, \overline{\mathtt{department}}, \overline{\mathtt{tmanager}}, \overline{\mathtt{material}}, \\
&\quad \overline{\mathtt{smaterial}}, \overline{\mathtt{storage}}, \overline{\mathtt{sstorage}}\} \\
\overline{\mathbf{V}} &= \{\mathtt{manager}, \mathtt{department}, \mathtt{tmanager}, \mathtt{material}, \\
&\quad \mathtt{smaterial}, \mathtt{storage}, \mathtt{sstorage}, \mathtt{query}\} \cup \\
&\quad \{\mathtt{r}_1^a \ldots \mathtt{r}_6^a\} \cup \{\mathtt{r}_1^c \ldots \mathtt{r}_6^c\} \cup \{n_1 \ldots n_8\}
\end{aligned}
$$

Riportiamo solo la descrizione dei nuovi tipi classe e valore introdotti da ODL-Designer in quanto gli altri rimangono invariati rispetto allo schema precedente.

$$
\begin{aligned}
\nu(\mathtt{query}) &= \overline{\mathtt{department}} \sqcap \overline{\mathtt{storage}} \sqcap \triangle t_{26} \\
\nu(b_{17}) &= 200\text{–}300 \\
\nu(b_{18}) &= 30 \div \infty \\
\nu(b_{19}) &= 50000\text{–}80000 \\
\nu(t_{24}) &= [\mathtt{name}\colon \mathtt{string}, \mathtt{risk}\colon b_{18}] \\
\nu(t_{25}) &= [\mathtt{level}\colon b_1, \mathtt{name}\colon \mathtt{string}, \mathtt{salary}\colon b_{19}] \\
\nu(t_{26}) &= [\mathtt{container}\colon \mathtt{string}, \mathtt{name}\colon \mathtt{string}, \mathtt{qty}\colon b_{17}, \\
&\quad \mathtt{stock}\colon n_7, \mathtt{managed\text{-}by}\colon n_8] \\
\nu(n_7) &= \overline{\mathtt{material}} \sqcap \triangle t_{24} \\
\nu(n_8) &= \overline{\mathtt{manager}} \sqcap \triangle t_{25}
\end{aligned}
$$

### Classificazione minimale

ODL-Designer calcola la classificazione minimale rispetto all'ereditarietà per le classi che sono state introdotte dall'utente.



Vogliamo mettere in evidenza prima le relazioni di sussunzione fra classi corrispondenti a dichiarazioni di ISA esplicite nella descrizione dello schema con regole e successivamente quelle calcolate effettivamente da ODL-Designer.

RELAZIONI DI ISA ESPLICITE

$$
\begin{aligned}
\mathtt{tmanager} &\ ISA\ \mathtt{manager} \\
\mathtt{smaterial} &\ ISA\ \mathtt{material} \\
\mathtt{storage} &\ ISA\ \mathtt{department} \\
\mathtt{sstorage} &\ ISA\ \mathtt{storage} \\
\mathtt{r}_1^a &\ ISA\ \mathtt{manager} \\
\mathtt{r}_1^c &\ ISA\ \mathtt{manager} \\
\mathtt{r}_2^a &\ ISA\ \mathtt{manager} \\
\mathtt{r}_2^c &\ ISA\ \mathtt{manager} \\
\mathtt{r}_3^a &\ ISA\ \mathtt{manager} \\
\mathtt{r}_3^c &\ ISA\ \mathtt{manager} \\
\mathtt{r}_4^a &\ ISA\ \mathtt{storage} \\
\mathtt{r}_4^c &\ ISA\ \mathtt{storage} \\
\mathtt{r}_5^a &\ ISA\ \mathtt{storage} \\
\mathtt{r}_5^c &\ ISA\ \mathtt{storage} \\
\mathtt{r}_6^c &\ ISA\ \mathtt{storage}
\end{aligned}
$$

RELAZIONI DI ISA CALCOLATE

$$
\begin{aligned}
\mathtt{r}_6^a &\ ISA\ \mathtt{r}_4^a \\
\mathtt{query} &\ ISA\ \mathtt{r}_5^a \\
\mathtt{query} &\ ISA\ \mathtt{r}_6^a
\end{aligned}
$$

### Interfaccia ODL-Designer GES

Il modello di interfaccia tra ODL-Designer e GES prevede che sia generato da ODL-Designer il file di tipo testo `ges_in.txt` contenente:

- la descrizione dei tipi che compaiono nella descrizione della `query` a qualsiasi livello di innestamento,

- i relativi insiemi GS (esclusi quelli dei `b_type` che non vengono calcolati esplicitamente da ODL-Designer poichè nessuna regola viene associata ad un `b_type`)

- la descrizione dei tipi che compaiono nella descrizione dei conseguenti di ciascuna regola a qualsiasi livello di innestamento.

Nel nostro esempio viene generato il file:

**File** `ges_in.txt`

```
[virt,query,=,-department,&,-storage,&,/,\,t26]
\*
      Le descrizioni che seguono sono quelle dei tipi che
      sono contenuti ricorsivamente nella query.
*\
[type,t26,=,[,container,:,string,,,name,:,string,,,qty,:, b17,,,
             stock,:,n7,,,managed-by,:,n8,]]
[virt,n7,=,-material,&,/,\,t24]
[type,t24,=,[,name,:,string,,,risk,:,b18,]]
[virt,n8,=,-manager,&,/,\,t25]
[type,t25,=,[,level,:,b1,,,name,:,string,,,salary,:, b19,]]
\*
      Riportiamo ora le descrizioni degli insiemi GS dei tipi
      precedentemente descritti.
*\
[gs,query,(department,prim,),(,storage,prim,),(,r4a,antev,),
          (,r5a,antev,),(,r6a,antev,)]
[gs,t26,(,t3,type,),(,t7,type,),(,t16,type,),
        (,t19,type,),(,t22,type,)]
[gs,n7,(,material,prim,),(,n3,virt,),(,n5,virt,)]
[gs,t24,(,t21,type,),(,t5,type,),(,t15,type,)]
[gs,n8,(,manager,prim,),(,n1,virt,),(,n6,virt,)]
```



```
[gs,t25,(,r7a,antet,),(,t1,type,),(,t2,type,)]
\*
    Riportiamo infine le descrizioni dei conseguenti di tutte
    le regole (compresi i tipi che compaiono nelle loro descrizioni),
    necessari poich      e potenzialmente si possono applicare tutte le regole.
*\
[consv,r1c,=,-manager,&,/,\,t10]
[type,t10,=,[,level,:,integer,,,name,:,string,,,salary,:,b8,]]
[consv,r2c,=,-manager,&,/,\,t12]
[type,t12,=,[,level,:,integer,,,name,:,string,,,salary,:,b10,]]
[consv,r3c,=,-manager,&,/,\,t14]
[type,t14,=,[,level,:,integer,,,name,:,string,,,salary,:,b12,]]
[consv,r4c,=,-department,&,-storage,&,/,\,t18]
[type,t18,=,[,container,:,string,,,name,:,string,,,qty,:,b6,,,
             stock,:,material,,,managed-by,:,n4,]]
[prim,material,=,-material,&,/,\,t5]
[type,t5,=,[,name,:,string,,,risk,:,integer,]]
[virt,n4,=,-manager,&,-tmanager,&,/,\,t17]
[type,t17,=,[,level,:,b5,,,name,:,string,,,salary,:,b4,]]
[consv,r5c,=,-department,&,-storage,&,/,\,t20]
[type,t20,=,[,container,:,b15,,,name,:,string,,,qty,:,b6,,,
             stock,:,material,,,managed-by,:,n1,]]
[virt,n1,=,-manager,&,/,\,t2]
[type,t2,=,[,level,:,integer,,,name,:,string,,,salary,:,b4,]]
[consv,r6c,=,-department,&,-storage,&,-sstorage,&,/,\,t23]
[type,t23,=,[,container,:,string,,,name,:,string,,,qty,:,b6,,,
             stock,:,n2,,,managed-by,:,n6,]]
[virt,n2,=,-material,&,-smaterial,&,/,\,t6]
[type,t6,=,[,feature,:,string,,,name,:,string,,,risk,:,integer,]]
[virt,n6,=,-manager,&,/,\,t2]
[const,r7c,=,[,name,:,b2,,,salary,:,b3,]]
.
```

### Esecuzione di GES

Il controllo delle operazioni viene lasciato a GES il quale si pone l'obiettivo di riconoscere le regole applicabili e generare il file di output `ges_out.pl` contenente la nuova descrizione della `query` mutata in ragione dell'applicazione

delle regole (nel caso in cui nessuna regola sia stata applicata `ges_out.pl` non contiene alcuna descrizione).
Per compiere le operazioni di controllo sui tipi GES necessita di una rappresentazione dei tipi stessi differente da quella contenuta in `ges_in.txt`; in particolare attraverso un'analisi sintattica ciascun tipo e ciascun GS vengono rappresentati da una lista nella quale ogni elemento specifica un componente. Le liste che riguardano i tipi sono individuate da un array di puntatori di nome *first_classe[i], mentre i GS dai puntatori *first_gs[i]: la numerazione degli indici parte da 0 ed è progressiva rispettando l'ordine dei tipi in `ges_in.txt`. Sotto, a titolo di esempio, vengono riportate alcune di queste liste:

LISTE relativa ai TIPI (Classe e valore):

*first_classe[0]

| NOME | query | NOME | -department |
|---|---|---|---|
| TIPO | N_Classe | TIPO | Atomo_Fittizio |
| C_TIPO | virt | C_TIPO | |
| D_ATTRIBUTO | | D_ATTRIBUTO | |
| NEXT | $\longrightarrow$ | NEXT | $\longrightarrow$ |

| NOME | -storage | NOME | t26 |
|---|---|---|---|
| TIPO | Atomo_Fittizio | TIPO | Nome_Virtuale |
| C_TIPO | | C_TIPO | |
| D_ATTRIBUTO | | D_ATTRIBUTO | |
| NEXT | $\longrightarrow$ | NEXT | NULL |

*first_classe[2]

| NOME | n7 | NOME | -material |
|---|---|---|---|
| TIPO | N_Classe | TIPO | Atomo_Fittizio |
| C_TIPO | virt | C_TIPO | |
| D_ATTRIBUTO | | D_ATTRIBUTO | |
| NEXT | $\longrightarrow$ | NEXT | $\longrightarrow$ |



| NOME | t24 |
|---|---|
| TIPO | Nome_Virtuale |
| C_TIPO | |
| D_ATTRIBUTO | |
| NEXT | NULL |

*first_classe[3]

| NOME | t24 | NOME | name |
|---|---|---|---|
| TIPO | N_Classe | TIPO | Attributo |
| C_TIPO | type | C_TIPO | |
| D_ATTRIBUTO | | D_ATTRIBUTO | string |
| NEXT | $\longrightarrow$ | NEXT | $\longrightarrow$ |

| NOME | risk |
|---|---|
| TIPO | Attributo |
| C_TIPO | |
| D_ATTRIBUTO | b18 |
| NEXT | NULL |

∗first_classe[5]

| NOME | t25 | NOME | level |
|---|---|---|---|
| TIPO | N_Classe | TIPO | Attributo |
| C_TIPO | type | C_TIPO | |
| D_ATTRIBUTO | | D_ATTRIBUTO | b1 |
| NEXT | ⟶ | NEXT | ⟶ |

| NOME | name | NOME | salary |
|---|---|---|---|
| TIPO | Attributo | TIPO | Attributo |
| C_TIPO | | C_TIPO | |
| D_ATTRIBUTO | string | D_ATTRIBUTO | b19 |
| NEXT | ⟶ | NEXT | NULL |

∗first_classe[6]

| NOME | r1c | NOME | -manager |
|---|---|---|---|
| TIPO | N_Classe | TIPO | Atomo_Fittizio |
| C_TIPO | consv | C_TIPO | |
| D_ATTRIBUTO | | D_ATTRIBUTO | |
| NEXT | ⟶ | NEXT | ⟶ |

| NOME | t10 |
|---|---|
| TIPO | Nome_Virtuale |
| C_TIPO | |
| D_ATTRIBUTO | |
| NEXT | NULL |

∗first_classe[8]

| NOME | r2c | NOME | -manager |
|---|---|---|---|
| TIPO | N_Classe | TIPO | Atomo_Fittizio |
| C_TIPO | consv | C_TIPO | |
| D_ATTRIBUTO | | D_ATTRIBUTO | |
| NEXT | ⟶ | NEXT | ⟶ |



| NOME | t12 |
|---|---|
| TIPO | Nome_Virtuale |
| C_TIPO | |
| D_ATTRIBUTO | |
| NEXT | NULL |

∗first_classe[27]

| NOME | r7c | NOME | name |
|---|---|---|---|
| TIPO | N_Classe | TIPO | Attributo |
| C_TIPO | const | C_TIPO | |
| D_ATTRIBUTO | | D_ATTRIBUTO | b2 |
| NEXT | ⟶ | NEXT | ⟶ |

| NOME | salary |
|---|---|
| TIPO | Attributo |
| C_TIPO | |
| D_ATTRIBUTO | b3 |
| NEXT | NULL |

LISTE relative agli insiemi GS :

∗first_gs[0]

| NOME | query | NOME | department |
|---|---|---|---|
| TIPO | N_Classe | TIPO | |
| C_TIPO | gs | C_TIPO | prim |
| D_ATTRIBUTO | | D_ATTRIBUTO | |
| NEXT | ⟶ | NEXT | ⟶ |

| NOME | storage | NOME | r4a |
|---|---|---|---|
| TIPO | | TIPO | |
| C_TIPO | prim | C_TIPO | antev |
| D_ATTRIBUTO | | D_ATTRIBUTO | |
| NEXT | ⟶ | NEXT | ⟶ |

| NOME | r5a | NOME | r6a |
|---|---|---|---|
| TIPO | | TIPO | |
| C_TIPO | antev | C_TIPO | antev |
| D_ATTRIBUTO | | D_ATTRIBUTO | |
| NEXT | $\longrightarrow$ | NEXT | NULL |

$*$first_gs[4]

| NOME | n8 | NOME | manager |
|---|---|---|---|
| TIPO | | TIPO | |
| C_TIPO | gs | C_TIPO | prim |
| D_ATTRIBUTO | | D_ATTRIBUTO | |
| NEXT | $\longrightarrow$ | NEXT | $\longrightarrow$ |

| NOME | n1 | NOME | n6 |
|---|---|---|---|
| TIPO | | TIPO | |
| C_TIPO | virt | C_TIPO | virt |
| D_ATTRIBUTO | | D_ATTRIBUTO | |
| NEXT | $\longrightarrow$ | NEXT | NULL |

$*$first_gs[5]

| NOME | t25 | NOME | r7a |
|---|---|---|---|
| TIPO | | TIPO | |
| C_TIPO | gs | C_TIPO | antet |
| D_ATTRIBUTO | | D_ATTRIBUTO | |
| NEXT | $\longrightarrow$ | NEXT | $\longrightarrow$ |

| NOME | t2 | NOME | t1 |
|---|---|---|---|
| TIPO | | TIPO | |
| C_TIPO | type | C_TIPO | type |
| D_ATTRIBUTO | | D_ATTRIBUTO | |
| NEXT | $\longrightarrow$ | NEXT | NULL |



### Controllo di incoerenza

Prima di eseguire l'espansione semantica vera e propria GES controlla che non vi siano tipi incoerenti tra quelli inviati nel file `ges_in.txt`. Nel nostro esempio non ve ne sono quindi l'esecuzione prosegue normalmente con l'applicazione delle regole.

### Applicazione delle regole

Ora GES può effettivamente eseguire i controlli per determinare le regole applicabili; ricordiamo che una regola puó essere applicata a un tipo $S$ attraverso la funzione $,^{i}(S)$ definita da:

$$,^{i}(S) \;=\; \begin{cases} S \sqcap \bigsqcap_k S_k^c & \forall R_k:\ S_k^a \in GS(S),\ S_k^a \notin GS(S) \\ S & \text{altrimenti} \end{cases}$$

Questa funzione deve essere applicata a ciascun tipo che compare nella descrizione della query a qualsiasi livello di innestamento; si compiono quindi le seguenti operazioni:

- Analisi di $GS(\texttt{query})$:
  $\texttt{r}_4^a, \texttt{r}_5^a, regola_6^a \in GS(\texttt{query})$;
  $\texttt{r}_1^c \ldots \texttt{r}_7^c \notin GS(\texttt{query})$;
  posso applicare $\texttt{r}_4, \texttt{r}_5$ ed $\texttt{r}_6$ a `query`.
  Si ottiene il nuovo schema relativo a `query`:

$$\begin{aligned} ,^{1}(\texttt{query}) &= \nu(\texttt{query}) \sqcap \nu(\texttt{r}_4^c) \sqcap \nu(\texttt{r}_5^c) \sqcap \nu(\texttt{r}_6^c) \\ &= \overline{\texttt{department}} \sqcap \overline{\texttt{storage}} \sqcap \triangle t_{26} \sqcap \\ &\quad \overline{\texttt{department}} \sqcap \overline{\texttt{storage}} \sqcap \triangle t_{18} \sqcap \\ &\quad \overline{\texttt{department}} \sqcap \overline{\texttt{storage}} \sqcap \triangle t_{20} \sqcap \\ &\quad \overline{\texttt{department}} \sqcap \overline{\texttt{storage}} \sqcap \overline{\texttt{sstorage}} \sqcap \triangle t_{23} \end{aligned}$$

- Analisi di $GS(t_{26})$:
  $\mathtt{r}_1^a \ldots regola_7^a \notin GS(t_{26})$;
  $\mathtt{r}_1^c \ldots \mathtt{r}_7^c \notin GS(t_{26})$;
  non posso applicare alcuna regola a $t_{26}$.

- Analisi di $GS(n_7)$:
  $\mathtt{r}_1^a \ldots regola_7^a \notin GS(n_7)$;
  $\mathtt{r}_1^c \ldots \mathtt{r}_7^c \notin GS(n_7)$;
  non posso applicare alcuna regola a $n_7$.

- Analisi di $GS(t_{24})$:
  $\mathtt{r}_1^a \ldots regola_7^a \notin GS(t_{24})$;
  $\mathtt{r}_1^c \ldots \mathtt{r}_7^c \notin GS(t_{24})$;
  non posso applicare alcuna regola a $t_{24}$.

- Analisi di $GS(n_8)$:
  $\mathtt{r}_1^a \ldots regola_7^a \notin GS(n_8)$;
  $\mathtt{r}_1^c \ldots \mathtt{r}_7^c \notin GS(n_8)$;
  non posso applicare alcuna regola a $n_8$.

- Analisi di $GS(t_{25})$:
  $\mathtt{r}_7^a \in GS(t_{25})$;
  $\mathtt{r}_1^c \ldots \mathtt{r}_7^c \notin GS(t_{25})$;
  posso applicare $\mathtt{r}_7$ a $t_{25}$.
  Si ottiene il nuovo schema relativo a $t_{25}$:

$$
\begin{aligned}
, ^1(t_{25}) &= \nu(t_{25}) \sqcap \nu(\mathtt{r}_7^c) \\
&= [\mathtt{level}\colon b_1, \mathtt{name}\colon \mathtt{string}, \mathtt{salary}\colon b_{19}] \sqcap \\
&\quad\ [\mathtt{name}\colon b_2, \mathtt{salary}\colon b_3]
\end{aligned}
$$

GES può ora rappresentare la funzione $, ^1(\mathtt{query})$ sostituendo ai tipi-valore ed ai nomi virtuali creati da ODL-Designer la loro descrizione. Questa sostituzione si rende necessaria poichè ovviamente ODL-Designer non mantiene



l'associazione tra un nome inserito dall'utente (come `query`) che cambia la propria descrizione, con i tipi-valore e nomi virtuali creati da ODL-Designer che precedentemente lo descrivevano e quindi vi sarebbero inconsistenze sui dati. Il risultato che abbiamo ottenuto è il calcolo del passo i (con i=1) della funzione di espansione semantica della `query` che vale:

$$
\begin{aligned}
, ^1(\mathtt{query}) = {} & \overline{\overline{\mathtt{department}} \sqcap \overline{\mathtt{storage}}} \sqcap \\
& \triangle[\mathtt{container}\colon \mathtt{string}, \\
& \quad \mathtt{name}\colon \mathtt{string}, \\
& \quad \mathtt{qty}\colon b_{17}, \\
& \quad \mathtt{stock}\colon \overline{\overline{\mathtt{material}} \sqcap} \\
& \qquad \triangle[\mathtt{name}\colon \mathtt{string}, \\
& \qquad\ \ \mathtt{risk}\colon b_{18}], \\
& \quad \mathtt{managed\text{-}by}\colon \overline{\mathtt{manager}} \sqcap \\
& \qquad \triangle[\mathtt{level}\colon b_1, \\
& \qquad\ \ \mathtt{name}\colon \mathtt{string}, \\
& \qquad\ \ \mathtt{salary}\colon b_{19}] \sqcap \\
& \qquad \triangle[\mathtt{name}\colon b_2, \\
& \qquad\ \ \mathtt{salary}\colon b_3] \\
& \quad ] \sqcap \\
& \overline{\overline{\mathtt{department}} \sqcap \overline{\mathtt{storage}}} \sqcap \\
& \triangle[\mathtt{container}\colon \mathtt{string}, \\
& \quad \mathtt{name}\colon \mathtt{string}, \\
& \quad \mathtt{qty}\colon b_6, \\
& \quad \mathtt{stock}\colon \overline{\overline{\mathtt{material}} \sqcap} \\
& \qquad \triangle[\mathtt{name}\colon \mathtt{string}, \\
& \qquad\ \ \mathtt{risk}\colon \mathtt{integer}], \\
& \quad \mathtt{managed\text{-}by}\colon \overline{\mathtt{manager}} \sqcap \\
& \qquad \overline{\mathtt{tmanager}} \sqcap \\
& \triangle[\mathtt{level}\colon b_5, \\
& \quad \mathtt{name}\colon \mathtt{string}, \\
& \quad \mathtt{salary}\colon b_4] \\
& ] \sqcap \\
& \overline{\overline{\mathtt{department}} \sqcap \overline{\mathtt{storage}}} \sqcap \\
& \triangle[\mathtt{container}\colon b_{15}, \\
& \quad \mathtt{name}\colon \mathtt{string}, \\
& \quad \mathtt{qty}\colon b_6,
\end{aligned}
$$

```
   stock: material ⊓
          △[name: string,
            risk: integer],
   managed-by: manager ⊓
               △[level: integer,
                 name: string,
                 salary: b4]
   ] ⊓
department ⊓ storage ⊓ sstorage ⊓
△[container: string,
   name: string,
   qty: b6,
   stock: material ⊓
          smaterial ⊓
          △[feature: ,
            name: string,
            risk: integer],
   managed-by: manager ⊓
               △[level: integer,
                 name: string,
                 salary: b4]
   ]
```

GES trasforma la funzione , [1](query) ottenuta in una classe virtuale che ha il nome query e l'indice incrementato di 1 (cioè $\texttt{query}_1$) e ne scrive la descrizione nel file ges_out.pl secondo la sintassi ODL-Designer:

**File** ges_out.pl

```
leggiinput(FLAG) :-
    aggiungi(
    s([virt,query1,=,
          '-department',&,'-storage',&,
          /,\,'[',container,:,string,
              ',',name,:,string,
              ',',qty,:,b17,
              ',',stock,:,'-material',&,
                    /,\,'[',name,:,string,
                        ',',risk,:,b18,
                        ']',
              ',',managed-by,:,'-manager',&,
                        /,\,'[',level,:,b1,
                            ',',name,:,string,
                            ',',salary,:,b19,
                            ']',&,
                        /,\,'[',name,:,b2,
                            ',',salary,:,b3,
                            ']',
              ']',
          &,'-department',&,'-storage',&,
            /,\,'[',container,:,string,
                ',', name,:,string,
                ',',qty,:,b6,',',
                stock,:,'-material',&,
                      /,\,'[',name,:,string,
                          ',',risk,:,integer,
                          ']',',',
                managed-by,:,'-manager',&,
                            '-tmanager',&,
                            /,\,'[',level,:,b5,
                                ',', name,:,string,
                                ',', salary,:,integer,
                                ']',
            ']',
          &,'-department',&,'-storage',&,
            /,\,'[',container,:,b15,
            ',', name,:,string,
            ',', qty,:,b6,
            ',', stock,:,'-material',&,
                 /,\,'[',name,:,string,
                     ',', risk,:,integer,
                     ']',',',
                       managed-by,:,'-manager',&
                                  /,\,'[',level,:,integer,
                                      ',',name,:,string,
```

```
                                        ',', salary,:,b4,
                                        ']',
            ']',
          &,'-department',&,'-storage',&,'-sstorage',&,
            /,\,'[',container,:,string,
                ',', name,:,string,
                ',', qty,:,b6,
                ',', stock,:,'material',&,
                           '-smaterial',&,
                           /,\,'[',feature,:,string,
                               ',', name,:,string,
                               ',', risk,:,integer,
                               ']',
                ',',managed-by,:,'-manager',&,
                                 /,\,'[',level,:,integer,
                                     ',', name,:,string,
                                     ',', salary,:,b4,
                                     ']',
            ']',
]),FLAG),
aggiungi(fine,FLAG).
```

### Visualizzazione

Durante l'esecuzione di GES su video vengono inviati messaggi che specificano le operazioni compiute; in questa prima esecuzione si visualizza:

```
Esecuzione di GES
Controllo sintattico ...  OK
Controllo incoerenza ...  OK
Applicazione delle regole ...
Applico la regola r7
Applico la regola r4
Applico la regola r5
Applico la regola r6
```



### Controllo di terminazione

Il comando sulle operazioni ritorna ad ODB-QOptimizer il quale controlla il valore intero di ritorno di GES: poichè sono state applicate delle regole GES torna con valore 0 che viene interpretato da ODB-QOptimizer come la condizione di "non terminato". Questo significa che occorre richiamare sequenzialmente ODL-Designer e GES per cercare di espandere ulteriormente l'interrogazione.

## 6.2.2 Seconda esecuzione dell'espansione semantica

### Esecuzione di ODL-Designer

ODB-QOptimizer manda nuovamente in esecuzione ODL-Designer il quale legge il file `ges_out.pl` in modalità AGGIUNGI inserendo quindi anche la classe $\texttt{query}_1$ nella base di conoscenza. ODL-Designer inizia poi a controllare la coerenza dello schema ed a generare lo schema canonico.

### SCHEMA CANONICO

$$
\begin{aligned}
\overline{\mathbf{T}} &= \{\mathtt{r}_7^a, \mathtt{r}_7^c\} \cup \{b_1 \ldots b_{20}\} \cup \{t_1 \ldots t_{29}\} \\
\overline{\mathbf{C}} &= \{\overline{\mathtt{manager}}, \overline{\mathtt{department}}, \overline{\mathtt{tmanager}}, \overline{\mathtt{material}}, \\
&\quad \overline{\mathtt{smaterial}}, \overline{\mathtt{storage}}, \overline{\mathtt{sstorage}}\} \\
\overline{\mathbf{V}} &= \{\mathtt{manager}, \mathtt{department}, \mathtt{tmanager}, \mathtt{material}, \\
&\quad \mathtt{smaterial}, \mathtt{storage}, \mathtt{sstorage}, \mathtt{query}, \\
&\quad \mathtt{query}_1\} \cup \{\mathtt{r}_1^a \ldots \mathtt{r}_6^a\} \cup \\
&\quad \{\mathtt{r}_1^c \ldots \mathtt{r}_6^c\} \cup \{n_1 \ldots n_{12}\}
\end{aligned}
$$

Riportiamo solo la descrizione dei nuovi tipi classe e valore introdotti da ODL-Designer in quanto gli altri rimangono invariati rispetto allo schema precedente.

$$
\begin{aligned}
\nu(\mathtt{query}_1) &= \overline{\mathtt{department}} \sqcap \overline{\mathtt{storage}} \sqcap \overline{\mathtt{storage}} \sqcap \triangle t_{29} \\
\nu(b_{20}) &= 11\text{–}12 \\
\nu(t_{27}) &= [\mathtt{feature}\colon \mathtt{string}, \mathtt{name}\colon \mathtt{string}, \mathtt{risk}\colon b_{18}] \\
\nu(t_{28}) &= [\mathtt{level}\colon b_{20}, \mathtt{name}\colon b_2, \mathtt{salary}\colon b_3]
\end{aligned}
$$

$$
\begin{aligned}
\nu(t_{29}) &= [\texttt{container}: b_{15}, \texttt{name}: \texttt{string}, \texttt{qty}: b_{17}, \\
&\quad \texttt{stock}: n_{10}, \texttt{managed-by}: n_{12}] \\
\nu(n_9) &= \triangle t_{27} \\
\nu(n_{10}) &= \overline{\texttt{material}} \sqcap \overline{\texttt{smaterial}} \sqcap \triangle t_{27} \\
\nu(n_{11}) &= \triangle t_{28} \\
\nu(n_{12}) &= \overline{\texttt{manager}} \sqcap \overline{\texttt{tmanager}} \sqcap \triangle t_{28}
\end{aligned}
$$

### Classificazione minimale

Come prima ODL-Designer calcola la classificazione minimale rispetto all'ereditarietà per le classi che sono state introdotte dall'utente.

RELAZIONI DI ISA ESPLICITE

| | | |
|---|---|---|
| tmanager | $ISA$ | manager |
| smaterial | $ISA$ | material |
| storage | $ISA$ | department |
| sstorage | $ISA$ | storage |
| $\texttt{r}_1^a$ | $ISA$ | manager |
| $\texttt{r}_1^c$ | $ISA$ | manager |
| $\texttt{r}_2^a$ | $ISA$ | manager |
| $\texttt{r}_2^c$ | $ISA$ | manager |
| $\texttt{r}_3^a$ | $ISA$ | manager |
| $\texttt{r}_3^c$ | $ISA$ | manager |
| $\texttt{r}_4^a$ | $ISA$ | storage |
| $\texttt{r}_4^c$ | $ISA$ | storage |
| $\texttt{r}_5^a$ | $ISA$ | storage |
| $\texttt{r}_5^c$ | $ISA$ | storage |
| $\texttt{r}_6^c$ | $ISA$ | storage |
| $\texttt{query}_1$ | $ISA$ | sstorage |



RELAZIONI DI ISA CALCOLATE

| | | |
|---|---|---|
| $\texttt{r}_6^a$ | $ISA$ | $\texttt{r}_4^a$ |
| query | $ISA$ | $\texttt{r}_5^a$ |
| query | $ISA$ | $\texttt{r}_6^a$ |
| $\texttt{query}_1$ | $ISA$ | $\texttt{r}_4^c$ |
| $\texttt{query}_1$ | $ISA$ | $\texttt{r}_5^c$ |
| $\texttt{query}_1$ | $ISA$ | $\texttt{r}_6^c$ |
| $\texttt{query}_1$ | $ISA$ | query |

### Interfaccia ODL-Designer GES

A partire dalla nuova descrizione canonica ODL-Designer genera il file:

**File** `ges_in.txt`

```
[virt,query1,=,-department,&,-storage,&,/,\,t29]
\*
      Le descrizioni che seguono sono quelle dei tipi che
      sono contenuti ricorsivamente nella query.
*\
[type,t29,=,[,container,:,b15,,,name,:,string,,,qty,:, b17,,,
             stock,:,n10,,,managed-by,:,n12,]]
[virt,n10,=,-material,&,/,\,t27]
[type,t27,=,[,name,:,string,,,risk,:,b18,]]
[virt,n12,=,-manager,&,/,\,t28]
[type,t28,=,[,level,:,b20,,,name,:,b2,,,salary,:, b3,]]
\*
      Riportiamo ora le descrizioni degli insiemi GS dei tipi
      precedentemente descritti.
*\
[gs,query1,(department,prim,),(,storage,prim,),(,sstorage,prim,),
           (,query,virt,),(,r4a,antev,),(,r5a,antev,),(,r6a,antev,),
           (,r4c,consv,),(,r5c,consv,),(,r6c,consv,)]
[gs,t29,(,t3,type,),(,t7,type,),(,t8,type,),(,t16,type,),
```

```
        (,t18,type,),(,t19,type,),(,t22,type,),(,t23,type,),
        (,t26,type,)]
[gs,n10,(,material,prim,),(,smaterial,prim,),(,n2,virt,),
        (,n3,virt,),(,n5,virt,),(,n7,virt,)]
[gs,t27,(,t5,type,),(,t6,type,),(,t15,type,),(,t21,type,),
        (,t24,type,)]
[gs,n12,(,manager,prim,),(,tmanager,prim,),(,n1,virt,),(,n4,virt,),
        (,n6,virt,),(,n8,virt,),(,r2a,antev,)]
[gs,t28,(,r7a,antet,),(,r7c,const,),(,t1,type,),(,t2,type,),
    (,t4,type,),(,t11,type,),(,t17,type,),(,t25,type,)]
\*
    Riportiamo infine le descrizioni dei conseguenti di tutte
    le regole (compresi i tipi che compaiono nelle loro descrizioni),
    necessari poich      e potenzialmente si possono applicare tutte le regole.
*\
[consv,r1c,=,-manager,&,/,\,t10]
[type,t10,=,[,level,:,integer,,,name,:,string,,,salary,:,b8,]]
[consv,r2c,=,-manager,&,/,\,t12]
[type,t12,=,[,level,:,integer,,,name,:,string,,,salary,:,b10,]]
[consv,r3c,=,-manager,&,/,\,t14]
[type,t14,=,[,level,:,integer,,,name,:,string,,,salary,:,b12,]]
[consv,r4c,=,-department,&,-storage,&,/,\,t18]
[type,t18,=,[,container,:,string,,,name,:,string,,,qty,:,b6,,,
            stock,:,material,,,managed-by,:,n4,]]
[prim,material,=,-material,&,/,\,t5]
[type,t5,=,[,name,:,string,,,risk,:,integer,]]
[virt,n4,=,-manager,&,-tmanager,&,/,\,t17]
[type,t17,=,[,level,:,b5,,,name,:,string,,,salary,:,b4,]]
[consv,r5c,=,-department,&,-storage,&,/,\,t20]
[type,t20,=,[,container,:,b15,,,name,:,string,,,qty,:,b6,,,
            stock,:,material,,,managed-by,:,n1,]]
[virt,n1,=,-manager,&,/,\,t2]
[type,t2,=,[,level,:,integer,,,name,:,string,,,salary,:,b4,]]
[consv,r6c,=,-department,&,-storage,&,-sstorage,&,/,\,t23]
[type,t23,=,[,container,:,string,,,name,:,string,,,qty,:,b6,,,
            stock,:,n2,,,managed-by,:,n6,]]
[virt,n2,=,-material,&,-smaterial,&,/,\,t6]
[type,t6,=,[,feature,:,string,,,name,:,string,,,risk,:,integer,]]
```



```
[virt,n6,=,-manager,&,/,\,t2]
[const,r7c,=,[,name,:,b2,,,salary,:,b3,]]
```

### Esecuzione di GES

Come prima GES compie l'analisi sintattica del file `ges_in.txt` costruendo le appropriate liste che sono del tutto simili a quelle giá mostrate.
Poichè la sintassi di `ges_in.txt` è corretta e non vi sono classi incoerenti si passa all'

### Applicazione delle regole

Viene calcolata la funzione , $^2$(`query`) attraverso il controllo sulla condizione di applicabilità delle regole:

- Analisi di $GS(\texttt{query}_1)$:
  $\texttt{r}_4^a, \texttt{r}_5^a, regola_6^a \in GS(\texttt{query}_1)$;
  $\texttt{r}_4^c, \texttt{r}_5^c, \texttt{r}_6^c \in GS(\texttt{query}_1)$;
  non posso applicare alcuna regola a $\texttt{query}_1$.

- Analisi di $GS(t_{29})$:
  $\texttt{r}_1^a \ldots \texttt{r}_7^a \notin GS(t_{29})$;
  $\texttt{r}_1^c \ldots \texttt{r}_7^c \notin GS(t_{29})$;
  non posso applicare alcuna regola a $t_{29}$.

- Analisi di $GS(n_{10})$:
  $\texttt{r}_1^a \ldots regola_7^a \notin GS(n_{10})$;
  $\texttt{r}_1^c \ldots \texttt{r}_7^c \notin GS(n_{10})$;
  non posso applicare alcuna regola a $n_{10}$.

- Analisi di $GS(t_{27})$:
  $\texttt{r}_1^a \ldots regola_7^a \notin GS(t_{27})$;
  $\texttt{r}_1^c \ldots \texttt{r}_7^c \notin GS(t_{27})$;
  non posso applicare alcuna regola a $t_{27}$.

- Analisi di $GS(n_{12})$:

  $\mathtt{r}_2^a \in GS(n_{12})$;
  $\mathtt{r}_1^c \ldots \mathtt{r}_7^c \notin GS(n_{12})$;
  posso applicare $\mathtt{r}_2$ a $n_{12}$;
  Si ottiene il nuovo schema relativo a $n_{12}$:

$$\begin{aligned} ,^2(n_{12}) &= \nu(n_{12}) \sqcap \nu(\mathtt{r}_2^c) \\ &= \overline{\mathtt{manager}} \sqcap \overline{\mathtt{tmanager}} \sqcap \triangle t_{28} \sqcap \\ &\quad \overline{\mathtt{manager}} \sqcap \overline{\mathtt{tmanager}} \sqcap \triangle t_{12} \sqcap \end{aligned}$$

- Analisi di $GS(t_{28})$:

  $\mathtt{r}_7^a \in GS(t_{28})$;
  $\mathtt{r}_7^c \in GS(t_{28})$;
  non posso applicare alcuna regola a $t_{28}$;

Possiamo ora scrivere la nuova funzione $,^2(\mathtt{query})$:

$$\begin{aligned} ,^2(\mathtt{query}) = \overline{\mathtt{department}} \sqcap\ & \overline{\mathtt{storage}} \sqcap\ \overline{\mathtt{sstorage}} \sqcap \\ & \triangle[\mathtt{container}: b_{15}, \\ & \quad \mathtt{name}: \mathtt{string}, \\ & \quad \mathtt{qty}: b_{17}, \\ & \quad \mathtt{stock}: \overline{\mathtt{material}} \sqcap \\ & \qquad \overline{\mathtt{smaterial}} \sqcap \\ & \qquad \triangle[\mathtt{feature}: \mathtt{string}, \\ & \qquad\quad \mathtt{name}: \mathtt{string}, \\ & \qquad\quad \mathtt{risk}: b_{18}], \\ & \quad \mathtt{managed\text{-}by}: \overline{\mathtt{manager}} \sqcap \\ & \qquad\qquad \overline{\mathtt{tmanager}} \sqcap \\ & \qquad\qquad \triangle[\mathtt{level}: b_{20}, \\ & \qquad\qquad\quad \mathtt{name}: b_2, \\ & \qquad\qquad\quad \mathtt{salary}: b_3] \sqcap \\ & \qquad\qquad \overline{\mathtt{manager}} \sqcap \\ & \qquad\qquad \triangle[\mathtt{level}: \mathtt{integer}, \\ & \qquad\qquad\quad \mathtt{name}: \mathtt{string}, \\ & \qquad\qquad\quad \mathtt{salary}: b_{10}] \\ & ] \end{aligned}$$



Infine GES trasforma la query ottenuta con l'espansione in una classe virtuale che ha il nome query con l'indice incrementato di 1 (cioè $\mathtt{query}_2$) e ne scrive la descrizione nel file `ges_out.pl` secondo la sintassi ODL-Designer:

`ges_out.pl`

```
leggiinput(FLAG) :-
                    aggiungi(
                    s([virt,query2,=,
                            '-department',&,'-storage',&,'-sstorage'
                            /,\,'[',container,:,b15,
                                ',',name,:,string,
                                ',',qty,:,b17,
                                ',',stock,:,'-material',&,
                                          '-smaterial',&,
                                          /,\,'[',feature,:,string,
                                              ',',name,:,string,
                                              ',',risk,:,l8,
                                              ']',
                                ',',managed-by,:,'-manager',&,
                                                  '-tmanager',&,
                                                  /,\,'[',level,:,b20,
                                                      ',',name,:,b2,
                                                      ',',salary,:,b3,
                                                      ']',&,
                                                    '-manager',&,
                                                  /,\,'[',level,:,integer,
                                                      ',',name,:,string,
                                                      ',',salary,:,b10,
                                                      ']',
                            ']',
                    ]),FLAG),
                    aggiungi(fine,FLAG).
```

### Visualizzazione

Durante l' esecuzione si visualizza:

```
Esecuzione di GES
Controllo sintattico ...  OK
Controllo incoerenza ...  OK
Applicazione delle regole ...
Applico la regola r2
```

#### Controllo di terminazione

Il controllo delle operazioni ritorna ad ODB-QOptimizer il quale controlla il valore intero di ritorno di GES: poichè è stata applicata una regola GES torna con valore 0 che viene interpretato da ODB-QOptimizer come la condizione di "non terminato". Questo significa che occorre richiamare sequenzialmente ODL-Designer e GES per cercare di espandere ulteriormente l'interrogazione.

### 6.2.3 Terza esecuzione dell'espansione semantica

#### Esecuzione di ODL-Designer

ODB-QOptimizer manda nuovamente in esecuzione ODL-Designer il quale legge il file `ges_out.pl` in modalità AGGIUNGI inserendo quindi anche la classe $\texttt{query}_2$ nella base di conoscenza. ODL-Designer inizia poi a controllare la coerenza dello schema ed a generare lo schema canonico.

#### SCHEMA CANONICO

$$
\begin{aligned}
\overline{\mathbf{T}} &= \{\mathtt{r}_7^a, \mathtt{r}_7^c\} \cup \{b_1 \ldots b_{20}\} \cup \{t_1 \ldots t_{31}\} \\
\overline{\mathbf{C}} &= \{\overline{\texttt{manager}}, \overline{\texttt{department}}, \overline{\texttt{tmanager}}, \overline{\texttt{material}}, \\
&\quad \overline{\texttt{smaterial}}, \overline{\texttt{storage}}, \overline{\texttt{sstorage}}\} \\
\overline{\mathbf{V}} &= \{\texttt{manager}, \texttt{department}, \texttt{tmanager}, \texttt{material}, \\
&\quad \texttt{smaterial}, \texttt{storage}, \texttt{sstorage}, \texttt{query}, \\
&\quad \texttt{query}_1, \texttt{query}_2\} \cup \{\mathtt{r}_1^a \ldots \mathtt{r}_6^a\} \cup \\
&\quad \{\mathtt{r}_1^c \ldots \mathtt{r}_6^c\} \cup \{n_1 \ldots n_{16}\}
\end{aligned}
$$

Riportiamo solo la descrizione dei nuovi tipi classe e valore introdotti da ODL-Designer in quanto gli altri rimangono invariati rispetto allo schema



precedente.

$$
\begin{aligned}
\nu(\texttt{query}_2) &= \overline{\texttt{department}} \sqcap \overline{\texttt{storage}} \sqcap \overline{\texttt{storage}} \sqcap \triangle t_{31} \\
\nu(b_{20}) &= 11\text{–}12 \\
\nu(t_{27}) &= [\texttt{feature}\colon \texttt{string}, \texttt{name}\colon \texttt{string}, \texttt{risk}\colon b_{18}] \\
\nu(t_{28}) &= [\texttt{level}\colon b_{20}, \texttt{name}\colon b_2, \texttt{salary}\colon b_3] \\
\nu(t_{31}) &= [\texttt{container}\colon b_{15}, \texttt{name}\colon \texttt{string}, \texttt{qty}\colon b_{17}, \\
&\quad \texttt{stock}\colon n_{14}, \texttt{managed-by}\colon n_{16}] \\
\nu(n_{13}) &= \triangle t_{27} \\
\nu(n_{14}) &= \overline{\texttt{material}} \sqcap \overline{\texttt{smaterial}} \sqcap \triangle t_{27} \\
\nu(n_{15}) &= \triangle t_{30} \\
\nu(n_{16}) &= \overline{\texttt{manager}} \sqcap \overline{\texttt{tmanager}} \sqcap \triangle t_{30}
\end{aligned}
$$

#### Classificazione minimale

Come prima ODL-Designer calcola la classificazione minimale rispetto all'ereditarietà per le classi che sono state introdotte dall'utente.

RELAZIONI DI ISA ESPLICITE

$$
\begin{aligned}
\texttt{tmanager} &\ ISA\ \texttt{manager} \\
\texttt{smaterial} &\ ISA\ \texttt{material} \\
\texttt{storage} &\ ISA\ \texttt{department} \\
\texttt{sstorage} &\ ISA\ \texttt{storage} \\
\mathtt{r}_1^a &\ ISA\ \texttt{manager} \\
\mathtt{r}_1^c &\ ISA\ \texttt{manager} \\
\mathtt{r}_2^a &\ ISA\ \texttt{manager} \\
\mathtt{r}_2^c &\ ISA\ \texttt{manager} \\
\mathtt{r}_3^a &\ ISA\ \texttt{manager} \\
\mathtt{r}_3^c &\ ISA\ \texttt{manager} \\
\mathtt{r}_4^a &\ ISA\ \texttt{storage} \\
\mathtt{r}_4^c &\ ISA\ \texttt{storage}
\end{aligned}
$$

$\mathtt{r}_5^a$ $ISA$ storage
$\mathtt{r}_5^c$ $ISA$ storage
$\mathtt{r}_6^c$ $ISA$ storage
$\mathtt{query}_1$ $ISA$ sstorage
$\mathtt{query}_2$ $ISA$ sstorage

RELAZIONI DI ISA CALCOLATE

$\mathtt{r}_6^a$ $ISA$ $\mathtt{r}_4^a$
query $ISA$ $\mathtt{r}_5^a$
query $ISA$ $\mathtt{r}_6^a$
$\mathtt{query}_1$ $ISA$ $\mathtt{r}_4^c$
$\mathtt{query}_1$ $ISA$ $\mathtt{r}_5^c$
$\mathtt{query}_1$ $ISA$ $\mathtt{r}_6^c$
$\mathtt{query}_1$ $ISA$ query
$\mathtt{query}_2$ $ISA$ $\mathtt{query}_1$

### Interfaccia ODL-Designer GES

A partire dalla nuova descrizione canonica viene ora generato il file:

**File** `ges_in.txt`

```
[virt,query2,=,-department,&,-storage,&,/,\,t31]
\*
      Le descrizioni che seguono sono quelle dei tipi che
      sono contenuti ricorsivamente nella query.
*\
[type,t31,=,[,container,:,b15,,,name,:,string,,,qty,:, b17,,,
            stock,:,n14,,,managed-by,:,n16,]]
[virt,n14,=,-material,&,/,\,t27]
[type,t27,=,[,name,:,string,,,risk,:,b18,]]
[virt,n16,=,-manager,&,/,\,t30]
```



```
[type,t30,=,[,level,:,b20,,,name,:,b2,,,salary,:, b10,]]
\*
      Riportiamo ora le descrizioni degli insiemi GS dei tipi
      precedentemente descritti.
*\
[gs,query2,(department,prim,),(,storage,prim,),(,sstorage,prim,),
           (,query,virt,),(,query1,virt,),(,r4a,antev,),(,r5a,antev,),
           (,r6a,antev,),(,r4c,consv,),(,r5c,consv,),(,r6c,consv,)]
[gs,t31,(,t3,type,),(,t7,type,),(,t8,type,),(,t16,type,),
        (,t18,type,),(,t19,type,),(,t22,type,),(,t23,type,),
        (,t26,type,),(,t29,type,)]
[gs,n14,(,material,prim,),(,smaterial,prim,),(,n2,virt,),
        (,n3,virt,),(,n5,virt,),(,n7,virt,)]
[gs,t27,(,t5,type,),(,t6,type,),(,t15,type,),(,t21,type,),
        (,t24,type,)]
[gs,n16,(,manager,prim,),(,tmanager,prim,),(,n1,virt,),(,n4,virt,),
        (,n6,virt,),(,n8,virt,),(,r2a,antev,),(,n10,virt,),(,r2c,consv,)]
[gs,t30,(,r7a,antet,),(,r7c,const,),(,t1,type,),(,t2,type,),
    (,t4,type,),(,t11,type,),(,t17,type,),(,t25,type,),
    (,t28,type,)]
\*
    Riportiamo infine le descrizioni dei conseguenti di tutte
    le regole (compresi i tipi che compaiono nelle loro descrizioni),
    necessari poich      e potenzialmente si possono applicare tutte le regole.
*\
[consv,r1c,=,-manager,&,/,\,t10]
[type,t10,=,[,level,:,integer,,,name,:,string,,,salary,:,b8,]]
[consv,r2c,=,-manager,&,/,\,t12]
[type,t12,=,[,level,:,integer,,,name,:,string,,,salary,:,b10,]]
[consv,r3c,=,-manager,&,/,\,t14]
[type,t14,=,[,level,:,integer,,,name,:,string,,,salary,:,b12,]]
[consv,r4c,=,-department,&,-storage,&,/,\,t18]
[type,t18,=,[,container,:,string,,,name,:,string,,,qty,:,b6,,,
              stock,:,material,,,managed-by,:,n4,]]
[prim,material,=,-material,&,/,\,t5]
[type,t5,=,[,name,:,string,,,risk,:,integer,]]
[virt,n4,=,-manager,&,-tmanager,&,/,\,t17]
[type,t17,=,[,level,:,b5,,,name,:,string,,,salary,:,b4,]]
```

```
[consv,r5c,=,-department,&,-storage,&,/,\,t20]
[type,t20,=,[,container,:,b15,,,name,:,string,,,qty,:,b6,,,
            stock,:,material,,,managed-by,:,n1,]]
[virt,n1,=,-manager,&,/,\,t2]
[type,t2,=,[,level,:,integer,,,name,:,string,,,salary,:,b4,]]
[consv,r6c,=,-department,&,-storage,&,-sstorage,&,/,\,t23]
[type,t23,=,[,container,:,string,,,name,:,string,,,qty,:,b6,,,
            stock,:,n2,,,managed-by,:,n6,]]
[virt,n2,=,-material,&,-smaterial,&,/,\,t6]
[type,t6,=,[,feature,:,string,,,name,:,string,,,risk,:,integer,]]
[virt,n6,=,-manager,&,/,\,t2]
[const,r7c,=,[,name,:,b2,,,salary,:,b3,]]
```

### Esecuzione di GES

Come prima GES compie l'analisi sintattica del file `ges_in.txt` costruen-le appropriate liste che sono del tutto simili a quelle giá mostrate.

Poichè la sintassi di `ges_in.txt` è corretta e non vi sono classi incoerenti si passa all'

### Applicazione delle regole

Viene calcolata la funzione , $^3$(`query`) attraverso il controllo sulla condizione di applicabilità delle regole:

- Analisi di $GS(\texttt{query}_2)$:

  $\texttt{r}_4^a, \texttt{r}_5^a, regola_6^a \in GS(\texttt{query}_2)$;
  $\texttt{r}_4^c, \texttt{r}_5^c, \texttt{r}_6^c \in GS(\texttt{query}_2)$;
  non posso applicare alcuna regola a $\texttt{query}_2$.

- Analisi di $GS(t_{31})$:

  $\texttt{r}_1^a \ldots regola_7^a \notin GS(t_{31})$;
  $\texttt{r}_1^c \ldots \texttt{r}_7^c \notin GS(t_{31})$;
  non posso applicare alcuna regola a $t_{31}$.



- Analisi di $GS(n_{14})$:

  $\texttt{r}_1^a \ldots regola_7^a \notin GS(n_{14})$;
  $\texttt{r}_1^c \ldots \texttt{r}_7^c \notin GS(n_{14})$;
  non posso applicare alcuna regola a $n_{14}$.

- Analisi di $GS(t_{27})$:

  $\texttt{r}_1^a \ldots regola_7^a \notin GS(t_{27})$;
  $\texttt{r}_1^c \ldots \texttt{r}_7^c \notin GS(t_{27})$;
  non posso applicare alcuna regola a $t_{27}$.

- Analisi di $GS(n_{16})$:

  $\texttt{r}_2^a \in GS(n_{16})$;
  $\texttt{r}_2^c \in GS(n_{16})$;
  non posso applicare alcuna regola a $n_{16}$.

- Analisi di $GS(t_{30})$:

  $\texttt{r}_7^a \in GS(t_{30})$;
  $\texttt{r}_7^c \in GS(t_{30})$;
  non posso applicare alcuna regola a $t_{30}$;

L'analisi compiuta non ha portato all'applicazione di nessuna regola: ció significa che , $^3$(`query`) $=$ , $^2$(`query`) ed è stato raggiunto il punto fisso della funzione di espansione semantica $\tilde{,}$(`query`).
GES si accorge di non essere riuscito ad applicare alcuna regola e come conseguenza genera il file `ges_out.pl` senza inserire la descrizione di una nuova `query`:

**File** `ges_out.pl`

```
leggiinput(FLAG) :-
                    aggiungi(fine,FLAG).
```

### Visualizzazione

Durante l'esecuzione si visualizza:

```
Esecuzione di GES
Controllo sintattico ...  OK
Controllo incoerenza ...  OK
Applicazione delle regole ...
Non ho applicato nessuna regola,
l'ottimizzazione e' terminata correttamente.

la query e':  [-sstorage,&,-storage,&,-department,&,
/,\,[,container,:,b15,,,name,:,string,,,qty,:,b17,,,
stock,:,-material,&,-smaterial,&,/,\,[,feature,:,string,,,
name,:,string,,,risk,:,b18,],,,managed-by,:,-manager,&,
-tmanager,&,/,\,[,level,:,b20,,,name,:,b2,,,salary,:,b3,],
&,-manager,&,/,\,[,level,:,integer,,,name,:,string,,,
salary,:,b10,],]]
```

### Controllo di terminazione

ODB-QOptimizer legge il valore di ritorno di GES che ora vale 1: da ció capisce che il calcolo dell'espansione semantica della `query` è finito e quindi termina anche il programma.
La $\tilde{\nu}$(`query`) è resa disponibile in forma canonica in ambiente ODL-Designer dalla descrizione di $\nu(\texttt{query}_2)$.

Lo schema vale:

$$
\begin{aligned}
\tilde{\nu}(\texttt{query}) &= \overline{\texttt{department}} \sqcap \overline{\texttt{storage}} \sqcap \triangle t_{31} \\
\tilde{\nu}(t_{31}) &= [\texttt{container}\colon b_{15}, \texttt{name}\colon \texttt{string}, \texttt{qty}\colon b_{17}, \\
&\quad\ \texttt{stock}\colon n_{14}, \texttt{managed-by}\colon n_{16}] \\
\tilde{\nu}(b_{15}) &= [65, 50] \\
\tilde{\nu}(b_{17}) &= 200\text{–}300 \\
\tilde{\nu}(n_{14}) &= \overline{\texttt{material}} \sqcap \overline{\texttt{smaterial}} \sqcap \triangle t_{27} \\
\tilde{\nu}(t_{27}) &= [\texttt{feature}\colon \texttt{string}, \texttt{name}\colon \texttt{string}, \texttt{risk}\colon b_{18}] \\
\tilde{\nu}(b_{18}) &= 30 \div \infty \\
\tilde{\nu}(n_{16}) &= \overline{\texttt{manager}} \sqcap \overline{\texttt{tmanager}} \sqcap \triangle t_{30}
\end{aligned}
$$



$$
\begin{aligned}
\tilde{\nu}(t_{30}) &= [\texttt{level}\colon b_{20}, \texttt{name}\colon b_2, \texttt{salary}\colon b_{10}] \\
\nu(b_{20}) &= 11\text{–}12 \\
\tilde{\nu}(b_2) &= [83, 105, 108, 118, 97, 110, 111] \\
\tilde{\nu}(b_{10}) &= 50000\text{–}70000
\end{aligned}
$$

Quindi lo strumento ODB-QOptimizer è riuscito ad ottenere l'ottimizzazione dell' interrogazione di partenza attraverso il calcolo della forma canonica dell'espansione semantica della query stessa.

## 6.3 Esempio di interrogazione incoerente

Supponiamo ora di voler inserire una nuova interrogazione:

`query` : "Seleziona i massimi dirigenti (`tmanager`) il cui livello è superiore a 20 ".

L'interrogazione viene trasformata nel formalismo ODL fornendo il seguente risultato:

$$\sigma_{\mathrm{V}}(\texttt{query}) \;=\; \texttt{tmanager} \sqcap (\triangle.\texttt{level}\colon 20 \div \infty)$$

L'operazione preliminare è quella di scrivere il file di input contenente l'interrogazione nella corretta sintassi del linguaggio ODL-Designer:

**File** `input.pl`

```
leggiinput(FLAG) :-
aggiungi(s([virt,query,=,tmanager,&,/,\,'[',level,:,level,:,20,-,']',
          ])FLAG),
aggiungi(fine,FLAG).
```

Come prima, l'esecuzione del'ottimizzazione viene attivata dal comando `odb_qopt.` : il sistema elimina i tipi generati dalla precedente espansione semantica ed attiva ODL-Designer. in funzione AGGIUNGI che inserisce la `query` nella base di dati già presente.

### 6.3.1 Prima esecuzione dell'espansione semantica

ODL-Designer, attivato in funzione AGGIUNGI con funzionalità F1 ed F2, legge il file `ges_out.pl` ed inserisce la classe `query` nella base di conoscenza. ODL-Designer inizia poi a controllare la coerenza dello schema ed a generare lo schema canonico.



**Esecuzione di ODL-Designer**

**SCHEMA CANONICO**

$$\begin{aligned}
\overline{\mathbf{T}} &= \{\mathtt{r}_7^a, \mathtt{r}_7^c\} \cup \{b_1 \ldots b_{16}\} \cup \{t_1 \ldots t_{23}\} \\
\overline{\mathbf{C}} &= \{\overline{\texttt{manager}}, \overline{\texttt{department}}, \overline{\texttt{tmanager}}, \overline{\texttt{material}}, \\
&\quad \overline{\texttt{smaterial}}, \overline{\texttt{storage}}, \overline{\texttt{sstorage}}\} \\
\overline{\mathbf{V}} &= \{\texttt{manager}, \texttt{department}, \texttt{tmanager}, \texttt{material}, \\
&\quad \texttt{smaterial}, \texttt{storage}, \texttt{sstorage}, \texttt{query}\} \cup \\
&\quad \{\mathtt{r}_1^a \ldots \mathtt{r}_6^a\} \cup \{\mathtt{r}_1^c \ldots \mathtt{r}_6^c\} \cup \{n_1 \ldots n_6\}
\end{aligned}$$

Riportiamo solo la descrizione dei nuovi tipi classe e valore introdotti da ODL-Designer in quanto gli altri rimangono invariati rispetto allo schema iniziale privo delle interrogazioni.

$$\nu(\texttt{query}) \;=\; \bot$$

**Classificazione minimale**

Come prima ODL-Designer calcola la classificazione minimale rispetto all'ereditarietà per le classi che sono state introdotte dall'utente.

RELAZIONI DI ISA ESPLICITE

$$\begin{aligned}
\texttt{tmanager} &\; ISA \; \texttt{manager} \\
\texttt{smaterial} &\; ISA \; \texttt{material} \\
\texttt{storage} &\; ISA \; \texttt{department} \\
\texttt{sstorage} &\; ISA \; \texttt{storage} \\
\mathtt{r}_1^a &\; ISA \; \texttt{manager} \\
\mathtt{r}_1^c &\; ISA \; \texttt{manager} \\
\mathtt{r}_2^a &\; ISA \; \texttt{manager} \\
\mathtt{r}_2^c &\; ISA \; \texttt{manager} \\
\mathtt{r}_3^a &\; ISA \; \texttt{manager}
\end{aligned}$$

$$\begin{array}{lll} \mathtt{r}_3^c & ISA & \mathtt{manager} \\ \mathtt{r}_4^a & ISA & \mathtt{storage} \\ \mathtt{r}_4^c & ISA & \mathtt{storage} \\ \mathtt{r}_5^a & ISA & \mathtt{storage} \\ \mathtt{r}_5^c & ISA & \mathtt{storage} \\ \mathtt{r}_6^c & ISA & \mathtt{storage} \end{array}$$

RELAZIONI DI ISA CALCOLATE

$$\mathtt{r}_6^a \quad ISA \quad \mathtt{r}_4^a$$

### Interfaccia ODL-Designer GES

A partire dalla nuova descrizione canonica viene ora generato il file:

**File** `ges_in.txt`

```
[virt,query2,=,bottom]
\*
      Riportiamo ora la descrizione dell' insieme GS(query)
*\
[gs,query,(department,prim,),(,storage,prim,),(,sstorage,prim,),
          (,manager,prim,),(,tmanager,prim,),(,material,prim,),
          (,smaterial,prim,),(,r1a,antev,),(,r1c,consv,),(,r2a,antev,),
          (,r2c,consv,),(,r3a,antev,),(,r3c,consv,),(,r4a,antev,),
          (,r4c,consv,), (,r5a,antev,),(,r5c,consv,),(,r6a,antev,),
          (,r6c,consv,),(,n1,virt,),(,n2,virt,),(,n3,virt,),(,n4,virt,),
          (,n5,virt,),(,n6,virt,)]
\*
          Riportiamo infine le descrizioni dei conseguenti di tutte
          le regole (compresi i tipi che compaiono nelle loro descrizioni),
          necessari poiche potenzialmente si possono applicare tutte le regole.
*\
[consv,r1c,=,-manager,&,/,\,t10]
```



```
[type,t10,=,[,level,:,integer,,,name,:,string,,,salary,:,b8,]]
[consv,r2c,=,-manager,&,/,\,t12]
[type,t12,=,[,level,:,integer,,,name,:,string,,,salary,:,b10,]]
[consv,r3c,=,-manager,&,/,\,t14]
[type,t14,=,[,level,:,integer,,,name,:,string,,,salary,:,b12,]]
[consv,r4c,=,-department,&,-storage,&,/,\,t18]
[type,t18,=,[,container,:,string,,,name,:,string,,,qty,:,b6,,,
             stock,:,material,,,managed-by,:,n4,]]
[prim,material,=,-material,&,/,\,t5]
[type,t5,=,[,name,:,string,,,risk,:,integer,]]
[virt,n4,=,-manager,&,-tmanager,&,/,\,t17]
[type,t17,=,[,level,:,b5,,,name,:,string,,,salary,:,b4,]]
[consv,r5c,=,-department,&,-storage,&,/,\,t20]
[type,t20,=,[,container,:,b15,,,name,:,string,,,qty,:,b6,,,
             stock,:,material,,,managed-by,:,n1,]]
[virt,n1,=,-manager,&,/,\,t2]
[type,t2,=,[,level,:,integer,,,name,:,string,,,salary,:,b4,]]
[consv,r6c,=,-department,&,-storage,&,-sstorage,&,/,\,t23]
[type,t23,=,[,container,:,string,,,name,:,string,,,qty,:,b6,,,
             stock,:,n2,,,managed-by,:,n6,]]
[virt,n2,=,-material,&,-smaterial,&,/,\,t6]
[type,t6,=,[,feature,:,string,,,name,:,string,,,risk,:,integer,]]
[virt,n6,=,-manager,&,/,\,t2]
[const,r7c,=,[,name,:,b2,,,salary,:,b3,]]
```

### Esecuzione di GES

GES si accorge che l'interrogazione è incoerente e quindi termina con valore di ritorno uguale a 4 senza applicare le regole.

### Visualizzazione

Durante l'esecuzione si visualizza:

```
Esecuzione di GES
Controllo sintattico ...  OK
Controllo incoerenza ...
WARNING: esiste almeno un tipo incoerente,
```

```
non si procede con l'applicazione delle regole.

la query e':  [tmanager,&,/,\,[,level,:,20,-,max,]]
```

### Controllo di terminazione

ODB-QOptimizer legge il valore di ritorno di GES che vale 4: da ció capisce che l'interrogazione non ha subito espansione semantica essendo incoerente. L'ottimizzazione è terminata avendo trovato che l'interrogazione rappresenta una classe con dominio vuoto (e questo risultato viene trovato senza accedere fisicamente al database).

# Appendice A

# GES: il sorgente

```
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#include <string.h>
#include <time.h>

#define MAX_LISTE 100
#define MAX 500
#define LEN_WORD 50


/* Il FILE contiene il programma completo per la gestione dell'espansione
   semantica (GES).
   Il modulo analisi() contiene la parte riguardante la trasformazione della
   stringa letta da file esterno (puntato da *streamin) nella struttura dati
   a lista che viene utilizzato negli altri moduli del programma GES.
   Il modulo applicaregole() cerca le regole applicabili e le applica generando
   il corretto FILE di out (puntato da *stream_out) che viene successivamente
   letto da ODL-Designer.
   Questa versione gestisce l'applicazione di regole sia virtuali che type,
   con la convenzione che gli antecedenti ed i conseguenti di tali regole
   sono classificati come antev, antet, consv, const.
*/

int analisi(void);
int caso_ennupla(int i);
int caso_gs(int j);
int caso_insieme(int i);
int caso_nome_fittizio(int i);
int caso_nome_virtuale(int i);
int caso_sequenza(int i);
int caso_tipo_semplice(int i);
int confronta_nomi(char DATO1[], char DATO2[]);
int converti_stringa_odl_ges(int i, int j);
int ges(void);
int scelta(void);
int seleziona_classe(char VAR[]);
void aggiungi_lista_classe_gs(char NOME1[], char NOME2[], char NOME3[],
      char NOME4[], int i ,int j);
void aggiungi_regola(char VAR[], int i);
```



```
int applica_regole(void);
void azzera_list_group (void);
void converti_ges_odl(int modo_conversione);
void errore(void);
void leggi_nome(char name[], int modo);
void leggi_stringa_ascii(char NOME[]);
void scrivi_nome(char DATO[] , FILE *stream);
void svuota_liste(void);
void togli_F_ante(void);
void togli_F_cons(void);
void togli_F_first_classe(int i);
void togli_F_first_gs(int i);
void unisci_nome (char DATO1[], char DATO2[]);

typedef struct elemento {

  char    NOME[LEN_WORD] ;
  char    TIPO[LEN_WORD];
  struct  elemento  *NEXT;
  char    C_TIPO[LEN_WORD];
  char   D_ATTRIBUTO[LEN_WORD];

} ELEMENTO;

int converti_ges_odl_tipo(int modo_conversione, ELEMENTO *puntaclasse1,
  int cerco_regole);
int trova_regole(ELEMENTO *tipo);


typedef struct regola {

  char    NOME[LEN_WORD] ;
  struct  regola  *NEXT;

} REGOLA;

ELEMENTO  *first_classe[MAX_LISTE],
          *last_classe[MAX_LISTE],
          *first_gs[MAX_LISTE],
          *last_gs[MAX_LISTE],
```

```
          *gruppo,
          *puntaclasse,
          dato,dato1;

REGOLA   *ante_group,
          *cons_group,
  *ante_group_last,
          *cons_group_last,
  *navigo_cons;

FILE      *stream_in,
          *stream_out,
          *stream_out1;

char      a, b, *riga_letta,
          nome_query[LEN_WORD];

int       offset_riga,
          regole_applicate = 0,
          applicazione_regole = 0,
          query_length = 0,
          conta_classe = 0,
  conta_gs = 0,
          incoerenti = 0,
          no_errore;

clock_t   start1,
          stop1,
          start2,
          stop2;

int main(void)
{
  /*
     Programmma principale che genera l'espansione semantica chiamando ges()
     e simulando il controllo di terminazione verificando il parametro di
     ritorno di GES.
     Il main deve essere tolto quando si usa con ODL-Designer.
     */
```



```
  int z = 0;

  while (z == 0)
    {
      z = ges();

      if (z == 0) {
        printf("\nTempo analisi sintassi e generazione liste = %6.3f msec.",
       (stop1 - start1) * .001);

printf("\nTempo applicazione regole = %6.3f msec.\n",
       (stop2 - start2) * .001);

printf("\nTempo totale = %6.3f msec.\n",
       (stop2 - start1) * .001);
        printf("Premi return dopo aver salvato il nuovo file di input\n");
        getchar();
      }
    }


  printf("\nTempo analisi sintassi e generazione liste = %6.3f msec.",
 (stop1 - start1) * .001);

  printf("\nTempo applicazione regole = %6.3f msec.\n",
 (stop2 - start2) * .001);

  switch (z)
    {

    case '1' :
      printf("Ottimizzazione corretta\n");
      break;

    case '2' :
      printf("Sintassi scorretta\n");
      break;

    case '3' :
      printf("Sintassi scorretta\n");
```

```
      break;

    case '4' :
      printf("Incoerenza\n");
      break;

    }

  return z;
}

int ges(void)
{
  /*
   Programma GES che ottiene l'espansione semantica della query
   leggendo un FILE puntato da *stream_in e generando infine un FILE
   puntato da *stream_out che contiene le nuove classi alle quali sono
   state applicate regole e che verra' utilizzato da ODL-Designer.

   Il programma termina con valore intero:

   0   se l'esecuzione e' avvenuta correttamente con applicazione di
       regole, visualizzando il file di output generato.

   1   se l'esecuzione e' avvenuta correttamente senza applicare regole ,
       visualizzando il file di output generato.

   2   se i dati di ingresso non sono corretti, visualizzando un messaggio di
       errore,

   3   se i dati di ingresso sono incompleti, visualizzando un messaggio di
       errore,

   4   se tra i tipi in ingresso ne esiste almeno uno incoerente.

   */

  int   z, ris_analisi;

  stream_in = fopen("ges_in.txt", "r");
```



```
  stream_out = fopen("input.pl","w+");

  printf("\nEsecuzione di GES");
  printf("\nControllo sintattico ... ");

  no_errore = 0;
  start1 = clock();

  ris_analisi = analisi();
  stop1 = clock();

  fclose(stream_in);

  if (ris_analisi == 1)
    {
      printf("Non si procede con l'applicazione delle regole.\n");
      z = 2;
    }
  else
    {
      printf("OK");
      printf("\nControllo incoerenza ... ");
      if (ris_analisi == 4)
{
  printf("\nWARNING: l'interrogazione e' incoerente,\n
                 non si procede con l'applicazione delle regole.\n");
  z = 4;
}
      else
{
  if (z == 5)
    printf("\nErrore non riconosciuto");
  else
    {
      printf("OK");
      printf("\nApplicazione delle regole ... \n");
      start2 = clock();
      z = applica_regole();
    }
}
```

```
    }

  svuota_liste();
  azzera_list_group();
  stop2 = clock();

  return z;
}


int analisi(void)
{
  /*
   Il modulo analisi() scandisce l'intero FILE di input ottenendo la
   completa conversione delle CLASSI e degli insiemi GS nelle liste
   tipo GES.
   Le liste delle CLASSI sono memorizzate in un array di puntatori di
   nome *first_classe[i], il quale punta al primo ELEMENTO dell'iesima
   lista, mentre gli insiemi GS in liste puntate da *first_gs[i].
   */

  int   z;

  conta_classe = 0;
  conta_gs = 0;

  /* Viene visualizzato il file di input
   */
  /*=======================================================
    Questa parte dovra' essere tolta nel momento di esecuzione
    con ODL.

    printf("IL FILE PRESCELTO CONTIENE IL TESTO :\n");

    do {
    a = fgetc(stream_in);
    putchar(a);
    } while (a != EOF);

    printf("\n\nPREMI RETURN PER CONTINUARE");
```



```
    getchar();
    */
  /* ritorna all'inizio del file */
  fseek(stream_in,0,0);
  /* ================================================== */
  /* Prima di cominciare la conversione da ODL a GES occorre
     svuotare le eventuali liste presenti.
     */
  svuota_liste();

  do {

    z = converti_stringa_odl_ges(conta_classe,conta_gs);

    /* La variabile r e' un intero di ritorno dal sottomodulo
       converti_stringa_odl_ges(int i, int j) che vale
       0 se viene convertita una classe,
       1 se viene convertito un insieme gs,
       2 se viene trovato un errore sintattico.
       */

    if (z == 0)
      conta_classe++;
    else
      if (z == 1)
conta_gs++;

  }  while ((z == 0 || z == 1) && !feof(stream_in)) ;

  /*
     Il programma continua a convertire una classe (con il while)
     finche' le classi (corrette sintatticamente) sono separate da
     punto e virgola oppure il FILE e' terminato.
     */


  if (*(riga_letta + offset_riga) == '.' && z == 3 && feof(stream_in))
    {

      /* Solo se l'ultimo carattere e' il punto si considera una
```

```
 conversione completata in modo corretto
 */

      if ( z != 2 && incoerenti > 0 )
return 4;

      return 0;
    }
  else
    {
      if ( z == 2 )
return 1;

      if (!feof(stream_in))
{
  printf("\nERRORE: La sintassi di terminazione del file di input
                  e' scorretta.\n");
  return 1;
}
    }
  return 5;
}

int converti_stringa_odl_ges(int i, int j)
{
  /*
   Il sottomodulo legge il FILE puntato da *stream_in e genera la conversione
   da ODL-Designer a GES di una singola classe o di un insieme gs.
   Se viene convertita una classe la lista creata e' puntata da
   *first_classe[i],
   se viene convertito un insieme gs la lista e' puntata da *first_gs[j].

   Ritorna  0 se viene convertita una classe correttamente,
    1 se viene convertito un insieme gs correttamente,
    2 se trova dati non sintatticamente corretti.
  */

  char   change_name_query[LEN_WORD];
  int    z, sig = 10, cont = 0;
  double query_number;
```



```
  first_classe[i] = NULL;
  last_classe[i] = NULL;
  first_gs[j] = NULL;
  last_gs[j] = NULL;

  riga_letta = malloc (MAX);

  for (z = 1; z < MAX; z++)
    *(riga_letta + z) = '\0';

  fscanf(stream_in,"%s\n",riga_letta);

  /*
     Viene letta un'intera riga del file e memorizzata dal
     puntatore *riga_letta.
  */

  offset_riga = 0;
  /*
     L'intero offset_riga viene utilizzato per muoversi all'interno
     della riga letta da file di input.
  */

  if (*riga_letta == '.')
    {
      for (z = 1; z < MAX; z++)
{
  if (*(riga_letta + z) != '\0')
    {
      errore();
      return 2;
    }
}
      return 3;
    }
  if (strncmp(riga_letta,"[",1))
    {
      errore();
      return 2;
```

```
    }

  leggi_nome(dato.C_TIPO,0);

  if ( *(riga_letta + offset_riga) != ',')
    {
      errore();
      return 2;
    }

  leggi_nome(dato.NOME,0);

  if (i == 0)
    {
      while ( isdigit(dato.NOME[query_length]) == 0 &&
      dato.NOME[query_length] != '\0' ) {

nome_query[query_length] = dato.NOME[query_length];
query_length++;
      }
      for (cont = query_length; cont < LEN_WORD; cont++)
nome_query[cont] = '\0';
    }

  if (!confronta_nomi(dato.C_TIPO , "gs"))
    {
      if (strncmp(dato.NOME,nome_query,query_length) == 0)
{
  for (z = 0; z < LEN_WORD; z++)
    change_name_query[z] = dato.NOME[z + query_length];

  for (z = query_length; z < LEN_WORD; z++)
    dato.NOME[z] = '\0';

  query_number = atof(change_name_query);
  query_number++;
  gcvt(query_number,sig,change_name_query);
  strcat(dato.NOME,change_name_query);
}
```



```
      z = caso_gs(j);
    }
  else
    {
      if (strncmp(riga_letta + offset_riga,",=,",3))
{
  errore();
  return 2;
}
      else
{
  /* Continuo poiche' la sintassi  ",=," e' corretta */

  offset_riga = offset_riga + 2;

  /* confronta_nomi(s1,s2) ritorna
     0 se le stringhe coincidono,
     1 altrimenti.
     */

  if ((confronta_nomi(dato.C_TIPO,"prim")) &&
      (confronta_nomi(dato.C_TIPO,"virt")) &&
      (confronta_nomi(dato.C_TIPO,"type")) &&
      (confronta_nomi(dato.C_TIPO,"b_type")) &&
      (confronta_nomi(dato.C_TIPO,"antev")) &&
      (confronta_nomi(dato.C_TIPO,"antet")) &&
      (confronta_nomi(dato.C_TIPO,"consv")) &&
      (confronta_nomi(dato.C_TIPO,"const")) )
    {
      errore();
      return 2;
    }
  else
    {
      /* continuo perche' il C_TIPO e' corretto */

      if (strncmp(dato.NOME,nome_query,query_length) == 0)
{
  for (z = 0; z < LEN_WORD; z++)
    change_name_query[z] = dato.NOME[z + query_length];
```

```
  for (z = query_length; z < LEN_WORD; z++)
    dato.NOME[z] = '\0';

  query_number = atof(change_name_query);
  query_number++;
  gcvt(query_number,sig,change_name_query);
  strcat(dato.NOME,change_name_query);
}

      aggiungi_lista_classe_gs
(dato.NOME,"\0",dato.C_TIPO,"N_Classe",i,0);

      if (*(riga_letta + (++offset_riga)) == '/' ||
  *(riga_letta +offset_riga) == '-')
{
  do
    {
      if (*(riga_letta + offset_riga) == '/')
z = caso_nome_virtuale(i);

      else
{
  if (*(riga_letta + offset_riga) == '-')
    (z = caso_nome_fittizio(i));
}

      if (z == 2)
{
  errore();
  return 2;
}

    } while (*(riga_letta + offset_riga) == '/' ||
     *(riga_letta + offset_riga) == '-');

}
      else
{
  switch (*(riga_letta + (offset_riga)))
```



```
    {
    case '[' :
      offset_riga++;
      z = 0;
      if (*(riga_letta + offset_riga) == ',')
{
  while (*(riga_letta + offset_riga) == ','
 &&  z != 2 )
    {
      z = caso_ennupla(i);
    }
}
      else
{
  leggi_stringa_ascii(dato.NOME);

  if (*(riga_letta + offset_riga) != ']')
    {
      errore();
      z = 2;
    }
  aggiungi_lista_classe_gs (dato.NOME,"\0","\0",
    "Semplice",i,0);

  offset_riga++;
  z = 0;
}
      break;

    case '{' :
      z = caso_insieme(i);
      break;

    case '<' :
      z = caso_sequenza(i);
      break;

    default  :
      z = caso_tipo_semplice(i);
      if (z == 4)
```

```
{
  incoerenti++;
  z = 0;
}
    }
}
    }
}
    }

  if (strncmp(riga_letta + offset_riga,"]\0",2))
    {
      if (no_errore == 0)
errore();

      return 2;
    }

  return z;

}

int caso_nome_virtuale(int i)
{
  /*
   Il sottomodulo analizza la sintassi di un nome di oggetto
   (definito da /,\), che e' l'ultima classe della congiunzione
   per definire una classe virt o prim.
   Ritorna 0 se la conversione del nome virtuale e' corretta,
   2 altrimenti.
   */

  if (strncmp(riga_letta + (++offset_riga),",\\,",3)) \
    {
      errore();
      return 2;
    }
  else
    {
      offset_riga = offset_riga + 2;
```



```
      leggi_nome(dato.NOME,0);
      aggiungi_lista_classe_gs(dato.NOME,"\0","\0","Nome_Virtuale",i,0);

      /*  La conversione e' terminata con successo avendo
  trovato classi oggetto.
  */
      return 0;
    }
}



int caso_nome_fittizio(int i)
{
  /*
     Il sottomodulo analizza la sintassi di atomo fittizio presente nella
     definizione di classi primitive o virtuali.
     Ritorna 0 se la conversione del nome fittizio e' corretta,
     2 altrimenti.
     */

  leggi_nome(dato.NOME,1);
  aggiungi_lista_classe_gs(dato.NOME,"\0","\0","Atomo_Fittizio",i,0);

  if (!strncmp(riga_letta + offset_riga,",&,",3))
    {
      offset_riga = offset_riga + 3;
      return 0;

      /* Il sottomodulo ha trovato un atomo fittizio
 corretto e si aspetta ancora altre classi congiunte
 */

    }

  /* La sintassi e' scorretta perche' dopo un atomo fittizio
     occorre sempre congiungere una classe oggetto (oppure
     altri atomi fittizi).
     */
```

```
  errore();
  return 2;
}


int caso_ennupla(int i)
{
  /* Il sottomodulo analizza la sintassi di un campo di un tipo attributo,
     individuandone sia il NOME che il D_ATTRIBUTO.
     Ritorna 0 se la conversione del campo della ennupla e‘ corretta,
     2 altrimenti.
     */

  if (strncmp(riga_letta + (offset_riga),",",1))
    {
      errore();
      return 2;
    }
  else
    {
      leggi_nome(dato.NOME,0);

      if (strncmp(riga_letta + offset_riga,",:,",3))
{
  errore();
  return 2;
}

      else
{
  offset_riga = offset_riga + 2;
  leggi_nome(dato.D_ATTRIBUTO,0);

  if (*(riga_letta + offset_riga) != ’,’)
    {
      errore();
      return 2;
    }
  else
    {
```



```
      aggiungi_lista_classe_gs(dato.NOME,dato.D_ATTRIBUTO,
       "\0","Attributo",i,0);
      offset_riga++;
      if (!strncmp(riga_letta + offset_riga,"]",1))
{
  offset_riga++;
  return 0;
}
      /* La conversione e’ terminata avendo
 trovato un tipo Attributo.
 */
      else
{
  /* Ancora un altro CAMPO */

  if (*(riga_letta + (offset_riga)) != ’,’)
    {
      errore();
      return 2;
    }
  else
    offset_riga++;
}
    }
}
    }
  return 0;
}

int caso_gs(int j)
{
  /*
     Il sottomodulo analizza la sintassi di un insieme gs
     e ritorna
     1 se la conversione  corretta,
     2 altrimenti.
     */

  aggiungi_lista_classe_gs(dato.NOME,"\0",dato.C_TIPO,"\0",j,1);
```

```
  if (!strncmp(riga_letta + offset_riga,",(,",3))
    {
      /*
 L'insieme gs non e' vuoto
 */
      do {

offset_riga = offset_riga + 3;

leggi_nome(dato.NOME,1);

if (*(riga_letta + offset_riga) != ',')
  {
    errore();
    return 2;
  }
else
  {
    leggi_nome(dato.C_TIPO,0);
    aggiungi_lista_classe_gs(dato.NOME,"\0",dato.C_TIPO,"\0",j,1);
    if (strncmp(riga_letta + offset_riga,",)",2))
      {
errore();
return 2;
      }

    offset_riga = offset_riga + 2;
  }

      }  while  (!strncmp(riga_letta + offset_riga,",(,",3));

    }
  return 1;
}


int caso_insieme(int i)
{
  /*
     Il sottomodulo analizza la sintassi di un tipo insieme.
```



```
     Ritorna 0 se la conversione del tipo insieme e' corretta,
     2 altrimenti.
     */

  if (strncmp(riga_letta + (++offset_riga),",",1))
    {
      errore();
      return 2;
    }
  else
    {
      leggi_nome(dato.NOME,0);
      aggiungi_lista_classe_gs(dato.NOME,"\0","\0","Insieme",i,0);

      if (strncmp(riga_letta + offset_riga,",}",2))
{
  errore();
  return 2;
}
      else
{
  offset_riga = offset_riga + 2;
  return 0;

  /* La conversione e' terminata correttamente
     avendo trovato un tipo insieme.
     */
}
    }
}

int caso_sequenza(int i)
{
  /*
     Il sottomodulo analizza la sintassi di un tipo sequenza.
     Ritorna 0 se la conversione del tipo sequenza e' corretta,
     2 altrimenti.
     */

  if (strncmp(riga_letta + (++offset_riga),",",1))
```

```
    {
      errore();
      return 2;
    }
  else
    {
      leggi_nome(dato.NOME,0);
      aggiungi_lista_classe_gs(dato.NOME,"\0","\0","Sequenza",i,0);

      if (strncmp(riga_letta + offset_riga,",>",2))
{
  errore();
  return 2;
}
      else
{
  offset_riga = offset_riga + 2;
  return 0;
  /* La conversione e' terminata correttamente
     avendo trovato un tipo insieme.
     */
}
    }
}

int caso_tipo_semplice(int i)
{
  /*
     Il sottomodulo analizza la sintassi di un nome singolo.
     Ritorna 0 se la conversione del nome e‘ corretta,
     2 altrimenti.
     */

  leggi_nome(dato.NOME,1);

  if (!strncmp(riga_letta + offset_riga,",-,",3))
    {
      offset_riga = offset_riga + 2;

      leggi_nome(dato1.NOME,0);
```



```
      unisci_nome(dato.NOME,dato1.NOME);

      /* La conversione e' terminata correttamente
 avendo trovato un tipo range.
 */
    }

  /* La conversione  terminata correttamente
     avendo trovato un tipo semplice.
     */

  if ( !strncmp(dato.NOME,"bottom",6) )
    return 4;

  if ( dato.NOME[0] == '\0')
    {
      errore();
      return 2;
    }

  aggiungi_lista_classe_gs (dato.NOME,"\0","\0","Semplice",i,0);
  return 0;
}


void errore(void)

{
  printf("\nERRORE : La sintassi della stringa n. %d e' scorretta:\n%s \n",
 conta_gs + conta_classe +1, riga_letta);
  no_errore++;
  return ;
}

void leggi_nome(char name[],int modo)
{
  /*
     Il sottomodulo legge una stringa alfanumerica da FILE puntato
     da *stream_in e la scrive nel vettore di chiamata name[].
     Con modo = 0 Scandisce dal prossimo char di ingresso
```

```
     Con modo = 1 Scandisce dal presente char di ingresso
     */

  int   n = 0 ,i;

  if (modo == 0)
    (offset_riga++) ;

  while (isalnum(*(riga_letta + offset_riga)) ||
 *(riga_letta + offset_riga) == '_' ||
 *(riga_letta + offset_riga) == '-' ||
 *(riga_letta + offset_riga) == '.' ||
 *(riga_letta + offset_riga) == '"' )
    {
      name[n++] = *(riga_letta + (offset_riga++));
    }

  for (i = n ; i < LEN_WORD ; name[i++] = '\0');
}

void leggi_stringa_ascii(char NOME[])
{
  /*
     Il sottomodulo legge una stringa di caratteri rappresentati
     dal codice ASCII dei singoli caratteri separati tra loro da
     una virgola e terminante con la parentesi quadra (]).
     La stringa  puntata da riga_letta e viene memorizzata in NOME[].
     */

  int   n = 0 ,i;

  NOME[n++] = *(riga_letta + offset_riga - 1);
  do {

    NOME[n++] = *(riga_letta + offset_riga);

  } while (isdigit(*(riga_letta + (++offset_riga))) ||
   *(riga_letta + offset_riga) == ',' );
```



```
  NOME[n++] = *(riga_letta + offset_riga);
  for (i = n ; i < LEN_WORD ; NOME[i++] = '\0');
  return ;
}

int applica_regole(void)
{
  /*
     Il modulo cerca le regole applicabili scandendo gli insiemi GS
     ricevuti da ODL e, nel caso esistano, le applica scrivendo le nuove
     classi nel FILE puntato da *stream_out.
     Ritorna:

0    Se ho applicato almeno una regola con esecuzione corretta,

2    Se non ho applicato alcuna una regola con esecuzione corretta,

3    Se manca la descrizione i un tipo o di un GS

*/

  int   q = 0;

  regole_applicate = 0;
  fputs("leggiinput(FLAG) :-\n",stream_out);
  fputs("aggiungi(s([",stream_out);

  converti_ges_odl_tipo(0,first_classe[0],0);
  if (applicazione_regole == 1)
    {
      printf("\nL'esecuzione non e' stata completata correttamente.\n");
      return 3;
    }

  fputs("]),FLAG),\n",stream_out);

  if (regole_applicate == 0)
    {
      printf("Non ho applicato nessuna regola,\n
              l'ottimizzazione e' terminata correttamente.\n");
```

```
      q = 1;
    }

  fputs("aggiungi(fine,FLAG).",stream_out);

  /*
    Chiudo il file di output
    */

  fclose(stream_out);
  return q;

}



int confronta_nomi(char DATO1[],char DATO2[])
{
  /*
     Il sottomodulo confronta i due array di input DATO1[] e DATO2[]
     (al piu' di LEN_WORD char) trascurando gli eventuali caratteri nulli,
     cioe'  '\0'.
     Ritorna 0 se le stringhe coincidono,
     1 altrimenti.
     */

  int   k = 0, c;

  while (DATO2[k] != '\0')
    if (DATO1[k] != DATO2[k++])
      return 1;

  for (c = k; c < LEN_WORD; c++)
    if (DATO1[c] != '\0')
      return 1;

  return 0;
}
```



```
void unisci_nome( char DATO1[], char DATO2[])
{
  /*
     Il sottomodulo unisce in un solo array di caratteri le parole inviate
     come DATO1[] e DATO2[] interponendo i caratteri ,-, tra le due parole.
     */

  int c = 0, k = 0;

  while ( DATO1[c] != '\0')
    dato.NOME[c] = DATO1[c++];

  dato.NOME[c++] = ',';
  dato.NOME[c++] = '-';
  dato.NOME[c++] = ',';

  while (DATO2[k] != '\0')
    dato.NOME[c++] = DATO2[k++];
}

void aggiungi_lista_classe_gs
(char NOME1[], char NOME2[], char NOME3[], char NOME4[], int i, int j)
{
  /*
     Il sottomodulo interpreta i NOMI inviati come i campi di una struttura
     di tipo ELEMENTO ed aggiunge in coda ad una lista l'intera struttura
     cosi' interpretata.
     La lista che viene aggiornata dipende dagli interi i ed j inviati :

     j = 0 viene aggiornata la lista di tipo CLASSE puntata da *first_classe[i]
     j = 1 viene aggiornata la lista di tipo GS puntata da *first_gs[i]
     */

  ELEMENTO   *newnode;
  int   c;

  newnode = (struct elemento *) malloc(sizeof *newnode);
  newnode -> NEXT = NULL;

  for (c = 0; c < LEN_WORD; c++)
```

```
    {
      newnode -> NOME[c] = '\0';
      newnode -> D_ATTRIBUTO[c] = '\0';
      newnode -> C_TIPO[c] = '\0';
      newnode -> TIPO[c] = '\0';
    }

  strcpy(newnode -> NOME, NOME1);
  strcpy(newnode -> D_ATTRIBUTO, NOME2);
  strcpy(newnode -> C_TIPO, NOME3);
  strcpy(newnode -> TIPO, NOME4);

  if (j == 0)
    {
      if (first_classe[i] == NULL)
{
  first_classe[i] = newnode;
  last_classe[i] = newnode;
}
      else
{
  last_classe[i] -> NEXT = newnode;
  last_classe[i] = newnode;
}
    }
  else
    {
      if (j == 1)
{
  if (first_gs[i] == NULL)
    {
      first_gs[i] = newnode;
      last_gs[i] = newnode;
    }
  else
    {
      last_gs[i] -> NEXT = newnode;
      last_gs[i] = newnode;
    }
}
```



```
    }
}

int seleziona_classe( char VAR[])
{
  /*
     Il sottomodulo cerca sequenzialmente nell'array first_classe[] l'elemento
     che ha NOME = VAR.
     Se la ricerca ha successo pone *puntaclasse coincidente con
     l'elemento trovato e ritorna con il valore 0 ,in caso diverso
     torna con 1.
     */

  int   i = 0;

  while ( first_classe[i] != NULL ) {

    if ( ( confronta_nomi(first_classe[i] -> NOME , VAR)) == 0 )
      {
puntaclasse = first_classe[i];
return 0;
      }

    i++;
  }

  return 1;
}

void aggiungi_regola( char VAR[], int i)
{
  /*
     Il sottomodulo aggiunge una struttura di tipo REGOLA in coda ad una
     lista di blocchi di REGOLE.
     Se i = 0  aggiunge la regola come antecedente nella lista *ante_group
        i = 1  aggiunge la regola come conseguente nella lista *cons_group
*/

  REGOLA   *newnode;
  int   c;
```

```
  newnode = (struct regola *) malloc(sizeof *newnode);
  newnode -> NEXT = NULL;

  for (c = 0; c < LEN_WORD; newnode -> NOME[c++] = '\0');

  strcpy(newnode -> NOME, VAR);

  if (i == 0)
    {
      if (ante_group == NULL)
{
  ante_group = newnode;
  ante_group_last = newnode;
}
      else
{
  ante_group_last -> NEXT = newnode;
  ante_group_last = newnode;
}
    }
  else
    {
      if (i == 1)
{
  if (cons_group == NULL)
    {
      cons_group = newnode;
      cons_group_last = newnode;
    }
  else
    {
      cons_group_last -> NEXT = newnode;
      cons_group_last = newnode;
    }
}
    }
}

void svuota_liste(void)
```



```
{
  /* Il sottomodulo svouta le liste *first_classe[] e *first_gs[]
   */

  int   i;

  for (i = 0; i < MAX_LISTE; i++)
    {
      while (first_classe[i] != NULL)
togli_F_first_classe(i);

      while (first_gs[i] != NULL)
togli_F_first_gs(i);
    }

  return ;
}

void togli_F_first_classe(int i)
{
  /* Il sottomodulo toglie il primo elemento della lista *first_classe[i]
   */

  ELEMENTO   *temp = first_classe[i];

  if (first_classe[i] == NULL)
    return;

  else
    {
      if (first_classe[i] == last_classe[i])
{
  first_classe[i] = NULL;
  last_classe[i] = NULL;
}
      else
first_classe[i] = temp -> NEXT;

      free ((struct elemento *) temp);
      return ;
```

```
    }
}

void togli_F_first_gs(int i)
{
  /* Il sottomodulo toglie il primo elemento della lista *first_gs[i]
   */

  ELEMENTO   *temp = first_gs[i];

  if (first_gs[i] == NULL)
    return;

  else
    {
      if (first_gs[i] == last_gs[i])
{
  first_gs[i] = NULL;
  last_gs[i] = NULL;
}
      else
first_gs[i] = temp -> NEXT;

      free ((struct elemento *) temp);
      return ;
    }
}



void azzera_list_group (void)
{
  /* Il sottomodulo azzera le liste puntate da *ante_group e *cons_group
   */
  while (ante_group != NULL )
    togli_F_ante();

  while (cons_group != NULL )
    togli_F_cons();
```



```
  return ;
}

void togli_F_ante()
{
  /* Il sottomodulo toglie il primo elemento della lista *ante_group
   */

  REGOLA   *temp = ante_group;

  if (ante_group == NULL)
    return;

  else
    {
      if (ante_group == ante_group_last)
{
  ante_group = NULL;
  ante_group_last = NULL;
}
      else
ante_group = temp -> NEXT;

      free ((struct regola *) temp);
      return ;
    }
}

void togli_F_cons()
{
  /* Il sottomodulo toglie il primo elemento della lista *cons_group
   */

  REGOLA   *temp = cons_group;

  if (cons_group == NULL)
    return;

  else
    {
```

```
      if (cons_group == cons_group_last)
{
  cons_group = NULL;
  cons_group_last = NULL;
}
      else
cons_group = temp -> NEXT;

      free ((struct regola *) temp);
      return ;
    }
}

int converti_ges_odl_tipo(int modo_conversione, ELEMENTO *puntaclasse1,
  int cerco_regole)
{
  /*
     Il sottomodulo converte una lista di tipo GES puntata da *puntaclasse1
     in una stringa secondo la sintassi di ODL-Designer.

     Se modo_conversione=0 la conversione e' globale,
     Se modo_conversione=1 non viene convertito il primo elemento della lista.

     Per la conversione di *puntaclasse per ciascun componente si ricerca
     la sua desrizione e si itera il procedimento ottenendo cosi
     l'intera descrizione del tipo *puntaclasse1.
     Infine viene chiamato il sottomodulo trova_regole(*p) che applica le
     regole al tipo in considerazione.

     Ritorna:

     0   Se esegue correttamante,

     1   Se manca la descrizione di un tipo,

     2   Se manca la descrizione di un GS.
     */

  int   i, f;
  ELEMENTO *tipo_now;
```



```
  tipo_now = puntaclasse1;

  if (modo_conversione == 0)
    {
      scrivi_nome(puntaclasse1 -> C_TIPO,stream_out);
      fputs(",",stream_out);
      scrivi_nome(puntaclasse1 -> NOME,stream_out);
      fputs(",=,",stream_out);
    }

  puntaclasse1 = puntaclasse1 -> NEXT;
  i = 0;

  while (puntaclasse1 != NULL)
    {
      if (i != 0)
fputs(",",stream_out);

      if ((confronta_nomi(puntaclasse1 -> TIPO,"Nome_Virtuale")) == 0)
{
  fputs("/,\\,",stream_out);
  if (seleziona_classe(puntaclasse1 -> NOME) == 1)
    {
      printf("\nERRORE: Nel file di input manca la descrizione
                      del tipo %s.",puntaclasse1 -> NOME);
      applicazione_regole = 1;
      return 1;
    }
  if(converti_ges_odl_tipo(1,puntaclasse,cerco_regole)== 2)
    return 2;

  if (puntaclasse1 -> NEXT != NULL)
    errore();
}
      else
{
  if ((confronta_nomi(puntaclasse1 -> TIPO,"Attributo")) == 0 )
    {
      if ( i == 0)
```

```
fputs("'[',",stream_out);

      scrivi_nome(puntaclasse1 -> NOME,stream_out);
      fputs(",:,",stream_out);

      if (seleziona_classe(puntaclasse1 -> D_ATTRIBUTO) == 0)
{
  if(converti_ges_odl_tipo(1,puntaclasse,cerco_regole)== 2)
    return 2;

}
      else
scrivi_nome(puntaclasse1 -> D_ATTRIBUTO,stream_out);

      if (puntaclasse1 -> NEXT == NULL)
fputs(",']'",stream_out);

      else
fputs(",',',stream_out);
    }
  else
    {
      if ((confronta_nomi(puntaclasse1 -> TIPO,"Insieme"))== 0 )
{
  fputs("'{',",stream_out);
  if (seleziona_classe(puntaclasse1 -> NOME) == 0)
    {
      if(converti_ges_odl_tipo(1,puntaclasse,cerco_regole)== 2)
return 2;
    }
  else
    {
      scrivi_nome(puntaclasse1 -> NOME,stream_out);
    }

  fputs(",'}'",stream_out);
}
      else
{
  if ((confronta_nomi(puntaclasse1 -> TIPO,"Sequenza")) == 0 )
```



```
    {
      fputs("'<',",stream_out);
      if (seleziona_classe(puntaclasse1 -> NOME) == 0)
{
  if(converti_ges_odl_tipo(1,puntaclasse,
   cerco_regole)== 2)
    return 2;
}
      else
{
  scrivi_nome(puntaclasse1 -> NOME,stream_out);
}
      fputs(",'>'",stream_out);
    }
  else
    {
      if (confronta_nomi
  (puntaclasse1 -> TIPO,"Atomo_Fittizio") == 0 )
{
  fputs("'",stream_out);
  scrivi_nome(puntaclasse1 -> NOME,stream_out);
  fputs("'",stream_out);
}
      else
{
  scrivi_nome(puntaclasse1 -> NOME,stream_out);
}
      if (puntaclasse1 -> NEXT != NULL)
fputs(",&",stream_out);
    }
}
    }
}
      puntaclasse1 = puntaclasse1 -> NEXT;
      i++;
    }

  if (cerco_regole == 0)
    {
      f = trova_regole(tipo_now);
```

```
      if ( f ==  2)
{
  applicazione_regole = 1;
  return 2;
}
    }

  return 0;
}



int trova_regole(ELEMENTO *tipo)
{
  /*

    Ritorna:

         0    Se non esistono regole applicabili,

 1    Se esistono regole applicabili,

 2    Se manca la decsrizione di un GS.
 */

  char   regola[LEN_WORD];
  int    i, cont = 0, regole_applicabili;

  /*Devo azzerare le liste ante_group e cons_group */

  azzera_list_group();

  for(i=0; i<LEN_WORD; i++)
    regola[i] = '\0';

  i = 0;
  while ( first_gs[i] != NULL && cont == 0)
    {
      if (( confronta_nomi(first_gs[i] -> NOME , tipo -> NOME)) == 0 )
{
```



```
  tipo = first_gs[i];
  cont = 1;
}
      else
i++;
    }

  if (cont == 0)
    {
      printf("\nERRORE: Nel file di input manca la descrizione del GS(%s).",
     tipo -> NOME);
      return 2;
    }
  tipo = tipo -> NEXT;

  while (tipo != NULL)
    {
      if ((confronta_nomi(tipo -> C_TIPO,"antev")) == 0 ||
  (confronta_nomi(tipo -> C_TIPO,"antet")) == 0)
{
  cont = 0;

  do {
    tipo -> NOME[cont] = tipo -> NOME[cont];
  } while (tipo-> NOME[++cont] != '\0');

  for (cont=cont-1; cont < LEN_WORD; cont++ )
    tipo -> NOME[cont] ='\0';

  aggiungi_regola(tipo -> NOME,0);
}
      else
{
  if ((confronta_nomi(tipo -> C_TIPO,"consv")) == 0 ||
      (confronta_nomi(tipo -> C_TIPO,"const")) == 0)
    {
      cont = 0;
      do {
tipo -> NOME[cont] = tipo -> NOME[cont];
      } while (tipo-> NOME[++cont] != '\0');
```

```
      for (cont=cont-1; cont < LEN_WORD; cont++ )
tipo -> NOME[cont] ='\0';

      aggiungi_regola(tipo -> NOME,1);
    }

}
      tipo = tipo -> NEXT;
    }
  /*Ora ho a disposizione le liste ante_group per poter applicare le
    regole alla classe puntata da tipo, quindi lanciando converti_GES_ODL1
    adesso per le regole applicabili dovrei scrivere la regola al posto
    giusto.
    Inoltre bisogna mettere a posto conv_GES_ODL per Seq,Ins e tipi Semplici.
    */

  while (ante_group != NULL)
    {
      /* Decido quali sono le regole che posso applicare, perche' lo
 sono solo quelle per le quali l'insieme GS contiene l'antecedente
 ma non il conseguente.
 */

      regole_applicabili = 1;
      navigo_cons = cons_group;

      while (navigo_cons != NULL && regole_applicabili == 1)
{
  if ( ( confronta_nomi(ante_group ->
NOME,navigo_cons -> NOME)) == 0 )

    regole_applicabili = 0;

  navigo_cons = navigo_cons -> NEXT;
}

      if (regole_applicabili == 1)
{
  cont = 0;
```



```
  /*      while (ante_group -> NOME[++cont] != '\0');  */

  do {
    regola[cont] = ante_group -> NOME[cont];
  } while (ante_group -> NOME[++cont] != '\0');


  ante_group -> NOME[cont] = 'c';
  seleziona_classe(ante_group -> NOME);
  fputs(",&,",stream_out);
  if ( (confronta_nomi(puntaclasse -> C_TIPO,"const")) == 0)
    {
      fputs("/,\\,",stream_out);
    }
  printf("Applico la regola %s\n",regola);
  if (converti_ges_odl_tipo(1,puntaclasse,1) == 1)
    return 1;
  regole_applicate++;
}

      ante_group = ante_group -> NEXT;
    }

  return 0;

}

void scrivi_nome( char DATO[] , FILE *stream)
{
  /* Il sottomodulo scrive una stringa di caratteri passata come DATO[]
     in un file specificato da un puntatore passato come *stream
     */

 fputs(DATO,stream);
}
```

# Bibliografia


[A+89] M. Atkinson et al. The object-oriented database system manifesto. In *1nd Int. Conf. on Deductive and Object-Oriented Databases*. Springer-Verlag, 1989.

[AB91] S. Abiteboul and A. Bonner. Objects and views. In *SIGMOD*, pages 238–247. ACM Press, 1991.

[AK89] S. Abiteboul and P. Kanellakis. Object identity as a query language primitive. In *SIGMOD*, pages 159–173. ACM Press, 1989.

[Atz93] P. Atzeni, editor. *LOGIDATA+: Deductive Databases with Complex Objects*. Springer-Verlag: LNCS n. 701, Heidelberg - Germany, 1993.

[Bal92] J.P. Ballerini. *ODL-Designer: un sistema per il progetto automatico di schemi di basi di dati ad oggetti complessi*. Tesi di Laurea, Facoltà di Scienze MM. FF. NN., Corso di Laurea in Scienze dell'Informazione, Università di Bologna, Bologna, 1992.

[BB93] J.P. Ballerini and S. Bergamaschi. Automatic building and validation of complex object database schemata. Technical Report 5/124, CNR - Progetto Finalizzato Sistemi Informatici e Calcolo Parallelo, July 1993.

[BBLS94a] D. Beneventano, S. Bergamaschi, S. Lodi, and C. Sartori. Consistency checking in complex object database schemata with integrity constraints. Technical Report 103, CIOC–CNR, Viale Risorgimento, 2 Bologna, Italia, September 1994.

[BBLS94b] D. Beneventano, S. Bergamaschi, S. Lodi, and C. Sartori. Constraints in complex object database models. In *Int. Workshop on Description Logics*, Bonn, Germany, June 1994.







[BBMR89] A. Borgida, R.J. Brachman, D.L. McGuinness, and L.A. Resnick. CLASSIC: A structural data model for objects. In *SIGMOD*, pages 58–67, Portland, Oregon, 1989.

[BBS91a] D. Beneventano, S. Bergamaschi, and C. Sartori. Taxonomic reasoning in complex object data models. In *COMAD 91*, Bombay, India, December 1991.

[BBS91b] D. Beneventano, S. Bergamaschi, and C. Sartori. Taxonomic reasoning in LOGIDATA+. In V.A. Monaco and R. Negrini, editors, *COMPEURO 91*, pages 894–899, Bologna - Italy, May 1991. IEEE Computer Society Press.

[BBS91c] S. Bergamaschi, D. Beneventano, and C. Sartori. Ragionamento tassonomico in LOGIDATA+. Technical Report 5/53, CNR - Progetto Finalizzato Sistemi Informatici e Calcolo Parallelo, Roma, February 1991.

[BBS93] D. Beneventano, S. Bergamaschi, and C. Sartori. Taxonomic reasoning with cycles in LOGIDATA+. In P. Atzeni, editor, *LOGIDATA+: Deductive Databases with Complex Objects*. Springer-Verlag, Heidelberg - Germany, 1993. Lecture Notes in CS - N. 701.

[BBS94] D. Beneventano, S. Bergamaschi, and C. Sartori. Using subsumption for semantic query optimization in oodb. In *Int. Workshop on Description Logics*, Bonn, Germany, June 1994.

[Bee90] C. Beeri. Formal models for object-oriented databases. In W. Kim, J.M. Nicolas, and S. Nishio, editors, *Deductive and Object-Oriented Databases*, page 405:430. Elsevier Science Publisher, B.V. - North-Holland, 1990.

[Ben94] D. Beneventano. *Uno strumento di inferenza nelle basi di dati ad oggetti: la sussunzione*. PhD thesis, Dip. di Elettronica, Informatica e Sistemistica, Università di Bologna, Bologna, 1994.

[BGN89] H.W. Beck, S.K. Gala, and S.B. Navathe. Classification as a query processing technique in the CANDIDE data model. In *5th Int. Conf. on Data Engineering*, pages 572–581, Los Angeles, CA, 1989.

[BM92] E. Bertino and D. Musto. Query optimization by using knowledge about data semantics. *Data and Knowledge Engineering*, 9(2):121–155, December 1992.

[BN91] S. Bergamaschi and B. Nebel. Theoretical foundations of complex object data models. Technical Report 5/91, CNR, Progetto Finalizzato Sistemi Informatici e Calcolo Parallelo, Roma, January 1991.

[BN94] S. Bergamaschi and B. Nebel. Acquisition and validation of complex object database schemata supporting multiple inheritance. *Journal of Applied Intelligence*, 4:185–203, 1994.

[BS92] S. Bergamaschi and C. Sartori. On taxonomic reasoning in conceptual design. *ACM Transactions on Database Systems*, 17(3):385–422, September 1992.

[Car84] L. Cardelli. A semantics of multiple inheritance. In *Semantics of Data Types - Lecture Notes in Computer Science N. 173*, pages 51–67. Springer-Verlag, 1984.

[CW91] N. Coburn and G.E. Weddel. Path constraints for graph-based data models: Towards a unified theory of typing constraints, equations and functional dependencies. In *2nd Int. Conf. on Deductive and Object-Oriented Databases*, pages 312–331, Heidelberg, Germany, December 1991. Springer-Verlag.

[DD89] L.M.L. Delcambre and K.C. Davis. Automatic validation of object-oriented database structures. In *5th Int. Conf. on Data Engineering*, pages 2–9, Los Angeles, CA, 1989.

[FS86] T. Finin and D. Silverman. Interactive classification as a knowledge acquisition tool. In L. Kershberg, editor, *Expert Database Systems*, pages 79–90. The Benjamin/Cummings Publishing Company, 1986.

[GR83] A. Goldberg and D. Robson. *Smalltalk–80: the language and its implementation*. Addison∼Wesley, 1983.

[HZ80] M.M. Hammer and S. B. Zdonik. Knowledge based query processing. In *6th Int. Conf. on Very Large Databases*, pages 137–147, 1980.

[JK84] M. Jarke and J. Koch. Query optimization in database systems. *ACM Computing Surveys*, 16(2):111–152, June 1984.

[KC86] S. Khoshafian and G. Copeland. Object identity. In *1st Int. Conf. on Object–Oriented Programming Systems, Languages, and Applications*, Portland, 1986.

[KFL89] W. Kim and editors F. Lochovsky. *Object-Oriented Concepts, Databases and Applications*. Addison-Wesley, 1989.



[Kin81a] J. J. King. *Query optimization by semantic reasoning*. PhD thesis, Dept. of Computer Science, Stanford University, Palo Alto, 1981.

[Kin81b] J. J. King. Quist: a system for semantic query optimization in relational databases. In *7th Int. Conf. on Very Large Databases*, pages 510–517, 1981.

[LR89a] C. Lecluse and P. Richard. Modelling complex structures in object-oriented databases. In *Symp. on Principles of Database Systems*, pages 362–369, Philadelphia, PA, 1989.

[LR89b] C. Lécluse and P. Richard. The $O_2$ data model. In *Int. Conf. On Very Large Data Bases*, pages 411–422, Amsterdam, 1989.

[Neb90] B. Nebel. *Reasoning and Revision in Hybrid Representation Systems*, volume 422 of *Lecture Notes in Artificial Intelligence*. Springer-Verlag, Berlin, Heidelberg, New York, 1990.