FLORID – Ein Prototyp für F-Logik

Report 84

Jürgen Frohn Rainer Himmeröder* Paul-Th. Kandzia
Georg Lausen Christian Schlepphorst*

Institut für Informatik der Universität Freiburg i. Br.
{frohn,himmeroe,kandzia,lausen,schlepph}@informatik.uni-freiburg.de

Zusammenfassung F-Logik [17] ist eine mächtige Sprache, die deduktive und objektorientierte Konzepte (Objekte, Methoden, Klassenhierarchie, Vererbung, Signaturen) integriert und eine Reihe weiterer Entwicklungen (z.B. ROL, Gulog, OrLog) beeinflußt hat. Ein Vorzug der F-Logik gegenüber anderen Ansätzen ist die Fähigkeit zur uniformen Behandlung von Daten und Metadaten (z.B. Klassenhierarchie), die sie insbesondere als Grundlage zur Schemamanipulation und Integration heterogener Datenbestände interessant macht. Mit FLORID – F-LOgic Reasoning In Databases – stellen wir eine Implementierung von F-Logik vor. Die Mächtigkeit der Sprache erwies sich als eine große Herausforderung für die effiziente algorithmische Umsetzung. Im vorliegenden Text geben wir anhand eines Beispiels aus dem Bereich Schemaintegration einen Eindruck von der Programmierung mit FLORID und gehen ferner auf die Architektur des Prototyps und einige spezielle Aspekte der Sprache und der Implementierung ein.

1 Einleitung

Im Rahmen des Projektes *F(rame)-Logik* wurde ein Prototyp[1] für die objektorientierte, deduktive Datenbanksprache F-Logik [17] entwickelt. F-Logik ist eine Sprache, die zwei verschiedene Ansätze zur Erweiterung des relationalen Modells miteinander verbindet:

– Der *deduktive* Ansatz versucht, die Lücke zwischen Datenbanken und Anwendungen (impedance mismatch) zu schließen oder zumindest zu verkleinern, indem die Mächtigkeit der Datenbankabfragesprache erweitert wird. Hierbei können Daten durch Regeln definiert sein, wodurch die Möglichkeit zu rekursiven Anfragen gegeben ist. Insbesondere kann der deduktive Ansatz mit einer deklarativen, modelltheoretischen Semantik verbunden werden.

* Unterstützt von der Deutschen Forschungsgemeinschaft, Aktenzeichen La 598/3-2.

[1] Die aktuelle Version von FLORID kann per anonymous ftp über den Server `ftp.informatik.uni-freiburg.de` bezogen werden. Die entsprechenden Dateien befinden sich in dem Verzeichnis `/pub/florid`. FLORID läßt sich auch per WWW über folgende URL beziehen:
`http://www.informatik.uni-freiburg.de/~dbis/flogic-project.html`

– Beim *objektorientierten* Ansatz steht die Erweiterung des Datenmodells im Vordergrund. Hierbei werden komplex strukturierte Objekte mit einer Objektidentität eingeführt, die hierarchisch angeordneten Klassen zugeordnet werden können. Die Modellierung von Beziehungen erfolgt mit (typisierten) Methoden, die vererbt werden können. Dadurch wird es möglich, Anwendungen zu modellieren, die sich mit den flachen Relationen des Relationenmodells nicht adäquat repräsentieren lassen und komplexere Strukturen erfordern.

Während in den meisten anderen Arbeiten die Integration unvollständig bleibt (vgl. Abschnitt 5), stellt F-Logik eine der am weitesten fortgeschrittenen Kombinationen beider Paradigmen dar.

Ein naheliegender Ansatz (etwa [15]) besteht darin, das Problem der Integration objektorientierter Konzepte in einen deduktiven Kontext durch eine Abbildung auf relationale Strukturen (etwa Datalog) anzugehen. Eine solche Abbildung ist im Falle von F-Logik wegen der größeren Ausdrucksstärke nicht ausreichend. Insbesondere die Semantik von nichtmonotoner und multipler Vererbung ist auf diese Weise nicht zu erfassen (vgl. Abschnitt 3). Konsequenterweise entschieden wir uns bei der Entwicklung von FLORID dafür, eine an die komplexen Eigenschaften von F-Logik angepaßte objektorientierte Datenstruktur zugrundezulegen. Statt das Objektmodell als Herbrandbasis (Menge von hergeleiteten Grundatomen) zu repräsentieren, wurde eine speziell auf F-Logik zugeschnittene Frame/Slot-Darstellung der Objekte verwendet. Abschlußregeln der Logik (z.B. transitive Hüllenbildung über die Klassenhierarchie) werden nicht durch Hinzunahme expliziter Regeln zum Programm realisiert, sondern in die Objektmodell-Zugriffskomponente (ObjectManager) integriert, so daß das Modell redundanzarm gehalten wird, und nicht benötigte Folgerungen aus den Abschlußregeln gar nicht erst hergeleitet werden müssen.

Verschiedene neuere Ansätze aus dem Bereich deduktiver objektorientierter Datenbanken sind von F-Logik inspiriert worden. Gulog[7] und ORLog[14] versuchen, die Triggersemantik der Vererbung durch eine deklarativere Semantik zu ersetzen. Gulog beschränkt sich dazu auf funktionsfreie Programme mit statischer Klassenhierarchie und versieht Regeln mit Typeinschränkungen der verwendeten Variablen. Nichtmonotone Vererbung wird durch Ausblenden von Regeln für bestimmte Objekte realisiert. ORLog erweitert die Sprachsyntax durch Hinzunahme von *locality-, inheritability-* und *withdrawal-* Atomen, um Metainformationen über Vererbung in der Sprache selbst ausdrücken zu können. Auf ROL[20] wird in Abschnitt 5 näher eingegangen.

Im folgenden möchten wir zunächst die Möglichkeiten von F-Logik und FLORID an einem Beispiel vorstellen. Abschnitt 2 demonstriert die Fähigkeiten zur Metadatenbehandlung anhand eines Problems aus dem Bereich der Integration heterogener Datenbanken. In Abschnitt 3 gehen wir – wieder anhand von Beispielen – auf die F-Logik-Semantik ein, so wie sie in FLORID umgesetzt wurde. Abschnitt 4 stellt kurz die Architektur von FLORID vor, und Abschnitt 5 vergleicht FLORID mit anderen Entwicklungen. Eine Zusammenfassung der Ergebnisse findet sich in Abschnitt 6.

2 Anwendungsbeispiel Schemaintegration

Beim Entwurf von F-Logik war eines der wesentlichen Ziele, eine Sprache zur Verfügung zu stellen, die die traditionell strenge Trennung zwischen Schema und Datenbestand einer Objektbank aufhebt. Besonders im Zusammenhang mit Anwendungen aus dem Problembereich des Data-Warehousing und der Integration heterogener Datenbanken ist die Behandlung von Schema und Daten auf einer Ebene oft unverzichtbar. Zur Beschreibung der beteiligten Datenbanken verwenden manche neuere Ansätze spezialisierte Sprachen, die aber mit F-Logik eng verwandt sind [13, 19]. Andere Arbeiten setzen F-Logik selbst ein, zumindest für Teile des Systems [4]. Um einen ersten Eindruck von F-Logik und FLORID zu geben, wählen wir ein vereinfachtes Beispiel aus diesem Gebiet. Es sei aber betont, daß FLORID kein spezialisiertes System darstellt, sondern als Umgebung für beliebige Datenbankanwendungen konzipiert wurde.

Das Beispiel nutzt ausgiebig die Möglichkeit der F-Logik, Gleichheiten zwischen Objektnamen explizit anzugeben oder herzuleiten. Zur Erhaltung der Eindeutigkeit von Methodenergebnissen können sogar vom System implizit Gleichheiten erzeugt werden (siehe dazu auch den nächsten Abschnitt). Da die Gleichheiten durch Regeln definiert und datenabhängig sein können, geht ihre Verarbeitung in FLORID über das reine Verwalten einer statischen Gleichheitstheorie weit hinaus.

Für unser Beispiel existieren in einem Betrieb zwei bisher unabhängige relationale Datenbanken in der Personalplanung und der Buchhaltung, die für eine gemeinsame Anwendung zusammengeführt werden sollen. Wir gehen davon aus, daß die technischen Probleme unterschiedlicher Formate und Hardware bereits gelöst seien und somit die Daten in einer für FLORID geeigneten Form vorliegen.

Aus der ersten Datenbank interessiert uns die Relation `PERSON1`, die Personalnummer, Name und Abteilung für verschiedene Personen umfaßt, aus der zweiten Datenbank die Relation `PERSON2`, die ebenfalls Personalnummern und Namen, zusätzlich aber das Gehalt auflistet. Teilweise beschreiben die Relationen dieselben Personen, wobei die Personalnummern, auf die in den Relationen mit den Attributen `PNO` bzw. `PNR` verwiesen wird, betriebsweit eindeutig seien.

Datenbank 1	Datenbank 2
`PERSON1(PNO, PNAME, DEPT)`	`PERSON2(PNR, PNAME, SALARY)`

Die Transformation der relationalen Datenbank in objektorientierte Konstrukte folgt im wesentlichen bereits bekannten Ansätzen [2, 8, 16]. Relationen werden zu Klassen, Tupel zu Instanzen dieser Klassen. Tupel sind damit Objekte, die einen eindeutigen Objektnamen erhalten, hier `t1`, `t2`, usw. Attribute und Werte eines Tupels modellieren wir als Methoden und deren Ergebnis:

$$
\begin{array}{l}
\mathsf{t1 : person1[pno{\rightarrow}1001;\ pname{\rightarrow}"Schmidt";\ dept{\rightarrow}"Personal"].}\\
\mathsf{t2 : person1[pno{\rightarrow}1002;\ pname{\rightarrow}"Berger";\ dept{\rightarrow}"Personal"].}\\
\mathsf{t3 : person1[pno{\rightarrow}1003;\ pname{\rightarrow}"Weber";\ dept{\rightarrow}"Verkauf"].}\\
\mathsf{t4 : person1[pno{\rightarrow}1004;\ pname{\rightarrow}"Tschechow";\ dept{\rightarrow}"Forschung"].}\\
\mathsf{t5 : person1[pno{\rightarrow}1005;\ pname{\rightarrow}"Schulz";\ dept{\rightarrow}"Verkauf"].}\\
\\
\mathsf{t8 : person2[pnr{\rightarrow}1001;\ pname{\rightarrow}"Schmidt";\ salary{\rightarrow}4000].}\\
\mathsf{t9 : person2[pnr{\rightarrow}1002;\ pname{\rightarrow}"Berger";\ salary{\rightarrow}8000].}\\
\mathsf{t10 : person2[pnr{\rightarrow}1003;\ pname{\rightarrow}"Weber";\ salary{\rightarrow}5000].}\\
\mathsf{t11 : person2[pnr{\rightarrow}1004;\ pname{\rightarrow}"Chechov";\ salary{\rightarrow}7000].}\\
\mathsf{t12 : person2[pnr{\rightarrow}1007;\ pname{\rightarrow}"Mayer";\ salary{\rightarrow}4000].}
\end{array}
\tag{2.1}
$$

Zusätzlich wird auch das Schema der relationalen Datenbanken und weitere Metainformation in F-Logik dargestellt. Die Zugehörigkeit einer Relation zu einer bestimmten Datenbank drücken wir durch eine Subklassenbeziehung, z.B. person1 :: db1, aus.[2] Signaturen, in F-Logik durch Doppelpfeile ($\Rightarrow$) notiert, entsprechen dem Schema einer Relation. Weiter besitzt jede Relation eine Methode key, die das Schlüsselattribut angibt.[3]

$$
\begin{array}{l}
\mathsf{person1 : relation.}\\
\mathsf{person1 :: db1.}\\
\mathsf{person1[key{\rightarrow}pno].}\\
\mathsf{person1[pno{\Rightarrow}integer;\ pname{\Rightarrow}string;\ dept{\Rightarrow}string].}\\
\\
\mathsf{person2 : relation.}\\
\mathsf{person2 :: db2.}\\
\mathsf{person2[key{\rightarrow}pnr].}\\
\mathsf{person2[pnr{\Rightarrow}integer;\ pname{\Rightarrow}string;\ salary{\Rightarrow}integer].}
\end{array}
\tag{2.2}
$$

Wir ergänzen unser Programm um Regeln, die allgemeines Wissen über relationale Schemata unabhängig von einer bestimmten relationalen Datenbank darstellen. Wir benötigen für dieses Beispiel nur die folgende Regel, die für Relationen die (mehrwertige) Methode attr definiert. Diese liefert alle Attribute der Relation. Wie üblich werden Variablen großgeschrieben, Konstanten dagegen klein.

$$
\mathsf{R[attr{\twoheadrightarrow}A] :- R : relation[A{\Rightarrow}X].} \tag{2.3}
$$

Nachdem die Transformation der relationalen Datenbanken in F-Logik abgeschlossen ist, soll nun die gemeinsame neue Datenbank dbNew entwickelt werden. Dazu definieren wir die neue Relation person als Subklasse von dbNew. Die Signatur der neuen Relation stellen wir noch etwas zurück.

[2] Aufgrund der Abschlußeigenschaften sind dann automatisch die Tupel auch Instanzen der Superklassen, z.B. t1 : db1.

[3] In unserem Beispiel beschränken wir uns auf einelementige Schlüssel. Mehrelementige Schlüssel können durch Verwendung von Funktoren auf den elementaren Fall zurückgeführt werden.

person :: dbNew.
person : relation. (2.4)

Mittels Regeln können wir die (zunächst disjunkte) Vereinigung der Tupel aus person1 und person2 bilden. Die neuen Attributnamen werden so gewählt, daß erkennbar bleibt, aus welcher Relation die Werte jeweils übernommen wurden.[4] Namenskonflikte werden so vermieden. Ferner vergeben wir für jedes Tupel einen neuen Objektnamen mittels der Skolemfunktion s. Es ist klar, daß auf jedem der neuen Tupel die Attribute genau einer Ausgangsrelation Werte liefern, die Attribute der jeweils anderen können wir uns als mit Nullwerten besetzt vorstellen.

```
s(T1) : person[attr(person1,Attr)→Val] :–
        T1 : person1[Attr→Val],
        person1[attr→→Attr].
                                                    (2.5)
s(T2) : person[attr(person2,Attr)→Val] :–
        T2 : person2[Attr→Val],
        person2[attr→→Attr].
```

Zur besseren Lesbarkeit benennen wir jetzt die neuen Attribute um, indem wir die Möglichkeit von F-Logik nutzen, durch explizite Angabe von Gleichheiten Objektnamen (unter die auch Methodenbezeichner fallen) als synonym zu erklären.

```
attr(person1,pname) = pname.
attr(person2,pname) = pname.

attr(person1,pno) = pnr.
attr(person2,pnr) = pnr.                            (2.6)

attr(person1,dept) = dept.
attr(person2,salary) = salary.
```

Da jedes Tupel der Relation person durch die Skolemisierung eindeutig den Ausgangstupeln in person1 bzw. person2 zuzuordnen ist, gibt es durch die Attributumbenennung keine Konflikte. Weiter geben wir jetzt auch die Signatur und den Schlüssel der neuen Relation an.

```
person[pnr⇒integer; pname⇒string; dept⇒integer; salary⇒integer].
person[key→pnr].                                    (2.7)
```

Nach Eingabe all dieser Fakten und Regeln veranlassen wir mit dem Systemkommando ?- sys.eval[ ]. die Auswertung, also die Bottom-Up-Berechnung des minimalen Modells. Um die neue Relation auszugeben, sind zwei Anfragen nötig, da auf jedem Tupel jeweils eine der Methoden dept oder salary undefiniert ist.

[4] In F-Logik wird zwischen Methoden- und Objektnamen nicht unterschieden. Hier werden die Methodennamen mittels Funktoren und Variablen gebildet.

```
?– sys.eval[ ].
?– T : person[pnr→P; pname→N; dept→D].
?– T : person[pnr→P; pname→N; salary→S].
```
(2.8)

Wir erhalten auf die beiden Fragen folgende Antwortmengen:

```
T/s(t1) P/1001 N/"Schmidt" D/"Personal"
T/s(t2) P/1002 N/"Berger" D/"Personal"
T/s(t3) P/1003 N/"Weber" D/"Verkauf"
T/s(t4) P/1004 N/"Tschechow" D/"Forschung"
T/s(t5) P/1005 N/"Schulz" D/"Verkauf"

T/s(t8) P/1001 N/"Schmidt" S/4000
T/s(t9) P/1002 N/"Berger" S/8000
T/s(t10) P/1003 N/"Weber" S/5000
T/s(t11) P/1004 N/"Chechov" S/7000
T/s(t12) P/1007 N/"Mayer" S/4000
```
(2.9)

Damit ist unser Ziel aber noch nicht erreicht: wir möchten die Information zu einer Person in jeweils einem Tupel zusammenfassen. Dies läßt sich durch folgende Regel erreichen:

```
T = U :–
        T : person[pnr→P],
        U : person[pnr→P].
```
(2.10)

Damit eine solche Verschmelzung korrekt verläuft, müssen wir allerdings vorher dafür sorgen, daß die Schlüsseleigenschaft von pnr in person nicht verletzt ist. Zur Demonstration geben wir eine generische Anfrage an, die Verletzungen der funktionalen Abhängigkeit eines beliebigen Attributs vom Schlüssel in einer beliebigen Relation auffindet:

```
?– R : relation[key→K; attr↠Attr],
    T1 : R[K→V1; Attr→V2],
    T2 : R[K→V1; Attr→V3],
    not V2 = V3.
```
(2.11)

Die Antwort zeigt nicht nur, daß tatsächlich eine Abhängigkeitsverletzung vorliegt, sondern gibt zugleich die beteiligten Relationen und Tupel an. Aufgrund der Symmetrie der Anfrage erhält man zwei Antworten:

```
R/person K/pnr T1/s(t11) V1/1004 Attr/pname V2/"Chechov"
T2/s(t4) V3/"Tschechow"

R/person K/pnr T1/s(t4) V1/1004 Attr/pname V2/"Tschechow"
T2/s(t11) V3/"Chechov"
```
(2.12)

Es ist anzunehmen, daß in diesem Fall zwei Schreibweisen für den gleichen Namen verwendet wurden. Dies ist allerdings kein Problem, das allein in der Datenbank gelöst werden kann, sondern muß zuerst in der Realität geklärt werden. Wertet man trotz dieses Problems Regel (2.10) aus, setzt Florid unter Ausgabe einer Warnung die Objekte "Tschechow" und "Chechov" gleich, um eine Verletzung der Funktionalität der Methode pname zu vermeiden[5]. Abschließend stellen wir eine Anfrage an die so gefundene endgültige Version:

$$\begin{array}{l} ?\!-\ \mathsf{sys.eval}[\,].\\ ?\!-\ \mathsf{X : person[pnr}\!\to\!\mathsf{P;\ pname}\!\to\!\mathsf{N;\ dept}\!\to\!\mathsf{D;\ salary}\!\to\!\mathsf{S].} \end{array} \tag{2.13}$$

Wir erhalten folgende Antworten (wobei nur die Tupel auftauchen, auf denen tatsächlich alle gefragten Attribute einen Wert liefern, vgl. die Antwort (2.9)):

```
X/s(t8) P/1001 N/"Schmidt" D/"Personal" S/4000
X/s(t9) P/1002 N/"Berger" D/"Personal" S/8000
X/s(t10) P/1003 N/"Weber" D/"Verkauf" S/5000
X/s(t11) P/1004 N/"Chechov" D/"Forschung" S/7000
```
(2.14)

Existieren mehrere Objektnamen für dasselbe Objekt, z.B. "Chechov" und "Tschechow", so wird in der Antwortsubstitution einer der möglichen Objektnamen verwendet, hier "Chechov".

3 Semantik

Für positiv definite Programme implementiert FLORID – abgesehen von der Typprüfung – die Semantik gemäß [17]. Außerdem wurde die Sprache um Pfadausdrücke [12] erweitert. Auf einige Besonderheiten der Semantik und der praktischen Umsetzung soll hier anhand von Beispielen kurz eingegangen werden. Eine ausführlichere Beschreibung findet sich in [11].

Die folgenden Fakten geben das Schema einer F-Logik-Datenbank an:

$$\begin{array}{l} \mathsf{manager :: employee.}\\ \mathsf{topmanager :: manager.}\\ \mathsf{employee[boss}\Rightarrow\mathsf{manager].}\\ \mathsf{manager[(boss.inv)}\Rightarrow\!\!\!>\mathsf{employee].}\\ \mathsf{boss : baseMethod.} \end{array} \tag{3.1}$$

F-Logik unterscheidet zwischen Subklassenbeziehung (manager :: employee) und Instanzbeziehung (boss : baseMethod). Die Subklassenbeziehung ist reflexiv und transitiv (zu den Abschlußeigenschaften vgl. Abschnitt 4.1). Implizit folgt also auch topmanager :: employee aus dem Schema. Als Konsequenz sind die Instanzen einer Klasse auch in den Superklassen enthalten, das Fakt jane : manager impliziert damit auch jane : employee.

[5] Wir unterscheiden nicht zwischen Werten und Objekten, auch Integers und Strings sind Objekte. FLORID hat zur Zeit keine eingebauten Stringfunktionen. Diese wären mit einer solchen Gleichsetzung nicht verträglich, so daß – wie im Fall der Built-In-Klasse Integer – die Gleichsetzung zu einem Laufzeitfehler führen müßte.

Das erste Signaturfakt besagt, daß auf der Klasse employee die Methode boss definiert ist und die Ergebnisse in der Klasse manager liegen. Das zweite Signaturfakt deklariert die mehrwertige Methode boss.inv auf der Klasse manager. Der Term boss.inv ist ein *Pfadausdruck* und steht für das Resultatobjekt der Methode inv, angewendet auf das Objekt boss. Da in F-Logik auch an Methodenposition beliebige Objekte stehen dürfen, können wir diesen Pfadausdruck als Methodennamen für die Umkehrung der boss-Methode einsetzen.

Der Begriff des wohlgetypten Programms nach [17] ist über die korrekte Typisierung *aller* kanonischen Modelle definiert und damit nicht praktikabel. Weiter unter beschreiben wir jedoch eine Typprüfung des *berechneten* Modells im Sinne einer Konsistenzbedingung.

Die Datenbasis bestehe aus einer Anzahl von Angestellten und Managern mit der (partiell definierten) Methode boss:

```
john : employee[boss→bill].
jack : employee[boss→jane].
bill : manager[boss→jane].
jim[boss→jane].
jane : manager.
```
(3.2)

Durch die folgende generische Regel wird zu jeder funktionalen Methode (die Instanz der Klasse baseMethod ist) die entsprechende inverse Methode erzeugt. Da eine funktionale Methode nicht notwendigerweise injektiv ist, ist ihre inverse Methode mehrwertig:

```
X[(M.inv)↠Y] :– Y[(M:baseMethod)→X].
```
(3.3)

Wenn für eine Instanz von baseMethod (angenommen, es gelte das Fakt dept : baseMethod), die an die Variable M gebunden wird, noch keine Methode inv definiert war, so wird bei Auswertung der Regel automatisch ein neues Objekt erzeugt und mit dem Pfadausdruck, hier dept.inv, bezeichnet.

Wir wollen jetzt die Klasse topmanager als die Menge aller Manager definieren, die ihrerseits keine weiteren Vorgesetzten haben. Dazu benötigen wir eine Regel mit einem negierten Teilziel. FLORID wertet Programme durch Iteration des T_P-Operators (Folgerungsoperator zu einem Programm P) unter Berücksichtigung möglicher Vererbung aus (vgl. Abschnitt 4.2). Negation von Teilzielen hat damit zunächst eine inflationäre Semantik [18], die in vielen Fällen nicht das intendierte Ergebnis liefert. Eine oft gewählte Lösung ist die Beschränkung auf stratifizierte Programme. Da aber die mächtigen Sprachkonstrukte von F-Logik häufig zu nichtstratifizierten Programmen führen, ermöglicht FLORID es dem Benutzer, mit dem Systemkommando ?- sys.strat.doIt[]. eine Partitionierung des Programms in Strata explizit anzugeben. Im Beispiel schreiben wir jetzt:

```
?– sys.strat.doIt[].
X : topmanager :– X : manager, not X.boss[].
```
(3.4)

Durch das Stratifizierungskommando wird das Modell der vorhergehenden Regeln (3.1)–(3.3) berechnet und diese Regeln danach entfernt. Damit sind alle Fakten, die das negierte Teilziel erfüllen können, bereits berechnet, wenn Regel (3.4) betrachtet wird. Interessant ist hier noch, daß das negierte Teilziel ohne Verwendung eines Pfadausdrucks not X[boss→Y] lauten müßte und die Variable Y unsicher wäre. Pfadausdrücke können also die Programmierung deutlich vereinfachen.

Wie bereits erwähnt, erfolgt die Typprüfung anhand der definierten Signaturen in FLORID nicht automatisch. Ein Test auf Konsistenz eines berechneten Modells kann jedoch über ein Systemkommando erfolgen oder durch die folgenden Regeln in F-Logik selbst realisiert werden:

```
?– sys.strat.doIt[ ].
typeSafe(O,M,R,funct) :– O:C[M→R], C[M⇒()].
?– sys.strat.doIt[ ].                                                    (3.5)
typeUnsafe(O,M,R,funct) :– O[M→R], not typeSafe(O,M,R,funct).
typeIncorrect(O,M,R,funct) :– O:C[M→R], C[M⇒D], not R:D.
```

Die analogen Regeln für mehrwertige Methoden erhalten wir durch Ersetzen von funct durch multi sowie von einfachen Pfeilen (→,⇒) durch Doppelpfeile (↠,⇛). Nach der Evaluierung dieser Regeln kann nun die korrekte Typisierung des berechneten Modells überprüft werden:

```
?– sys.eval[ ].
?– typeUnsafe(O,M,R,K).                                                  (3.6)
?– typeIncorrect(O,M,R,K).
```

Aus der Antwortmenge wird ersichtlich, welche Methoden bezüglich welcher Objekte keine entsprechende Signatur besitzen (mangelnde Typsicherheit), bzw. eine gegebene Signatur nicht einhalten (keine Typkorrektheit):

```
Answer to query: ?– typeUnsafe(O,M,R,K).
O/boss M/inv R/boss.inv K/funct
O/jim M/boss R/jane K/funct
                                                                         (3.7)
Answer to query: ?– typeIncorrect(O,M,R,K).
O/jane M/boss.inv R/jim K/multi
```

Zum Schluß erfragen wir alle Topmanager sowie deren unmittelbare Untergebene:

```
?– X : topmanager[(boss.inv)↠Y].                                         (3.8)
```

In unserem Fall ergibt sich folgende Antwortmenge:

```
Answer to query: ?– X : topmanager[(boss.inv)↠Y].
X/jane Y/bill
X/jane Y/jack                                                            (3.9)
X/jane Y/jim
```

Die Unterteilung eines Programms in Strata kann auch benutzt werden, um die Erzeugung einer Gleichheitsrelation in einem bestimmten Programmteil durchzuführen und Gleichsetzungen von Objekten im Rest des Programms dann als Inkonsistenzen zu behandeln (vgl. [17], Section 11.1). Das folgende Beispiel berücksichtigt, daß eine Person auch über einen Spitznamen referenziert werden kann:

$$
\begin{array}{l}
\mathsf{john[spouse{\rightarrow}mary;\ nickname{\twoheadrightarrow}johnny].} \\
\mathsf{mary[spouse{\rightarrow}johnny].} \\
\mathsf{peter[spouse{\rightarrow}sara].} \\
\mathsf{sara[spouse{\rightarrow}james].} \\
\\
\mathsf{X = Y :- X[nickname{\twoheadrightarrow}Y].} \\
\\
\mathsf{?\text{-}\ sys.strat.doIt[\,].} \\
\mathsf{?\text{-}\ sys.theOM.eqTraceOn[\,].}
\end{array}
\tag{3.10}
$$

Das letzte Systemkommando weist den Object Manager an, bei weiteren Gleichsetzungen von Objekten eine Warnung zu erzeugen. Interessant ist, daß eine Gleichsetzung auch *implizit* erfolgen kann, etwa über die Funktionalität von Methoden. Mit der folgenden Regel machen wir die Methode spouse symmetrisch:

$$
\begin{array}{l}
\mathsf{X[spouse{\rightarrow}Y] :- Y[spouse{\rightarrow}X].} \\
\mathsf{?\text{-}\ sys.eval[\,].}
\end{array}
\tag{3.11}
$$

Da bereits sara[spouse→james] gilt, durch die Regel aber auch noch das Fakt sara[spouse→peter] abgeleitet wird, kann das Programm wegen der Funktionalität der Methode spouse nur konsistent sein, wenn james = peter gilt. Daher gibt FLORID bei der Evaluierung die folgende Warnung aus:

```
Equality: james made equal to peter
```

4 Architektur

Nachdem wir FLORID auf Sprachebene betrachtet haben, gehen wir nun auf die konkrete Umsetzung der Konzepte ein. Abbildung 1 gibt einen Überblick über die Struktur des Systems, dessen interessanteste Komponenten wir im folgenden kurz betrachten werden. Die Implementation von FLORID erfolgte in C++. Der objektorientierte Entwurf spiegelt sich an der Benutzerschnittstelle wider. So wird das System vom Benutzer (bzw. einem Konfigurationsprogramm) aus einzelnen Komponenten zusammengesetzt. Die dadurch gewonnene Flexibilität ist sowohl für die Integration neuer Komponenten bei der Weiterentwicklung des Prototyps als auch für die individuelle Anpassung des Systems von Vorteil. Metabefehle (für Konfiguration und Betrieb des Systems) und Anfragesprache sind integriert, d.h. alle Systemkommandos sind syntaktisch F-Logik-Anfragen an ein Pseudoobjekt sys. Details finden sich im FLORID–Benutzerhandbuch [10].

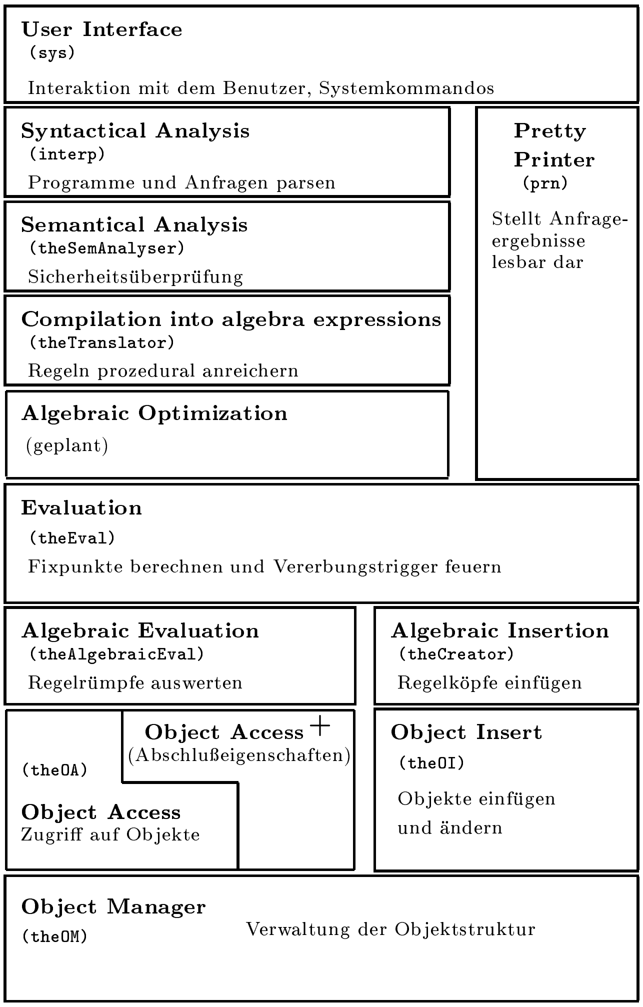

Abbildung1. Struktur des Prototyps

4.1 Repräsentation des Objektmodells

Wie bereits erwähnt, ist es für F-Logik nicht möglich, die Objektstrukturen auf die übliche Weise auf relationale Strukturen abzubilden, indem man Klassen als Relationen und Objekte als Tupel darstellt. Die Klassenhierarchie ist i.a. zu Beginn der Evaluierung nicht bekannt: sie kann ihrerseits durch Regeln definiert werden. Instanzen einer Klasse müssen nicht alle dieselben Methoden aufweisen. Daher mußte eine für F-Logik adäquate Repräsentation gefunden werden.

Die Basis von FLORID bildet der *Object Manager* (OM), der die betrachtete Objektwelt in Form einer Frame/Slot-Darstellung verwaltet. Diese Darstellung geht auf Ansätze der Wissensrepräsentation in der Künstlichen Intelligenz [26] zurück. Jeder Objektname wird intern auf eine Objektidentität (OID) abgebildet (in Abb. 2 z.B. o_{john}). Diese Abbildung ist i.a. nicht injektiv; indem mehrere Objektnamen auf dieselbe OID abgebildet werden, verwaltet der OM die Gleichheitsrelation zwischen Objektnamen. Wenn über das Objekt Informationen bekannt sind, werden diese in einem Objektframe abgelegt, der aus einer Anzahl Slots besteht. Es gibt Slots für Klassenzugehörigkeit, Superklassen und Methodendefinitionen. Slots enthalten OIDs oder Verweise auf andere Frames. Dies sind Parameterframes (in Abb. 2 p_1, p_2) für parametrisierte Methoden oder Mengenframes (in Abb. 2 s_1, s_2, s_3) für mehrwertige Methoden. Zusätzlich gibt es Prädikatframes für die Verwaltung von Prädikaten.

Abbildung 2 verdeutlicht die Darstellung eines Ausschnittes eines Datenbankzustandes mittels Frames.[6]

```
john : student.
john : employee.
john[age→24;
     spouse→mary;
     children→→{peter, sally};
     salary@(1993,flogic)→50000;
     salary@(1993,updates)→60000;
     projects@(1993)→→{flogic,updates}].
```

o_{john}		
	:	s_1
	o_{age}	i_{24}
	$o_{children}$	s_2
	o_{salary}	p_1
	$o_{projects}$	p_2
	o_{spouse}	o_{mary}

s_1	
	$o_{student}$
	$o_{employee}$

s_2	
	o_{peter}
	o_{sally}

p_1			
	i_{1993}	o_{flogic}	i_{50000}
	i_{1993}	$o_{updates}$	i_{60000}

p_2		
	i_{1993}	s_3

s_3	
	o_{flogic}
	$o_{updates}$

Abbildung2. Beispiel eines Datenbankzustandes und der entsprechenden Frame-Struktur

[6] Zur besseren Lesbarkeit schreiben wir die OID zu *objektname* mit $o_{objektname}$. Integerwerte werden durch Built-In-Namen (z.B. i_1) bezeichnet. Die Klassenzugehörigkeit wird im OM analog einer mehrwertigen Methode gespeichert.

Da die Abschlußeigenschaften von F-Logik (Transitive Hülle über die Klassenhierarchie, Reflexivität der Subklassenbeziehung, usw.) nicht durch Hinzunahme von Abschlußregeln zum Benutzerprogramm realisiert werden sollten, wird der Zugriff auf den OM durch die Komponenten Object Accessor (OA) und OA^+ gekapselt, die diese Abschlußeigenschaften als virtuelle Sicht auf den OM umsetzen. Auf die gleiche Weise wurden die Prädikate $<, >, =$ sowie integer() und die arithmetischen Funktionen fest eingebaut. Der Schreibzugriff auf den OM wird durch den Object Inserter (OI) gekapselt.

4.2 Evaluierung von Programmen

Ein F-Logik-Programm ist eine Menge von Regeln. Fakten und Anfragen sind Spezialfälle von Regeln, wobei Anfragen in FLORID getrennt vom übrigen Programm behandelt werden. Über einen Parser liest FLORID Regel für Regel von einem Programmfile ein (*Syntactical Analysis*) und führt einen Sicherheitscheck durch (*Semantical Analysis*). Gegebenenfalls wird die Anordnung der Teilziele verändert, um bei Auswertung von links nach rechts geeignet Bindungen propagieren zu können. Fakten und Regeln werden dem internen Programm P hinzugefügt, Anfragen (Systemkommandos oder Anfragen an das Modell) werden direkt ausgeführt.

FLORID verwendet ein mengenorientiertes Bottom-Up-Evaluierungsverfahren, das die in [17] beschriebene Triggersemantik der Vererbung realisiert. Der Operator T_P zum Programm P nimmt zum Objektmodell alle Fakten hinzu, die durch einmaliges Anwenden einer Programmregel daraus gefolgert werden können. T_P wird solange angewendet, bis ein Fixpunkt $T_P^\infty(\emptyset)$ erreicht ist. Nun wird versucht, einen aktiven Vererbungstrigger zu finden, d.h. ein Fakt, das von einer Klasse auf eine Subklasse oder eine Instanz vererbt werden kann, noch nicht überschrieben und für die weitere Modellberechnung auch von Belang ist, d.h. für die Instanziierung eines Teilziels einer Regel oder Anfrage verwendet werden kann. Gibt es einen solchen Trigger, so wird er gefeuert (das Fakt eingefügt) und wieder T_P bis zum nächsten Fixpunkt angewendet. Dieses Vorgehen wird solange wiederholt, bis kein Trigger mehr aktiv ist. Damit ist das Modell zu P berechnet. Wenn mehrere Trigger gleichzeitig aktiv sind, so kann das Feuern des einen den anderen deaktivieren, so daß das berechnete Modell von der Reihenfolge der Trigger abhängt. Dies entspricht der (nichtdeterministischen) Auswahl eines möglichen kanonischen Modells zu P (vgl. [17] Kapitel 14).

Die mächtigen Ausdrucksmittel von F-Logik – zu nennen wären hier insbesondere die Möglichkeit, Variablen an Methoden- und Klassenpositionen zu verwenden, regelgesteuerte Gleichsetzungen von Objekten sowie Definition der Klassenhierarchie über Regeln – erschweren die Anwendung gängiger Evaluierungs- und Optimierungstechniken (wie seminaive Evaluierung, Magic Sets, stratifizierte Negation) ganz erheblich. Viele dieser Strategien erfordern eine Einschränkung auf Teilklassen von F-Logik. Grund dafür ist, daß die semantische Abhängigkeit zwischen Teilzielen und Regelköpfen nicht durch eine syntaktische Unifikation erfaßt werden kann. Über Mechanismen wie Vererbung oder Hüllenbildung der Subklassenbeziehung können Abhängigkeiten auch zwischen syntaktisch völlig unterschiedlichen Atomen bestehen. Dies macht syntaktische Programmanalysen und -transformationen in uneingeschränkter F-Logik unmöglich.

Es gelang jedoch, einen seminaiven Evaluierungsalgorithmus für F-Logik zu entwickeln. Die Grundidee besteht darin, die in der aktuellen T_P-Runde hergeleitete Information in Delta-Prädikaten abzulegen (z.B. $\Delta^1_{\rightarrow}(m, o, a_1, r)$ für $o[m@(a_1){\rightarrow}r]$ und $\Delta_{isa}(o,c)$ für $o : c$). Aufgrund der Abschlußeigenschaften kann der Fall eintreten, daß hergeleitete Information andere Fakten impliziert, die dann nicht in den Δ-Prädikaten enthalten ist. Dieses Problem lösen wir, indem die Abschlußeigenschaften bei der Transformation der Teilziele berücksichtigt werden. So wird $o : c$ unter Einführung einer neuen Variablen X zur Konjunktion $\Delta_{isa}(o, X) \wedge X :: c$ (vgl. das folgende Beispiel). Das Objekt o ist neue Instanz von der Klasse c, wenn es in der letzten Auswertungsrunde neue Instanz einer Subklasse von c wurde.[7]

Die Regeltransformation sei hier am Beispiel eines Tierstammbaumes demonstriert:

$$\begin{array}{l} \mathsf{animal[father{\Rightarrow}animal;\ mother{\Rightarrow}animal].} \\ \mathsf{horse :: animal.} \\ \mathsf{mare :: horse.} \\ \mathsf{stallion :: horse.} \end{array} \tag{4.1}$$

$$\begin{array}{l} \mathsf{Y : mare :\!-\ X : horse[mother{\rightarrow}Y].} \\ \mathsf{Y : stallion :\!-\ X : horse[father{\rightarrow}Y].} \end{array}$$

Enthalte die EDB einen Abstammungsbaum von Tieren, über die aber nicht bekannt ist, ob sie Pferde sind. Angewendet auf ein Fakt, etwa fury : stallion, berechnen die beiden Regeln die Klassenzugehörigkeit aller Vorfahren von fury. Man beachte, daß die Rekursion hier über die Isa-Beziehung verläuft. Dabei muß berücksichtigt werden, daß Instanzen einer Klasse (z.B. mare) auch Instanzen all ihrer Superklassen (horse und animal) sind. Die Transformation der Regeln liefert

$$\begin{array}{l} \mathsf{Y : mare :\!-\ } \Delta_{isa}\mathsf{(X,Z),\ Z :: horse,\ Z[mother{\rightarrow}Y].} \\ \mathsf{Y : stallion :\!-\ } \Delta_{isa}\mathsf{(X,Z),\ Z :: horse,\ Z[father{\rightarrow}Y].} \end{array} \tag{4.2}$$

Die Delta-Prädikate werden beim Einfügen in den OM automatisch aktualisiert. Wenn also fury : stallion neu hergeleitet wird, wird auch Δ_{isa}(fury,stallion) eingefügt, nicht jedoch Δ_{isa}(fury,horse). Daher ist das zusätzliche Teilziel Z :: horse in den Delta-Regeln notwendig, so daß Z mit der jeweiligen Subklasse von horse belegt wird.

4.3 Benutzerschnittstelle

Als Basisschnittstelle verfügt FLORID über eine textorientierte Ein- und Ausgabe ähnlich der Prolog-Shell. Darüber läßt sich auch die Konfiguration des Systems durch das Versenden von Nachrichten an sogenannte Interfaceobjekte

[7] In dieser Skizze beschränken wir uns auf den elementaren Fall: Keine Herleitung von neuen Subklassenbeziehungen oder Gleichheiten durch Regeln.

steuern, etwa die Einstellung von Systemparametern wie Debug- oder Ausgabe-Modi. Der Ausgabe vorgeschaltet ist außerdem ein *PrettyPrinter*, der die Anfrageergebnisse von interner Objektdarstellung in für den Benutzer lesbare Darstellungen umwandelt.

Basierend auf der Terminalschnittstelle erfolgte eine Anbindung an die Emacs-Umgebung und an eine mit Tcl/Tk realisierte graphische Oberfläche [10]. Ferner steht FLORID dem Benutzer auch on-line über eine WWW-Schnittstelle zur Verfügung.[8]

5 Vergleich mit anderen Forschungsarbeiten

In den letzten Jahren ist eine Reihe von Prototypen und Systementwicklungen im Bereich deduktive objektorientierte Datenbanken entstanden, die jeweils ganz unterschiedliche Schwerpunkte setzen und die beiden Paradigmen auf verschiedene Weise miteinander zu verbinden suchen.

Verschiedene Systeme (z.B. Aditi [25], LOLA [9], $\mathcal{LDL}$ [5, 21], CORAL [23, 24]) erweitern Datalog^{neg} um komplexere Datenstrukturen (Mengen, Listen) und weitere Sprachkonstrukte (z.B. Aggregation). CORAL erlaubt zudem die Unterteilung des Programms in Module, für die der Programmierer Auswertungsstrategien in Form von *Annotationen* vorgeben kann. All diese Systeme können aber nicht als objektorientiert im engeren Sinne bezeichnet werden.

ROCK & ROLL [1] kombiniert eine imperative Datendefinitions- und Manipulationssprache (ROCK) mit einer deduktiven Anfragesprache (ROLL). Einen ähnlichen Weg verfolgt Glue-Nail! [6, 22]. Im Gegensatz zu F-Logik ist also keine uniforme Daten- und Schemaverarbeitung möglich. ConceptBase [15] kombiniert Deduktion mit Objektorientierung, indem die objektorientierten Strukturen auf Datalog^{neg} (d.h. auf relationale Strukturen) abgebildet werden. Diese Transformation hat den Vorteil, daß die für Datalog bekannten Auswertungsalgorithmen (stratifizierte Negation, seminaive Evaluierung, Magic Sets) direkt übernommen werden können. Anders als in F-Logik ist in ConceptBase keine regelgesteuerte Objektkreation (Erzeugung neuer Objekte in Regelköpfen) möglich. Um den Einfluß beider Ansätze auf die Effizienz der Evaluierung zu untersuchen, wollen wir in Kürze einen Performancevergleich mit FLORID durchführen.

Ein neuerer Entwurf einer deduktiven objektorientierten Datenbank ist ROL [20], das sich nahe an F-Logik anlehnt, deren Ausdrucksmöglichkeiten jedoch stark einschränkt. So sind keine Variablen an Methodenposition erlaubt, Gleichsetzung von Objekten wird als Inkonsistenz behandelt, es erfolgt eine scharfe Trennung zwischen Datenbankschema (d.h. Klassenhierarchie) und Daten, so daß keine regelgesteuerten Schemadefinitionen oder -veränderungen möglich sind und Anfragen an das Schema als Metaanfragen außerhalb der Logik selbst gestellt werden müssen. ROL erlaubt nur strukturelle Vererbung (Vererbung von Signaturausdrücken), die Klassenhierarchie im ROL-Prototyp unterliegt zudem Einschränkungen (Halbverband), um Vererbungskonflikte zu vermeiden. Diese

[8] URL: `http://www.informatik.uni-freiburg.de/~dbis/flsys/`

Restriktionen gestatten andererseits eine statische Abhängigkeitsanalyse und somit mittels einer Stratifizierung eine Mengenbehandlung, wie sie bereits in $\mathcal{LDL}$ vorgeschlagen wurde. Die Unterscheidung von partiellen und vollständigen Mengen ermöglicht z.B. direkte Mengenvergleiche. In FLORID müssen diese vom Benutzer mittels Stratifikation umgesetzt werden.

6 Zusammenfassung

Wegen ihrer mächtigen Sprachkonstrukte und der Einbindung des Objektparadigmas in den Kontext einer logischen Sprache stellt F-Logik einen der fortgeschrittensten Ansätze zur Integration deklarativer und objektorientierter Konzepte dar. Mit FLORID wurde nun zum ersten Mal der gesamte F-Logik-Sprachumfang implementiert, wobei eine mengenorientierte Bottom-Up-Evaluierungsstrategie zum Einsatz kam. Dies ermöglicht die praktische Anwendung von F-Logik insbesondere in Bereichen, in denen objektorientierte Modellierung und Metadatenbehandlung vorteilhaft genutzt werden können, wie wir am Beispiel einer Schemaintegration zweier relationaler Datenbanken aufzeigten. Weiterhin gingen wir auf die Architektur des Prototyps und die Umsetzung der theoretischen Grundlagen aus [17] ein. Derzeit arbeiten wir an einer CORBA-Anbindung von FLORID an persistente Datenbanken sowie an der Implementierung von Optimierungsstrategien (Magic Sets).

Danksagung

Unser besonderer Dank gilt Heinz Uphoff für wertvolle Anregungen und Beiträge zum F-Logik-Projekt. Ferner danken wir auch den weiteren Mitgliedern unserer Arbeitsgruppe, Wolfgang May und Bertram Ludäscher, sowie den wissenschaftlichen Hilfskräften und Diplomanden, die an FLORID mitgearbeitet haben.

References

1. M. L. Barja, N. W. Paton, A. A. A. Fernandes, M. H. Williams, and A. Dinn. An effective deductive object-oriented database through language integration. In Bocca et al. [3], pages 463–474.
2. J. Biskup, R. Menzel, and T. Polle. Transforming an entity-relationship schema into object-oriented database schemas. In J. Eder and L. A. Kalinichenko, editors, *Proceedings of the Second International Workshop on Advances in Databases and Information Systems*, Workshops in Computing. Springer, 1995.
3. J. B. Bocca, M. Jarke, and C. Zaniolo, editors. *Proc. Intl. Conference on Very Large Data Bases*, Santiago de Chile, 1994.
4. Y. Chang, L. Raschid, and B. Dorr. Transforming queries from a relational schema to an equivalent object schema: a prototype based on F-logic. In *International Symposium on Methodologies in Information Systems, (ISMIS-94)*, Oct. 1994.
5. D. Chimenti, R. Gamboa, R. Krishnamurphy, S. Naqvi, S. Tsur, and C. Zaniolo. The $\mathcal{LDL}$ system prototype. *IEEE Transactions on Knowledge and Data Engineering*, 2(1), Mar. 1990.

6. M. Derr, S. Morishita, and G. Phipps. The glue-nail! deductive database system: Design, implementation, and evaluation. *The VLDB Journal*, 3(2):123–160, Apr. 1994.
7. G. Dobbie and R. Topor. On the declarative and procedural semantics of deductive object-oriented systems. *Journal of Intelligent Information Systems*, 4(2):193–219, 1995.
8. C. Fahrner and G. Vossen. Transforming relational database schemas into object-oriented schemas according to ODMG-93. In *Proc. Intl. Conference on Deductive and Object-Oriented Databases (DOOD)*, 1995.
9. B. Freitag, H. Schütz, G. Specht, and R. Bayer. LOLA — ein deduktives Datenbanksystem. In R. Bayer, T. Härder, and P. Lockemann, editors, *Objektbanken für Experten*. Springer, 1992.
10. J. Frohn, R. Himmeröder, P.-T. Kandzia, and C. Schlepphorst. FLORID, Version 1.0, user manual, 1996. Available from ftp://ftp.informatik.uni-freiburg.de/pub/florid/manual.ps.gz.
11. J. Frohn, R. Himmeröder, P.-T. Kandzia, and C. Schlepphorst. How to write F-Logic programs in FLORID, 1996. Available from ftp://ftp.informatik.uni-freiburg.de/pub/florid/tutorial.ps.gz.
12. J. Frohn, G. Lausen, and H. Uphoff. Access to objects by path expressions and rules. In Bocca et al. [3].
13. H. Garcia-Molina, Y. Papakonstantinou, D. Quass, A. Rajaraman, Y. Sagiv, J. Ullman, and J. Widom. The TSIMMIS approach to mediation: Data models and languages (extended abstract). In *Next Generation Information Technologies and Systems*, 1995.
14. M. H. Jamil and L. V. S. Lakshmanan. A declarative semantics for behavioral inheritance and conflict resolution. In *Proc. of the 12th International Logic Programming Symposium (ILPS)*, pages 130–144, 1995.
15. M. Jarke, R. Gallersdörfer, M. A. Jeusfeld, and M. Staudt. ConceptBase - a deductive object base for meta data management. *Journal of Intelligent Information Systems*, 4(2):167–192, 1995.
16. D. A. Keim, H.-P. Kriegel, and A. Miethsam. Object-oriented querying of existing relational databases. In *Proc. Intl. Conference on Database and Expert Systems Applications (DEXA)*, 1993.
17. M. Kifer, G. Lausen, and J. Wu. Logical foundations of object-oriented and frame-based languages. *Journal of the ACM*, 42(4):741–843, July 1995.
18. P. Kolaitis and C. Papadimitriou. Why not negation by fixpoint? In *Proc. ACM Symposium on Principles of Database Systems*, pages 231–239, 1988.
19. L. V. Lakshmanan, F. Sadri, and I. N. Subramanian. On the logical foundations of schema integration and evolution in heterogenous database systems. In *Proc. Intl. Conference on Deductive and Object-Oriented Databases (DOOD)*, 1993.
20. M. Liu. ROL: A typed deductive object base language. *Proc. Intl. Conference on Database and Expert Systems Applications (DEXA)*, 1996.
21. S. Naqvi and S. Tsur. *A logical Language for Data and Knowledge Bases*. Computer Science Press, New York, 1989.
22. G. Phipps, M. A. Derr, and K. A. Ross. Glue-Nail : A deductive database system. In *Proc. of the ACM SIGMOD Conf. on Management of Data*, 1991.
23. R. Ramakrishnan, D. Srivastava, and S. Sudarshan. CORAL: Control, relations and logic. In L.-Y. Yuan, editor, *Proc. Intl. Conference on Very Large Data Bases*, Vancouver, 1992.

24. R. Ramakrishnan, D. Srivastava, S. Sudarshan, and P. Seshadri. Implementation of the CORAL deductive database system. In *Proc. ACM SIGMOD Intl. Conference on Management of Data*, 1993.
25. J. Vaghani, K. Ramamohanarao, D. B. Kemp, Z. Somogyi, P. J. Stuckey, T. S. Leask, and J. Harland. The Aditi deductive database system. *The VLDB Journal*, 3(2):245–287, Apr. 1994.
26. P. H. Winston. *Artificial Intelligence.* Addison Wesley, 1993.