

EGE ÜNİVERSİTESİ FEN BİLİMLERİ ENSTİTÜSÜ

(DOKTORA TEZİ)

**Anlamsal Web için Kural Tabanlı bir
Melez Çıkarsama Yöntemi**

Övünç Öztürk

Tez Danışmanı: Yrd.Doç.Dr. Murat Osman Ünahr

Bilgisayar Mühendisliği Anabilim Dalı

Bilim Dalı Kodu: 619.01.00

Sunuş Tarihi: 18.11.2009

Bornova-İZMİR

2009

Övünç Öztürk tarafından doktora tezi olarak sunulan "**Anlamsal Web için Kural Tabanlı bir**

Melez Çıkarsama Yöntemi" başlıklı bu çalışma E.Ü. Lisansüstü Eğitim ve Öğretim Yönetmeliği ile E.Ü. Fen Bilimleri Enstitüsü Eğitim ve Öğretim Yönergesi'nin ilgili hükümleri uyarınca tarafımızdan değerlendirilerek savunmaya değer bulunmuş ve **18 Kasım 2009** tarihinde yapılan tez savunma sınavında aday oy birliği/oy çokluğu ile başarılı bulunmuştur.

Jüri Üyeleri:

İmza

Jüri Başkanı: Yrd.Doç.Dr. Murat Osman Ünalır

.....

Raportör Üye: Prof.Dr. Oğuz Dikenelli

.....

Üye: Yrd.Doç.Dr. Adil Alpkoçak

.....

Üye: Yrd.Doç.Dr. Murat Komesli

.....

Üye: Yrd.Doç.Dr. R. Cenk Erdur

.....

ÖZET

Anlamsal Web için Kural Tabanlı bir Melez Çıkarsama Yöntemi

Öztürk, Övünç

Doktora Tezi, Bilgisayar Mühendisliği Bölümü

Tez Yöneticisi: Yrd.Doç.Dr. Murat Osman Ünalır

Kasım 2009, 175 sayfa

Anlamsal Web üzerinde, büyük bir kısmı örneksel verilerden meydana gelen büyük ontolojiler kullanılacaktır. Bu ontolojiler üzerinde ölçeklenebilir çıkarsama yapabilecek yöntemleri geliştirmek Anlamsal Web'in gerçekleştirimi önündeki önemli engellerden biridir. Bu tez kapsamında Anlamsal Web ontolojileri üzerinde ölçeklenebilir çıkarsama ve sorgulama yapabilecek yeni bir yöntem sunulmaktadır. Bu yöntem üç temel parçadan oluşmaktadır: uzantı tabanlı bilgi modeli, bu modele uygun bir veritabanı şeması ve bu model üzerinde çalışan çıkarsama ve sorgulama algoritmaları. Yöntemin dayandığı uzantı tabanlı bilgi modeli ontolojileri özetleyerek çıkarsamanın bu özet üzerinde gerçekleştirilmesini sağlar. Çıkarsama yalnızca bu özet üzerinde gerçekleştiği için bellekte gerçekleştirilen ileriye zincirlemenin süre ve bellek gereksinimleri önemli ölçüde düşer. Sorgulama algoritması ise veritabanında saklanan ontolojik veriyi ve çıkarsanmış veriyi kullanarak sorguların eksiksiz ve doğru cevaplanabilmesi için özelleşmiş bir geriye zincirleme algoritmasıyla eksik kalan çıkarsamayı tamamlar. Tez kapsamında bu yöntemin gerçekleştirimi ve başarımları deneyleri yapılmıştır. Ayrıca yöntemin nitelikleri ve başarımları, varolan benzer uygulamalarla karşılaştırılmıştır.

Anahtar Sözcükler: Ölçeklenebilirlik, pD*, kural tabanlı çıkarsama, uzantı-tabanlı model, ileriye zincirleme, geriye zincirleme

ABSTRACT**A Hybrid Rule Based Reasoning Algorithm
for Semantic Web**

Öztürk, Övünç

PhD in Computer Eng.

Supervisor: Yrd.Doç.Dr. Murat Osman Ünalır

November 2009, 175 pages

Ontologies with large scale of instance data will be used on Semantic Web. Developing scalable reasoning methods on these ontologies is one of the main obstacles in the realization of Semantic Web. This thesis presents a novel reasoning and querying method that scales well with ontologies having large scale of instance data. This method consists of three parts: extension based knowledge model, a database schema for this knowledge model and inference and query answering algorithms for the extension based knowledge model. The extension based knowledge model is an ontology summarization mechanism. The reasoning process is conducted on the ontology summary; therefore the time and memory consumptions of the forward chaining process decreases significantly. The query answering algorithm completes the reasoning process using the inferred data and the ontological data stored in database. In this thesis, the implementation and performance tests of the proposed method are completed. Besides, the features and performance of the method are compared with related applications in the literature.

Keywords: Scalability, pD*, rule based reasoning, extension-based model, forward chaining, backward chaining

TEŞEKKÜR

Bu tez çalışması sırasında göstermiş olduğu ilgi ve sabrından dolayı değerli danışmanım Yrd.Doç.Dr. Murat Osman Ünalır'a teşekkürlerimi sunarım. Sevgi ve desteklerinden dolayı eşime, sevgili anneme, babama ve kardeşime minnettar olduğumu belirtmek isterim. Ege Üniversitesi Bilgisayar Mühendisliği'nin değerli öğretim üyelerine ve saygıdeğer jüri üyelerime teşekkür etmek isterim.

İÇİNDEKİLER

ÖZET	v
ABSTRACT	vii
TEŞEKKÜR	ix
İÇİNDEKİLER	xi
ŞEKİLLER DİZİNİ	xvii
ÇİZELGELER DİZİNİ	xxi
ALGORİTMALAR DİZİNİ	xxvi
1 . Giriş	1
2 . Temel Kavramlar	5
2.1 Betim Mantığı ve Ontoloji Kavramı	5
2.2 Ontoloji Dilleri	9
2.2.1 OWL DLP	12
2.2.2 pD^*	13
2.3 Çıkarsama Yaklaşımları	14
3 . Uzantı Tabanlı Bilgi Modeli	19
3.1 Uzantı Tabanlı Bilgi Modeli Yapıtaşları	20
3.2 Uzantı Tabanlı Bilgi Modeli Veritabanı Şeması	22
3.3 Uzantı Tabanlı Bilgi Modeli Dönüşüm Algoritması	23
4 . Uzantı Tabanlı Çıkarsama ve Sorgulama	25
4.1 Uzantı Tabanlı Bilgi Modeli Çıkarsama Kuralları	28
4.1.1 Ontoloji Diline Ait Kurallar	28
4.1.2 Uzantı Tabanlı Bilgi Modelinin Gerektirdiği Kurallar	42
4.2 Çıkarsama Süreci	44

4.2.1	Üçlülerin Filtrelenmesi	44
4.2.2	İleriye Zincirleme Algoritmasının Uygulanması . . .	44
4.2.3	Veritabanında Uzantıların İşlenmesi ve Genişletilmesi	45
4.3	Sorgulama Süreci	49
4.3.1	Hazırlık Aşaması	54
4.3.2	Geçiş Aşaması	54
4.3.3	Sonuç Aşaması	56
5 .	Örnek Uygulama	59
5.1	Uzantı Tabanlı Bilgi Modeline Dönüştürme	60
5.2	Üçlüler Filtreleme	67
5.3	İleriye Zincirleme	68
5.4	Veritabanında Uzantıları İşleme ve Genişletme	69
5.5	Sorgulama	72
5.5.1	Örnek Sorgu 1	72
5.5.2	Örnek Sorgu 2	79
5.5.3	Örnek Sorgu 3	81
5.5.4	Örnek Sorgu 4	85
6 .	Uzantı Tabanlı Bilgi Modelinin Analizi ve Doğrulanması	87
6.1	Ontoloji Ölçütlerini Kullanarak Faydalanma Oranını Tahminleme	87
6.2	Faydalanma Oranının Hesaplanması	88
6.3	Sınama ve Doğrulama	90
7 .	Deneyler	107
7.1	Standart Çıkarsama Motoru	107
7.2	LUBM Değerlendirme Uygulaması	109
7.3	UOBM Değerlendirme Uygulaması	115
7.4	Başarım Değerlendirme Ölçütleri	119
7.5	Deney Sonuçları	120
7.5.1	Birinci Deney	121
7.5.2	İkinci Deney	131
7.5.3	Üçüncü Deney	133

7.5.4 Dördüncü Deney	141
8 . Varolan Sistemlerle ve Tekniklerle Karşılaştırma	143
8.1 SHER	143
8.2 Minerva	144
8.3 OWLDB	145
8.4 DLDB	146
8.5 Jena	148
8.6 3Store	149
8.7 Niteliksel Karşılaştırma	150
8.8 Başarım Karşılaştırması	153
8.8.1 Sesame	153
8.8.2 Jena	155
8.8.3 OWLIM	157
9 . Sonuçlar	159
Kaynakça	163
Ek A. TÜRKÇE-İNGİLİZCE TERİMLER SÖZLÜĞÜ	169
ÖZGEÇMİŞ	175

ŞEKİLLER DİZİNİ

<u>Şekil</u>		<u>Sayfa</u>
2.1	OWL altdilleri ve bunlarla ilişkili diller.	12
3.1	Sınıf örneği gruplama yapıtaşları.	20
3.2	Nitelik gruplama yapıtaşları.	22
4.1	Uzantı tabanlı çıkarsama süreci.	27
4.2	Kuralların örüntüler doğrultusunda işlenmesi.	31
4.3	Nitelik sıradüzen örneği.	48
4.4	Sorgu ağacı bileşenleri ve aralarındaki ilişkiler.	53
5.1	Örnek ontoloji şeması.	59
5.2	<i>Örnek Sorgu 1</i> ağacı (ε_{α_1} açıldıktan sonra).	73
5.3	<i>Örnek Sorgu 1</i> ağacı (ε_{α_2} açıldıktan sonra).	75
5.4	<i>Örnek Sorgu 1</i> sorgu ağacı (ε_{α_3} açıldıktan sonra).	76
5.5	<i>Örnek Sorgu 1</i> sorgu ağacı (ε_{α_4} açıldıktan sonra).	78
5.6	<i>Örnek Sorgu 2</i> ağacı.	80
5.7	<i>Örnek Sorgu 3</i> hazırlık aşaması.	83
5.8	<i>Örnek Sorgu 3</i> geçiş aşaması.	83
5.9	<i>Örnek Sorgu 4</i> hazırlık aşaması.	86
7.1	Örnek bir Rete ağı.	109
7.2	Univ-Bench şeması.	110
7.3	LUBM şeması.	115
7.4	UOBM şeması.	116
7.5	Standart ve eniyilenmiş uygulamaya ait ontoloji açılış sürelerinin karşılaştırması.	122
7.6	Standart ve eniyilenmiş uygulama arasındaki açılış başarımının iyileşme oranı.	123
7.7	Standart uygulama ontoloji açılış süreleri.	123
7.8	Eniyilenmiş uygulama ontoloji açılış süreleri.	124

7.9	Çıkarsamaya katılan üçlü sayısının toplam üçlü sayısına oranının artan veri miktarı ile değişimi.	125
7.11	Standart ve eniyilenmiş uygulamanın bellek kullanım miktarları.	126
7.10	Saniyede yüklenen üçlü sayısının artan veri miktarıyla değişimi.	126
7.12	Standart ve eniyilenmiş çıkarsama motorunun LUBM değerlendirme uygulaması için sorgu sonuçları.	127
7.13	Standart ve eniyilenmiş çıkarsama motorunun LUBM değerlendirme uygulaması için sorgu sonuçları (Devamı).	128
7.14	Standart ve eniyilenmiş çıkarsama motorunun LUBM değerlendirme uygulaması sorguları için sonuç başına sorgu işletme süresi değerleri.	131
7.15	Dönüşümün başarımlar oranının LUBM 1-50 üniversite veri kümesi ile değişimi.	132
7.16	LUBM(50,0) veri kümesi için sorgu başarımlarının değişimi. . .	133
7.17	Standart ve eniyilenmiş uygulamaya ait ontoloji açılış sürelerinin karşılaştırması.	134
7.18	Standart ve eniyilenmiş uygulama arasındaki açılış başarımlarının iyileşme oranı.	135
7.19	Standart uygulama ontoloji açılış süreleri (ms).	135
7.20	Eniyilenmiş uygulama ontoloji açılış süreleri (ms).	136
7.21	Çıkarsamaya katılan üçlü sayısının toplam üçlü sayısına oranının artan veri miktarı ile değişimi.	136
7.22	Saniyede yüklenen üçlü sayısının artan veri miktarıyla değişimi.	137
7.23	Standart ve eniyilenmiş uygulamanın bellek kullanım miktarları.	137
7.24	Standart ve eniyilenmiş çıkarsama motorunun LUBM değerlendirme uygulaması için sorgu sonuçları.	140
7.25	Standart ve eniyilenmiş çıkarsama motorunun LUBM değerlendirme uygulaması için sorgu sonuçları (Devamı).	141
7.26	UOBM değerlendirme uygulaması için model dönüşümünün başarımlarının artan veri kümesi ile değişimi.	142

7.27	UOBM değerlendirme uygulaması sorgu başarımının artan veri kümesi ile değişimi.	142
8.1	OWLDB şeması	146
8.2	3store veritabanı şeması.	150
8.3	Sesame, standart çıkarsama motoru ve eniyilenmiş çıkarsama motorunun 1, 4 ve 14 numaralı sorgu başarımları.	154
8.4	Sesame ve eniyilenmiş çıkarsama motorunun artan veri miktarıyla 1 numaralı sorgu başarımları.	154
8.5	Sesame ve eniyilenmiş çıkarsama motorunun artan veri miktarıyla 4 numaralı sorgu başarımları.	155
8.6	Sesame ve eniyilenmiş çıkarsama motorunun artan veri miktarıyla 14 numaralı sorgu başarımları.	155
8.7	Jena, standart çıkarsama motoru ve eniyilenmiş çıkarsama motorunun bir üniversite için sorgu başarımları.	156
8.8	Jena ve eniyilenmiş çıkarsama motorunun beş üniversite için sorgu başarımları.	157
8.9	Standart çıkarsama motoru, eniyilenmiş çıkarsama motoru ve OWLIM'in ontoloji açılış süreleri.	158
8.10	Eniyilenmiş çıkarsama motoru ve OWLIM'in bellek kullanımı.	158

ÇİZELGELER DİZİNİ

Çizelge		Sayfa
3.1	<i>contains</i> ve <i>statements</i> tablolarına ait sahalar.	23
4.1	Kuralların işleme zamanlarını belirleyen örüntüler.	30
4.2	<i>D*</i> gerektirme kuralları.	35
4.3	Dönüştürülmüş <i>D*</i> gerektirme kuralları.	36
4.4	<i>p</i> gerektirme kuralları.	40
4.5	Dönüştürülmüş <i>p</i> gerektirme kuralları.	41
4.6	<i>p</i> dili uzantı tabanlı bilgi modeli gerektirme kuralları.	42
4.7	<i>D*</i> dili uzantı tabanlı bilgi modeli gerektirme kuralları.	43
4.8	Karmaşık sınıf uzantı bilgilerini saklayan veri yapıları.	46
4.9	Nitelik uzantılarının karakteristiklerini saklayan veri yapıları.	47
4.10	Örnek niteliklerin nihai uzantılarının hesaplanması.	48
4.11	Bağlantı düğümlerine karşılık gelen <i>SQL</i> sorguları.	57
5.1	Örnek ontolojideki sınıf örnekleri.	60
5.2	Örnek ontolojideki nitelik örnekleri.	61
5.3	<i>classExtensionResearchGroup</i> uzantısının elemanlarını belirten üçlüler.	62
5.4	<i>classExtensionDepartment</i> uzantısının elemanlarını belirten üçlüler.	63
5.5	<i>classExtensionUniversity</i> uzantısının elemanlarını belirten üçlüler.	63
5.6	<i>classExtensionProfessor</i> uzantısının elemanlarını belirten üçlüler.	63
5.7	<i>subjectExtensionsubOrganizationOf</i> uzantısının elemanlarını belirten üçlüler.	64
5.8	<i>objectExtensionsubOrganizationOf</i> uzantısının elemanlarını belirten üçlüler.	64
5.9	<i>subjectExtensionworksFor</i> uzantısının elemanlarını belirten üçlüler.	65

5.10	<i>objectExtensionworksFor</i> uzantısının elemanlarını belirten üçlüler.	65
5.11	<i>subjectExtensiondegreeFrom</i> uzantısının elemanlarını belirten üçlüler.	65
5.12	<i>objectExtensiondegreeFrom</i> uzantısının elemanlarını belirten üçlüler.	66
5.13	<i>subjectExtensionheadOf</i> uzantısının elemanlarını belirten üçlüler.	66
5.14	<i>objectExtensionheadOf</i> uzantısının elemanlarını belirten üçlüler.	66
5.15	Yaklaşım uygulandıktan sonra çıkarsamaya katılan üçlü sayısı.	67
5.16	Yaklaşım uygulanmadan önce çıkarsanan üçlüler.	68
5.17	Yaklaşım uygulandıktan sonra çıkarsanan üçlüler.	69
5.18	Anonim sınıflar ve tanımları.	70
5.19	<i>RestrictionComplexExtensions</i> veri yapısı.	70
5.20	<i>SetComplexExtensions</i> veri yapısı.	70
5.21	<i>ComplexExtensions</i> veri yapısı.	71
5.22	<i>PropertiesWithCharacteristics</i> veri yapısı.	71
5.23	Geçişkenlik kuralının işletilmesi sonucu bulunan nitelik örnekleri.	71
5.24	Örnek Sorgu 1 sonucu.	74
5.25	Örnek Sorgu 2 sonucu.	81
5.26	Örnek Sorgu 3 sonucu.	83
5.27	Örnek Sorgu 4 sonucu.	86
6.1	Sınama sonuçları.	91
7.1	1, 5, 10, 20 ve 50 üniversiteye ait veri setlerinin boyutları. . .	110
7.2	LUBM ve UOBM test uygulamalarına ait sınıf, nitelik ve üniversite başına örnek sayıları.	116
7.3	Standart uygulama için ontoloji açılış süreleri (ms).	122
7.4	Eniyilenmiş uygulama için ontoloji açılış süreleri (ms). . . .	122

7.5	Standart ve eniyilenmiş uygulama için saniyede açılan üçlü sayısı.	125
7.6	Standart çıkarsama motoru için sorgu çalışma süreleri (ms). .	129
7.7	Eniyilenmiş çıkarsama motoru için sorgu çalışma süreleri (ms).	130
7.8	Standart uygulama için ontoloji açılış süreleri (ms).	133
7.9	Eniyilenmiş uygulama için ontoloji açılış süreleri (ms). . . .	134
7.10	Standart ve eniyilenmiş uygulama için saniyede açılan üçlü sayısı.	136
7.11	Sorgu çalışma süreleri (ms).	138
7.12	Sorgu çalışma süreleri (ms).	139
8.1	Tez çalışmasının diğerleri ile karşılaştırılması.	152

ALGORİTMALAR VE SORGULAR DİZİNİ

<u>Algoritma</u>	<u>Sayfa</u>
3.1 Uzantı tabanlı modele dönüşüm algoritması.	24
4.1 Düğüm açılım algoritması.	55
5.1 ε_{α_1} karmaşık uzantısının SQL sorgusu karşılığı.	74
5.2 ε_{α_2} karmaşık uzantısının SQL sorgusu karşılığı.	74
5.3 ε_{α_3} karmaşık uzantısının SQL sorgusu karşılığı.	77
5.4 ε_{α_4} karmaşık uzantısının SQL sorgusu karşılığı.	77
5.5 <i>Örnek Sorgu 1</i> 'in SQL karşılığı.	77
5.6 <i>Örnek Sorgu 2</i> 'nin SQL karşılığı.	81
5.7 <i>Örnek Sorgu 3</i> 'ün SQL karşılığı.	84
5.8 <i>Örnek Sorgu 4</i> 'ün SQL karşılığı.	86
6.1 <i>owl:FunctionalProperty 001</i> numaralı sına ma durumu için öncül ontoloji.	92
6.2 <i>owl:FunctionalProperty 001</i> numaralı sına ma durumu için sonuç.	92
6.3 <i>owl:FunctionalProperty 002</i> numaralı sına ma durumu için öncül ontoloji.	93
6.4 <i>owl:FunctionalProperty 002</i> numaralı sına ma durumu için sonuç.	93
6.5 <i>owl:InverseFunctionalProperty 001</i> numaralı sına ma durumu için öncül ontoloji.	94
6.6 <i>owl:InverseFunctionalProperty 001</i> numaralı sına ma durumu için sonuç.	94
6.7 <i>owl:InverseFunctionalProperty 002</i> numaralı sına ma durumu için öncül ontoloji.	95
6.8 <i>owl:InverseFunctionalProperty 002</i> numaralı sına ma durumu için sonuç.	95
6.9 <i>owl:SymmetricProperty 001</i> numaralı sına ma durumu için öncül ontoloji.	96

6.10	<i>owl:SymmetricProperty 001</i> numaralı sinama durumu için sonuç.	96
6.11	<i>owl:TransitiveProperty 001</i> numaralı sinama durumu için öncül ontoloji.	97
6.12	<i>owl:TransitiveProperty 001</i> numaralı sinama durumu için sonuç.	97
6.13	<i>owl:allValuesFrom 001</i> numaralı sinama durumu için öncül ontoloji.	98
6.14	<i>owl:allValuesFrom 001</i> numaralı sinama durumu için sonuç.	98
6.15	<i>owl:equivalentClass 001</i> numaralı sinama durumu için öncül ontoloji.	99
6.16	<i>owl:equivalentClass 001</i> numaralı sinama durumu için sonuç.	99
6.17	<i>owl:equivalentClass 002</i> numaralı sinama durumu için öncül ontoloji.	100
6.18	<i>owl:equivalentClass 002</i> numaralı sinama durumu için sonuç.	100
6.19	<i>owl:equivalentClass 003</i> numaralı sinama durumu için öncül ontoloji.	100
6.20	<i>owl:equivalentClass 003</i> numaralı sinama durumu için sonuç.	101
6.21	<i>owl:equivalentProperty 001</i> numaralı sinama durumu için öncül ontoloji.	101
6.22	<i>owl:equivalentProperty 001</i> numaralı sinama durumu için sonuç.	102
6.23	<i>owl:equivalentProperty 002</i> numaralı sinama durumu için öncül ontoloji.	102
6.24	<i>owl:equivalentProperty 002</i> numaralı sinama durumu için sonuç.	103
6.25	<i>owl:equivalentProperty 003</i> numaralı sinama durumu için öncül ontoloji.	103
6.26	<i>owl:equivalentProperty 003</i> numaralı sinama durumu için sonuç.	103
6.27	<i>owl:equivalentProperty 006</i> numaralı sinama durumu için öncül ontoloji.	104

6.28	<i>owl:equivalentProperty</i> 006 numaralı sinama durumu için sonuç.	104
6.29	<i>owl:inverseOf</i> 001 numaralı sinama durumu için öncül ontoloji.	105
6.30	<i>owl:inverseOf</i> 001 numaralı sinama durumu için sonuç. . . .	105
6.31	<i>owl:sameAs</i> 001 numaralı sinama durumu için öncül ontoloji.	106
6.32	<i>owl:sameAs</i> 001 numaralı sinama durumu için sonuç.	106
7.1	LUBM Sorgu 1.	111
7.2	LUBM Sorgu 2.	111
7.3	LUBM Sorgu 3.	111
7.4	LUBM Sorgu 4.	111
7.5	LUBM Sorgu 5.	112
7.6	LUBM Sorgu 6.	112
7.7	LUBM Sorgu 7.	112
7.8	LUBM Sorgu 8.	112
7.9	LUBM Sorgu 9.	113
7.10	LUBM Sorgu 10.	113
7.11	LUBM Sorgu 11.	113
7.12	LUBM Sorgu 12.	114
7.13	LUBM Sorgu 13.	114
7.14	LUBM Sorgu 14.	114
7.15	UOBM Sorgu 1.	117
7.16	UOBM Sorgu 2.	117
7.17	UOBM Sorgu 3.	117
7.18	UOBM Sorgu 4.	117
7.19	UOBM Sorgu 5.	117
7.20	UOBM Sorgu 6.	118
7.21	UOBM Sorgu 7.	118
7.22	UOBM Sorgu 8.	118
7.23	UOBM Sorgu 9.	118
7.24	UOBM Sorgu 10.	118
7.25	UOBM Sorgu 11.	118
7.26	UOBM Sorgu 12.	119

7.27	UOBM <i>Sorgu</i> 13.	119
------	-------------------------------	-----

1 Giriş

Anlamsal Web günümüzde yaygın bir kullanıma sahip olan Web'e bir eklenti olarak tasarlanmıştır. Anlamsal Web, yapılandırılmış bilgiye dayanarak, insan ve makine işbirliğini daha ileri götürmektedir. Anlamsal Web üzerinde bilgi anlamı ile birlikte yer alacaktır, başka bir deyişle, yazılım etmenleri Web üzerinde yer alan varlıkların anlamlarını bilecek ve bu bilgiyi kullanabilecektir.

Anlamsal Web'in tanıtıldığı makale olan (Berners-Lee et al., 2001)'de, ontolojilerin bilgi gösterimi ve bilginin anlamının paylaşımı için kullanılması önerilmektedir. Ontoloji kavramı temellerini felsefeden almasına rağmen bilgi gösterimi alanında uzun yıllardır kullanılmaktadır. (Gruber, 1995)'te ontoloji ortak bir kavramsallaştırmanın biçimsel gösterimi olarak tanımlanmaktadır. Ontolojiler, Anlamsal Web kullanıcılarına dünyayı modelleyebilecekleri bir temel sunmaktadır. Anlamsal Web çalışmalarına önderlik eden W3C¹, Web üzerinde ontolojilerin yaratılması ve paylaşılması için OWL (Smith et al., 2004) dilini önermektedir. Bir Anlamsal Web ontolojisi iki parçadan oluşmaktadır. Bu parçalardan ilki örneklerin tanımlanabilmesi için gereken sınıfların, niteliklerin ve bunlar arasındaki ilişkilerin tanımlarının yapıldığı şema kısmıdır. Bu kısma ontolojinin “işlemsel” parçası veya TBox adı da verilir. İkinci parça ise örnek tanımlarının bulunduğu kısımdır. Bu kısma “dışlamsal” kısım veya ABox denir.

Günümüzde Web üzerinde yer alan kaynaklara ilişkin verinin büyüklüğü gözönüne alındığında sıradan bir Anlamsal Web kullanıcısının üzerinde çalışması gereken ontolojinin ABox kısmının ne kadar büyük olacağı tahmin edilebilir (Hitler and Vrandečić, 2006). Gerçek dünya ontolojilerinin, göreceli olarak çok küçük bir kısmı TBox bilgisi olacaktır. Ontolojilerde ABox aynı zamanda ontolojinin en değişken kısmıdır (Weithöner et al., 2007). Ontolojilerin ABox kısmı büyüdükçe çıkarsama sürecindeki tekrarlar artar ve bunun sonucunda çıkarsama yavaşlar. Bu nedenle Anlamsal Web üzerinde yapılacak çıkarsamanın karmaşıklığı, ABox çıkarsamanın karmaşıklığı ile çok yakından ilişkilidir. ABox kısmının karmaşıklığına aynı zamanda “veri karmaşıklığı” adı da verilmektedir. Çıkarsamanın karmaşıklığındaki artışa bağlı olarak ölçeklenebilirliğin düşmesi, Anlamsal Web'in en

¹W3C Semantic Web Activity (<http://www.w3.org/2001/sw/>)

önemli problemlerinden biridir.

OWL ontoloji dilinin dayandığı temellerden biri olan Betim Mantığı, karmaşık sınıflar, aralarındaki ilişkiler ve bunların örnekleri üzerinde çıkarsama yapabilmek için tasarlanmış mantıksal dillerden biridir. Betim Mantığı üzerinde yapılan araştırmalar, yeni çıkarsama ve sorgulama yöntemlerinin aranması üzerinde yoğunlaşmaktadır. Bu alandaki yapılmış olan çalışmaların, Anlamsal Web üzerinde uygulanamamasının en temel nedeni, çalışmalar kapsamında üzerinde çalışılan veri miktarının, Anlamsal Web üzerinde çalışması beklenen veri miktarının yanında çok küçük kalmasıdır. Anlamsal Web ontolojilerinde örneksel veri miktarı çok büyük oranlarda artış göstermektedir. Bu nedenle şimdiye kadar tasarlanan çıkarsama algoritmaları ölçeklenebilir olmaktan uzaktır. Ayrıca varolan algoritmalar, bellek tabanlıdır. Diğer bir deyişle, bu algoritmalar ontolojiler üzerinde çıkarsama yapabilmek için ontolojinin tamamını belleğe yüklemek zorundadırlar. Günümüzde yaygın olarak kullanılan bilgisayarların sunabildiği ortalama bellek kapasitesi bu uygulamalar için yetersiz kalmaktadır.

Anlamsal Web tanımı gereği dağıtık bir sistem olmasına karşın, Anlamsal Web üzerinde tüm uygulamalar dağıtık olmak zorunda değildir. Başka bir deyişle, Anlamsal Web üzerinde, yerel ontolojilerle çıkarsama yaparak, bunların diğer uygulamalar tarafından sorgulanmasını sağlayacak uygulamalara da ihtiyaç duyulmaktadır. Bu uygulamaların sunduğu servisler dağıtık çalışan uygulamalar tarafından kullanarak, yerel ontolojilerin Anlamsal Web ile bütünleştirilmesi gerçekleştirilecektir.

Bu tez kapsamında, yerel ontolojiler üzerinde ölçeklenebilir çıkarsama yapabilmek için bir algoritma sunulmaktadır. Bu algoritma, Anlamsal Web çalışmaları kapsamında geliştirilmiş ve W3C tarafından standartlaştırılmış OWL ontoloji diline özelleşmiş kural tabanlı bir melez çıkarsama algoritması sunmaktadır. Algoritmanın melez olarak tanımlanmasının nedeni, her ikisi de kural tabanlı çıkarsama yöntemleri olan ileriye ve geriye zincirleme tekniklerinin birlikte kullanılmasıdır. Tez kapsamında önerilen çıkarsama ve sorgulama algoritması büyük ölçekli ontolojik veriler için tasarlanmıştır. Sunulan algoritma, tüm çıkarsama ve sorgulama gereksinimlerini karşılamak üzere önerilmemiştir. Bu algoritmanın amacı düşük belleğe sahip ortamlarda bile verimli olarak çıkarsama ve sorgulama yapabilmektir. Tez

kapsamında, algoritmanın örnek bir gerçekleştirimi yapılmış ve belirli kısımlarla ilgili sözde kodlar sunulmuştur. Tezin sunduğu temel katkılar şunlardır:

- ontolojilerin veri karmaşıklığının düşürülmesi için değiştirilmiş bir bilgi gösterim yöntemi sunulmuştur.
- büyük ABox'lar üzerinde ölçeklenebilir çıkarsama ve sorgulama yapılabilmesi için tezde tanımlanan, değiştirilmiş bilgi gösterim yapısını kullanan bir algoritma tanımlanmış ve gerçekleştirimi yapılmıştır.
- güncel karşılaştırmalı değerlendirme uygulamaları kullanılarak algoritmanın başarımı sınanmış ve benzer uygulamalarla karşılaştırması yapılmıştır.

Tezin sağladığı katkıların, Anlamsal Web üzerinde çıkarsama ve sorgulama yapan uygulamalar açısından birden çok sonucu olacaktır. Önceki paragraflarda belirtildiği gibi Anlamsal Web ontolojileri büyük ontolojilerdir. Varolan çıkarsama yaklaşımlarının bellek tabanlı olması, uygulamaların bu tür büyük ontolojileri desteklemesini engellemektedir. Tezde önerilen uzantı tabanlı bilgi modeli, çıkarsamanın bir kısmını veritabanına taşıyarak, gereken bellek miktarını ontoloji şemasının karmaşıklığı ile sınırlandırmaktadır. Daha önce belirtildiği gibi, Anlamsal Web ontolojilerinde ontolojisi şeması, ontolojinin çok küçük bir kısmını oluşturduğundan, bellek ihtiyacı önemli ölçüde azalmaktadır. Çıkarsama karmaşıklığının örnek verisinin boyutundan bağımsız olmasının başka bir sonucu, ontolojideki örnek sayısı ne oranda artarsa artsın gereken bellek miktarının sabit kalmasıdır.

Ontolojinin veritabanında saklanarak, az bir kısmının belleğe aktarılması, çıkarsama ve sorgu yanıtlama süreçlerinin bellek gereksinimlerini önemli ölçüde düşürmüştür. Şema verisi üzerindeki çıkarsama bellekte yapılırken, örneklerle ilgili çıkarsama veritabanında saklanan bilgi üzerinde SQL sorguları ile gerçekleştirilmektedir. Böylelikle, çıkarsamanın tamamen bellek bağımlı bir süreç olması engellenmektedir. Veritabanı üzerinde çalıştırılan SQL sorgularının oluşturulması için bir sorgu algoritması önerilmektedir. Sorgu algoritması, desteklenen çıkarsama kurallarının bazılarının sorgu cevaplama anında işlenebilmesi için gereken veri yapılarını ve bunlar üzerinde çalışacak uygulamayı tanımlar. Bu yönüyle özelleşmiş bir geriye zincirleme algoritmasına benzer. Sorgu algoritmasının temel özelliği, kuralların sabit ve uygulamaya gömülü olmasıdır.

Tezin kalan kısmı şu şekilde düzenlenmiştir: İkinci bölüm, tez kapsamında sunulan çalışmayı anlatabilmek için gerekli olan temel kavramları vermektedir. Çıkarsama, sorgu cevaplama ve ontoloji kavramlarına ait tanımlar bu bölümde verilmektedir. Üçüncü bölümde uzantı tabanlı bilgi modeli anlatılmakta ve bir OWL ontolojisini, uzantı tabanlı modele dönüştürmek için gerekli adımlar tanımlanmaktadır. Dördüncü bölümde, uzantı tabanlı bilgi modelinden en üst düzeyde faydalanabilmek üzere özelleşmiş çıkarsama ve sorgulama algoritmaları önerilmektedir. Beşinci bölüm, örnek bir OWL ontolojisinin uzantı tabanlı modele dönüştürülmesini, bu model üzerinde özelleşmiş çıkarsama algoritmasının çalıştırılmasını ve örnek sorguların yanıtlanmasını göstermektedir. Altıncı bölümde ise uzantı tabanlı bilgi modelinin ve çıkarsama algoritmasının, çıkarsama sürecine olan etkilerinin analizi ve doğrulaması yapılmaktadır. Yedinci bölümde, LUBM ve UOBM adlı güncel karşılaştırmalı değerlendirme uygulamaları kullanılarak, algoritmanın ve gerçekleştirimin başarımı sınanmakta ve diğer benzer uygulamalarla karşılaştırması yapılmaktadır. Sekizinci bölümde, sunulan algoritmanın nitelikleri ve başarımı varolan çalışmalarla karşılaştırılmış ve çalışmanın literatürdeki yeri saptanmıştır. Dokuzuncu bölümde sonuçlar anlatılmış ve ileride yapılabilecek çalışmalar hakkında bilgi verilmiştir.

2 Temel Kavramlar

Bu bölüm tezin, devamında anlatılan konuların dayandığı temel kavramları anlatmaktadır. Bu bölümde öncelikle ontoloji kavramı tanımlanacak ve ontoloji modelleme dillerinin temel aldığı Betim Mantiğı konusu ele alınacaktır. Bölüm 2.2, Web-tabanlı ontoloji dillerini ve bu dillerin altdillerini anlatmaktadır. Son olarak, çıkarsama kavramı ve çıkarsama yaklaşımları sunulmaktadır.

2.1 Betim Mantiğı ve Ontoloji Kavramı

Bilgi gösterimi ve uzman sistem uygulamalarında yıllardır kullanılmakta olan ontolojiler, bir alandaki bilginin biçimsel ve yapılandırılmış gösterimi olarak tanımlanmaktadır (Gruber, 1995). İlgili alandaki bilgiye ait temel kavram tanımlarını ve bu kavramlar arasındaki ilişkileri, bilgisayarların da okuyabileceği bir biçimde saklayan ontolojiler, insanlar ve uygulamalar arasında paylaşılabilir, ortak bilgi oluşturabilmek için kullanılmaktadır.

Ontoloji tanımlamak için önerilmiş birçok dil bulunmaktadır. Bu diller, hedef sistemlerin ihtiyaçları doğrultusunda geliştirilmişlerdir. Dilin anlatımlılığı ile çıkarsama sürecinin maliyeti doğru orantılıdır. W3C, Anlamsal Web üzerinde kullanılacak standart ontoloji modelleme dilini OWL Web Ontoloji Dili olarak belirlemiştir (Smith et al., 2004). OWL dili, Betim Mantiğı'na temel almaktadır. Betim Mantiğı, anlamsal ağlarla ve çerçeve sistemlerle ilişkili bir mantıksal biçimciliktir (Baader et al., 2003).

Betim Mantiğı, bir uygulama alanına ait kavram tanımlarını yapılandırılmış ve biçimsel bir şekilde gösterebilmek için ortaya konmuş bilgi gösterim dilleri ailesidir. Betim Mantiğı dilleri, çerçevelerin ve anlamsal ağların bir uzantısı olarak tasarlanmışlardır. Eklemler mantığı ile birinci basamak mantık arasında bir çözüm sunarlar. Eklemler mantığına göre anlatım güçleri yüksektir, birinci basamak mantığa göre anlatım güçleri düşük fakat çıkarsama karmaşıklıkları daha azdır.

Aşağıda Betim Mantiğı'na ait sözdizimi sunulmaktadır:

- Sınıf adlarını belirten semboller
- Rol isimlerini belirten ikili ilişkiler kümesi

- Varolan sınıf ve rol isimlerinden özyinelemeli olarak yeni sınıf türeten yapıtaşları

Genel olarak, bir sınıf kendine ait sınıf örneklerini belirtir, roller ise sınıflar arası ilişkileri gösterir. Betim Mantığı'nın en temel özelliği varolan sınıf tanımlarından özyinelemeli olarak yeni sınıflar tanımlamayı sağlayan yapıtaşlarıdır. Bazı yaygın yapıtaşları birinci basamak mantık kapsamında kullanılan mantıksal operatörlere karşılık gelirler. Örneğin, sınıfların kesişimi ve birleşimi, sınıfların tümleyeni, tümel niceleyici ve tikel niceleyici gibi. Diğer yapıtaşları ikili ilişkiler için kullanılırlar. Örneğin, terslik, geçişkenlik, fonksiyonellik vb. Kullanılan bu yapıtaşları dilin anlatım gücünü belirler ve bu anlatım gücünü gösterebilmek için $\mathcal{AL}$ (Schmidt-Schaubband Smolka, 1991) dil ailesinden bir dil ile eşleştirme yapılıır. Aşağıdaki örnekte, en temel betim mantığı özelliklerini taşıyan $\mathcal{AL}$ dilinin sözdizimi sunulmaktadır:

A	atomik sınıf
$\top$	Evrensel sınıf
$\perp$	Boş sınıf
$\neg C$	C sınıfının tümleyeni
$C \sqcap D$	C ve D sınıflarının kesişimi
$\forall R.C$	C bir kavram, R bir rol olmak üzere tümel niceleyici
$\exists R.\top$	R bir rol ismi olmak üzere sınırlı tikel niceleyici

Örneğin, $\mathcal{AL}$ diline göre, $A \sqcap B$ bir kavramdır fakat $A \sqcup B$ bir kavram değildir. Benzer şekilde, $\exists S.\top$ bir kavramdır fakat $\exists S.A$ bir kavram değildir.

Betim Mantığı, sınıfları örnek kümesi, rolleri de ikili sınıf örneği çifti kümesi olarak yorumlar. Atomik olmayan sınıflar ve roller, atomik kavramlar ve roller üzerinden tanımlanırlar. Aşağıda atomik kavram ve rol yorumlamaları bulunmaktadır:

- Atomik sınıf, bir alana ait örnekler kümesidir.
- Atomik rol, sınıf örnekleri arası ikili ilişkiler kümesidir. Bu durumda, bir sınıf örneği x diğer bir sınıf örneği y ile R rolü üzerinden ilişkilendirilmişse, y örneği x 'in R -ardılı olarak adlandırılır.

Bu yorumlama, atomik olmayan sınıf ve rol yorumlamaları ile aşağıdaki şekilde genişletilir:

- Evrensel sınıf, alandaki tüm sınıf örnekleri kümesi
- Boş sınıf, hiçbir örneği olmayan boş küme
- $\neg C$, C sınıfına ait olmayan örnekler kümesi
- $C \sqcap D$, hem C hem de D sınıfına ait örnekler kümesi
- $\forall R.C$, R -ardıllarının tümü C sınıfına ait örnekler olan örnekler kümesi
- $\exists R.\top$ en az bir tane R -ardılı olan örnekler kümesi

Örnek. Eğer alana ait tüm insanların kümesi P , alana ait tüm kadınların kümesi F ise, kadın olmayan tüm insanlar kümesi aşağıdaki şekilde ifade edilir:

$$P \sqcap \neg F$$

Betim Mantığı ile ifade edilmiş bir bilgi tabanı iki kısımda incelenir: TBox (terminological box) ve ABox (assertional box). Genel olarak TBox kavram (sınıf ve rol) tanımlarını ve kavramlar arası ilişkileri içerir. ABox ise kavram örneklerine ait tanımları içerir. Örneğin,

- (1) "Her kadın bir insandır" ifadesi TBox'a aittir.
- (2) "Ayşe bir kadındır" ifadesi ABox'a aittir.

Böyle bir ayırım yapılması çıkarsama sürecinde faydalı olmaktadır. Örneğin, bir çıkarsama motoru TBox ve ABox kısımlarını ayrı ayrı işleyebilir, çünkü tanımlanmış temel çıkarsama problemlerinin her biri bu parçalardan yalnızca birini ilgilendirmektedir (örneğin, *kavram sınıflandırması* TBox ile, *örnek kontrolü* ABox ile ilgilidir). Bu ve benzeri nedenle, bu parçaların ayrı ayrı değerlendirilmesi faydalı olmaktadır. Ayrıca, böyle bir ayırım yapmak modelleme sırasında da faydalı olmaktadır. Modelleme yapan kişinin böyle bir ayırımın farkında olması, alana ait terimleri değerlendirmesine katkıda bulunmaktadır.

Betim Mantığı'nın birçok modelleme yaklaşımından farklı olan iki özelliği vardır:

- Betim Mantığı *Özgün İsim Kabullemesi* ("Unique Name Assumption"-UNA) yapmaz. Bunun anlamı, çıkarsama sonucunda aynı isme sahip olmayan iki kavramın eşit olduğu sonucuna varılabildiğidir.
- Betim Mantığı *Kapalı Dünya Kabullemesi* ("Closed World Assumption"-CWA) yapmaz, diğer bir deyişle *Açık Dünya Kabullemesi* ("Open World Assumption"-OWA) yapar. Bunun anlamı, bilgi tabanında bir olgunun olmaması, bu olgunun tersinin doğru olduğu sonucunu doğurmaz.

Farklı operatörleri destekleyen birçok Betim Mantığı çeşidi bulunmaktadır. Bu operatörler için formal olmasa da yaygın olarak kullanılan bir isimlendirme bulunmaktadır. Aşağıda bu isimlendirmeye göre, operatörleri temsil eden harfler listelenmektedir:

- $\mathcal{F}$ fonksiyonel nitelikler
- $\mathcal{E}$ tikel niceleyici (sınırlı tikel niceleyici en az bir tane *R-ardılı* olan örnekler kümesini belirtirken, tikel niceleyici *R-ardılı*nın belirli bir sınıfa ait bir örnek olmasını da gerektirir).
- $\mathcal{U}$ kavram birleşimi
- $\mathcal{C}$ atomik olmayan sınıf tümleyeni
- $\mathcal{S}$ $\mathcal{ALC}$ diline geçişken rollerin eklenmesi ile oluşan dile ait kısaltma.
- $\mathcal{H}$ rol hiyerarşisi.
- $\mathcal{R}$ karmaşık roller, dönüşlülük ve dönüşsüzlük, rol ayrıklığı.
- $\mathcal{O}$ listelenmiş sınıflar (*owl:oneOf*) ve niteliklere ait değerlerin bir sınıf örneği veya bir değer ile sınırlandırılması (*owl:hasValue*).
- $\mathcal{I}$ ters roller
- $\mathcal{N}$ nitelik sayı kısıtları (*owl:Cardinality*, *owl:MaxCardinality*)
- $\mathcal{Q}$ nitelendirilmiş nitelik sayı kısıtları (OWL 1.1 dili tarafından desteklenmektedir.)

($\mathcal{D}$) veritipleri ve bu tiplere ait değerler, datatype nitelikler

Bu düzene tam olarak uymayan betim mantıkları aşağıda listelenmektedir:

- $\mathcal{AL}$ (*Attributive language*): Taban dil olarak kabul gören bu dil atomik tümleme, kavram kesişimi, evrensel kısıtlar ve sınırlı tikel nicelemeye izin verir.
- $\mathcal{FL}_o$: $\mathcal{AL}$ dilinden atomik tümlemenin çıkarılması sonucu elde edilen altdil.

- $\mathcal{FL}^-$: $\mathcal{FL}_o$ dilinden sınırlı tikel nicelemenin çıkartılmasıyla elde edilen altdil.
- $\mathcal{EL}$: Sınıfların kesişimi ve tikel nicelemeye izin veren altdil.

Yukarıdaki betim mantıkları ve operatörler yardımıyla, daha başka betim mantıkları isimlendirilir. Örneğin, $\mathcal{ALC}$ temel olarak kabul edilen ve diğer betim mantıklarının karşılaştırıldığı bir altdildir. Adından da anlaşılacağı üzere, $\mathcal{AL}$ dilinin $\mathcal{C}$ operatörüyle genişletilmesi sonucu oluşmuştur, diğer bir deyişle $\mathcal{AL}$ dilinden farklı olarak karmaşık sınıf tümleyeni operatörünü destekler.

Bir başka örnek vermek gerekirse, $\mathcal{SHIQ}$ dili $\mathcal{ALC}$ dilinin nitelik sayı kısıtları, geçişken ve ters rollerle genişletilmesi sonucu oluşmuştur. İsimlendirme de harflerin dizilişinin bir önemi yoktur, diğer bir deyişle $\mathcal{ALCOIN}$ dili ile $\mathcal{ALCNIO}$ dili aynıdır. Bunun yanısıra anlatım güçleri eşit altdiller olabilir. Örneğin kavramların birleşimi ve tikel niceleyici kullanılarak tümleme ifade edilebileceği için, $\mathcal{ALC}$ ve $\mathcal{ALUE}$ dilleri eş kabul edilir ve birbirinin yerine kullanılabilir.

OWL 2 $\mathcal{SROIQ}^{(D)}$, OWL-DL $\mathcal{SHOIN}^{(D)}$ ve OWL-Lite $\mathcal{SHIF}^{(D)}$ dildir.

2.2 Ontoloji Dilleri

Anlamsal Web üzerinde kullanılacak ontolojiler, HTML ve XML sözdizimine dayalı oldukları için Web-tabanlı ontoloji dilleri olarak adlandırılırlar. Temel olarak, RDF veri modeli üçlülerle gösterilen ifadelere dayanmaktadır. Bir üçlü, özne, yüklem ve nesneden oluşan bir ifadedir. Her üçlü; özne ve nesnenin birer düğüm, yüklem ise bir yönlü kenar olduğu bir çizgedir. RDF basit bir veri modelidir. RDFS ve OWL bu modeli genişletecek yapıtaşları sunmaktadır. W3C, Anlamsal Web üzerinde kullanılacak standart ontoloji modelleme dilini OWL Web Ontoloji Dili olarak belirlemiştir (Smith et al., 2004).

OWL dili ile sınıf, nitelik, sınıf örneği tanımlanabilmektedir. Bunun yanısıra, OWL dili, sınıflar üzerinde tanımlı birçok operatörü desteklemektedir. Aşağıda OWL bileşenleri hakkında bilgi verilmektedir;

- **Sınıflar:** OWL dilinde tüm sınıflar *owl:Thing* sınıfının alt sınıfıdır. Sınıfların kendilerine ait veya diğer alt sınıflarına ait sınıf örnekleri olabilir.

- **Nitelikler:** Sınıf karakteristiklerini tanımlayan ikili ilişkilerdir. OWL dilinde iki tip nitelik tanımlanmaktadır: *datatype (veritipi)* ve *object (nesne tipi)* nitelikler. Veritipi nitelikler, sınıf örnekleri ile RDF sabitleri veya XML şema veritipleri arasındaki ilişkilerdir. Nesne tipi nitelikler ise, iki sınıfa ait örnekler arası ilişkilerdir. OWL dilinde, geçişkenlik, simetriklik, terslik ve fonksiyonellik gibi nitelik karakteristikleri tanımlanabilir. Niteliklere ait önalan ve artalan tanımlamaları yapılabilir.
- **Sınıf örnekleri:** Bir sınıfın sınırsız sayıda örneği tanımlanabilir.
- **Operasyonlar:** OWL dili sınıflar üzerinde birleşim, kesişim ve tümleme operasyonları tanımlamaktadır. Bunun yanı sıra listelenmiş sınıf, sınıfa ait nitelik üzerinde sayı kısıtı ve ayrık sınıf tanımlamalarını desteklemektedir.

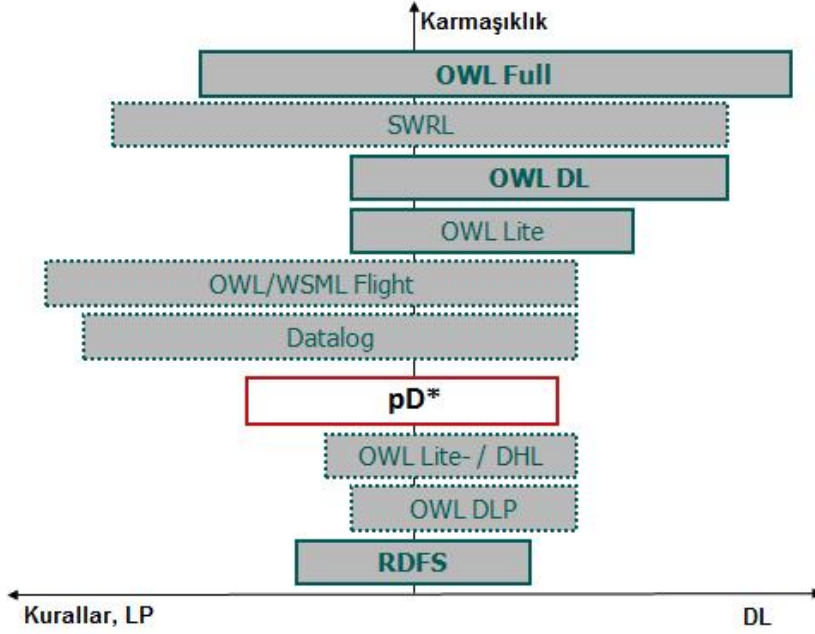
Anlamsal Web, ölçeklenebilir ve yüksek performanslı ontoloji depolarına ve çıkarsama motorlarına gereksinim duymaktadır. Bu anlamda temel problem, ontoloji dilinin anlatım gücünün yükselmesiyle çıkarsama karmaşıklığının da artış göstermesidir. Dil yığıtında yukarı katmanlara doğru çıkıldıkça, çıkarsama maliyeti yükselmektedir. Tez yazıldığı sırada, OWL dilinin en basit altdili olan OWL Lite için bile, milyonlarca üçlünden oluşan bir bilgi tabanı üzerinde, verimli çıkarsama ve sorgulama gerçekleştirebilecek algoritmalar üzerinde yapılan çalışmalar devam etmektedir. Aşağıda, OWL dilinin her biri değişik anlatımlılığa sahip üç altdili sunulmaktadır;

- **OWL Lite:** bir sınıflandırma hiyerarşisine ve basit kısıtlara ihtiyaç duyan kullanıcılar için tasarlanmıştır.
- **OWL DL:** eksiksizlikten (yapılabilecek tüm çıkarsamaların hesaplanacağı garantisi) ve sonlandırılabilirlikten (tüm çıkarsamaların sonlu zamanda biteceği garantisi) ödün vermeden, anlatımlılığı maksimum seviyeye çıkartan altdildir.
- **OWL Full:** eksiksizlik ve sonlandırılabilirlik garantisi olmadan, maksimum anlatımlılık gereksinimi olan kullanıcılar için tasarlanmıştır. Örneğin, OWL DL'den farklı olarak, OWL Full ile tasarlanmış bir ontolojide, bir kaynak,

hem bir sınıf örneği hem de başka örneklerle sahip bir sınıf olarak modellenilebilir. Bu gibi durumlardan dolayı, OWL Full ile eksiksiz çıkarsama yapabilecek bir çıkarsama aracının gerçekleştirimi imkansız gözükmektedir.

OWL Full, RDF(S) ile uyumludur, diğer bir deyişle her RDF dökümanı bir OWL Full dökümanıdır. Fakat her RDF dökümanı bir OWL DL veya OWL Lite dökümanı değildir (Grosz et al., 2003). Çünkü bu iki altdil, RDF dili üzerinde birtakım kısıtlar tanımlamışlardır, bu nedenle RDF dilinin uzantıları değildirler. Aşağıda bu kısıtlar listelenmektedir:

- *kaynakların bölünmesi*: Her bir kaynak bir sınıf, veritipi, datatype nitelik, object nitelik, sınıf örneği veya dilin tanımından gelen yapıtaşlarından biri olabilir. Kaynaklar sayılan gruplardan yalnızca birine ait olabilirler.
- *tiplerin belirlenmesi*: Kaynakların tipleri açıkça belirtilmelidir (örneğin *rdfs:subClassOf* niteliği ile ilişkilendirilecek bir kaynağın sınıf olduğu önceden tanımlanmalıdır).
- *niteliklerin ayrılması*: Object nitelikler ve datatype nitelikler kümesi ayrıktır. Datatype nitelikler aşağıdaki karakteristiklerle kullanılamaz:
 - *owl:inverseOf*
 - *owl:TransitiveProperty*
 - *owl:SymmetricProperty*
- *geçişken sayı kısıtlarının engellenmesi*: Geçişken nitelikler ve onların altnitelikleri üzerinde sayı kısıtı tanımlanamaz.
- *anonim sınıfların kısıtlanması*: Anonim sınıflar yalnızca aşağıdaki şekillerde kullanılabilirler:
 - *owl:equivalentClass* veya *owl:disjointWith* niteliklerinin önalın ve artalanlarında
 - *rdfs:subClassOf* niteliğinin artalanında



Şekil 2.1. OWL altdilleri ve bunlarla ilişkili diller.

Bu kısıtlamaların sonucunda, RDFS belgelerinin OWL belgelerine dönüştürülmesi zorlaşmaktadır. Diğer bir deyişle varolan birçok RDFS dökümanı doğrudan OWL Lite/DL ontolojilerine çevrilemez. RDFS dökümanlarının karşılığı OWL Full dökümanlarıdır. Fakat, OWL Full ile eksiksiz ve doğru çıkarsama yapmak mümkün değildir.

Şekil 2.1’de (Ontotext, 2009), OWL altdilleri ve bu dillerle ilişkili diğer diller görülmektedir. Bu dillerin Betim Mantığı (DL), mantık programları (LP) ve kurallar arasındaki yeri de şekilde görülmektedir. Takip eden iki alt başlık, şekilde OWL altdilleri ile ilişkilendirilmiş OWL DLP ve pD^* dillerini anlatmaktadır.

2.2.1 OWL DLP

Mantık programlama (LP), kural tabanlı diller ve sistemler için ortak olarak kullanılan bir terimdir. Datalog ve F-Logic en bilinen LP dillerine örnek gösterilebilir. LP tabanlı sistemler, DL tabanlı sistemlere göre daha verimli ve ölçeklenebilir. Fakat tam olarak bir karşılaştırma yapılamamaktadır. Bunun nedeni iki paradigmanın farklı mantıkları olmasıdır, diğer bir deyişle F-Logic ile ifade edilip OWL-DL ile ifade edilemeyen veya OWL-DL ile ifade edilip F-Logic ile ifade edilemeyen tanımlamalar bulunmaktadır.

OWL DLP (Grosz et al., 2003), anlatım gücü, verimli çıkarsama ve uyumluluk sağlayan bir altdil olarak önerilmiştir. OWL DLP'nin anlatım gücü, OWL DL ve LP dillerinin anlatım gücünün kesişimidir. Diğer bir deyişle, OWL DLP, Datalog dili ile eşleştirilebilen ve anlatım gücü en yüksek olan OWL DL altdilidir.

2.2.2 pD^*

RDF ve RDFS gerektirmelerinin tam bir listesi (Hayes, 2004) olmasına rağmen, OWL dilinin anlatım gücünün çok yüksek olmasından dolayı OWL diline ait tam bir gerektirme listesi tanımlanamamaktadır (Meditos and Bassiliades, 2009). Bu soruna bir çözüm getirmek için, (ter Horst, 2004) OWL Full dilinin zayıflatılmış bir versiyonunu tanımladı ve daha sonraki çalışmasında (ter Horst, 2005) pD^* dilini daha fazla OWL yapıtaşını destekleyecek şekilde genişletti. Horst'un bu çalışmaları RDF gerektirmelerini iki şekilde genişletmektedir:

- D^* , tiplendirilmiş sabitlerle çıkarsama desteği ekler.
- pD^* , OWL yapıtaşlarını kısmi olarak destekleyecek kurallar ekler. Desteklenen yapıtaşları, *FunctionalProperty*, *InverseFunctionalProperty*, *sameAs*, *SymmetricProperty*, *TransitiveProperty*, *inverseOf*, *equivalentClass*, *equivalentProperty*, *hasValue*, *someValuesFrom*, *allValuesFrom*, *differentFrom* ve *disjointWith* olarak listelenebilir. Son iki yapıtaşı tutarsızlık türeten kurallar yardımıyla desteklenmektedir.

Bu aşamada belirtilmesi gereken bir nokta, bazı yapıtaşlarının kısmi olarak desteklenmesidir. En belirgin örneklerden biri, *someValuesFrom* ve *allValuesFrom* yapıtaşları ile ilgili gerektirmelerin yalnızca tek yönlü olarak desteklenebilmesidir. RDFS diline ait tek yönlü gerektirmeler, RDFS çıkarsama motorlarında kullanılan basit kurallardır. Fakat OWL dilinin anlatım gücü, çift yönlü gerektirmelerin kullanılmasını gerektirir. Örneğin, tek yönlü bir gerektirme, c sınıfı d sınıfının alt sınıfı ise, c sınıfına ait her bir örneğin aynı zamanda d sınıfının bir örneği olmasını gerektirir. OWL dilinde bu gerektirme çift yönlü olarak tanımlanmalıdır, diğer bir deyişle, bu gerektirmenin tersi yönünde de bir çıkarsama yapılmalıdır. pD^* dili, OWL dilinin gerektirdiği çift yönlü gerektirmeleri gözönünde bulundurmaz, diğer

bir deyişle OWL dilinin zayıflatılmış bir altkumesidir ve OWL dili ile yapılacak tüm çıkarsamalar pD^* kuralları ile türetilemez.

pD^* dili için 23 gerektirme kuralı ve 2 tutarsızlık kuralı tanımlanmıştır (bkz. Bölüm 4.1.1). Örneğin, aşağıda *owl:TransitiveProperty* yapıtaşı ile ilgili bir kural gösterilmektedir (Antoniou and van Harmelen, 2004):

$$p\ rdfs : type\ owl : TransitiveProperty\ s\ p\ x\ x\ p\ z \Rightarrow s\ p\ z$$

pD^* dili, örneklerle ilgili çıkarsamaların ön planda olduğu birçok uygulama için yeterli olmaktadır (ter Horst, 2006). Ontoloji örnekleri ile ilgili birçok çıkarsamayı destekler ve çıkarsama karmaşıklığı OWL diline göre düşüktür. OWL DLP dilinden farklı olarak OWL DL dilinin bir altdili olmadığı için, RDF üzerinde bir kısıt tanımlamaz. pD^* dili, OWL 2 dilinin OWL RL altdili olarak kabul görmüştür (Motik et al., 2008).

2.3 Çıkarsama Yaklaşımları

Ontolojilerde, açıkça belirtilmiş bilgiden yola çıkarak, yeni bilgilere ulaşma işlemine *çıkarsama* denir. Bu yeni bilgiler çıkarsanmış veya dolaylı bilgi olarak, çıkarsama işlemi gerçekleştirilen uygulamalar da “*çıkarsama motoru*” olarak adlandırılır.

Ontolojiler üzerinde çıkarsama yapmak için önerilmiş iki paradigma bulunmaktadır; *klasik çıkarsama paradigması* ve *datalog paradigması* (Patel-Schneider and Horrocks, 2007). Klasik paradigma, Betim Mantığı temellidir (Baader et al., 2003). RacerPro , Pellet ve Fact++ gibi Betim Mantığı temelli çıkarsama motorları tablo algoritmalarını (Baader and Sattler, 2000) kullanırlar ve klasik paradigmayı benimserler. Geçmiş yıllarda Betim Mantığı için geçerli ve eksiksiz çıkarsama algoritmaları ve bunları kullanan sistemler geliştirilmiştir (Horrocks, 1998; Haarslev and Möller, 2001). Fakat bu çalışmalar genellikle TBox çıkarsamaya odaklanmışlardır. ABox çıkarsamayı gerçekleştirebilenler ise bunu ölçeklenebilir bir şekilde yapmayı başaramamışlardır. Ayrıca, bu sistemlerin desteklediği sorgulama dilleri yeterli anlatımlılığa sahip değildirler. Daha anlatımlı sorgulama dilleri getiren RACER gibi sistemler ise ABox’ın mutlaka ana bellekte olmasını gerektirmektedir ve ABox’ın disk üzerinde sorgulanabilmesine izin vermemektedir (Haarslev and Möller, 2001).

Datalog paradigması, kural tabanlı çıkarsamaya dayanır. Ontoloji gösterim

dilinin mantığı, kurallara dönüştürülür ve çıkarsama motoru bu kuralları kullanarak ontolojik veri üzerinde çıkarsama yapar. Jess, OWLIM, Jena ve O-DEVICE gibi çıkarsama motorları bu paradigmayı benimserler.

Ölçeklenebilirlik açısından, bu iki paradigma arasındaki fark ABox çıkarsama sürecinde ve sorgulama sürecinde ortaya çıkmaktadır. Klasik paradigma kapsamında incelenen çıkarsama motorları, gerçek uygulamalarda yer alan örnek veri boyutlarıyla ölçeklenebilir çıkarsamayı destekleyememektedir (Horrocks et al., 2004). Klasik yaklaşımda ölçeklenebilir ABox çıkarsama yapabilmek için çalışmalar devam etmektedir (Fokoue et al., 2006), (Dolby et al., 2007). Diğer taraftan, kural tabanlı yaklaşımın TBox çıkarsama ile ilgili bazı problemleri vardır. TBox çıkarsama, klasik yaklaşımdaki gibi eksiksiz değildir ve klasik yaklaşıma göre daha yavaştır. Fakat bu yaklaşımda, ABox çıkarsama ve sorgulama daha ölçeklenebilir yapılmaktadır. Ölçeklenebilirlik farkı, örnek verinin boyutu arttıkça daha da farkedilebilir olmaktadır.

Anlamsal Web uygulamaları için geliştirilmiş çıkarsama mekanizmalarının verimli ve ölçeklenebilir olabilmesi için eniyilenmiş depolama modellerine ihtiyaç duyulmaktadır. Depolama modellerinin ölçeklenebilir olması için, bellek tabanlı olarak değil, disk tabanlı olarak tasarlanmaları ve özelleşmiş şemalara sahip olmaları gerekmektedir. Bir başka deyişle, hem özelleşmiş veritabanı şemalarına hem de bu şemalar üzerinde bilgiye daha verimli erişebilmek için indeks tanımlamalarına ihtiyaç duyulacaktır.

Veritabanında saklı ontoloji örnekleri üzerinde çıkarsama yapmanın bazı zorlukları vardır. Veritabanları ontolojilerin aksine kapalı dünya varsayımına dayanmaktadır. Diğer bir deyişle, veritabanında olmayan bir bilginin yanlış olduğu kabullenmesi yapılır. Ontolojiye dayalı sistemlerde ise ontolojide varolan bilgiyle çelişmediği sürece, bir bilginin yanlışlığı ispatlanamaz. Bu durum, veritabanlarına yönelik çıkarsama mekanizmalarında mutlaka gözönünde bulundurulmalıdır. (Mei et al., 2006), veritabanları ve ontolojiler arasındaki diğer farkları da listelemektedir.

Tümdengelimli veritabanı sistemleri, sorgu dilleri ve modelleme yapıları, mantıksal bir modele dayanan veritabanı sistemleridir. SQL gibi ilişkisel dillerin kısıtlanmış bir mantıksal dile dayandıkları ve ilişkilerin de aslında mantıktaki yüklemeler gibi görülebileceği düşünülürse, tümdengelimli veritabanlarının ilişkisel ve-

ritabanlarının daha ileri bir formu olduğu söylenebilir. Tümdengelimli veritabanları, uzman sistemlere benzerler. Aralarındaki fark geleneksel uzman sistemlerin tüm bilgi tabanının ana belleğe yüklenmesini gerektirmesidir. Tümdengelimli veritabanları ise bilgi tabanlarını, veritabanında saklarlar ve bellek ihtiyaçları yoktur. Tümdengelimli veritabanlarının sorgulama ve kural tanımlama dili olan Datalog, sözdizim olarak Prolog'un altkümesidir. Bunun dışında, ontolojilerin veritabanlarında saklanması da sorgulama mekanizmasını etkilemektedir.

Çıkarsama ve sorgulama birbiriyle ilişkili süreçler olduklarından, çıkarsama algoritmalarının farklılıklarına bağlı olarak sorgulama yaklaşımları da değişmektedir. En büyük farklılık, bilgi tabanı kapanışını çevrimiçi hesaplayan algoritmalarda ortaya çıkmaktadır. Bu algoritmalar, gönderilen sorguların yeniden yazılmasını ve genişletilmesini gerektirmektedir.

Bu tez kapsamında, kural tabanlı bir çıkarsama yaklaşımı sunulmaktadır. Bu yaklaşım, hem çevrimiçi hem çevrimdışı çıkarsama algoritmalarını birlikte kullanmakta ve veritabanında saklanmış örnek verileri üzerinde çıkarsama yapmaktadır.

Kural tabanlı çıkarsama algoritmaları, ileriye zincirleme (çevrimdışı çıkarsama) ve geriye zincirleme (çevrimiçi çıkarsama) olmak üzere ikiye ayrılır. İki algoritma da bilgi tabanında olmayan yeni gerçekler türetmek için varolan gerçekler üzerinde kuralları çalıştırırlar. İkisi arasındaki fark, çıkarsama sürecinin uygulanma zamanıdır. İleriye zincirleme, ontoloji kapanışını tümüyle hesaplar ve daha sonra sorguları yanıtlar. Geriye zincirleme ise sorgu yanıtlanmadan hemen önce gerekli çıkarsamaları yapar. İki yaklaşımın da avantajları ve dezavantajları vardır. Geriye zincirleme algoritmasının bellek gereksinimi daha azdır fakat sorgu yanıtlama süresi daha uzundur. İleriye zincirleme algoritmasının ise sorgu yanıtlama süresi daha kısadır fakat bellek ihtiyacı daha fazladır.

İleriye zincirleme algoritması, çıkarsama kuralları içinde önbişeni doğru olan bir kural buluncaya kadar arama yapar. Böyle bir kural bulunduğunda, bu kuralın artbişenin doğru olduğu sonucuna varır ve yeni gerçeği bilgi tabanına ekler. Yeni ontoloji üçlöleri bilgi tabanına eklenir eklenmez, ileriye zincirleme algoritması tekrar tetiklenir ve yeni gerçekler kümesi üzerine kuralları baştan uygular. Bilgi tabanı üzerinde kuralların uygulanmasıyla ulaşılabilcek tüm dolaylı bilgi çıkarsanır. Sorgu zamanında hiçbir yeni çıkarsama yapılmasına gerek kalmadığı

için sorgu yanıtlama süresi hızlıdır, fakat ontoloji yükleme zamanı ve bellek ihtiyacı yüksektir. Bu yaklaşımın uygulanması pratik değildir, bunun nedeni (i) çıkarsanan her üçlünün, daha sonra sorgulanacağı kesin değildir ve (ii) çıkarsanan üçlülere saklamak için gerekli bellek gereksinimi çok yüksek olabilir ve bu üçlülere erişim maliyeti de buna bağlı olarak yüksek olabilir.

Diğer taraftan, geriye zincirleme algoritması, bir sorgu gönderildiğinde bu sorgu koşullarını hedef listesine ekleyerek, koşulların doğruluğunu ispatlamaya çalışır. Algoritma, artbileşeni hedefle eşleşen kuralları bulur. Eşleşen bir kural bulunduğunda, kuralın önbileşeni bilgi tabanında bulunamazsa, önbileşenler hedef listesine eklenir, çünkü hedefin doğrulanması için hedefle eşleşen kuralı doğrulayacak veriye gereksinim duyulmaktadır. İlgili çıkarsama kuralları sorgu zamanında tetiklendiği için, ileriye zincirleme algoritmasından farklı olarak, çıkarsanan üçlüler için bellek gereksinimine ihtiyaç yoktur. Fakat sorunun yanıtlanması daha uzun sürmektedir. Genel bir değerlendirme yapmak gerekirse, geriye zincirleme algoritmasının kullanılmasının avantajlı olacağı belirli durumlar vardır. Eğer az sayıda kural, fakat fazla miktarda veri varsa, sorgu zamanında az sayıda kuralı çalıştırmak diğer bir deyişle geriye zincirleme algoritmasını kullanmak avantajlı olacaktır.

Yukarıda anlatılanlara ek olarak, bazı yaklaşımlar ileriye ve geriye zincirleme algoritmalarını birleştirerek iki yaklaşımın da dezavantajlarından kurtulurlar (Ding et al., 2005). Bu alanda çalışmalar devam etmektedir. Araştırmacılar, tüm arama uzayının kapanışını hesaplamayı engelleyecek, aynı zamanda sorgu yanıtlama süresini de makul sınırlarda tutacak melez teknikler üzerinde çalışmaktadır.

Kural tabanlı çıkarsamanın eksiksizliğini ve sonlandırılabilirliğini garantilemek için bazı yaklaşımlar sunulmuştur. Bu yaklaşımlar OWL ve RDF dillerinin bir altkümesini tanımlayarak, çıkarsamanın karmaşıklığını ve süresini azaltmayı hedeflemektedir. RDF MT ve RDFS(FA) (Pan and Heflin, 2003) bu yaklaşımlara örnek gösterilebilir. Bu çalışmaların içinde en bilineni, pD^* 'dir (Hayes, 2004) (bkz. Bölüm 2.2.2). pD^* dili için özelleşmiş algoritmalar bulunmaktadır (Li et al., 2006). Bu tez kapsamında sunulan çıkarsama yaklaşımı pD^* dilini desteklemektedir.

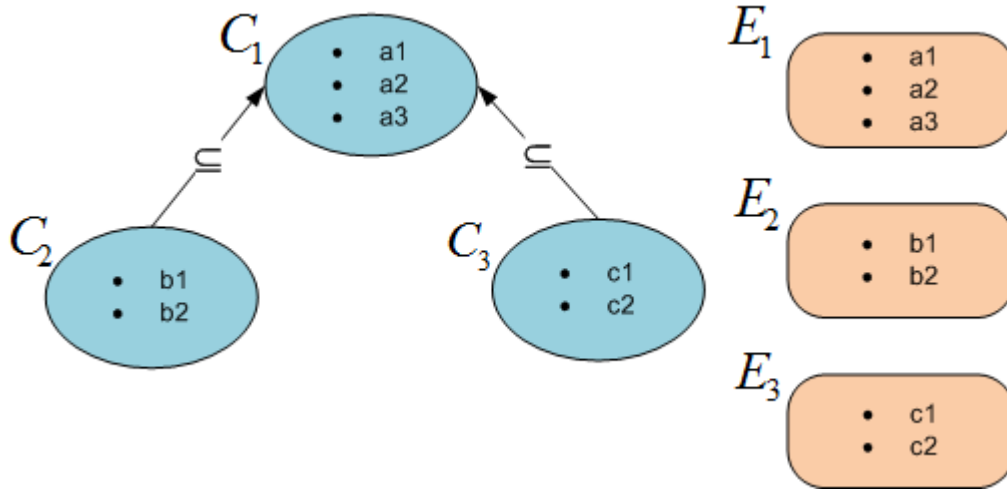
3 Uzantı Tabanlı Bilgi Modeli

Bilgi modelleri, bilgiye ait parçaları ve bu parçalar arasındaki ilişkileri göstermek için belirli semboller kullanan yapısal gösterimlerdir. Bu tanıma göre OWL dili ile tanımlanmış herhangi bir ontoloji bir bilgi modelidir. Bu tez kapsamında, OWL dili ile tanımlanmış bir ontoloji, uzantı tabanlı bilgi modeli olarak adlandırılan başka bir modele dönüştürülmektedir. Bu bölüm, uzantı tabanlı bilgi modeli bileşenlerini ve bir OWL ontolojisinin bu modele nasıl dönüştürüleceğini anlatmaktadır.

Bir kavrama ait örnekler kümesi bu kavramın uzantısı olarak adlandırılmaktadır (Dean et al., 2003). Uzantı tabanlı model, temel bir prensibe dayanır. OWL ontolojilerinde, her kavrama ait uzantılar tanımlayarak, kavramı bu uzantılarla ilişkilendirir. OWL ontolojisinin, uzantı tabanlı modele dönüştürülmesi ontolojide kavramlara ait uzantıların tanımlanması ve kavrama ait örneklerin bu uzantılarla ilişkilendirilmesinden ibarettir. Bu çalışmada, kavramları ve kavramlara ait uzantıları ilişkilendirmek için kullanılan yapıtaşları *gruplama yapıtaşları* olarak adlandırılacaktır.

Uzantı tabanlı bilgi modelinin amacı, ontoloji çıkarsama ve sorgulama süreçlerinin eniyilenmesidir. Bu eniyilemenin yapılabilmesi için uzantı tabanlı bilgi modeli, veritabanı teknolojisinden de faydalanır. Uzantı tabanlı bilgi modeline göre ontoloji iki parçaya ayrılır. İlk parça kavram tanımları, kavramlara ait uzantı tanımları ve kavramlar arası ilişkileri içeren şema kısmıdır (TBox). Bu kısım üzerinde çıkarsama yapılır. Çıkarsamaya katılmayan, ikinci parça genellikle birincisine oranla çok daha büyüktür. Bu kısım kavram örnekleri ve bu örnekler arası ilişkilerden oluşur (ABox). Bu kısım çıkarsamaya katılmaz ve sorgu yanıtlama aşamasında kullanılmak üzere veritabanında saklanır.

Takip eden alt başlıklarda, öncelikle uzantı tabanlı bilgi modeli yapıtaşları anlatılacaktır. Daha sonra modelin kullandığı veritabanı şeması gösterilecek ve son olarak bir OWL ontolojisinin, uzantı tabanlı bilgi modeline dönüştürülmesi anlatılacaktır.



Şekil 3.1. Sınıf örneği gruplama yapıtaşları.

3.1 Uzantı Tabanlı Bilgi Modeli Yapıtaşları

Uzantı tabanlı bilgi modeli yapıtaşları dört grupta incelenir;

Sınıf örneği gruplama yapıtaşları: *hasClassExtension*, bir sınıfı ve bu sınıfa ait uzantının bir altkümesini ilişkilendiren bir niteliktir. İki tane altniteliği bulunmaktadır; *hasExplicitClassExtension* ve *hasInferredClassExtension*. *hasExplicitClassExtension*, sınıfı ve sınıfla açıkça ilişkilendirilmiş tüm sınıf örneklerini içeren sınıfın açık uzantısını ilişkilendirir. *hasInferredClassExtension*, sınıfı ve sınıfa ait dolaylı (çıkarsanmış) örneklerin bir kısmını içeren dolaylı uzantılarından birisini ilişkilendirir. Uzantı ile her bir uzantı elemanı *contains* ilişkisi ile bağlanır.

Şekil 3.1, üç tane sınıf ve bu sınıflarla ilgili uzantı altkümelerini göstermektedir. C_1 sınıfının, C_2 ve C_3 adında iki alt sınıfı bulunmaktadır. E_1 , C_1 ile açıkça ilişkilendirilmiş örnekleri, E_2 , C_2 ile açıkça ilişkilendirilmiş örnekleri, E_3 ise C_3 ile açıkça ilişkilendirilmiş örnekleri göstermektedir. Bu durumda, C_1 sınıfı E_1 uzantısı ile, C_2 sınıfı E_2 uzantısı ile, C_3 sınıfı ise E_3 uzantısı ile, *hasExplicitClassExtension* ilişkisi ile bağlanır. C_2 ve C_3 sınıfları C_1 'in alt sınıfları olduğu için, çıkarsama sonucu E_2 ve E_3 uzantıları C_1 sınıfı ile *hasInferredClassExtension* ilişkisi ile bağlanır. Şekilde gösterilen tüm uzantılar ile uzantıların her bir elemanı *contains* ilişkisi ile bağlanmıştır (örn, " E_1 contains a_1 " şeklinde).

Nitelik öznesi gruplama yapıtaşları: *hasSubjectExtension*, bir niteliği (type niteliği hariç) ve bu niteliğin örneklerine ait özneler kümesinin bir altkümesini ilişkilendiren bir niteliktir. İki tane altniteliği bulunmaktadır; *hasExplicitSubjectExtension*

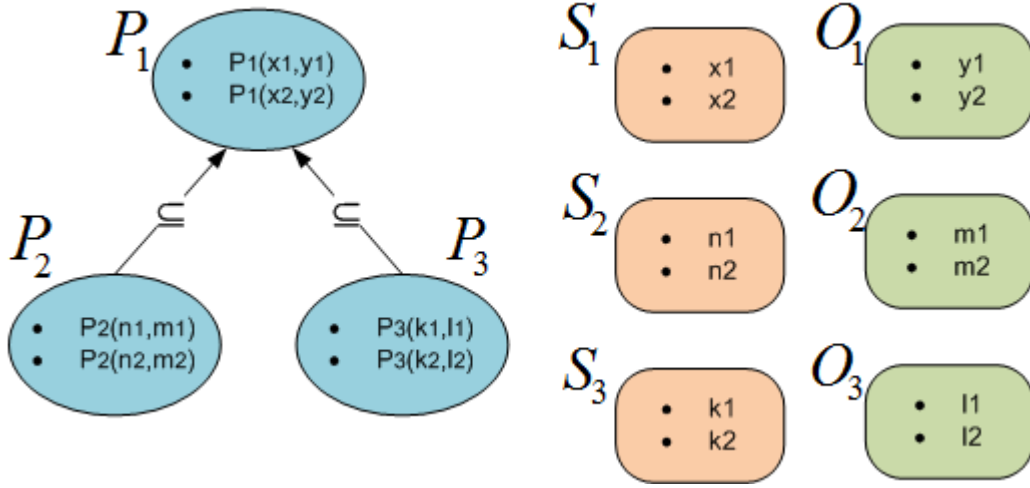
sion ve *hasInferredSubjectExtension*. *hasExplicitSubjectExtension*, niteliği ve nitelik ile açıkça ilişkilendirilmiş tüm nitelik örneklerine ait özneler kümesini ilişkilendirir. *hasInferredSubjectExtension*, niteliği ve niteliğin dolaylı (çıkarılmış) örneklerine ait özneler kümesinin bir altkümelerini ilişkilendirir.

Şekil 3.2, üç tane nitelik ve bu niteliklerle ilgili uzantı altkümelerini göstermektedir. P_1 niteliğinin, P_2 ve P_3 adında iki altniteliği bulunmaktadır. S_1 , P_1 ile açıkça ilişkilendirilmiş örneklere ($P_1(x_1, y_1)$ ve $P_1(x_2, y_2)$) ait özneleri, S_2 , P_2 ile açıkça ilişkilendirilmiş örneklere ($P_2(n_1, m_1)$ ve $P_2(n_2, m_2)$) ait özneleri, S_3 ise P_3 ile açıkça ilişkilendirilmiş örneklere ($P_3(k_1, l_1)$ ve $P_3(k_2, l_2)$) ait özneleri göstermektedir. Bu durumda, P_1 niteliği S_1 uzantısı ile, P_2 niteliği S_2 uzantısı ile, P_3 niteliği ise S_3 uzantısı ile *hasExplicitSubjectExtension* ilişkisi ile bağlanır. P_2 ve P_3 nitelikleri P_1 'in altnitilikleri olduğu için, çıkarsama sonucu S_2 ve S_3 uzantıları P_1 sınıfı ile *hasInferredSubjectExtension* ilişkisi ile bağlanır. Şekilde gösterilen tüm uzantılar ile uzantıların her bir elemanı *contains* ilişkisi ile bağlanmıştır (örn, " S_1 contains x_1 " şeklinde).

Nitelik nesnesi gruplama yapıtaşları: *hasObjectExtension*, bir niteliği (type niteliği hariç) ve bu niteliğin örneklerine ait nesneler kümesinin bir altkümelerini ilişkilendiren bir niteliktir. İki tane altniteliği bulunmaktadır; *hasExplicitObjectExtension* ve *hasInferredObjectExtension*. *hasExplicitObjectExtension*, niteliği ve nitelik ile açıkça ilişkilendirilmiş tüm nitelik örneklerine ait nesneler kümesini ilişkilendirir. *hasInferredObjectExtension*, niteliği ve niteliğin dolaylı (çıkarılmış) örneklerine ait nesneler kümesinin bir altkümelerini ilişkilendirir.

Şekil 3.2'de, O_1 , P_1 ile açıkça ilişkilendirilmiş örneklere ($P_1(x_1, y_1)$ ve $P_1(x_2, y_2)$) ait nesneleri, O_2 , P_2 ile açıkça ilişkilendirilmiş örneklere ($P_2(n_1, m_1)$ ve $P_2(n_2, m_2)$) ait nesneleri, O_3 ise P_3 ile açıkça ilişkilendirilmiş örneklere ($P_3(k_1, l_1)$ ve $P_3(k_2, l_2)$) ait nesneleri göstermektedir. Bu durumda, P_1 niteliği O_1 uzantısı ile, P_2 niteliği O_2 uzantısı ile, P_3 niteliği ise O_3 uzantısı ile *hasExplicitObjectExtension* ilişkisi ile bağlanır. P_2 ve P_3 nitelikleri P_1 'in altnitilikleri olduğu için, çıkarsama sonucu O_2 ve O_3 uzantıları P_1 sınıfı ile *hasInferredObjectExtension* ilişkisi ile bağlanır. Şekilde gösterilen tüm uzantılar ile uzantıların her bir elemanı *contains* ilişkisi ile bağlanmıştır (örn, " O_1 contains y_1 " şeklinde).

Nitelik örneği gruplama yapıtaşları: *hasPropertyExtension*, bir niteliği ve



Şekil 3.2. Nitelik gruplama yapıtaşları.

bu niteliğe ait uzantının bir altkümesini ilişkilendiren bir niteliktir. İki tane alt niteliği bulunmaktadır; *hasExplicitPropertyExtension* ve *hasInferredPropertyExtension*. *hasExplicitPropertyExtension*, niteliği ve nitelikle açıkça ilişkilendirilmiş tüm nitelik örneklerini temsil eden niteliğin açık uzantısını ilişkilendirir. *hasInferredPropertyExtension*, niteliği ve niteliğe ait dolaylı (çıkarsanmış) örneklerin bir kısmını temsil eden dolaylı uzantılarından birisini ilişkilendirir.

Nitelik örneği gruplama yapıtaşlarının diğer gruplama yapıtaşlarından en büyük farkı, uzantılar ve elemanları arasında bulunan *contains* ilişkisinin belirtilmemesidir. Bu nedenle *hasPropertyExtension* niteliği temsili bir yapıtaşdır. Bu şekilde tasarlanmasının nedeni, uzantı tabanlı modele dönüşümden sonra bilgi tabanındaki üçlü sayısının çok fazla artış göstermesini önlemektir.

3.2 Uzantı Tabanlı Bilgi Modeli Veritabanı Şeması

Ontolojilerin veritabanında saklanması, çıkarsamanın ölçeklenebilirliği açısından gereklidir. Çıkarsama işlemi gerçekleştirilirken gereken bellek miktarı ontolojinin büyüklüğüyle doğru orantılı olarak üstel bir şekilde artmaktadır. Büyük ontolojiler üzerinde çıkarsama yapılması için günümüzdeki bilgisayarların bellek kapasiteleri yetersiz kalmaktadır. Bu nedenle üzerinde çıkarsama gerçekleştirilen verinin veritabanında saklanması daha büyük ontolojilerin kullanılabilmesi için zorunludur.

Tablo Adı	Sütun Adı
contains	extension_name, resource
statements	subject, predicate, object

Çizelge 3.1. *contains* ve *statements* tablolarına ait sahalar.

Ontolojilerin veritabanında saklanması için üç farklı yöntem bulunmaktadır (Heymans et al., 2008).

1. Genel Üçlü Depoları : Ontoloji üçlülere özne, yüklem ve nesne sütunlarından oluşan tek bir tabloda tutulurlar. Jena (Wilkinson et al., 2003b) ve Oracle (Das et al., 2008) genel RDF depolarıdır.
2. İkili tablo tabanlı Üçlü Depoları : Bu tip depolar her kavram (sınıf ve nitelik) için bir tablo yaratırlar. DLDB (Pan and Heflin, 2003) buna örnek gösterilebilir.
3. Gelişmiş Üçlü Depoları : Bu tip depolar ise saklanan bilginin ontolojik niteliklerine göre daha ayırık olmasını sağlar. Örneğin, sınıf, nitelik bilgileri ve tip üçlülere diğer üçlülerden yalıtılmıştır. Genel RDF depolarının gelişmiş bir hali olarak düşünülebilirler. Minerva (Zhou et al., 2006) bu tip sistemlere örnek olarak gösterilebilir.

Tez kapsamında kullanılan veritabanı şeması genel üçlü depoları sınıfında yer alır. Veritabanında uzantılar ve uzantı elemanlarını birebir eşleşmiş biçimde tutan *contains* tablosu ve ontoloji üçlülerini tutan *statements* tablosu yer alır (Çizelge 3.1). Veritabanında gruplama yapıtaşlarının desteklenmesi için yardımcı tablolar da yer almaktadır (bkz. 4.2.3).

3.3 Uzantı Tabanlı Bilgi Modeli Dönüşüm Algoritması

OWL ile tanımlanmış bir ontolojinin, uzantı tabanlı bilgi modeline dönüştürülmesi, ontoloji üçlülere üzerinde belirli bir algoritmaya uygun olarak sözdizimsel dönüşümlerin yapılması ile ilgilidir. Bu dönüşüm sırasında, ilk adım her bir sınıfı, bu sınıfa açıkça bağlanmış sınıf örneklerini içeren *sınıf örneği uzantısı* ile; her bir niteliği ise bu nitelikle açıkça ilişkilendirilmiş nitelik örneklerini temsil

Algorithm 3.1 Uzantı tabanlı modele dönüşüm algoritması.

```

for each triple in ontology
{
    if predicate is rdf:type
    {
        if (extension_object is not defined)
            add triple object has extension_object
        add triple extension_object contains subject
    }
    else
    {
        if (subjectExtension_predicate is not defined)
            add triple predicate has subjectExtension_predicate
        add triple subjectExtension_predicate contains subject
        if (objectExtension_predicate is not defined)
            add triple predicate has objectExtension_predicate
        add triple objectExtension_predicate contains object
    }
}

```

eden bir *nitelik örneği uzantısı*, bu nitelik örneklerinin öznelerini içeren bir *nitelik öznesi uzantısı* ve yine bu nitelik örneklerinin nesnelerini içeren bir *nitelik nesnesi uzantısı* ile ilişkilendirmektir. İkinci adımda, her bir uzantı (*nitelik örneği uzantıları* hariç) elemanları ile *contains* ilişkisi ile bağlanır.

Yukarıda anlatılan dönüşümler Algoritma 3.1 uygulanması sonucu gerçekleştirilir. Bu dönüşüm, modelin anlamını değiştirmeyen, sözdizimsel seviyede bir dönüşümdür. Diğer bir deyişle dönüşüm sonunda modelde bir anlam kaybı veya artışı olmaz.

4 Uzantı Tabanlı Çıkarsama ve Sorgulama

Tez kapsamında önerilen uzantı tabanlı çıkarsama algoritması, kural tabanlı sistemler için tasarlanmıştır. Kural tabanlı sistemler, gerektirme kurallarını bilgi tabanı üzerinde uygulayarak bilgi tabanında açık olarak belirtilmeyen yeni olguları üretirler. Kural tabanlı çıkarsama algoritmaları, ileriye ve geriye zincirleme algoritmaları adlı iki ana grup altında incelenebilirler. Her iki ana grubun da birbirlerine üstün veya zayıf oldukları yönler bulunmaktadır. Örneğin, geriye zincirleme algoritmaları daha az belleğe gereksinim duyarken, ileriye zincirleme algoritmaları sorgu cevaplama aşamasında daha hızlıdır fakat çok yüksek miktarda belleğe gereksinim duyarlar.

Bu iki ana algoritma grubunun dışında her iki grubun olumlu taraflarını alarak, yeni bir çıkarsama algoritması tanımlama yönünde çalışmalar bulunmaktadır. Bu algoritmalarından bazılarının belli bağlamlarda başarılı sonuçlar verdikleri gösterilmiştir (Kiryakov, 2008). Jena (Wilkinson et al., 2003b) çıkarsama motoru da ileriye ve geriye zincirleme algoritmalarını birlikte kullandığı için bu bağlamda değerlendirilebilir. 3Store'da bu yöntemi kullanan bir RDFS çıkarsama motorudur (Harris and Gibbins, 2003). 3Store, üzerinde çıkarsama yapılacak aksiyomları gruplara ayırarak, göreceli olarak daha az miktarda olgu çıkarsanmasına neden olacak aksiyomlar üzerinde ileriye zincirleme yaparken, geri kalan tüm aksiyomlar üzerinde geriye zincirleme yöntemini uygular.

Benzer şekilde, tezde önerilen çıkarsama algoritması da ileriye zincirleme ve geriye zincirleme yöntemlerinin birleştirilmesi için önerilmiştir. Diğer çalışmalara göre en önemli farkı, Bölüm 3'te anlatılan uzantı tabanlı bilgi modelinin, çıkarsama sonucu üretilen aksiyomların sayısında sağladığı düşüştür. Önceki yıllarda, tez çalışmasıyla ilgili iki çalışma gerçekleştirilmiştir ((Öztürk et al., 2006) ve (Özacar et al., 2007)). (Öztürk et al., 2006)'da, OWL ontolojileri üzerinde ana bellekte, ileriye zincirleme algoritması kullanılmış ve sınıf örneği gruplama yapıtaşlarının kullanılmasıyla çıkarsanan "type" ilişkilerinin sayısında düşüş sağlanmıştır. (Özacar et al., 2007)'de ise, uzantı tabanlı bilgi modeli ile ileriye zincirleme algoritması birlikte kullanılmış ve dil karmaşıklık seviyesi RDFS olan bir çıkarsama motoru geliştirilmiştir. Tez çalışmasında ise veritabanı teknolojileri,

ileriye ve geriye zincirleme algoritmaları kullanarak, pD^* dilini destekleyecek bir çıkarsama algoritması ve bu karmaşıklık seviyesinde sorguların cevaplanmasını sağlayacak bir sorgulama algoritması sunulmaktadır. Uzantı tabanlı bilgi modelinin sürece katkısı aşağıdaki iki maddede özetlenmektedir;

- Çıkarsama sürecinde, ileriye zincirleme mekanizması şematik veri üzerinde çalışır. Şematik veri gerçek uygulamalarda kullanılan ontolojilerin çok küçük bir bölümüdür. Bu nedenle, ileriye zincirleme mekanizması, ontolojinin tümü üzerinde çıkarsama yapan yaklaşımlara göre çok daha hızlı çalışır. Bunun yanında, ileriye zincirleme mekanizması örneksel veri üzerinde çalışmadığı için, ileriye zincirleme mekanizmasının zaman gereksinimi örneksel verinin boyutundan bağımsızdır. Diğer bir deyişle, artan örnek verisi ile ileriye doğru zincirleme süresi artmaz.
- Yöntemin diğer bir faydası, örneksel veriyi ileriye zincirleme sürecine dahil etmemesine rağmen, örneksel verinin özeti olarak kabul edilen uzantılarla ilgili çıkarsamanın ileriye doğru zincirleme sürecinde tamamlanmasıdır. Bu yolla sıradan bir ontoloji şeması üzerinde, ileriye zincirleme algoritmasının yapabileceğinden daha fazla çıkarsama yapılmış olmaktadır. Sorgular sırasında, geriye zincirleme yöntemi uygulanarak çıkarsama tamamlanır. Şema üzerinde ileriye zincirleme algoritması uygulayıp, sorgulama aşamasında geriye zincirleme yöntemiyle çıkarsamayı tamamlayan benzer çalışmalar da bulunmaktadır (Stuckenschmidt and Broekstra, 2005). Fakat, tez çalışmasında diğer çalışmalardan farklı olarak uzantılarla ilgili çıkarsamanın ileriye zincirleme sürecinde tamamlanması sonucunda, geriye zincirleme sürecinde oluşturulan sorguların karmaşıklığı azaltılmaktadır. Tezde önerilen yöntem ile benzer çalışmadaki yöntem (*temel yöntem*) ait sorguların karmaşıklığı arasındaki fark aşağıda gösterilmektedir;

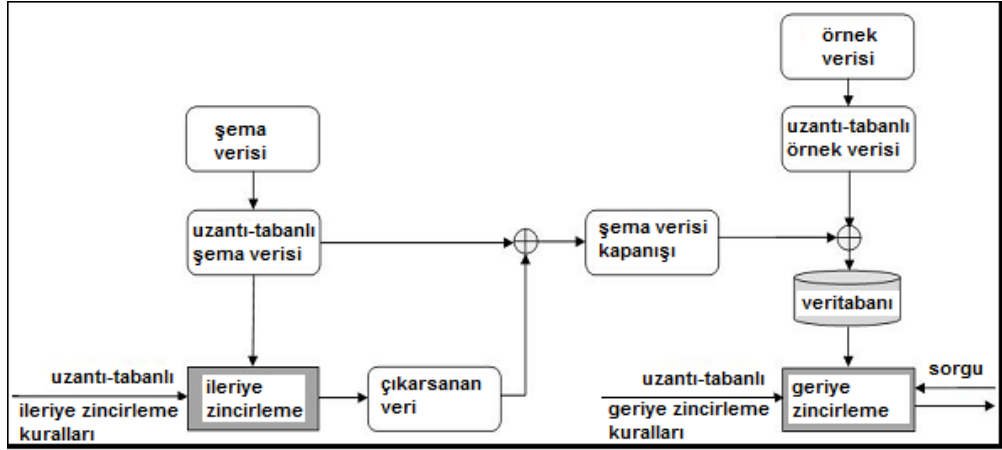
temel yöntem :

$type(?i, C) \Rightarrow$

$(subClassOf(?C_s, C) \wedge type(?i, C)) \vee$

$(subClassOf(?C_s, C) \wedge subPropertyOf(?p_s, p) \wedge domain(p, C_s) \wedge ?p_s(?i, ?x)) \vee$

$(subClassOf(?C_s, C) \wedge subPropertyOf(?p_s, p) \wedge range(p, C_s) \wedge ?p_s(?x, ?i))$



Şekil 4.1. Uzantı tabanlı çıkarsama süreci.

uzantı temelli :

$type(?i, C) \Rightarrow$

$hasClassExtension(?C, ?\varepsilon) \wedge contains(? \varepsilon, i)$

Bu örnekte, bir sınıfa ait örnekleri bulan tek koşullu bir sorgunun, geriye zincirleme sürecindeki dönüştürülmüş halleri karşılaştırılmaktadır. Temel yöntemde gösterilen sorguda $\vee$ bağlacı bulunmaktadır. Bunun anlamı tek bir koşulun üç ayrı sorguya dönüştürülmüş olduğudur. Bu nedenle, içinde n adet "type" koşulu bulunan bir sorgu, en az 3^n farklı sorguya dönüşmektedir. Fakat uzantı temelli sistemde $\vee$ bağlacı kullanılmasına gerek kalmamıştır, bu nedenle içinde n adet "type" koşulu bulunan bir sorgu her zaman tek bir sorguya dönüşür.

Şekil 4.1’de uzantı temelli çıkarsama sürecinin ayrıntıları gösterilmektedir. Uzantı temelli çıkarsama süreci, ontolojilerin uzantı temelli karşılıklarına çevrilmesiyle başlar. Bu çevrim sözdizimsel bir çevrimdir ve ontolojilere uzantı temelli çıkarsamayı destekleyecek yapıların eklenmesiyle gerçekleştirilir. Ardından, ontolojiler şema ve örnekler olarak iki bölüme ayrılır. Şema bölümü çıkarsama motoruna yüklenirken, örnekler bölümü Bölüm 3.2’de anlatılan veritabanı şemasına uygun olarak veritabanına aktarılır. Şematik veri üzerinde ileriye zincirleme yöntemi ile şema kapanışı çıkarılır ve sorgu zamanında uzantı tabanlı veri ile veritabanındaki örnek verileri birlikte kullanılır.

Sorgulama aşamasında kullanılan sorgulama algoritması, asıl sorgunun bir

ağaç yardımıyla yeniden yazımına dayanmaktadır. Burada kullanılan ağaç, SQL sorgularının başarımının artırılması için kullanılanlara (Ioannidis, 1996) yapısal açıdan benzemekle birlikte işlevsel açıdan tamamen farklıdır.

4.1 Uzantı Tabanlı Bilgi Modeli Çıkarsama Kuralları

Sistemde kullanılan çıkarsama motoru, kural tabanlı bir çıkarsama motordur. Kural tabanlı çıkarsama motorları tanımlanan kurallara bağlı olarak bilgi tabanı üzerinde çıkarsama yapmaktadırlar. OWL Reference (Dean et al., 2003) belgesinde belirtilen OWL gerektirme kuralları, uzantı tabanlı bilgi modeli ile dönüştürülmüş ontolojiler üzerinde eksiksiz çıkarsama yapamamaktadır. Bunun nedeni, bilgi modelinde kullanılan yapıların kurallarda yer almamasıdır. Kural tabanlı çıkarsama motorunun gruplama yapıtaşlarını destekleyebilmesi için ontolojiler üzerinde yapılan değişime benzer bir değişimin kurallar üzerinde de gerçekleştirilmesi gerekmektedir. Ayrıca, uzantı tabanlı bilgi modelini işleyebilmek için yalnızca uzantılar üzerinde işlem yapan ek kurallara da ihtiyaç bulunmaktadır.

4.1.1 Ontoloji Diline Ait Kurallar

pD^* , kural tabanlı sistemler ile tam ve eksiksiz çıkarsama yapabilmek için önerilmiş, en yüksek anlatımlılığa sahip betim mantığıdır (ter Horst, 2005). Bu tez kapsamında anlatılan kural tabanlı yaklaşım pD^* dilini desteklemektedir. Bu dille ilgili daha ayrıntılı bilgi, Bölüm 2.2.2’de verilmektedir.

Tez kapsamında pD^* diline ait gerektirme kuralları iki temel ilke gözetilerek, uzantı tabanlı modele uyumlu hale getirilmiştir. Aşağıda kural dönüşümünde kullanılan iki temel ilke sunulmaktadır;

- *Kuralların sınıflandırılması:* Kurallar etkiledikleri ontoloji parçasına göre iki gruba ayrılırlar: (a) TBox kuralları: ontolojinin şema kısmındaki veriyi kullanarak, ontoloji şeması hakkında yeni olgular türeten kurallar, (b) ABox kuralları: ontolojideki şema ve örnek verisinden yola çıkarak, örnek verisine yeni olgular ekleyen kurallar. Bu kurallar, *kural dönüşümü ilkesi* doğrultusunda dönüştürülerek, çıkarsamaya dahil edilirler.
- *Kural dönüşümü ilkesi:* Varolan kurallar, tek tek örnekler üzerinde değil,

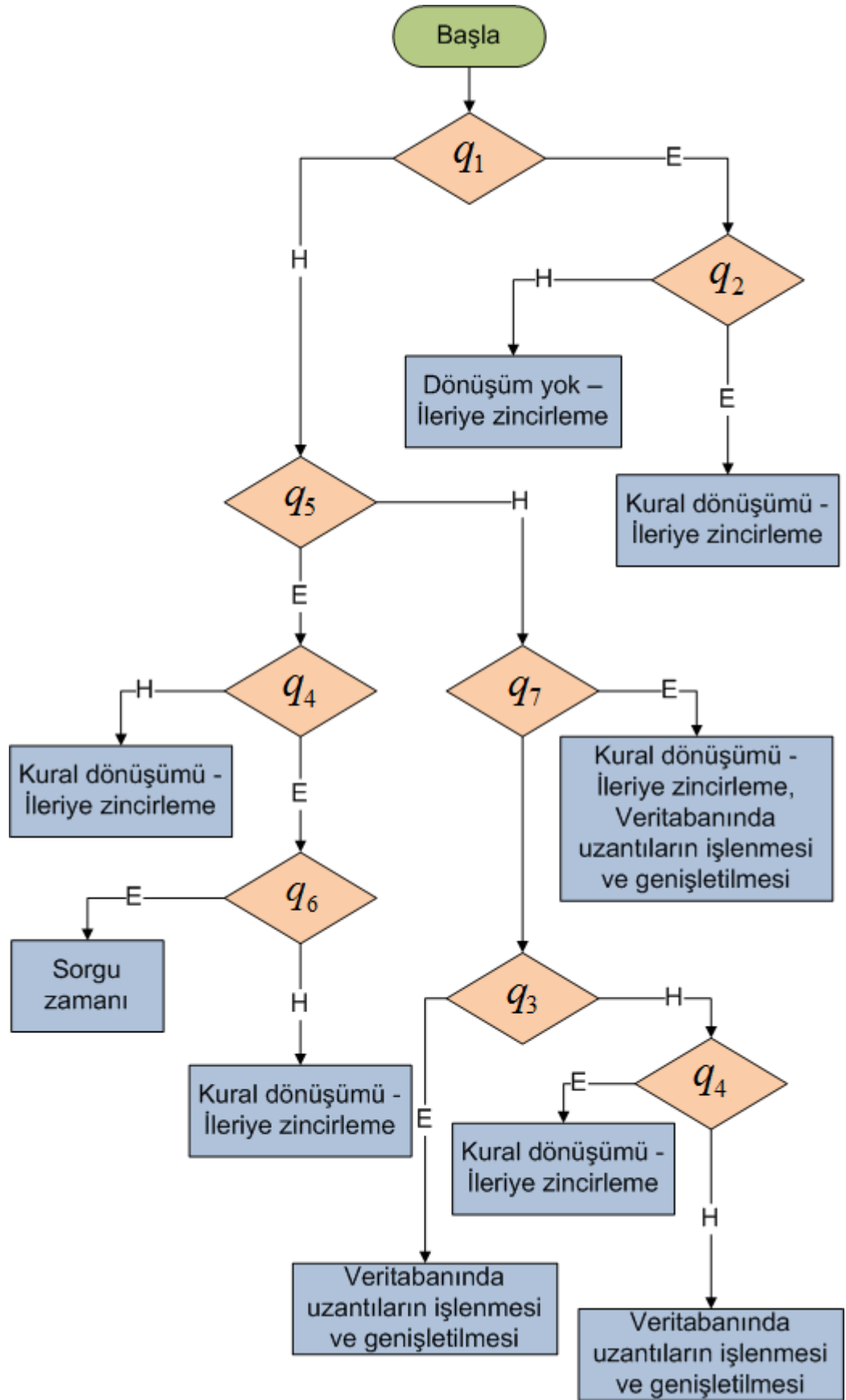
örneğin ait olduğu tüm uzantı üzerinde çalışacak şekilde dönüştürülür. Her kural üzerinde böyle bir dönüşüm uygulanması mümkün değildir. Çizelge 4.1’de verilen örüntüler doğrultusunda, dönüşüme uygun kurallar üzerinde dönüşüm uygulanır ve bu kurallar ileriye zincirleme yöntemiyle işlenir. Uzantı tabanlı modele uygun olarak dönüştürülemeyen kurallar ise veritabanında uzantıların işlenmesi ve genişletilmesi sırasında veya sorgu zamanında geriye zincirleme algoritmasıyla uygulanır. Şekil 4.2 kuralların örüntüler doğrultusunda işlenmesini gösteren bir akış şeması içermektedir. R bir kural, c bir koşul, $subj_c$ c koşulunun öznesi, $pred_c$ c koşulunun yüklemi, obj_c c koşulunun nesnesi, lhs_R , R kuralının sol el tarafındaki koşullar kümesi, rhs_R , R kuralının sağ el tarafındaki tek elemanlı koşullar kümesi, $Cond_{ABox}$ ABox ile ilgili koşullar kümesi ve $Cond_{TBox}$ TBox ile ilgili koşullar kümesi olmak üzere, kurallar aşağıdaki kriterlere göre gruplandırılır ve her bir gruptaki kural R , Çizelge 4.1’de karşılık gelen yöntemle işlenir;

- $q_1 : \forall c \in lhs_R (c \in Cond_{TBox})$: Bir kuralın sol el tarafındaki tüm koşullar TBox ile ilgilidir.
- $q_2 : \exists c_1 \in lhs_R (pred_{c_1} = rdf : type)$: Bir kuralın sol el tarafında yüklemi *type* olan en az bir koşul vardır.
- $q_3 : \forall c \in lhs_R (c \in Cond_{ABox})$: Bir kuralın sol el tarafındaki tüm koşullar ABox ile ilgilidir.
- $q_4 : \exists c_1 \in lhs_R (c_1 \in Cond_{ABox} \wedge pred_{c_1} \neq rdf : type)$: Bir kuralın sol el tarafında nitelik örnekleri ile örtüşen bir en az bir koşul vardır.
- $q_5 : \forall c \in rhs_R (pred_c = rdf : type)$: Bir kuralın sağ el tarafındaki koşulun yüklemi "type"tır.
- $q_6 : \exists c_1 \in lhs_R (pred_{c_1} = owl : hasValue \vee pred_{c_1} = owl : someValuesFrom \vee pred_{c_1} = owl : allValuesFrom)$: Kuralın sol el tarafında nitelik kısıtları ile ilgili en az bir koşul bulunmaktadır.
- $q_7 : \exists c_1 \in lhs_R (obj_{c_1} = owl : FunctionalProperty \vee obj_{c_1} = owl : InverseFunctionalProperty \vee obj_{c_1} = owl : SymmetricProperty \vee obj_{c_1} = owl : TransitiveProperty \vee pred_{c_1} = owl : inverseOf)$: Sol el tarafında nitelik karakteristikleri ile ilgili en az bir koşul bulunmaktadır.

Çıkarsama süresince uzantıların elemanları üzerinde değişiklik yapan bir kural çalışmamaktadır, yöntemin bu tür kurallara ihtiyacı yoktur. Yöntemin

ÖRÜNTÜ NO	ÖRÜNTÜ	ÖRÜNTÜYÜ DOĞRULAYAN KURALLAR KÜMESİ	İŞLENME ZAMANI
1	$q_1 \wedge \neg q_2$	$\{rdfs5, rdfs11, rdfp12a, rdfp12b, rdfp12c, rdfp13a, rdfp13b, rdfp13c\}$	Dönüşüm yok-ileriye zincirleme.
2	$q_1 \wedge q_2$	$\{rdfs6, rdfs8, rdfs10, rdfp9, rdfp10\}$	Kural dönüşümü-ileriye zincirleme.
3	$\neg q_1 \wedge \neg q_4 \wedge q_5$	$\{rdfs9, rdfs4a, rdfs4b\}$	Kural dönüşümü-ileriye zincirleme.
4	$\neg q_1 \wedge \neg q_3 \wedge q_4 \wedge \neg q_5 \wedge \neg q_7$	$\{rdfs7x\}$	Kural dönüşümü-ileriye zincirleme.
5	$\neg q_1 \wedge \neg q_3 \wedge \neg q_4 \wedge \neg q_5 \wedge \neg q_7$	$\{rdfp14bx\}$	Veritabanında uzantıların işlenmesi ve genişletilmesi.
6	$\neg q_1 \wedge q_4 \wedge q_5 \wedge q_6$	$\{rdfp14a, rdfp15, rdfp16\}$	Sorgu zamanı.
7	$\neg q_1 \wedge q_4 \wedge q_5 \wedge \neg q_6$	$\{rdfs2, rdfs3\}$	Kural dönüşümü-ileriye zincirleme.
8	$\neg q_1 \wedge q_3 \wedge \neg q_5 \wedge \neg q_7$	$\{rdfp5a, rdfp5b, rdfp6, rdfp7, rdfp11\}$	Veritabanında uzantıların işlenmesi ve genişletilmesi.
9	$\neg q_1 \wedge \neg q_5 \wedge q_7$	$\{rdfp1, rdfp2, rdfp3, rdfp4, rdfp8ax, rdfp8bx\}$	Kural dönüşümü-ileriye zincirleme, Veritabanında uzantıların işlenmesi ve genişletilmesi.

Çizelge 4.1. Kuralların işlenme zamanlarını belirleyen örüntüler.



Şekil 4.2. Kuralların örüntüler doğrultusunda işlenmesi.

sağladığı başarımlarının en önemli sebebi de budur. Uzantı elemanları üzerinde tek tek değişiklik yapılmasına gerek olmaması, çıkarsama yükünü çok büyük oranda azaltmaktadır. Bu nedenle, uzantı elemanları üzerinde tek tek çıkarsama yapan ve Çizelge 4.1’de belirtilen 2, 3, 4, 5, 8 ve 10 nolu örüntülere uyan kurallar üzerinde; uzantı elemanları değil, uzantıların kendisi üzerinde çalışmalarını sağlayacak dönüşümler gerçekleştirilmektedir.

Çizelge 4.2’de görülen kurallar pD^* dilinin D^* ile ilgili kural kümesidir. Bu kümede RDF dilinin semantiğinden gelen kurallar bulunmaktadır (Hayes, 2004). Bu kurallar arasında çıkarsama performansını en fazla etkileyen kurallar $rdfs9$ ve $rdfs7x$ kurallarıdır. Aşağıda gösterilen $rdfs9$ kuralı, bir sınıfa (v) ait örneklerin her biri (u) ile, bu sınıfın her bir üstsınıfı (w) arasında yeni bir "type" ilişkisi yaratmaktadır. Çıkarsama sonucu türetilen "type" ilişkilerinin çok büyük bir kısmı bu kural tarafından türetilmektedir (bkz. Bölüm 6.2).

- $rdfs9: v \text{ subClassOf } w \wedge u \text{ type } v \rightarrow u \text{ type } w$

Bu kural 4 numaralı örüntüyü karşılamaktadır. Bu kuralın, uzantı tabanlı modele uygun olarak dönüştürülmesi sonucunda, bir sınıfa ait her bir örneği, bu sınıfın her bir üst sınıfıyla ilişkilendirmek yerine, sınıfa (v) ait her bir örnek kümesi (e) ile her bir üstsınıf (w) arasında bir ilişki oluşturulur. Tahmin edileceği (ve Bölüm 5’te verilen örnek uygulamada görüleceği) üzere, bu kuralın dönüştürülmesi sonucunda çıkarsanan "type" ilişkilerinin sayısında büyük bir azalma olmaktadır. Aşağıda bu kuralın dönüştürülmüş hali görülmektedir;

- $v \text{ subClassOf } w \wedge v \text{ hasClassExtension } e \rightarrow w \text{ hasClassExtension } e$

Örüntü 5’e uyan, $rdfs7x$ kuralı ise bir niteliğe (p) ait örneklerin her biri ile, bu niteliğin her bir üstniteliğini (q) ilişkilendirerek, yeni bir nitelik örneği türetmektedir. Bu ilişki ABox ile ilgili olan çıkarsamanın en fazla yük getiren ikinci kuralıdır. Bunun nedeni, ontolojideki nitelik hiyerarşisinin derinliğiyle orantılı olarak yeni üçlü çıkarsanmasıdır.

- $rdfs7x: p \text{ subPropertyOf } q \wedge v \text{ p } w \rightarrow v \text{ q } w$

Bu kuralın, uzantı tabanlı modele uygun olarak dönüştürülmesi sonucunda, bir niteliğe ait her bir örneği, bu niteliğin her bir üstniteliğiyle ilişkilendirmek yerine,

niteliğe (p) ait her bir örnek kümesi (e) ile her bir üstnitelik (q) arasında bir ilişki oluşturulur. Tahmin edileceği (bkz. Bölüm 5'te verilen örnek uygulama) üzere, bu kuralın dönüştürülmesi sonucunda çıkarsanan nitelik örneklerinin sayısında büyük bir azalma olmaktadır. Aşağıda bu kuralın dönüştürülmüş hali görülmektedir;

- $p \text{ subPropertyOf } q \wedge p \text{ hasPropertyExtension } e \rightarrow q \text{ hasPropertyExtension } e$

D^* diline ait diğer bir kural seti, niteliklerin önalı ve artalanları ile ilgili çıkarsamaları gerçekleştiren $rdfs2$ ve $rdfs3$ kurallarıdır. Bu kurallar, 8 numaralı örüntüye uymaktadır. $rdfs2$ kuralı, bir niteliğe ait örnek kümesindeki her bir özneyi, niteliğin önalı olarak belirtilen sınıfla "type" ilişkisi ile bağlamaktadır;

- $rdfs2: p \text{ domain } u \wedge p \text{ w} \rightarrow v \text{ type } u$

Bu kuralın, aşağıda gösterildiği gibi uzantı tabanlı modele uyumlu olacak şekilde dönüştürülmesi sonucu elde edilen kural, niteliğe ait örneklerin öznelere içeren uzantılarla, niteliğin önalı olarak belirtilen sınıfı ilişkilendirmektedir;

- $p \text{ domain } u \wedge p \text{ hasSubjectExtension } e \rightarrow u \text{ hasClassExtension } e$

Benzer şekilde, $rdfs3$ kuralı, bir niteliğe ait örnek kümesindeki her bir nesneyi, niteliğin artalanı olarak belirtilen sınıfla "type" ilişkisi ile bağlamaktadır;

- $rdfs3: p \text{ range } u \wedge p \text{ w} \rightarrow w \text{ type } u$

Bu kuralın, aşağıda gösterildiği gibi uzantı tabanlı modele uyumlu olacak şekilde dönüştürülmesi sonucu elde edilen kural, niteliğe ait örneklerin nesnelere içeren uzantılarla, niteliğin artalanı olarak belirtilen sınıfı ilişkilendirmektedir;

- $p \text{ range } u \wedge p \text{ hasObjectExtension } e \rightarrow u \text{ hasClassExtension } e$

$rdfs4a$ ve $rdfs4b$ kuralları, ontolojideki her üçlünün öznesinin ve nesnesinin *Resource* sınıfının örnekleri olduğunu belirten ve ontolojideki üçlü sayısını genellikle iki kat, bazen daha fazla oranda arttıran kurallardır.

- $rdfs4a: v \text{ p } w \rightarrow v \text{ type Resource}$
- $rdfs4b: v \text{ p } w \rightarrow w \text{ type Resource}$

3 numaralı örüntüye uyan bu kuralların dönüştürülmesi sonucu çıkarsanan üçlü sayısında büyük oranda düşüş sağlanmaktadır. Dönüştürülen kurallar her bir özneyi ya da nesneyi tek tek *Resource* sınıfı ile bağlamak yerine, ontolojideki yüklemelere ait nitelik öznesi veya nesnesi gruplarını *Resource* sınıfı ile ilişkilendirmektedir;

- $p \text{ hasSubjectExtension } e \rightarrow \text{Resource hasClassExtension } e$
- $p \text{ hasObjectExtension } e \rightarrow \text{Resource hasClassExtension } e$

2 numaralı örüntü doğrultusunda dönüştürülmesi gereken diğer D^* kuralları, *rdfs6*, *rdfs8* ve *rdfs10* kurallarıdır. *rdfs6* ve *rdfs10*, her sınıfın ve niteliğin, kendisinin alt-sınıfı/altniteliği olduğunu belirtir. *rdfs8* ise her sınıfın *Resource* sınıfının alt-sınıfı olduğunu belirtir.

- $\text{rdfs6:}v \text{ type Property} \rightarrow v \text{ subPropertyOf } v$
- $\text{rdfs8:}v \text{ type Class} \rightarrow v \text{ subClassOf Resource}$
- $\text{rdfs10:}v \text{ type Class} \rightarrow v \text{ subClassOf } v$

Bu kuralların dönüştürülmesi, çıkarsanan üçlü sayısında bir azalma sağlamaz, fakat çıkarsamanın eksik olmaması için gereklidir. Dönüşüm sonunda her sınıf ve niteliğe, bir sınıf örneği ve nitelik örneği uzantısı eklenmiş olduğundan, sınıf örneği ve nitelik örneği uzantısına sahip her kaynağın, kendisinin alt-sınıfı/altniteliği olduğu çıkarsanabilir. Aynı mantıkla, kendisine sınıf örneği uzantısı eklenmiş her bir kaynak bir sınıftır, bu nedenle, bu kaynağın *Resource* sınıfının alt-sınıfı olduğu çıkarsanır. Aşağıda sırası ile *rdfs6*, *rdfs8* ve *rdfs10* kurallarının dönüştürülmüş halleri görülmektedir;

- $v \text{ hasExplicitPropertyExtension } e \rightarrow v \text{ subPropertyOf } v$
- $v \text{ hasExplicitClassExtension } e \rightarrow v \text{ subClassOf Resource}$
- $v \text{ hasExplicitClassExtension } e \rightarrow v \text{ subClassOf } v$

D^* kuralları içerisinde, *rdfs5* ve *rdfs11* şema ile ilgili kurallardır ve örneklerle ya da uzantılarla ilgili değildir. Dolayısıyla, bu kurallar üzerinde uzantı tabanlı bir dönüşüm yapılması gerekmez, çıkarsama sürecine doğrudan dahil edilirler. Çizelge 4.3, dönüştürülmüş D^* gerektirme kurallarını göstermektedir.

	Koşul	Sonuç
rdfs2	$p \text{ domain } u$ $v p w$	$v \text{ type } u$
rdfs3	$p \text{ range } u$ $v p w$	$w \text{ type } u$
rdfs4a	$v p w$	$v \text{ type Resource}$
rdfs4b	$v p w$	$w \text{ type Resource}$
rdfs5	$v \text{ subPropertyOf } w$ $w \text{ subPropertyOf } u$	$v \text{ subPropertyOf } u$
rdfs6	$v \text{ type Property}$	$v \text{ subPropertyOf } v$
rdfs7x	$p \text{ subPropertyOf } q$ $v p w$	$v q w$
rdfs8	$v \text{ type Class}$	$v \text{ subclassOf Resource}$
rdfs9	$v \text{ subclassOf } w$ $u \text{ type } v$	$u \text{ type } w$
rdfs10	$v \text{ type Class}$	$v \text{ subclassOf } v$
rdfs11	$v \text{ subclassOf } w$ $w \text{ subclassOf } u$	$v \text{ subclassOf } u$

Çizelge 4.2. D^* gerektirme kuralları.

	Koşul	Sonuç
rdfs2	$p \text{ domain } u$ $p \text{ hasSubjectExtension } e$	$u \text{ hasClassExtension } e$
rdfs3	$p \text{ range } u$ $p \text{ hasObjectExtension } e$	$u \text{ hasClassExtension } e$
rdfs4a	$p \text{ hasSubjectExtension } e$	$\text{Resource hasClassExtension } e$
rdfs4b	$p \text{ hasObjectExtension } e$	$\text{Resource hasClassExtension } e$
rdfs5	$v \text{ subPropertyOf } w$ $w \text{ subPropertyOf } u$	$v \text{ subPropertyOf } u$
rdfs6	$v \text{ hasExplicitPropertyExtension } e$	$v \text{ subPropertyOf } v$
rdfs7xa	$p \text{ subPropertyOf } q$ $p \text{ hasPropertyExtension } e$	$q \text{ hasInferredPropertyExtension } e$
rdfs7xb	$p \text{ subPropertyOf } q$ $p \text{ hasObjectExtension } e$	$q \text{ hasInferredObjectExtension } e$
rdfs7xc	$p \text{ subPropertyOf } q$ $p \text{ hasSubjectExtension } e$	$q \text{ hasInferredSubjectExtension } e$
rdfs8	$v \text{ hasExplicitClassExtension } e$	$v \text{ subClassOf } \text{Resource}$
rdfs9	$v \text{ subClassOf } w$ $v \text{ hasClassExtension } e$	$w \text{ hasInferredClassExtension } e$
rdfs10	$v \text{ hasExplicitClassExtension } e$	$v \text{ subClassOf } v$
rdfs11	$v \text{ subClassOf } w$ $w \text{ subClassOf } u$	$v \text{ subClassOf } u$

Çizelge 4.3. Dönüştürülmüş D^* gerektirme kuralları.

D^* kural kümesinin aksine, p kural kümesi (Çizelge 4.4) OWL dili ile ilgili kuralları içermektedir. p kuralları içerisinde, $rdfp12a$, $rdfp12b$, $rdfp12c$, $rdfp13a$, $rdfp13b$ ve $rdfp13c$ şema ile ilgili kurallardır ve örneklerle ya da uzantılarla ilgili değildir. Dolayısıyla, bu kurallar üzerinde uzantı tabanlı bir dönüşüm yapılması gerekmez, örüntü 1 doğrultusunda çıkarsama sürecine doğrudan dahil edilirler.

2 numaralı örüntü doğrultusunda dönüştürülmesi gereken diğer p kuralları, $rdfp9$ ve $rdfp10$ kurallarıdır. $rdfp9$ ve $rdfp10$, her sınıfın/niteliğin, $owl:sameAs$ ilişkisi ile bağlanmış olduğu sınıfın/niteliğin alt sınıfı/altniteliği olduğunu belirtir.

- $rdfp9: v \text{ type } Class \wedge v \text{ sameAs } w \rightarrow v \text{ subClassOf } w$
- $rdfp10: p \text{ type } Property \wedge p \text{ sameAs } q \rightarrow p \text{ subPropertyOf } q$

Bu kuralların dönüştürülmesi çıkarsanan üçlü sayısında bir azalma sağlamaz, fakat çıkarsamanın eksik olmaması için gereklidir. Dönüşüm sonunda her sınıf ve niteliğe bir sınıf örneği ve nitelik örneği uzantısı eklenmiş olduğundan, sınıf örneği ve nitelik örneği uzantısına sahip her kaynağın, $owl:sameAs$ ilişkisi ile bağlanmış olduğu sınıfın/niteliğin alt sınıfı/altniteliği olduğu çıkarsanabilir. Aşağıda sırası ile $rdfp9$ ve $rdfp10$ kurallarının dönüştürülmüş halleri görülmektedir;

- $v \text{ hasExplicitClassExtension } e \wedge v \text{ sameAs } w \rightarrow v \text{ subClassOf } w$
- $p \text{ hasExplicitPropertyExtension } e \wedge p \text{ sameAs } q \rightarrow p \text{ subPropertyOf } q$

$rdfp14bx$, $rdfp5a$, $rdfp5b$, $rdfp6$, $rdfp7$ ve $rdfp11$ kuralları 6 ve 9 numaralı örüntüler doğrultusunda, veritabanında uzantıların işlenmesi ve genişletilmesi sırasında çalıştırılırlar. Bu kuralların işlenmesi Bölüm 4.2.3'te detaylı olarak anlatılmaktadır.

$rdfp1$, $rdfp2$, $rdfp3$, $rdfp4$, $rdfp8ax$ ve $rdfp8bx$ kuralları kısmi olarak ileriye zincirleme sürecinde, kısmi olarak ise veritabanında uzantıların işlenmesi ve genişletilmesi (bkz. Bölüm 4.2.3) sürecinde çalıştırılırlar. Bu kurallar nitelik karakteristikleri ile ilgili kurallardır. Nitelik karakteristiklerine ve ters rollere bağlı olarak çıkarsanması gerekli nitelik örneklerinin hesaplanması, ileriye zincirleme sürecinden sonra, veritabanında uzantıların işlenmesi ve genişletilmesi aşamasında gerçekleştirilir. Fakat veritabanında uzantıların işlenmesi ve genişletilmesine yardımcı olacak bazı ilişkiler ileriye zincirleme sürecinde çıkarsanır. $rdfp1$, $rdfp2$, $rdfp3$, $rdfp4$,

rdfp8ax ve *rdfp8bx* kurallarının dönüştürülmesi sonucu oluşan bu kurallar, veritabanında uzantıların işlenmesi ve genişletilmesi sürecinde kullanılacak ilişkilerin türetilmesini sağlar.

Bu bölümde, nitelik karakteristiklerine ve ters rollere bağlı olarak, ilgili nitelik uzantıları ile niteliği, özelleşmiş grupta yapıtaşları ile bağlayan dönüştürülmüş kurallar gösterilmektedir. *rdfp1* kuralının dönüşmüş hali olan ilk kural fonksiyonel bir nitelik ile bu niteliğe ait nitelik uzantılarını *hasFunctionalPropertyExtension* ilişkisi ile bağlamaktadır. *rdfp2* kuralının dönüştürülmesi sonucu oluşan ikinci kural, ters-fonksiyonel bir nitelik ile bu niteliğe ait nitelik uzantılarını *hasInversePropertyExtension* ilişkisi ile bağlamaktadır. *rdfp3* kuralının dönüştürülmüş hali olan üçüncü kural, simetrik bir nitelik ile bu niteliğe ait nitelik uzantılarını *hasSymmetricPropertyExtension* ilişkisi ile bağlar. *rdfp4* kuralının dönüşmüş hali olan dördüncü kural, geçişken bir nitelik ile bu niteliğe ait nitelik uzantılarını *hasTransitivePropertyExtension* ilişkisi ile bağlar. *rdfp8ax* ve *rdfp8bx* ise son dört kurala dönüşürler. Bu dört kural birbirlerinin ters rolü olarak belirtilmiş iki role ait; (a) düz nitelik uzantılarını, ters rol ile *hasInversePropertyExtension* ilişkisi ile, (b) ters nitelik uzantılarını ise, ters rol ile *hasPropertyExtension* ilişkisi ile bağlar.

- $p \text{ type } FunctionalProperty \wedge p \text{ hasPropertyExtension } e \rightarrow p \text{ hasFunctionalPropertyExtension } e$
- $p \text{ type } InverseFunctionalProperty \wedge p \text{ hasPropertyExtension } e \rightarrow p \text{ hasInverseFunctionalPropertyExtension } e$
- $p \text{ type } SymmetricProperty \wedge p \text{ hasPropertyExtension } e \rightarrow p \text{ hasSymmetricPropertyExtension } e$
- $p \text{ type } TransitiveProperty \wedge p \text{ hasPropertyExtension } e \rightarrow p \text{ hasTransitivePropertyExtension } e$
- $p \text{ inverseOf } q \wedge p \text{ hasPropertyExtension } e \rightarrow q \text{ hasInversePropertyExtension } e$
- $p \text{ inverseOf } q \wedge q \text{ hasPropertyExtension } e \rightarrow p \text{ hasInversePropertyExtension } q$
- $p \text{ inverseOf } q \wedge p \text{ hasInversePropertyExtension } e \rightarrow q \text{ hasPropertyExtension } e$
- $p \text{ inverseOf } q \wedge q \text{ hasInversePropertyExtension } e \rightarrow p \text{ hasPropertyExtension } e$

rdfp14a, *rdfp15* ve *rdfp16* kuralları ise örüntü 7 doğrultusunda, tamamıyla sorgu zamanında işletilirler. Dolayısıyla bu kurallar çıkarsama sürecine dahil edilmezler.

	Koşul	Sonuç
rdfp1	$p \text{ type } FunctionalProperty$ $u \text{ p } v$ $u \text{ p } w$	$v \text{ sameAs } w$
rdfp2	$p \text{ type } InverseFunctionalProperty$ $u \text{ p } w$ $v \text{ p } w$	$u \text{ sameAs } v$
rdfp3	$p \text{ type } SymmetricProperty$ $v \text{ p } w$	$w \text{ p } v$
rdfp4	$p \text{ type } TransitiveProperty$ $u \text{ p } v$ $v \text{ p } w$	$u \text{ p } w$
rdfp5a	$v \text{ p } w$	$v \text{ sameAs } v$
rdfp5b	$v \text{ p } w$	$w \text{ sameAs } w$
rdfp6	$v \text{ sameAs } w$	$w \text{ sameAs } v$
rdfp7	$u \text{ sameAs } v$ $v \text{ sameAs } w$	$u \text{ sameAs } w$
rdfp8ax	$p \text{ inverseOf } q$ $v \text{ p } w$	$w \text{ q } v$
rdfp8bx	$p \text{ inverseOf } q$ $v \text{ q } w$	$w \text{ p } v$
rdfp9	$v \text{ type } Class$ $v \text{ sameAs } w$	$v \text{ subClassOf } w$
rdfp10	$p \text{ type } Property$ $p \text{ sameAs } q$	$p \text{ subPropertyOf } q$
rdfp11	$u \text{ p } v$ $u \text{ sameAs } u'$ $v \text{ sameAs } v'$	$u' \text{ p } v'$
rdfp12a	$u \text{ equivalentClass } w$	$u \text{ subClassOf } w$

rdfp12b	$u \text{ equivalentClass } w$	$w \text{ subclassOf } u$
rdfp12c	$u \text{ subclassOf } w$ $w \text{ subclassOf } u$	$u \text{ equivalentClass } w$
rdfp13a	$v \text{ equivalentProperty } w$	$w \text{ subPropertyOf } v$
rdfp13b	$v \text{ equivalentProperty } w$	$v \text{ subPropertyOf } w$
rdfp13c	$v \text{ subPropertyOf } w$ $w \text{ subPropertyOf } v$	$v \text{ equivalentProperty } w$
rdfp14a	$v \text{ hasValue } w$ $v \text{ onProperty } p$ $u \text{ p } w$	$u \text{ type } v$
rdfp14bx	$v \text{ hasValue } w$ $v \text{ onProperty } p$ $u \text{ type } v$	$u \text{ p } w$
rdfp15	$v \text{ someValuesFrom } w$ $v \text{ onProperty } p$ $u \text{ p } x$ $x \text{ type } w$	$u \text{ type } v$
rdfp16	$v \text{ allValuesFrom } w$ $v \text{ onProperty } p$ $u \text{ type } v$ $u \text{ p } x$	$x \text{ type } w$

Çizelge 4.4: p gerektirme kuralları.

	Koşul	Sonuç
rdfp1	$p \text{ type } \text{FunctionalProperty}$ $p \text{ hasPropertyExtension } e$	$p \text{ hasFunctionalPropertyExtension } e$
rdfp2	$p \text{ type } \text{InverseFunctionalProperty}$ $p \text{ hasPropertyExtension } e$	$p \text{ hasInverseFunctionalPropertyExtension } e$
rdfp3	$p \text{ type } \text{SymmetricProperty}$ $p \text{ hasPropertyExtension } e$	$p \text{ hasSymmetricPropertyExtension } e$
rdfp4	$p \text{ type } \text{TransitiveProperty}$	

	$p \text{ hasPropertyExtension } e$	$p \text{ hasTransitivePropertyExtension } e$
rdfp5a	$u \text{ } p \text{ } w$	$u \text{ sameAs } u$
rdfp5b	$u \text{ } p \text{ } w$	$w \text{ sameAs } w$
rdfp6	$v \text{ sameAs } w$	$w \text{ sameAs } v$
rdfp7	$u \text{ sameAs } v$ $v \text{ sameAs } w$	$u \text{ sameAs } w$
rdfp8ax	$p \text{ inverseOf } q$ $p \text{ hasPropertyExtension } e$	$q \text{ hasInversePropertyExtension } e$
rdfp8bx	$p \text{ inverseOf } q$ $q \text{ hasPropertyExtension } e$	$p \text{ hasInversePropertyExtension } e$
rdfp8cx	$p \text{ inverseOf } q$ $p \text{ hasInversePropertyExtension } e$	$q \text{ hasPropertyExtension } e$
rdfp8dx	$p \text{ inverseOf } q$ $q \text{ hasInversePropertyExtension } e$	$p \text{ hasPropertyExtension } e$
rdfp9	$v \text{ hasExplicitSubExtension } e$ $v \text{ sameAs } w$	$v \text{ subClassOf } w$
rdfp10	$p \text{ hasExplicitPropertyExtension } e$ $p \text{ sameAs } q$	$p \text{ subPropertyOf } q$
rdfp11	$u \text{ } p \text{ } v$ $u \text{ sameAs } u'$ $v \text{ sameAs } v'$	$u' \text{ } p \text{ } v'$
rdfp12a	$u \text{ equivalentClass } w$	$u \text{ subClassOf } w$
rdfp12b	$u \text{ equivalentClass } w$	$w \text{ subClassOf } u$
rdfp12c	$v \text{ subClassOf } w$ $w \text{ subClassOf } v$	$v \text{ equivalentClass } w$
rdfp13a	$v \text{ equivalentProperty } w$	$v \text{ subPropertyOf } w$
rdfp13b	$v \text{ equivalentProperty } w$	$w \text{ subPropertyOf } v$
rdfp13c	$v \text{ subPropertyOf } w$ $w \text{ subPropertyOf } v$	$v \text{ equivalentProperty } w$

Çizelge 4.5: Dönüştürülmüş p gerektirme kuralları.

	Koşul	Sonuç
rdfs:ebkm1	$p \text{ hasExplicitClassExtension } e$	$p \text{ hasClassExtension } e$
rdfs:ebkm2	$p \text{ hasExplicitPropertyExtension } e$	$p \text{ hasPropertyExtension } e$
rdfs:ebkm3	$p \text{ hasExplicitSubjectList } e$	$p \text{ hasSubjectList } e$
rdfs:ebkm4	$p \text{ hasExplicitObjectList } e$	$p \text{ hasObjectList } e$

Çizelge 4.6. p dili uzantı tabanlı bilgi modeli gerektirme kuralları.

Bu kuralların dönüştürülmesinin herhangi bir anlam kaybına neden olup olmadığını inceleyebilmek için OWL Sınama Durumları belgesi kullanılmıştır (Carroll and Roo, 2004). Bu konuyla ilgili ayrıntılı inceleme ve açıklama Bölüm 6.3’te verilmektedir.

4.1.2 Uzantı Tabanlı Bilgi Modelinin Gerektirdiği Kurallar

Uzantı tabanlı bilgi modeli, çıkarsamanın ontolojide varolan sınıf ve nitelik örnekleri bazında yapılması yerine örnek grupları üzerinde yapılmasını sağlayarak, çıkarsama performansını arttırmalar. Uzantı adı verilen bu örnek grupları ile ilgili dilin semantiğinden gelen kurallar Bölüm 4.1.1’de verilmiştir.

Uzantı tabanlı bilgi modelinde, uzantı bağlantılarının doğru ve tam kurulabilmesi, çıkarsamanın eksiksiz ve tam yapılabilmesi açısından önem taşımaktadır. Bu nedenle dönüştürülmüş kurallara ek olarak bazı ekstra kurallara gereksinim duyulmaktadır. Bu kurallar, Çizelge 4.6 ve Çizelge 4.7’de verilmektedir.

Çizelge 4.6’da verilen kurallar uzantıların daha kolay sorgulanabilmesi için konulmuştur. Bu kurallar çalıştırıldığı zaman bir kaynağın belirtik uzantıları ile örtük uzantılarının ayrı ayrı sorgulanmasına gerek kalmamaktadır; bir kaynağın tüm uzantıları tek bir sorgu ile elde edilebilmektedir.

Çizelge 4.7’de verilen kurallar ise nitelik uzantı karakteristiklerinin doğru işlenebilmesi ile ilgilidir. Nitelik uzantılarının, üst niteliklerle doğru olarak ilişkilendirilebilmelerini sağlarlar.

Koşul	Sonuç
rdfp:ebkm1 $p \text{ subPropertyOf } q$	
$p \text{ hasSymmetricPropertyExtension } e$	$q \text{ hasSymmetricPropertyExtension } e$
rdfp:ebkm2 $p \text{ subPropertyOf } q$	
$p \text{ hasTransitivePropertyExtension } e$	$q \text{ hasTransitivePropertyExtension } e$
rdfp:ebkm3 $p \text{ subPropertyOf } q$	
$p \text{ hasFunctionalPropertyExtension } e$	$q \text{ hasFunctionalPropertyExtension } e$
rdfp:ebkm4 $p \text{ subPropertyOf } q$	
$p \text{ hasInverseFunctionalPropertyExtension } e$	$q \text{ hasInverseFunctionalPropertyExtension } e$
rdfp:ebkm5 $p \text{ subPropertyOf } q$	
$p \text{ hasInverseExtension } e$	$q \text{ hasInverseExtension } e$

Çizelge 4.7. D^* dili uzantı tabanlı bilgi modeli gerektirme kuralları.

4.2 Çıkarsama Süreci

Tez kapsamında önerilen çıkarsama algoritması, herhangi bir ileriye zincirleme algoritmasının dönüştürülmüş kurallarla desteklenerek, uzantı tabanlı bilgi modelini işleyecek hale getirilmesi prensibine dayanır. Başka bir deyişle burada sunulan katkı, uzantı tabanlı bilgi modeline uygun olarak, eksiksiz, doğru ve verimli çıkarsama yapacak bir sürecin tanımlanmasıdır.

4.2.1 Üçlülerin Filtrelenmesi

İleriye zincirleme yöntemi ile çıkarsamaya başlamadan önce, veritabanında saklanan üçlüler analiz edilmeli ve çıkarsamaya katılacak üçlüler çıkarsama motorunun çalışma belleğine yüklenmelidir.

Çıkarsamaya katılan üçlü sayısı ontoloji şemasının boyutuna bağlıdır, başka bir deyişle, örnek verilerinin boyutundan bağımsızdır. Ontoloji örnekleri çıkarsama motorunun belleğine yüklenmez. Bunun sonucunda, artan örnek sayısı ile doğru orantılı olarak bellek kazancı sağlanır. Modelden faydalanma oranının, ontoloji ölçütleri cinsinden bir tahmini ve dönüştürülmüş modeldeki toplam üçlü sayısını hesaplamak için Bölüm 6.2’de bir formül verilmektedir. Aynı zamanda modelden faydalanma oranını gösterebilmek için, karşılaştırmalı değerlendirme uygulamaları kullanılmış ve sonuçlar Bölüm 7’de sunulmuştur.

4.2.2 İleriye Zincirleme Algoritmasının Uygulanması

Çıkarsama sürecinin amacı kavramlar, uzantılar ve bunların kendi aralarındaki ve birbirleriyle olan ilişkileri hakkındaki önermeleri, diğer bir deyişle, şematik önermeleri çıkarsayabilmektir. Uzantılarla ilgili ilişkilerin çıkarsanması, ontoloji şemasının tanımından türeyen sınıf ve nitelik örneklerinin çıkarsanmasını sağlayacaktır. Bu nedenlerden ötürü, çıkarsama sürecine yalnızca şematik önermeler dahil edilir. Örnek verisi çıkarsamaya dahil edilmediğinden, örnek verisindeki artış çıkarsamanın bellek ve zaman gereksimini arttırmamaktadır.

Çıkarsama sürecinde, uzantı tabanlı modele dönüştürülmüş ontoloji üzerinde, uzantı tabanlı modelin gereksinimleri doğrultusunda dönüştürülmüş kurallar, ileriye zincirleme algoritmasına uygun olarak uygulanır. Çıkarsama pD^* dilini (ter

Horst, 2005) desteklemektedir.

Çıkarsama süreci sonucunda, şematik önermeler çıkarsanmış olmaktadır. Örnek verileriyle ilgili çıkarsamalar sorgu zamanında tamamlanır. Fakat tüm örnek verilerinin sorgu zamanında hesaplanması, sorgu performansı açısından verimli olmayacaktır. Bu nedenle, çıkarsama ve sorgulama süreçleri arasına veritabanında uzantıların işlenmesi ve genişletilmesi (bkz. Bölüm 4.2.3) adında ek bir süreç konulmuştur. Bu ek süreçte, veritabanında bazı örnekssel verilerin çıkarsanması gerçekleştirilir.

4.2.3 Veritabanında Uzantıların İşlenmesi ve Genişletilmesi

Örnek verileriyle ilgili tüm çıkarsamaların sorgu zamanında hesaplanması, sorgu performansı açısından verimli olmayacaktır. Bu nedenle, sorgulama sürecinden önce, veritabanında bazı örnekssel verilerin çıkarsanması ve uzantılarla ilgili kimlikli bilgiler oluşturulması gerçekleştirilir. Bu süreçte sırası ile aşağıdaki işlemler gerçekleştirilir;

1. Nitelik kısıtları yardımıyla tanımlanmış sınıflara ait uzantıların *RestrictionComplexExtensions* veri yapısında, küme operatörleri yardımıyla tanımlanmış sınıflara ait uzantıların ise *SetComplexExtensions* veri yapısında saklanması. Tüm bu karmaşık sınıflara ait sınıf örneği uzantılarının *ComplexExtensions* veri yapısında saklanması,
2. Karakteristiği olan nitelikleri ve bu niteliklere ait karakteristikleri saklayan *PropertiesWithCharacteristics* veri yapısının tanımlanması,
3. *owl:hasValue* nitelik kısıtı sonucu çıkarsanan nitelik örneklerinin ilgili nitelik örneği uzantılarına eklenmesi,
4. Geçişken ve simetrik niteliklere ait uzantı kümelerinin bu karakteristikler doğrultusunda genişletilmesi.

İlk adımda, *RestrictionComplexExtensions*, *SetComplexExtensions* ve *ComplexExtensions* veri yapılarına ait sahalara doldurulur. Bu veri yapıları aşağıda tanımlanmakta ve sahaları belirtilmektedir (Çizelge 4.8);

Veri Yapısı	Sahalar
ComplexExtensions	extensionUri, extensionType
RestrictionComplexExtensions	extensionUri, restrictionType, onProperty, classOrValueUri
SetComplexExtensions	extensionUri, setOperator, listOfSetElements

Çizelge 4.8. Karmaşık sınıf uzantı bilgilerini saklayan veri yapıları.

- *RestrictionComplexExtensions*: Nitelik kısıtları (*owl:someValuesFrom*, *owl:allValuesFrom* ve *owl:hasValue*) yardımıyla tanımlanmış OWL sınıflarına ait sınıf örneği uzantılarını saklar. Bu uzantılar bir sınıf için bir niteliğin alabileceği değerleri tanımlarlar. Bu nedenle, *RestrictionComplexExtensions* veri yapısı sahalarında, uzantı URI'sinin (*extensionUri*) ve kısıt tipinin (*restrictionType*) yanısıra, kısıtın üzerinde tanımlandığı niteliğin URI'si (*onProperty*) ve niteliğin alabileceği değerleri belirten sınıfın veya değerin URI'si (*classOrValueUri*) de yer alır.
- *SetComplexExtensions*: Küme operatörleri yardımıyla tanımlanmış OWL sınıflarına ait sınıf örneği uzantılarını saklar. Bu sınıflar, bir dizi sınıfın birleşimi (*owl:unionOf*) ve kesişimi (*owl:intersectionOf*) ile tanımlanırlar. Bu nedenle, *SetComplexExtensions* veri yapısı sahalarında, uzantı URI'sinin (*extensionUri*) ve operatör adının (*setOperator*) yanısıra, küme operatörünün işlediği sınıf listesi de (*listOfSetElements*) yer alır.
- *ComplexExtensions*: Nitelik kısıtları veya küme operatörleri yardımıyla tanımlanmış karmaşık sınıflara ait sınıf örneği uzantılarını saklar. *ComplexExtensions* veri yapısı sahalarında, uzantı URI'si (*extensionUri*) ve "restrictionComplexExtension" veya "setComplexExtension" değerlerini alan uzantı tipi (*extensionType*) saklanır.

İkinci adımda, *PropertiesWithCharacteristics* veri yapısına ait sahalar doldurulur. Bu veri yapısı aşağıda tanımlanmakta ve sahaları belirtilmektedir;

- *PropertiesWithCharacteristics*: Karakteristiği (*owl:SymmetricProperty*, *owl:TransitiveProperty*, *owl:FunctionalProperty* ve

Veri Yapısı	Sahalar
PropertiesWithCharacteristics	propertyUri, sym, tran, func, invfunc

Çizelge 4.9. Nitelik uzantılarının karakteristiklerini saklayan veri yapıları.

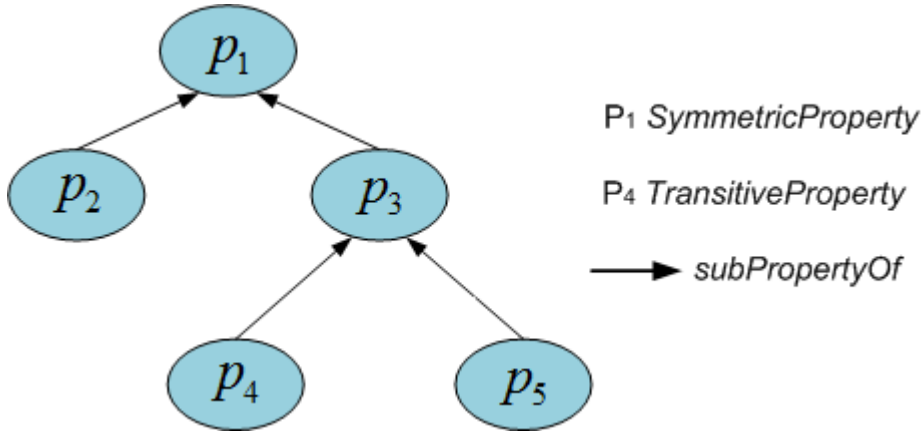
owl:InverseFunctionalProperty) olan nitelikleri ve bu niteliklere ait karakteristikleri saklar. *PropertiesWithCharacteristics* veri yapısı sahalarında, nitelik URI'sini saklayan *propertyUri* sahası ve niteliğin karakteristiklerine bağlı olarak "true" veya "false" değer alan *sym*, *tran*, *func* ve *invfunc* sahaları yer alır (Çizelge 4.9).

Üçüncü adımda, *owl:hasValue* nitelik kısıtı sonucu çıkarsanan nitelik örneklerinin ilgili nitelik örneği uzantısına eklenmesi işlemi gerçekleştirilmektedir. Aşağıda gösterilen *rdfp14bx* kuralı doğrultusunda, *statements* tablosuna ve ilgili niteliğe ait nitelik öznesi ve nitelik nesnesi uzantılarına eklemeler yapılmaktadır. Diğer bir deyişle, niteliğin temsili nitelik örneği uzantısı da genişletilmiş olmaktadır.

$$v \text{ hasValue } w \wedge v \text{ onProperty } p \wedge u \text{ type } v \rightarrow u \text{ p } w$$

Dördüncü adımda, geçişken ve simetrik niteliklere ait uzantı kümeleri, bu karakteristikler doğrultusunda genişletilmektedir. *PropertiesWithCharacteristics* veri yapısında, hangi niteliğin hangi karakteristiklere sahip olduğu saklanmaktadır. Bir niteliğin birden fazla uzantısı olabilir, diğer bir deyişle bir niteliğe ait tüm örnekler bu uzantıların birleşimidir. Bu nedenle, nitelik karakteristiklerinin, her bir uzantı üzerinde ayrı ayrı uygulanması yeterli değildir. Karakteristiklerin uzantı kümelerinin birleşimi üzerinde uygulanması gerekmektedir. Bu durumu bir örnekle açıklamak gerekirse, geçişken olarak tanımlanmış *p* niteliğinin $e_1 = a \text{ p } b, c \text{ p } d$ ve $e_2 = d \text{ p } e, f \text{ p } g$ olmak üzere, iki nitelik örneği uzantısı olduğunu varsayalım. Bu nitelik örneği uzantılarına ayrı ayrı geçişkenlik karakteristiği uygulandığında hiçbir üçlü çıkarsanmaz. Fakat bu iki uzantının birleşimine $e_1 \cup e_2 = a \text{ p } b, c \text{ p } d, d \text{ p } e, f \text{ p } g$ geçişkenlik karakteristiği uygulandığında, $c \text{ p } e$ üçlüsü çıkarsanacaktır.

Nitelik uzantılarının karakteristikler doğrultusunda genişletilmesi bir örnekle açıklanacaktır. Şekil 4.3'te p_1, p_2, p_3, p_4 ve p_5 adlarında beş nitelik bulunmaktadır. Beş niteliğin her birinin sırasıyla e_1, e_2, e_3, e_4 ve e_5 isminde nitelik



Şekil 4.3. Nitelik sıradüzen örneği.

Nitelik Adı	Uzantı Tanımı
p_1	$sym(e_2 + e_3 + tran(e_4))$
p_2	e_2
p_3	$tran(e_4) + e_3$
p_4	$tran(e_4)$
p_5	e_5

Çizelge 4.10. Örnek niteliklerin nihai uzantılarının hesaplanması.

uzantıları bulunmaktadır. Ayrıca, p_1 simetrik, p_4 ise geçişken nitelik olarak tanımlanmıştır.

Uzantıların genişletilmesine, sıradüzenin en altındaki niteliklerden başlanır. Böylelikle, altnitelikten üstniteliğe uzantı aktarılırken, aktarılma patikası üzerinde bulunan niteliklerin karakteristikleri de bu uzantılara etki eder. Tüm karakteristik etkilerinin tam ve eksiksiz olarak hesaplanabilmesi için, niteliklerin uzantılarının hesaplanmasına en alttaki nitelikten başlanması gereklidir. Çizelge 4.10'da, çıkarsama süreci sonunda niteliklerin nihai uzantı hesaplamaları verilmiştir.

Çizelgedeki tanımlarda bahsedilen *tran* ve *sym* işlemleri uzantıların geçişken ve simetrik hallerini hesaplarlar. Bu hesaplama sonucu elde edilen üçlüler *statements* tablosuna, üçlünün yüklemine ait nitelik öznesi uzantısına ve üçlünün yüklemine ait nitelik nesnesi uzantısına dahil edilirler. Böylelikle sorgu zamanında, bir niteliğe ait örnekleri sorgulamak için, yalnızca niteliğe ait belirtik uzantı (*hasExplicitPropertyExtension* ilişkisi ile niteliğe bağlanmış uzantılar) elemanlarına ulaşılması yeterli olacaktır.

Dördüncü adımda, fonksiyonel ve ters fonksiyonel karakteristiklere sahip nitelikler üzerinde *rdfp1* ve *rdfp2* kurallarının çalıştırılması sonucu çıkarsanan sınıf örnekleri arasındaki *sameAs* ilişkileri de hesaplanmaktadır.

$$rdfp1 : p \text{ type } FunctionalProperty \wedge u p v \wedge u p w \rightarrow v \text{ sameAs } w$$

$$rdfp2 : p \text{ type } InverseFunctionalProperty \wedge u p w \wedge v p w \rightarrow u \text{ sameAs } v$$

Sınıf örnekleri arası *sameAs* ilişkileri veritabanında, eşlenik örnekleri gösteren *individual1* ve *individual2* sahalarına sahip *sameAs* tablosunda saklanmaktadır.

4.3 Sorgulama Süreci

Uzantı tabanlı modele göre çıkarsama işlemi ontolojideki örnekler üzerinde değil uzantılar (örnek grupları) üzerinde yapılmaktadır. *RDFS* dili için yeterli olan bu durum, *OWL* dili için bazı sorunlar yaratmaktadır. Çünkü *OWL* dilinde, örnekler üzerinde tek tek çıkarsama yapılmasını gerektiren bazı kurallar vardır. Örneğin, *OWL* diline ait *owl:hasValue* yapıtaşı, belirli bir niteliğin değerinin varlığına bağlı olarak sınıf tanımı yapmayı sağlar. Başka bir deyişle, eğer *C*'nin *p* nitelik değeri *owl:hasValue* yapıtaşı ile *v* değerine kısıtlanmışsa ve bir sınıf örneğinin *p* değeri *v* ise, bu örneğin *C* sınıfının bir örneği olduğu çıkarsaması yapılabilir (bkz. Çizelge 4.4'te *rdfp14a* kuralı).

Bu tür bir çıkarsamanın yapılabilmesi için örnek grupları yerine örneklerin tek tek ele alınması gerekmektedir. Bu nedenle, bu çalışmada özelleşmiş bir sorgulama mekanizması kullanılmaktadır. Böylelikle, uzantı tabanlı bilgi modelinin, *RDFS*'in üzerine yeni yapılar ekleyerek *pD** dilini destekleyebilmesi sağlanır. Sorgulama mekanizması, bir tür geriye zincirleme algoritmasına dayanmaktadır.

Sorgulama mekanizması, herhangi bir sorgulama dilinde (*OWL-QL*, *SPARQL* vb.) verilmiş olan bir sorguyu, çıkarsama sürecinde tamamlanmamış bir takım çıkarsamaları gerçekleştirecek şekilde genişletmektedir. Sorgulama mekanizması, bir sorgu ağacının oluşturulması ve bu ağaca karşılık gelen bir *SQL* sorgusunun yaratılmasını sağlar. Bu sorgu ağacı, düğümlerden meydana gelir. Düğümler üç ana gruba ayrılırlar;

- *Kök Düğüm*: Her bir sorgu ağacının bir ve yalnız bir tane kök düğümü vardır.

Bu kök düğüm, temsil edilen sorguya ait koşullara ait kısıtları ve koşullar arası bağımlılıkları saklar. Bir koşula ait kısıtlar, o koşuldaki sabit değerleri içerir. Koşullar arası bağımlılıklarsa, koşullar arası ortak değişkenleri belirtir.

- *Varlık Düğümleri:* Her bir varlık düğümü, ilişkili olduğu varlıkla (v) ilgili bir sorgu bileşeni (Ω_v) içerir. Varlık düğümüne ait sorgu bileşenleri, *memory* olarak adlandırılan bir tablo içerirler. Varlık düğümleri içerdikleri sorgu bileşenlerine göre üç grupta incelenirler;

- *Sınıf Düğümleri:* Bir sınıf sorgu bileşenine ait *memory*, sınıf sorgu bileşeninin ilişkili olduğu sınıfa ait uzantı isimlerini saklar. Bu uzantılar, kavrama bağlanış şekline göre *belirtik* veya *çıkarsanmış uzantı* olabilirler. *Belirtik uzantılar* her kavram için bir tanedir ve *çıkarsanmış uzantılardan* farklı olarak başlangıçtan itibaren o kavrama bağlıdır. *Çıkarsanmış uzantılar* ise çıkarsama süreci ile sınıflara bağlanırlar. OWL dilinde iki tip sınıf bulunmaktadır; isimlendirilmiş sınıflar ve anonim sınıflar (Smith et al., 2004). *memory* içindeki uzantılar da isimlendirilmiş bir sınıfa veya anonim bir sınıfa ait olabilirler. Bu çalışmada, isimlendirilmiş bir sınıfa ait uzantılar *basit sınıf uzantıları*, anonim bir sınıfa ait uzantılar ise *karmaşık sınıf uzantıları* olarak adlandırılmıştır.

Eğer bir sınıf düğümüne ait *memory* içindeki tüm uzantılar basit sınıf uzantıları ise bu sınıf düğümü *basit sınıf düğümü* olarak adlandırılır. Bir sınıf düğümüne ait *memory* içinde herhangi bir uzantının *karmaşık sınıf uzantısı* olması durumunda, bu düğüm *karmaşık sınıf düğümü* olarak adlandırılır.

- *Nitelik Düğümleri:* Nitelik düğümlerine ait *memory*, nitelik sorgu bileşeninin ilişkili olduğu niteliğe ait uzantı isimlerini ve uzantılar ile nitelik arasındaki ilişkileri saklar. Bu uzantılar, kavrama bağlanış şekline göre *belirtik* veya *çıkarsanmış uzantı* olabilirler. *Belirtik uzantılar* her kavram için bir tanedir ve *çıkarsanmış uzantılardan* farklı olarak başlangıçtan itibaren o kavrama bağlıdır. *Çıkarsanmış uzantılar* ise çıkarsama süreci ile sınıflara bağlanırlar. Nitelik düğümlerinin sınıf

düğümlelerinden farkı, *memory* içindeki uzantıların her zaman isimlendirilmiş bir niteliğe ait olmasıdır, çünkü OWL dili ile anonim nitelik tanımlı yapılamamaktadır.

Diğer bir fark, *memory* içinde, uzantı isimlerine ek olarak, uzantı ile nitelik arasındaki ilişkinin de saklanmasıdır. OWL dilinde kullanılan *owl:inverseOf* yapıtaşından dolayı, bir nitelik, kendisi ile ilişkili uzantıya düz veya ters olarak bağlanmış olabilir. Bu nedenle, *memory* içinde, ilgili uzantının niteliğe bağlanış şeklini belirten ilişki ismi de tutulur (*hasPropertyExtension* düz bağlantıyı, *hasInversePropertyExtension* ters bağlantıyı belirtir).

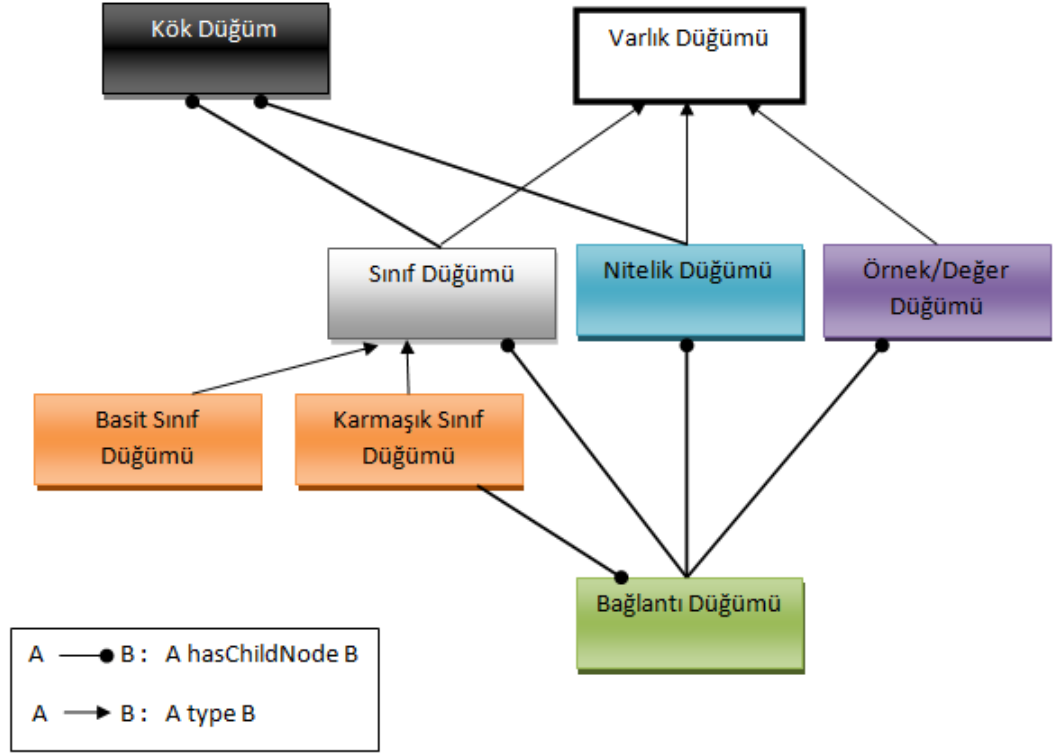
- *Örnek/Değer Düğümleri*: Her bir örnek/değer düğümü, bir örnek veya değer ismi saklar. Eğer düğüm bir örnek ismi saklıyorsa ve bu örnek *owl:sameAs* ilişkisi ile başka örneklerle bağlıysa, örneğin bağlandığı örnek isimleri de bu düğümde saklanır.
- *Bağlantı Düğümleri*: Bir sınıf düğümüne ait *memory* içerisinde yer alan her bir karmaşık sınıf uzantısı için, o sınıf düğümünün çocuklarına, bir *bağlantı* düğümü eklenir. Bağlantı düğümleri, varlık düğümlerinden farklı olarak *memory* içermezler. Bağlantı düğümleri, ilgili anonim sınıfın tanımında kullanılan OWL işlecine bağlı olarak yedi grupta incelenirler;
 - *kesişim bağlantı düğümü*: İlgili anonim sınıf, kesişim küme operatörü ($\sqcap$, *owl:intersectionOf*) ile tanımlanmışsa, bu bağlantı düğümü kullanılır. Bu düğümüne ait sorgu bileşeni ($\Omega_{intersectionOf}$) sembolü ile belirtilir.
 - *birleşim bağlantı düğümü*: İlgili anonim sınıf, birleşim küme operatörü ($\sqcup$, *owl:unionOf*) ile tanımlanmışsa, bu bağlantı düğümü kullanılır. Bu düğümüne ait sorgu bileşeni ($\Omega_{unionOf}$) sembolü ile belirtilir.
 - *tikel niceleme bağlantı düğümü*: İlgili anonim sınıf, tikel niceleme kısıtı ($\exists R.C$, *owl:someValuesFrom*) uygulanarak tanımlanmışsa, bu bağlantı düğümü kullanılır. Bu düğümüne ait sorgu bileşeni ($\Omega_{someValuesFrom}$) sembolü ile belirtilir.

- *tümel niceleme bağlantı düğümü*: İlgili anonim sınıf, tümel niceleme kısıtı ($\forall R.C$, *owl:allValuesFrom*) uygulanarak tanımlanmışsa, bu bağlantı düğümü kullanılır. Bu düğüme ait sorgu bileşeni ($\Omega_{allValuesFrom}$) sembolü ile belirtilir.
- *nitelik sayı kısıtı bağlantı düğümü*: İlgili anonim sınıf, bir nitelik üzerinde sayı kısıtı ($= nR$, *owl:cardinality*) uygulanarak tanımlanmışsa, bu bağlantı düğümü kullanılır. Bu düğüme ait sorgu bileşeni ($\Omega_{cardinality}$) sembolü ile belirtilir.
- *nitelik üst sayı kısıtı bağlantı düğümü*: İlgili anonim sınıf, bir nitelik üzerinde üst sayı kısıtı ($\leq nR$, *owl:maxCardinality*) uygulanarak tanımlanmışsa, bu bağlantı düğümü kullanılır. Bu düğüme ait sorgu bileşeni ($\Omega_{maxCardinality}$) sembolü ile belirtilir.
- *nitelik alt sayı kısıtı bağlantı düğümü*: İlgili anonim sınıf, bir nitelik üzerinde alt sayı kısıtı ($\geq nR$, *owl:minCardinality*) uygulanarak tanımlanmışsa, bu bağlantı düğümü kullanılır. Bu düğüme ait sorgu bileşeni ($\Omega_{minCardinality}$) sembolü ile belirtilir.
- *nitelik değer kısıtı bağlantı düğümü*: İlgili anonim sınıf, bir nitelik üzerinde değer kısıtı (*owl:hasValue*) uygulanarak tanımlanmışsa, bu bağlantı düğümü kullanılır. Bu düğüme ait sorgu bileşeni ($\Omega_{hasValue}$) sembolü ile belirtilir.

Bağlantı düğümlerinin çocuklarına, ilgili anonim sınıfın tanımında kullanılan OWL işleci ile işlenen varlıkların (sınıf, nitelik, sınıf örneği veya değer) her biri için varlığın tipine bağlı bir sınıf, nitelik veya örnek/değer düğümü eklenir.

Algoritmanın daha anlaşılır olması için, *Bölüm 4.3.1*'de sunulan sorgu ağacı bileşenleri üç katmanda incelenmektedir;

- *birinci katman* kök düğümü içerir.
- *ikinci katman* kök düğüme doğrudan bağlı çocuk düğümleri içerir.
- *üçüncü katman* ikinci katmandaki düğümlere ait çocuk düğümlerin tümünü içerir.



Şekil 4.4. Sorgu ağacı bileşenleri ve aralarındaki ilişkiler.

Yukarıda anlatılan sorgu ağacının oluşturulması ve karşılık gelen SQL sorgusunun oluşturulması için üç adım belirlenmiştir;

1. *Hazırlık Aşaması:* Hazırlık aşamasında asıl sorgu işlenerek sorgu ağacının ilk hali oluşturulur. Bu durumda sorguda kullanılan tüm kavramlar ayıklanarak karşılık gelen sorgu ağacı düğümleri oluşturulur.
2. *Geçiş Aşaması:* Geçiş aşaması sorgu ağacının ilk hali ile başlayarak ağacın her bir dalını genişleyebileceği en son sınıra kadar açarak büyütür. Ağaçta büyüyecek hiç bir dal kalmayınca bu aşama sona erer.
3. *Sonuç Aşaması:* Sonuç aşamasında son halini alan sorgu ağacı kullanılarak *SQL* sorgusu yazılır. Bu sorgu veritabanı üzerinde çalıştırılır ve *sameAs* tablosu (bkz. Bölüm 4.2.3) yardımıyla, sorgu yanıtlarındaki kaynaklarla eşdeğer olan kaynaklar tespit edilerek, sorgu yanıtına dahil edilir.

Takip eden alt başlıklarda her bir aşama detaylı olarak anlatılmaktadır.

4.3.1 Hazırlık Aşaması

Bu aşamada, birinci katmanda yer alan, kök düğüme doğrudan bağlı çocuk düğümler oluşturulur. Sorgudaki (q), her bir sorgu koşulu (q_c) için bir düğüm yaratılır ve kök düğümün çocuklarına eklenir. Yeni yaratılan bu düğümler bir sınıf veya bir nitelik düğümüdür. Bu düğümler, çıkarsama motorunun aşağıda gösterilen şekilde sorgulanmasıyla oluşturulurlar;

- *type* yüklemine sahip her bir sorgu koşulu ($type(?x, C)$) için, yeni bir sınıf düğümü yaratılır ve bu düğüme ait *memory* tablosu, aşağıda gösterilen sorgu yanıtları içerisindeki $?ε$ değerleri ile doldurulur;

– *hasClassExtension*($C, ?ε$)

- *type* yüklemi dışında herhangi bir yükleme sahip her bir sorgu koşulu ($?p(?x, C)$) için, yeni bir nitelik düğümü yaratılır ve bu düğüme ait *memory* tablosu, aşağıda gösterilen sorgu yanıtları içerisindeki $?ε$ değerleri ile doldurulur;

– *hasPropertyExtension*($?p, ?ε$) sorgusu sonucu elde edilen $?ε$ değerleri için *memory* tablosunda, uzantı ile nitelik arasındaki ilişki *hasPropertyExtension* olarak saklanır.

– *hasInversePropertyExtension*($?p, ?ε$) sorgusu sonucu elde edilen $?ε$ değerleri için *memory* tablosunda, uzantı ile nitelik arasındaki ilişki *hasInversePropertyExtension* olarak saklanır.

Bu aşamada yaratılacak yeni bir düğüm ($α$) ile kök düğüm arasındaki patikada, $α$ düğümü zaten varsa **Sonlandırma Yordamı** ($α$) devreye girer;

- **Sonlandırma Yordamı** ($α$): $α$ düğümüne ait *memory* tablosundaki tüm karmaşık sınıf uzantılarını silerek, $α$ düğümüne yeni bir çocuk düğüm eklenmesini engelle.

4.3.2 Geçiş Aşaması

Geçiş aşamasında, sorgu ağacı Algoritma 4.1'e uygun olarak, artık dalandırılabilir herhangi bir düğüm kalmayana kadar açılır. Ağacın açılması, her bir

Algorithm 4.1 D ğ m a ılım algoritması.

```

for every extension e in ememory
{
    if (e is complex class extension)
    {
        if (e in RestrictionComplexExtensions)
        {
            get type of restriction
            create related connector node
            get related property of restriction
            create a new property node
            get related value of restriction
            create a class or value node
        }
        else if (e in SetComplexExtensions)
        {
            get type of set operator
            create related connector node
            for every class in set
                create a class node
        }
    }
    else
        continue
}

```

yaprak bir * rnek/deęer d ę m *, *nitelik d ę m * veya *basit sınıf d ę m * olduęunda sonlanır.

Karmařık sınıf d ę mlerinin a ılması i in, B l m 4.2.3'te anlatılan karmařık sınıf uzantı tablolarına bařvurulur. Bu tabloların sahaları ařaęıda g sterilmektedir:

- *ComplexExtension(extensionUri, extensionType)*
- *RestrictionComplexExtensions(extensionUri, restrictionType, onProperty, classUri)*
- *SetComplexExtensions(extensionUri, setOperator, listOfSetElements)*

Karmařık sınıf d ę mlerine ait *ememory*'de yer alan her bir karmařık sınıf uzantısı i in, karmařık sınıf d ę m n n  ocuklarına bir *baęlantı d ę m * eklenir. Karmařık sınıf uzantısının tipine g re (*RestrictionComplexExtensions* tablosundaki *restrictionType* sahası veya *SetComplexExtensions* tablosundaki *setOperator* sahasına bakılarak karar verilir), *baęlantı d ę m *n n tipine karar verilir. Eęer

bağlantı düğümü kesişim veya birleşim bağlantı düğümü ise, *SetComplexExtensions* tablosundaki *listOfSetElements* sahasına ait her bir eleman için *bağlantı düğümünün* çocuklarına yeni bir *sınıf düğümü* eklenir. Eğer *bağlantı düğümü* diğer *bağlantı düğümü* çeşitlerinden biri ise, *RestrictionComplexExtensions* tablosundaki *onProperty* ve *classUri* sahalarında belirtilen nitelik ve sınıf için *bağlantı düğümünün* çocuklarına bir *nitelik* ve *sınıf düğümü* eklenir. Yeni eklenen çocuk düğümlerin içinde, *karmaşık sınıf düğümleri* olması durumunda, *karmaşık sınıf düğümlerinin* açılması işlemine özyinelemeli olarak devam edilir. Bu özyinelemeli algoritma, her bir çocuk düğümü bir *örnek/değer düğümü*, *nitelik düğümü* veya *basit sınıf düğümü* olduğunda sonlanır.

4.3.3 Sonuç Aşaması

Bu aşamada oluşturulan sorgu ağacı *SQL* sorgusuna dönüştürülür. İkinci katmandaki düğümlerin hesaplanması sonucu oluşan sonuçlar, *kök düğümde* belirtilen koşul kısıtlarına ve koşullar arası bağımlılıklara göre birleştirilir. İkinci katmanda yer alan *sınıf* ve *nitelik düğümleri* aşağıda gösterilen şekilde hesaplanmaktadır:

- Eğer *sınıf düğümü* (Ω_C) çocuğu olmayan, *basit sınıf düğümü* ise, $\gamma(\Omega_C)$ fonksiyonu Ω_C düğümüne ait *memory* içindeki basit sınıf uzantılarını hesaplar ve sonuçları birleştirir. Her bir basit sınıf uzantısı (ε_{basic}) aşağıdaki *SQL* sorgusu ($Q_{\varepsilon_{basic}}$) ile hesaplanır:

$$- Q_{\varepsilon_{basic}} : SELECT resource FROM contains WHERE extension = \varepsilon_{basic}$$

- $\delta(\Omega_p)$ fonksiyonu, bir niteliğe (p) ait nitelik düğümü (Ω_p) içindeki her bir nitelik uzantısını hesaplar ve sonuçları birleştirir. Niteliğe düz bağlı bir her bir nitelik uzantısı $Q_{\varepsilon_{p+}}$ sorgusu ile, niteliğe ters bağlı her bir nitelik uzantısı ise $Q_{\varepsilon_{p-}}$ sorgusu ile hesaplanır. $Q_{\varepsilon_{p+}}$ ve $Q_{\varepsilon_{p-}}$ *SQL* sorguları aşağıda gösterilmektedir:

$$\begin{aligned} - Q_{\varepsilon_{p+}} : & SELECT subject, object \\ & FROM statements WHERE predicate = p \\ - Q_{\varepsilon_{p-}} : & SELECT subject AS object, object AS subject \\ & FROM statements WHERE predicate = p \end{aligned}$$

$hasValue(C, p, v) \implies$	$SELECT T.subject$ $FROM \delta(\Omega_p) AS T$ $WHERE T.object = v$
$someValuesFrom(C_X, p, C_Y) \implies$	$SELECT T_2.subject$ $FROM \gamma(\Omega_{C_Y}) AS T_1, \delta(\Omega_p) AS T_2$ $WHERE T_1.resource = T_2.object$
$allValuesFrom(C_X, p, C_Y) \implies$	$SELECT T_2.subject$ $FROM \gamma(\Omega_{C_Y}) AS T_1, \delta(\Omega_p) AS T_2$ $GROUP BY T_2.subject$ $HAVING object IN T_1$
$intersectionOf(C, C_1, \dots, C_n) \implies$	$((\gamma(\Omega_{C_1}) INTERSECT \gamma(\Omega_{C_2})) \dots) INTERSECT \gamma(\Omega_{C_n})$
$unionOf(C, C_1, \dots, C_n) \implies$	$((\gamma(\Omega_{C_1}) UNION \gamma(\Omega_{C_2})) \dots) UNION \gamma(\Omega_{C_n})$

Çizelge 4.11. Bağlantı düğümlerine karşılık gelen *SQL* sorguları.

- Sınıf düğümü çocukları olan karmaşık bir sınıf düğümü ise, her biri bir bağlantı düğümü olan çocuklar hesaplanır. Bunların sonuçları, *memory* içindeki basit sınıf uzantılarının $Q_{\varepsilon_{basic}}$ sorgusu ile hesaplanması sonucunda oluşan listelerle birleştirilerek nihai sonuç bulunur. Her bir bağlantı düğümü Çizelge 4.11’de karşılık gelen *SQL* sorgusu S_1 ile hesaplanır. S_1 sorgusunun işlediği değerler, basit sınıf düğümleri veya nitelik düğümleri olduğunda $Q_{\varepsilon_{basic}}$, $Q_{\varepsilon_{p+}}$ ve $Q_{\varepsilon_{p-}}$ sorguları yardımıyla düğümler hesaplanır ve işlem sonlanır. Bu değerlerin karmaşık sınıf düğümleri olması durumunda, bu düğümün çocuğu olan yeni bir bağlantı düğümünün hesaplanması gerekmektedir. Bunun için Çizelge 4.11’de, yeni bağlantı düğümüne karşılık gelen *SQL* sorgusu S_2 , oluşturulur ve S_1 ’in içine gömülür. Sorguların içiçe yuvalandırılması özyinelemeli olarak devam eder ve ağacın en alt düğümlerinde (yapraklarında) sonlanır. Her bir yaprak bir *örnek/değer düğümü*, *nitelik düğümü* veya *basit sınıf düğümü*dür.

5 Örnek Uygulama

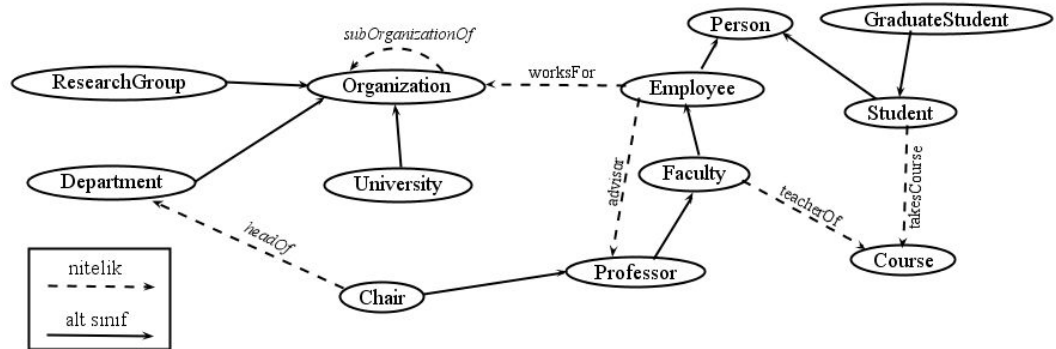
Bu bölümde, tez kapsamında sunulmuş çıkarsama yaklaşımının, örnek bir ontoloji üzerinde uygulanması anlatılmaktadır. Örnek ontoloji olarak, LUBM karşılaştırmalı değerlendirme uygulamasına ait ontoloji şemasının bir altkümesi kullanılacaktır. Sorgulama aşamasında, basit bir örnek sorguya (Bölüm 5.5.1) ek olarak LUBM uygulamasına ait 11 (Bölüm 5.5.2), 12 (Bölüm 5.5.3) ve 13 (Bölüm 5.5.4) numaralı sorgular örnek sorgu olarak seçilecektir. 11, 12 ve 13 numaralı sorguların seçilmesinin nedeni OWL çıkarsama desteği gerektiren sorgular olmasıdır.

Şekil 5.1’de, elipsler sınıfları, oklar ise sınıflar arası ilişkileri göstermektedir. Düz oklar sınıflar arası alt sınıf ilişkilerini, kesikli oklar ise sınıflar arası diğer *Objecttype* ilişkileri göstermektedir. Şemada bulunan niteliklerle ilgili özellikler şunlardır:

- *subOrganizationOf* geçişken bir niteliktir.
- *hasAlumnus* niteliği, *degreeFrom* niteliğinin ters niteliğidir.

Çizelge 5.1 ve Çizelge 5.2, örnek ontolojiye ait örnek verileri göstermektedir. Çizelge 5.1, sınıf örneklerini gösterir. Her bir sınıf örneği, ontolojide bir "type" ilişkisi yardımı ile gösterilmektedir. Örneğin, çizelgedeki *rg01* örneği ile *ResearchGroup* sınıf "type" niteliği ile aşağıdaki şekilde ilişkilendirilmiştir;

- *rg01 rdf : type ResearchGroup*



Şekil 5.1. Örnek ontoloji şeması.

<i>Sınıf Adı</i>	<i>Örnek Sayısı</i>	<i>Örnek isimleri</i>
<i>ResearchGroup</i>	2	<i>rg01, rg02</i>
<i>Department</i>	3	<i>dep01, dep02, dep03</i>
<i>University</i>	5	<i>univ01, ..., univ05</i>
<i>Professor</i>	10	<i>prof01, ..., prof10</i>

Çizelge 5.1. Örnek ontolojideki sınıf örnekleri.

Örnek ontoloji ayrıştırılıp, ontolojideki tüm üçlüler veritabanındaki *statements* tablosuna aktarıldıktan sonra, sırası ile aşağıdaki adımlar gerçekleştirilir;

- *Uzantı Tabanlı Bilgi Modeline Dönüştürme (Bölüm 5.1)*
- *Üçlüleri Filtreleme (Bölüm 5.2)*
- *İleriye Zincirleme (Bölüm 5.3)*
- *Veritabanında Uzantıları İşleme ve Genişletme (Bölüm 5.4)*
- *Sorgulama (Bölüm 5.5)*

Takip eden alt başlıklarda, bu adımlar örnek ontoloji üzerinde sırası ile gerçekleştirilmiş ve sonuçlar sunulmuştur.

5.1 Uzantı Tabanlı Bilgi Modeline Dönüştürme

Ontoloji ayrıştırıldıktan ve ontoloji üçlüleri veritabanındaki *statements* tablosuna aktarıldıktan sonra, bu üçlüler üzerinde uzantı tabanlı model doğrultusunda gerekli dönüşümler gerçekleştirilir. Dönüşümün ilk adımı, ontolojideki her bir sınıf için bir uzantı yaratılması ve bu uzantı ile sınıf arasında aşağıdaki örnekte görüldüğü gibi *hasExplicitClassExtension* ilişkisi kurulmasıdır;

- *Professor hasExplicitClassExtension ExtensionProfessor*

Sınıf ve sınıf örneği uzantısı arasındaki ilişki *statements* tablosuna eklenir. Çizelge 5.1’de gösterilen 4 sınıf için, *statements* tablosuna 4 adet yeni üçlü eklenmektedir.

Benzer şekilde, tüm nitelikler için de bir nitelik özne uzantısı, nitelik nesne uzantısı ve nitelik örneği uzantısı yaratılır ve her bir uzantı ile nitelik arasında sırası

<i>Özne</i>	<i>Yüklem</i>	<i>Nesne</i>
<i>rg01</i>	<i>subOrganizationOf</i>	<i>dep01</i>
<i>rg02</i>	<i>subOrganizationOf</i>	<i>dep02</i>
<i>dep01</i>	<i>subOrganizationOf</i>	<i>univ01</i>
<i>dep02</i>	<i>subOrganizationOf</i>	<i>univ01</i>
<i>dep03</i>	<i>subOrganizationOf</i>	<i>univ01</i>
<i>prof01</i>	<i>worksFor</i>	<i>univ01</i>
<i>prof02</i>	<i>worksFor</i>	<i>univ01</i>
<i>prof03</i>	<i>worksFor</i>	<i>univ01</i>
<i>prof04</i>	<i>worksFor</i>	<i>univ01</i>
<i>prof05</i>	<i>worksFor</i>	<i>univ01</i>
<i>prof06</i>	<i>worksFor</i>	<i>univ01</i>
<i>prof07</i>	<i>worksFor</i>	<i>univ01</i>
<i>prof08</i>	<i>worksFor</i>	<i>univ02</i>
<i>prof09</i>	<i>worksFor</i>	<i>univ03</i>
<i>prof10</i>	<i>worksFor</i>	<i>univ03</i>
<i>prof01</i>	<i>degreeFrom</i>	<i>univ05</i>
<i>prof02</i>	<i>degreeFrom</i>	<i>univ04</i>
<i>prof03</i>	<i>degreeFrom</i>	<i>univ03</i>
<i>prof04</i>	<i>degreeFrom</i>	<i>univ02</i>
<i>prof05</i>	<i>degreeFrom</i>	<i>univ01</i>
<i>prof06</i>	<i>degreeFrom</i>	<i>univ05</i>
<i>prof07</i>	<i>degreeFrom</i>	<i>univ04</i>
<i>prof08</i>	<i>degreeFrom</i>	<i>univ03</i>
<i>prof09</i>	<i>degreeFrom</i>	<i>univ02</i>
<i>prof10</i>	<i>degreeFrom</i>	<i>univ01</i>
<i>prof03</i>	<i>headOf</i>	<i>dep01</i>
<i>prof05</i>	<i>headOf</i>	<i>dep02</i>
<i>prof07</i>	<i>headOf</i>	<i>dep03</i>

Çizelge 5.2. Örnek ontolojideki nitelik örnekleri.

<i>Özne</i>	<i>Yüklem</i>	<i>Nesne</i>
<i>ExtensionResearchGroup</i>	<i>contains</i>	<i>rg01</i>
<i>ExtensionResearchGroup</i>	<i>contains</i>	<i>rg02</i>

Çizelge 5.3. *classExtensionResearchGroup* uzantısının elemanlarını belirten üçlüler.

ile *hasExplicitSubjectExtension*, *hasExplicitObjectExtension* ve *hasExplicitPropertyExtension* ilişkileri kurulur. Aşağıda *headOf* niteliği için tanımlanmış uzantılar gösterilmektedir;

- *headOf hasExplicitSubjectExtension SubjectExtensionheadOf*
- *headOf hasExplicitObjectExtension ObjectExtensionheadOf*
- *headOf hasExplicitPropertyExtension ExtensionheadOf*

Nitelik ve ilgili nitelik uzantıları arasındaki ilişki *statements* tablosuna eklenir. Çizelge 5.1’de gösterilen 4 niteliğin her biri için üç yeni ilişki yaratılır ve *statements* tablosuna toplamda 12 adet yeni üçlü eklenir.

Dönüşümün ikinci adımı, uzantıları ve uzantı elemanlarını ilişkilendirmek ve bu ilişkileri veritabanındaki *contains* tablosuna aktarmaktır. Sınıf örneği uzantıları ile ilgili uzantı elemanlarını eşleştirmek için, "type" ilişkileri kullanılır. Örneğin, *Professor* sınıfına ait sınıf örneği uzantısı ile *Professor* sınıfının her bir örneği için *contains* tablosuna yeni bir bir saha eklenir;

- *ExtensionProfessor contains prof01*

Bu işlemin sonunda *contains* tablosuna, *Professor* sınıfı için 10 adet üçlü eklenmektedir. Diğer bir deyişle, *contains* tablosuna, her sınıfın örnek sayısı kadar saha eklenir. Çizelge 5.3, Çizelge 5.4, Çizelge 5.5 ve Çizelge 5.6’da sınıf uzantılarının içerikleri açık olarak verilmiştir. Sınıf örneklerinin, uzantı tabanlı modelle dönüştürülmesi sonunda, *contains* tablosuna eklenen toplam üçlü sayısı Çizelge 5.1’de belirtilen toplam örnek sayısına eşittir ($2+3+5+10=20$).

Nitelikler örneği uzantısı temsili bir uzantıdır. Diğer bir deyişle, bu uzantı ve elemanları arasında bağlantı kurulmaz. Nitelik öznesi ve nitelik nesnesi uzantıları ise elemanları ile *contains* ilişkisi ile bağlanır, diğer bir deyişle, her bir nitelik örneği için *contains* tablosuna 2 adet yeni üçlü eklenir. Örneğin, *prof03 headOf dep01*

Özne	Yüklem	Nesne
<i>ExtensionDepartment</i>	<i>contains</i>	<i>dep01</i>
<i>ExtensionDepartment</i>	<i>contains</i>	<i>dep02</i>
<i>ExtensionDepartment</i>	<i>contains</i>	<i>dep03</i>

Çizelge 5.4. *classExtensionDepartment* uzantısının elemanlarını belirten üçlüler.

Özne	Yüklem	Nesne
<i>ExtensionUniversity</i>	<i>contains</i>	<i>univ01</i>
<i>ExtensionUniversity</i>	<i>contains</i>	<i>univ02</i>
<i>ExtensionUniversity</i>	<i>contains</i>	<i>univ03</i>
<i>ExtensionUniversity</i>	<i>contains</i>	<i>univ04</i>
<i>ExtensionUniversity</i>	<i>contains</i>	<i>univ05</i>

Çizelge 5.5. *classExtensionUniversity* uzantısının elemanlarını belirten üçlüler.

Özne	Yüklem	Nesne
<i>ExtensionProfessor</i>	<i>contains</i>	<i>prof01</i>
<i>ExtensionProfessor</i>	<i>contains</i>	<i>prof02</i>
<i>ExtensionProfessor</i>	<i>contains</i>	<i>prof03</i>
<i>ExtensionProfessor</i>	<i>contains</i>	<i>prof04</i>
<i>ExtensionProfessor</i>	<i>contains</i>	<i>prof05</i>
<i>ExtensionProfessor</i>	<i>contains</i>	<i>prof06</i>
<i>ExtensionProfessor</i>	<i>contains</i>	<i>prof07</i>
<i>ExtensionProfessor</i>	<i>contains</i>	<i>prof08</i>
<i>ExtensionProfessor</i>	<i>contains</i>	<i>prof09</i>
<i>ExtensionProfessor</i>	<i>contains</i>	<i>prof10</i>

Çizelge 5.6. *classExtensionProfessor* uzantısının elemanlarını belirten üçlüler.

<i>Özne</i>	<i>Yüklem</i>	<i>Nesne</i>
<i>SubjectExtensionSubOrganizationOf</i>	<i>contains</i>	<i>rg01</i>
<i>SubjectExtensionSubOrganizationOf</i>	<i>contains</i>	<i>rg01</i>
<i>SubjectExtensionSubOrganizationOf</i>	<i>contains</i>	<i>dep01</i>
<i>SubjectExtensionSubOrganizationOf</i>	<i>contains</i>	<i>dep02</i>
<i>SubjectExtensionSubOrganizationOf</i>	<i>contains</i>	<i>dep03</i>

Çizelge 5.7. *subjectExtensionSubOrganizationOf* uzantısının elemanlarını belirten üçlüler.

<i>Özne</i>	<i>Yüklem</i>	<i>Nesne</i>
<i>ObjectExtensionSubOrganizationOf</i>	<i>contains</i>	<i>dep01</i>
<i>ObjectExtensionSubOrganizationOf</i>	<i>contains</i>	<i>dep02</i>
<i>ObjectExtensionSubOrganizationOf</i>	<i>contains</i>	<i>univ01</i>

Çizelge 5.8. *objectExtensionSubOrganizationOf* uzantısının elemanlarını belirten üçlüler.

nitelik örneği için, *headOf* niteliğinin nitelik öznesi ve nitelik nesnesi uzantılarına aşağıdaki elemanlar eklenir;

- *SubjectExtensionheadOf contains prof03*
- *ObjectExtensionheadOf contains dep01*

Sınıf örneklerinin, uzantı tabanlı modele dönüştürülmesi sonunda, Çizelge 5.2’de gösterilen 28 üçlünün her biri için 2 adet yeni *contains* ilişkisi yaratılır ve *contains* tablosuna toplamda 56 adet yeni üçlü eklenir. Çizelge 5.7, Çizelge 5.9, Çizelge 5.11 ve Çizelge 5.13’te nitelik özne uzantılarının içerikleri; Çizelge 5.8, Çizelge 5.10, Çizelge 5.12 ve Çizelge 5.14’te ise nitelik nesne uzantılarının içerikleri verilmiştir.

Bu örnekte, dönüşüm sonunda veritabanındaki *statements* tablosuna toplamda 16 (4+12), *contains* tablosuna ise toplamda 76 (20+56) yeni üçlü eklenir ve veritabanına toplamda 92 (16+76) yeni üçlü eklenmiş olur.

Özne	Yüklem	Nesne
<i>SubjectExtensionWorksFor</i>	<i>contains</i>	<i>prof01</i>
<i>SubjectExtensionWorksFor</i>	<i>contains</i>	<i>prof02</i>
<i>SubjectExtensionWorksFor</i>	<i>contains</i>	<i>prof03</i>
<i>SubjectExtensionWorksFor</i>	<i>contains</i>	<i>prof04</i>
<i>SubjectExtensionWorksFor</i>	<i>contains</i>	<i>prof05</i>
<i>SubjectExtensionWorksFor</i>	<i>contains</i>	<i>prof06</i>
<i>SubjectExtensionWorksFor</i>	<i>contains</i>	<i>prof07</i>
<i>SubjectExtensionWorksFor</i>	<i>contains</i>	<i>prof08</i>
<i>SubjectExtensionWorksFor</i>	<i>contains</i>	<i>prof09</i>
<i>SubjectExtensionWorksFor</i>	<i>contains</i>	<i>prof10</i>

Çizelge 5.9. *subjectExtensionworksFor* uzantısının elemanlarını belirten üçlüler.

Özne	Yüklem	Nesne
<i>ObjectExtensionWorksFor</i>	<i>contains</i>	<i>univ01</i>
<i>ObjectExtensionWorksFor</i>	<i>contains</i>	<i>univ02</i>
<i>ObjectExtensionWorksFor</i>	<i>contains</i>	<i>univ03</i>

Çizelge 5.10. *objectExtensionworksFor* uzantısının elemanlarını belirten üçlüler.

Özne	Yüklem	Nesne
<i>SubjectExtensionDegreeFrom</i>	<i>contains</i>	<i>prof01</i>
<i>SubjectExtensionDegreeFrom</i>	<i>contains</i>	<i>prof02</i>
<i>SubjectExtensionDegreeFrom</i>	<i>contains</i>	<i>prof03</i>
<i>SubjectExtensionDegreeFrom</i>	<i>contains</i>	<i>prof04</i>
<i>SubjectExtensionDegreeFrom</i>	<i>contains</i>	<i>prof05</i>
<i>SubjectExtensionDegreeFrom</i>	<i>contains</i>	<i>prof06</i>
<i>SubjectExtensionDegreeFrom</i>	<i>contains</i>	<i>prof07</i>
<i>SubjectExtensionDegreeFrom</i>	<i>contains</i>	<i>prof08</i>
<i>SubjectExtensionDegreeFrom</i>	<i>contains</i>	<i>prof09</i>
<i>SubjectExtensionDegreeFrom</i>	<i>contains</i>	<i>prof10</i>

Çizelge 5.11. *subjectExtensiondegreeFrom* uzantısının elemanlarını belirten üçlüler.

<i>Özne</i>	<i>Yüklem</i>	<i>Nesne</i>
<i>ObjectExtensionDegreeFrom</i>	<i>contains</i>	<i>univ01</i>
<i>ObjectExtensionDegreeFrom</i>	<i>contains</i>	<i>univ02</i>
<i>ObjectExtensionDegreeFrom</i>	<i>contains</i>	<i>univ03</i>
<i>ObjectExtensionDegreeFrom</i>	<i>contains</i>	<i>univ04</i>
<i>ObjectExtensionDegreeFrom</i>	<i>contains</i>	<i>univ05</i>

Çizelge 5.12. *objectExtensiondegreeFrom* uzantısının elemanlarını belirten üçlüler.

<i>Özne</i>	<i>Yüklem</i>	<i>Nesne</i>
<i>SubjectExtensionHeadOf</i>	<i>contains</i>	<i>prof03</i>
<i>SubjectExtensionHeadOf</i>	<i>contains</i>	<i>prof05</i>
<i>SubjectExtensionHeadOf</i>	<i>contains</i>	<i>prof07</i>

Çizelge 5.13. *subjectExtensionheadOf* uzantısının elemanlarını belirten üçlüler.

<i>Özne</i>	<i>Yüklem</i>	<i>Nesne</i>
<i>ObjectExtensionHeadOf</i>	<i>contains</i>	<i>dep01</i>
<i>ObjectExtensionHeadOf</i>	<i>contains</i>	<i>dep02</i>
<i>ObjectExtensionHeadOf</i>	<i>contains</i>	<i>dep03</i>

Çizelge 5.14. *objectExtensionheadOf* uzantısının elemanlarını belirten üçlüler.

<i>Yaklaşım uygulanmadan önce</i>		<i>Yaklaşım uygulandıktan sonra</i>	
<i>statements</i>	η_S	<i>statements</i>	η_S
	20		20
	28		28
			<u>16</u>
<i>contains</i>	0	<i>contains</i>	76
<i>çıkarsamaya katılan=</i>	$\eta_S + 48$	<i>çıkarsamaya katılan=</i>	$\eta_S + 16$

Çizelge 5.15. Yaklaşım uygulandıktan sonra çıkarsamaya katılan üçlü sayısı.

5.2 Üçlüleri Filtreleme

Filtreleme işleminin amacı, veritabanındaki üçlüler içerisinde çıkarsama sürecine dahil edilecek üçlüleri seçip, çıkarsama motoruna yüklemektir. Böylece ileriye zincirleme algoritmasının üzerinde çalışacağı veri kümesi elde edilmiş olur. Filtreleme işlemi sırasında seçilen üçlüler iki grupta incelenir;

- ontoloji şemasına ait üçlüler
- dönüşüm sonrasında *statements* tablosuna eklenen üçlüler

Bu aşamada, çıkarsamaya katılan her üçlü sayısının azalması, ileriye zincirleme algoritmasının zaman ve depolama başarımını arttırmaktadır. Uzantı tabanlı yaklaşımın uygulanması sonucunda, örnek ontolojide, çıkarsamaya katılan üçlü sayısındaki azalma Çizelge 5.15'te gösterilmektedir. η_S ontoloji şemasındaki üçlü sayısını belirtmektedir. Yaklaşım uygulanmadan önce, *statements* tablosundaki tüm üçlüler çıkarsamaya katılmaktadır ($\eta_S + 48$). Yaklaşım uygulandıktan sonra, şema üçlüleri ve altı çizili olarak belirtilen 16 üçlü çıkarsamaya katılır ($\eta_S + 16$). Bu 16 üçlü, dönüşüm sonrasında *statements* tablosuna eklenen üçlüleri, diğer bir deyişle, sınıf ve nitelikler ile bu sınıf ve niteliklere ait uzantıları bağlayan üçlüleri belirtir.

Bu bölümde kullanılan örnek ontolojiye ait şema verisinin, örnek verisine oranı oldukça yüksektir. Gerçek uygulamalarda kullanılan ontolojilerde bu oran çok daha azdır. Yöntemin başarımı, bu oranla ters orantılıdır. Kısacası, gerçek uygulamalarda kullanılan ontolojilerde yöntemin başarımı büyük ölçüde artış gösterecektir. Yöntemin başarımı ile ilgili tahminlemeler Bölüm 6.2'de, değerlendirme uygulamalarıyla elde edilen sonuçlar ise Bölüm 7.5'te verilmektedir.

5.3 İleriye Zincirleme

İleri zincirleme algoritmasının çalıştırılması sonucunda uzantılar hakkında çıkarsama işlemi tamamlanmaktadır. Çıkarsamanın tek tek örnekler üzerinde yapılması yerine uzantılar üzerinde yapılması hem çıkarsamaya katılan hem de çıkarsama sonucu oluşan üçlü sayısını azaltmaktadır. Bu örnekte, $\subseteq$ *subClassOf*, $\in$ *subPropertyOf* niteliğini göstermek üzere, uzantı tabanlı model kullanılmadan, ileriye zincirleme uygulandığında çıkarsanan üçlü sayısı Çizelge 5.16’da hesaplanmaktadır.

<i>İlişki</i>	<i>İlişki sonucu çıkarsanan üçlü</i>
<i>ResearchGroup</i> $\subseteq$ <i>Organization</i>	2 adet "type" ilişkisi
<i>Department</i> $\subseteq$ <i>University</i> $\subseteq$ <i>Organization</i>	6 adet "type" ilişkisi
<i>University</i> $\subseteq$ <i>Organization</i>	5 adet "type" ilişkisi
<i>Professor</i> $\subseteq$ <i>Faculty</i> $\subseteq$ <i>Employee</i> $\subseteq$ <i>Person</i>	30 adet "type" ilişkisi
<i>worksFor</i> $\in$ <i>memberOf</i>	10 adet nitelik örneği
<i>headOf</i> $\in$ <i>worksFor</i> $\in$ <i>memberOf</i>	6 adet nitelik örneği
<i>subOrganizationOf</i> type <i>TransitiveProperty</i>	2 adet nitelik örneği
<i>degreeFrom</i> inverseOf <i>hasAlumnus</i>	10 adet nitelik örneği
Toplam	71

Çizelge 5.16. Yaklaşım uygulanmadan önce çıkarsanan üçlüler.

Çizelge 5.17 ise yaklaşım uygulandıktan sonra çıkarsanacak üçlü sayılarını göstermektedir.

<i>İlişki</i>	<i>İlişki sonucu çıkarsanan üçlü</i>
$ResearchGroup \subseteq Organization$	1 adet "hasInferredClassExtension" ilişkisi
$Department \subseteq University \subseteq Organization$	2 adet "hasInferredClassExtension" ilişkisi
$University \subseteq Organization$	1 adet "hasInferredClassExtension" ilişkisi
$Professor \subseteq Faculty \subseteq Employee \subseteq Person$	3 adet "hasInferredClassExtension" ilişkisi
$worksFor \in memberOf$	1 adet "hasInferredPropertyExtension" ilişkisi
$headOf \in worksFor \in memberOf$	2 adet "hasInferredPropertyExtension" ilişkisi
$subOrganizationOf$ type <i>TransitiveProperty</i>	0
$degreeFrom$ inverseOf <i>hasAlumnus</i>	1 adet nitelik örneği
Toplam	11

Çizelge 5.17. Yaklaşım uygulandıktan sonra çıkarsanan üçlüler.

Yaklaşım uygulanmadan önce 71 adet yeni üçlü çıkarsanırken, yaklaşım uygulandıktan sonra 11 adet yeni üçlü çıkarsanmaktadır. Bu kadar küçük boyuttaki örnekssel veri üzerinde dahi çıkarsanan üçlü sayısındaki düşüş görülebilmektedir. Örnekssel verinin boyutundaki artışla birlikte, çıkarsanan üçlü sayısındaki düşüş daha fazla olacaktır. Çıkarsanan üçlü sayısındaki azalma, ileriye zincirleme sürecindeki zaman ve depolama kazancını etkileyen önemli bir faktördür.

5.4 Veritabanında Uzantıları İşleme ve Genişletme

Çıkarsamanın tamamlanmasının ardından, veritabanında uzantıların işlenmesi ve genişletilmesi aşaması gelir. Bunun için öncelikle Çizelge 5.18’de gösterilen anonim sınıfların uzantılarına ait verilerin, bellekteki *RestrictionComplexExtensions* (Çizelge 5.19), *SetComplexExtensions* (Çizelge 5.20) ve *ComplexExtensions* (Çizelge 5.21) veri yapılarına aktarılması gerekmektedir.

Bu aşamada, ikinci adım karakteristiği olan nitelikleri ve bu niteliklere ait karakteristikleri saklayan *PropertiesWithCharacteristics* veri yapısının tanımlanmasıdır. Bu örneğe ait, *PropertiesWithCharacteristics* veri yapısı, Çizelge 5.22’de gösterilmektedir.

Örnekte, *owl:hasValue* nitelik kısıtı ile tanımlanmış bir sınıf olmadığından, *owl:hasValue* sonucu çıkarsanan nitelik örneklerinin ilgili nitelik örneği uzantısına

<i>Uzantı Adı</i>	<i>İlgili olduğu isimlendirilmiş sınıf</i>	<i>Bağlı olduğu anonim sınıf</i>
ε_{α_1}	<i>Chair</i>	$\exists \text{ headOf } \text{Department}$
ε_{α_2}	<i>Employee</i>	$\exists \text{ worksFor } \text{Organization}$
ε_{α_3}	<i>GraduateStudent</i>	$\exists \text{ takesCourse } \text{GraduateCourse}$
ε_{α_4}	<i>Student</i>	$\exists \text{ takesCourse } \text{Course}$
ε_{α_5}	<i>Chair</i>	$\text{Person} \cap \exists \text{ headOf } \text{Department}$
ε_{α_6}	<i>Employee</i>	$\text{Person} \cap \exists \text{ worksFor } \text{Organization}$
ε_{α_7}	<i>GraduateStudent</i>	$\text{Person} \cap \exists \text{ takesCourse } \text{GraduateCourse}$
ε_{α_8}	<i>Student</i>	$\text{Person} \cap \exists \text{ takesCourse } \text{Course}$

Çizelge 5.18. Anonim sınıflar ve tanımları.

<i>extensionUri</i>	<i>restrictionType</i>	<i>onProperty</i>	<i>classOrValueUri</i>
ε_{α_1}	someValuesFrom	headOf	Department
ε_{α_2}	someValuesFrom	worksFor	Organization
ε_{α_3}	someValuesFrom	takesCourse	GraduateCourse
ε_{α_4}	someValuesFrom	takesCourse	Course

Çizelge 5.19. *RestrictionComplexExtensions* veri yapısı.

<i>extensionUri</i>	<i>setOperator</i>	<i>listOfSetElements</i>
ε_{α_5}	intersection	Person, α_5
ε_{α_6}	intersection	Person, α_6
ε_{α_7}	intersection	Person, α_7
ε_{α_8}	intersection	Person, α_8

Çizelge 5.20. *SetComplexExtensions* veri yapısı.

<i>extensionUri</i>	<i>extensionType</i>
ε_{α_1}	set
ε_{α_2}	set
ε_{α_3}	set
ε_{α_4}	set
ε_{α_5}	restriction
ε_{α_6}	restriction
ε_{α_7}	restriction
ε_{α_8}	restriction

Çizelge 5.21. *ComplexExtensions* veri yapısı.

<i>propertyUri</i>	<i>transitive</i>	<i>symmetric</i>	<i>functional</i>	<i>inv-functional</i>
subOrganizationOf	+	-	-	-

Çizelge 5.22. *PropertiesWithCharacteristics* veri yapısı.

eklenmesi adımı sonunda hiçbir değişiklik olmamaktadır.

Geçişken ve simetrik niteliklere ait uzantı kümelerinin bu karakteristikler doğrultusunda genişletilmesidir. Çizelge 5.22’de gösterilen nitelik karakteristiklerinin ilgili uzantılara uygulanması halinde aşağıdaki üçlüler çıkarsanmaktadır:

- $rg01 \text{ subOrganizationOf } univ01$
- $rg02 \text{ subOrganizationOf } univ01$

Her bir üçlü *statements* tablosuna, üçlünün öznesi üçlü yüklemine ait nitelik özne uzantısına, üçlünün nesnesi ise üçlü yüklemine ait nitelik nesne uzantısına eklenir. Nitelik özne ve nitelik nesne uzantılarına ekleme yapmak için, *contains* tablosuna Çizelge 5.23’te görülen sahaların eklenmesi gerekmektedir;

<i>extension_name</i>	<i>resource</i>
<i>SubjectExtensionssubOrganizationOf</i>	<i>rg01</i>
<i>SubjectExtensionssubOrganizationOf</i>	<i>rg02</i>
<i>ObjectExtensionssubOrganizationOf</i>	<i>univ01</i>

Çizelge 5.23. Geçişkenlik kuralının işletilmesi sonucu bulunan nitelik örnekleri.

5.5 Sorgulama

Veritabanında uzantıların işlenmesi ve genişletilmesinden sonra çevrimdışı çıkarsama süreci tamamlanmaktadır. Geriye kalan tüm çıkarsamalar, sorgulama sürecinde çevrimiçi olarak gerçekleştirilmektedir. Sorgulama algoritması, aşağıda listelenen üç kaynaktan gelen bilgileri birleştirir ve bu bilgiler üzerinde geriye zincirleme algoritmasını uygulayarak, sorgu yanıtlarını döndürür;

1. Veritabanındaki üçlüler
2. İleriye zincirleme sürecinde bellekte oluşan ontoloji kapanışı
3. Uzantılara ait bilgiler

Sorgulama aşamasında, basit bir örnek sorguya (*Bölüm 5.5.1*) ek olarak LUBM uygulamasına ait 11 (*Bölüm 5.5.2*), 12 (*Bölüm 5.5.3*) ve 13 (*Bölüm 5.5.4*) numaralı sorgular örnek sorgu olarak kullanılacaktır. Takip eden alt başlıklarda bu sorguların işletilmesi anlatılmaktadır.

5.5.1 Örnek Sorgu 1

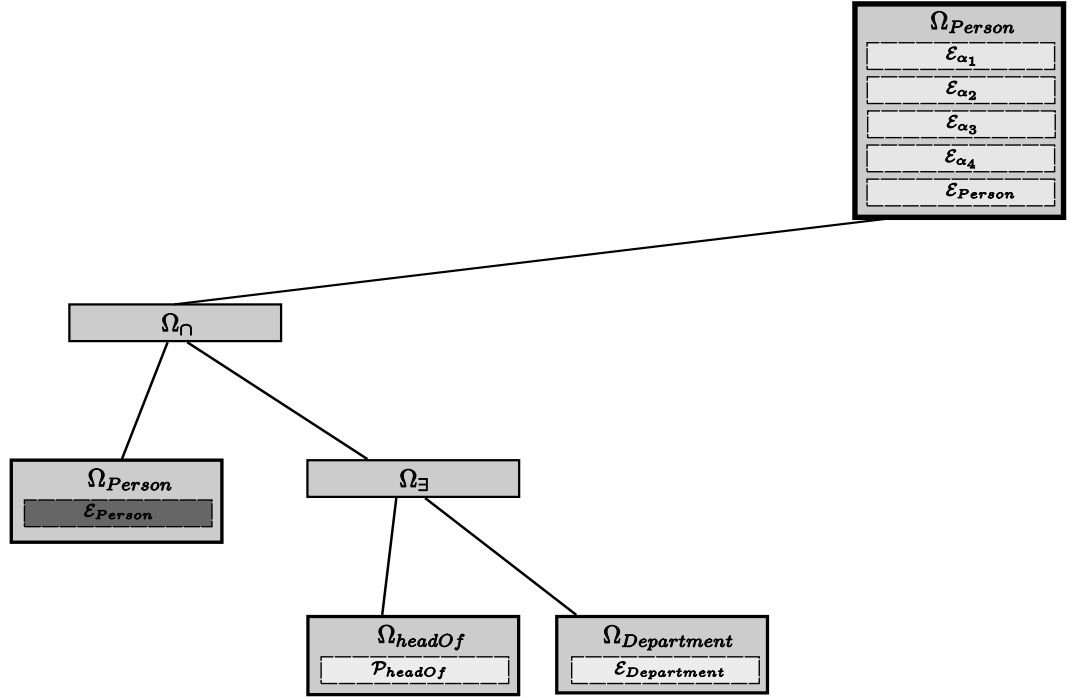
Aşağıda gösterilen *Örnek Sorgu 1*, *Person* sınıfına ait örnekleri döndüren tek koşullu bir sorgudur;

1. ($?X \text{ type } Person$)

Bu sorgunun yanıtlanabilmesi için, *Person* sınıfına ait bir sınıf düğümü yaratılması ve bu düğüme ait *memory* tablosunun *Person* sınıfına ait sınıf örneği uzantıları ile doldurulması gerekmektedir. *Person* sınıfına ait sınıf örneği uzantılarını bulmak için aşağıdaki sorgu kullanılır;

$hasClassExtension(Person, ?\epsilon)$

Bu sorgunun cevabı, ϵ_{Person} , ϵ_{α_1} , ϵ_{α_2} , ϵ_{α_3} , ϵ_{α_4} olarak hesaplanır ve *Person* sınıf düğümünün *memory* tablosuna bu değerler eklenir. Bundan sonra ilk karmaşık uzantı olan ϵ_{α_1} 'in ağaca eklenmesi gerekmektedir. α_1 'in sınıfı $Person \cap \{\exists headOf Department\}$ şeklinde tanımlanmıştır. Şekil 5.2'de, bu karmaşık sınıfın ağaca eklenmesinden sonra sorgu ağacının durumu gösterilmektedir. Karmaşık sınıf



Şekil 5.2. Örnek Sorgu 1 ağacı (ε_{α_1} açıldıktan sonra).

uzantısının içinde bir kez daha *Person* sınıfı kullanıldığı için burada sonlandırma prosedürü (bkz. Bölüm 4.3.1) devreye girer.

Diğer karmaşık sınıfların açılımları yapıldıktan sonra sorgu ağacının durumu sırasıyla Şekil 5.3, Şekil 5.4 ve Şekil 5.5'te gösterilmektedir. Diğer karmaşık sınıfların açılımı da ε_{α_1} sınıfının açılımına benzer şekilde yapılmaktadır.

Sorgu ağacından SQL sorgusu oluşturma algoritması özyinelemeli olarak çalışır ve ağaçtaki her bir düğümün Bölüm 4.3.3'teki karşılıkları eklenerek sorgu büyütülür. Her karmaşık uzantının sorgusu, *Person* uzantısının (*cextOfPerson*) sorgusu ile birleştirilerek sonuç sorgusu bulunur. Sorgu 5.5'te gösterilen sorguda *Query01* Sorgu 5.1'e, *Query02* Sorgu 5.2'ye, *Query03* Sorgu 5.3'e ve *Query04* Sorgu 5.4'e karşılık gelmektedir ve sorgunun daha kolay anlaşılabilir olması açısından bu şekilde yazılmıştır. Bu sorgunun çalıştırılması sonucu elde edilen cevap Çizelge 5.24'te verilmektedir.

Örnek Sorgu 1, yöntemin açıklanması bakımından önemlidir. *Person* sınıfının ontolojide temel bir sınıf olması nedeniyle yaratılan sorgunun karmaşıklık düzeyi yüksektir.

Sorgu 5.1 ε_{α_1} karmaşık uzantısının SQL sorgusu karşılığı.

```

SELECT T1.resource FROM
(SELECT resource FROM contains
WHERE extension = cextOfPerson) AS T1
(SELECT T3.subject FROM
(SELECT resource FROM contains
WHERE extension = cextOfDepartment) AS T2
(SELECT subject ,object FROM statements
WHERE predicate = headOf) AS T3
WHERE T3.object = T2.resource) AS T4
WHERE
T1.resource = T4.subject

```

Sorgu 5.2 ε_{α_2} karmaşık uzantısının SQL sorgusu karşılığı.

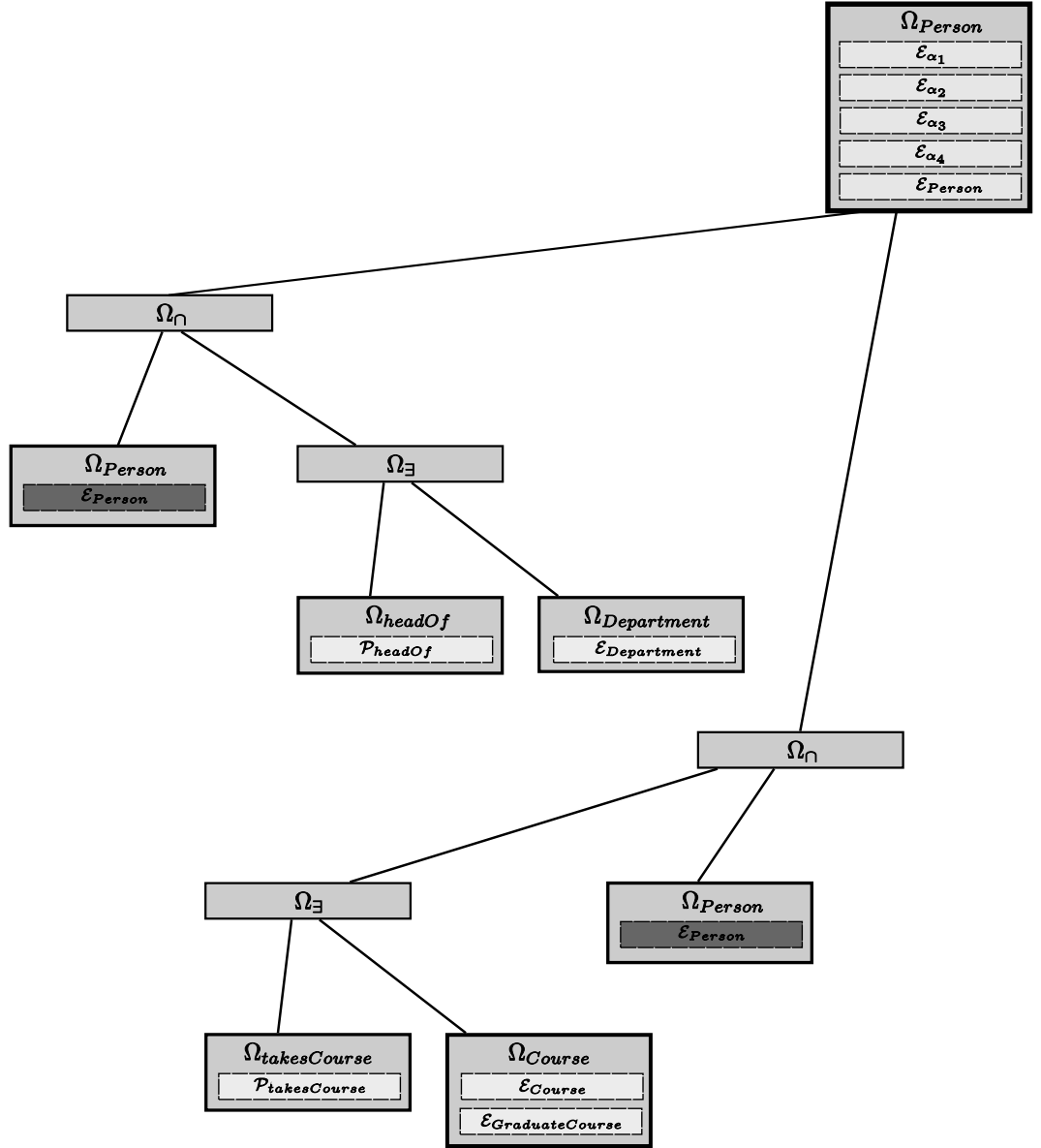
```

SELECT T1.resource FROM
(SELECT resource FROM contains
WHERE extension = cextOfPerson) AS T1
(SELECT T3.subject FROM
(SELECT resource FROM contains
WHERE extension = cextOfCourse
OR extension = cextOfGraduateCourse) AS T2
(SELECT subject ,object FROM statements
WHERE predicate = takesCourse) AS T3
WHERE T3.object = T2.resource) AS T4
WHERE
T1.resource = T4.subject

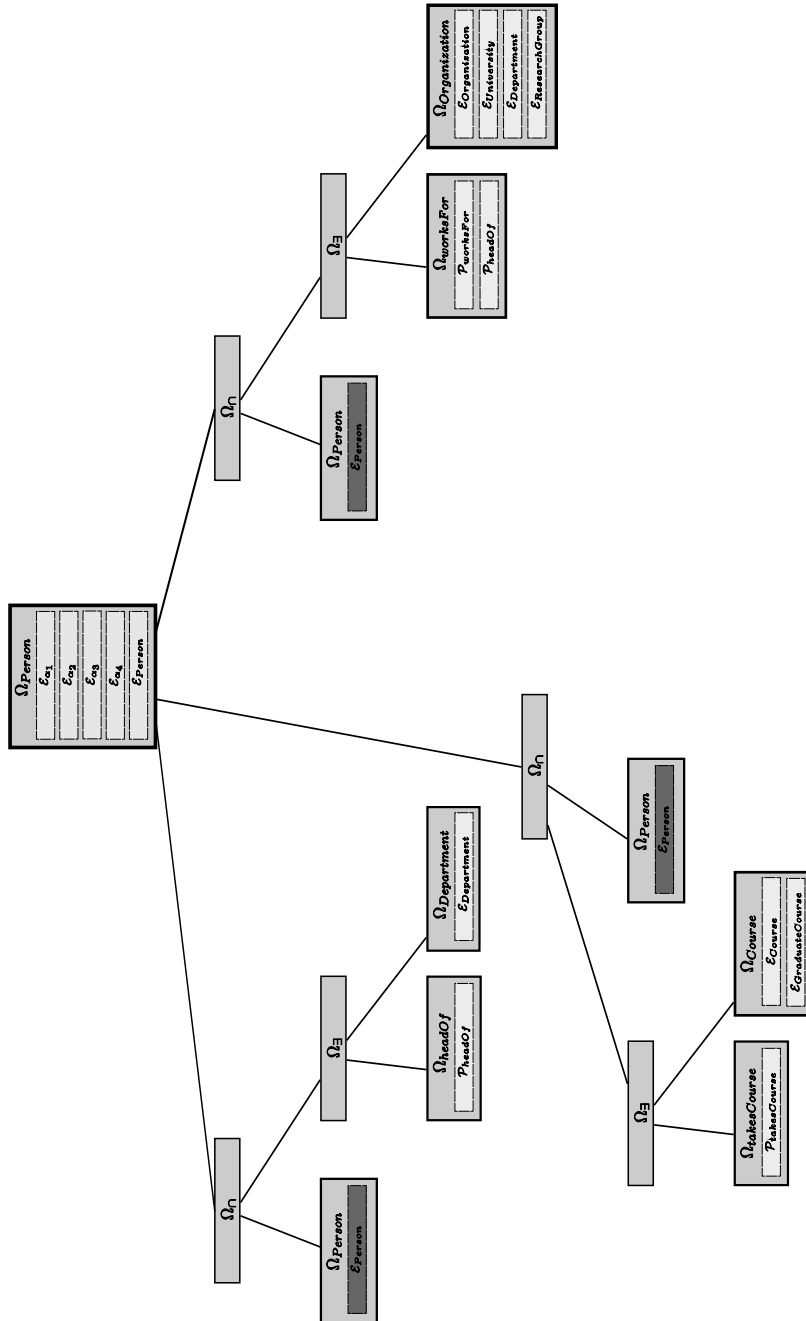
```

<i>Sorgu</i>	<i>Yanıt</i>
<i>cextOfPerson</i>	<i>prof01, ..., prof10</i>
<i>Query01</i>	<i>prof03, prof05, prof07</i>
<i>Query02</i>	-
<i>Query03</i>	<i>prof01, ..., prof10</i>
<i>Query04</i>	-
Ana Sorgu	<i>prof01, ..., prof10</i>

Çizelge 5.24. Örnek Sorgu 1 sonucu.



Şekil 5.3. Örnek Sorgu 1 ağacı (ϵ_{α_2} açıldıktan sonra).



Şekil 5.4. *Örnek Sorgu 1* sorgu ağacı (ε_{α_3} açıldıktan sonra).

Sorgu 5.3 ε_{α_3} karmaşık uzantısının SQL sorgusu karşılığı.

```

SELECT T1.resource FROM
(SELECT resource FROM contains
WHERE extension = cextOfPerson) AS T1
(SELECT T3.subject FROM
(SELECT resource FROM contains
WHERE extension = cextOfOrganization
OR extension = cextOfUniversity
OR extension = cextOfDepartment
OR extension = cextOfResearchGroup) AS T2,
(SELECT subject , object FROM statements
WHERE predicate = worksFor OR predicate = headOf) AS T3
WHERE T3.object = T2.resource) AS T4
WHERE
T1.resource = T4.subject

```

Sorgu 5.4 ε_{α_4} karmaşık uzantısının SQL sorgusu karşılığı.

```

SELECT T1.resource FROM
(SELECT resource FROM contains
WHERE extension = cextOfPerson) AS T1
(SELECT T3.subject FROM
(SELECT resource FROM contains
WHERE extension = cextOfGraduateCourse) AS T2
(SELECT subject , object FROM statements
WHERE predicate = takesCourse) AS T3
WHERE T3.object = T2.resource) AS T4
WHERE
T1.resource = T4.subject

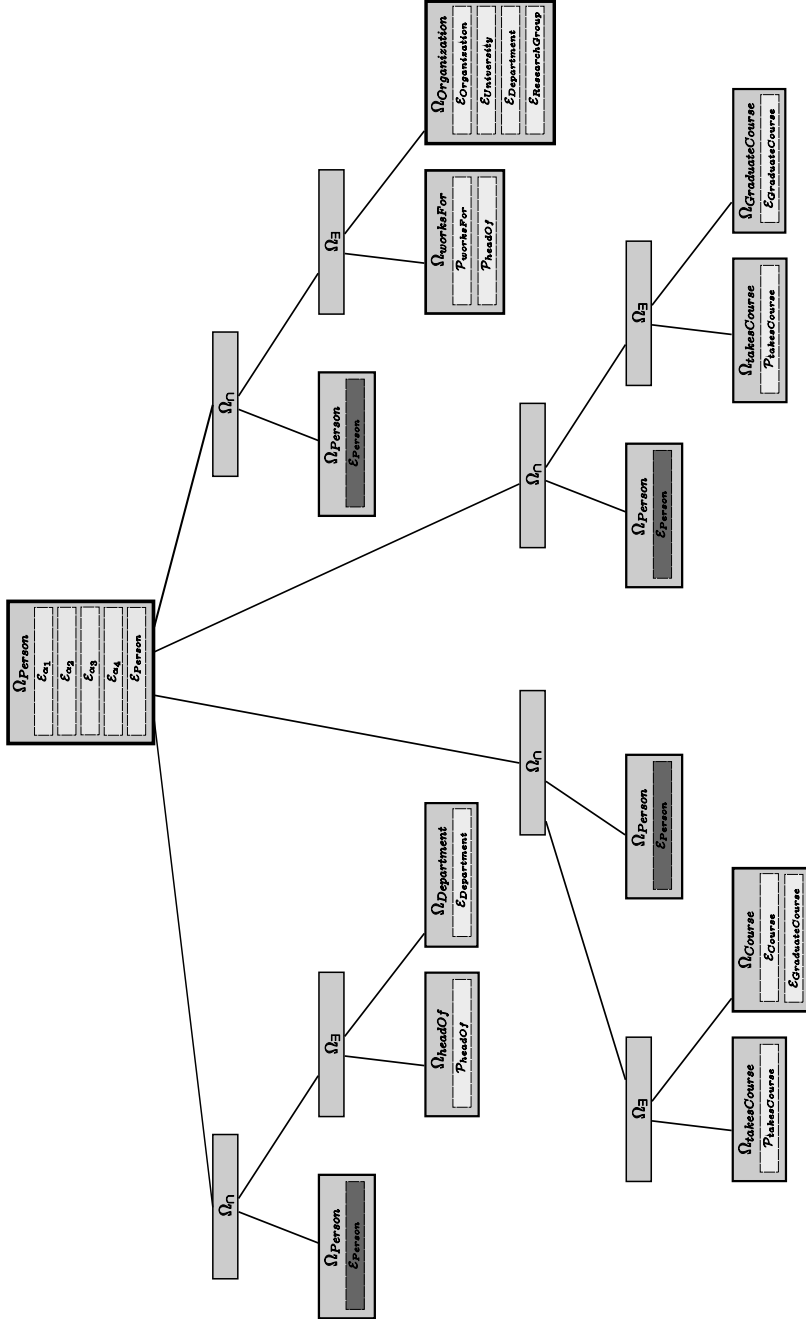
```

Sorgu 5.5 *Örnek Sorgu 1*'in SQL karşılığı.

```

(SELECT resource FROM contains
WHERE extension = cextOfPerson)
UNION Query01
UNION Query02
UNION Query03
UNION Query04

```



Şekil 5.5. Örnek Sorgu 1 sorgu ağacı (ϵ_{α_4} açıldıktan sonra).

5.5.2 Örnek Sorgu 2

Örnek Sorgu 2, LUBM uygulamasına ait 11 numaralı sorgudur. Aşağıda bu sorguya ait iki koşul gösterilmektedir;

1. (*?X type ResearchGroup*)
2. (*?X subOrganizationOf "http://www.University0.edu"*)

Bu sorgu, veritabanında uzantıların işletilmesinin ve genişletilmesinin (bkz. Bölüm 4.2.3) sorgu sürecine nasıl etki ettiğini gösteren bir örnektir. Bu sorguda kullanılan *subOrganizationOf* niteliği örnek ontolojide geçişken bir nitelik olarak tanımlanmıştır. Bu niteliğin önalın ve artalanından yeralan *Organization* sınıfının üç adet alt sınıfı bulunmaktadır; *ResearchGroup*, *Department* ve *University*. *ResearchGroup* sınıfı ve *Department* sınıfı ile *Department* sınıfı ise *University* sınıfı ile *subOrganizationOf* niteliği ile ilişkilendirilmiştir. *subOrganizationOf* niteliği geçişken bir nitelik olduğundan, *ResearchGroup* ve *University* sınıflarına ait örnekler arasındaki *subOrganizationOf* ilişkisinin çıkarsanması gerekmektedir.

Hazırlık aşamasında, ağacın ilk seviyesindeki düğümler oluşturulur. İlk koşul için bir sınıf düğümü oluşturulur ve bu düğüme ait *memory* tablosu aşağıdaki sorgunun cevapları ile doldurulur:

hasClassExtension(ResearchGroup, ?ε)

Sorgunun çalıştırılması sonucu, tabloya $\varepsilon_{ResearchGroup}$ uzantısı eklenir. İkinci koşul için ağaca bir nitelik düğümü eklenir. Nitelik düğümlerinin *memory* tabloları aşağıdaki iki sorgu ile doldurulur. İlki düz uzantıları bulurken ikincisi ters uzantıları elde etmektedir:

hasPropertyExtension(subOrganizationOf, ?ε)

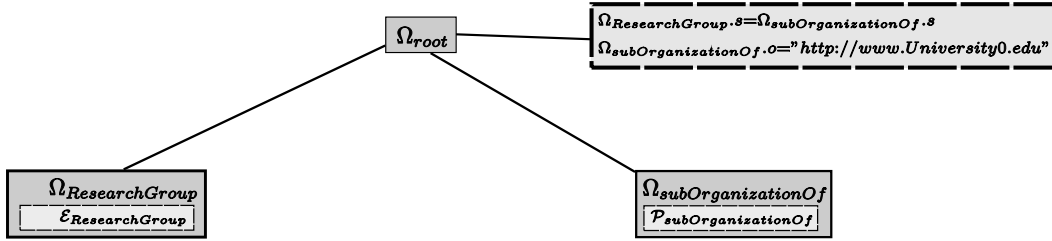
hasInversePropertyExtension(subOrganizationOf, ?ε)

Örnekte, ilk sorgunun cevabı $\mathcal{P}_{subOrganizationOf}$ 'dir. İkinci sorgunun cevabı ise boş dönecektir, çünkü *subOrganizationOf*'un ters bir uzantısı yoktur. Her koşula karşılık gelen düğümler oluşturulduktan sonra bu koşullar arasındaki bağlantıların kök düğüme eklenmesi gerekmektedir. Bu sorguda, birinci ve ikinci koşulların öznelere aynı olması gerekmektedir. Aynı zamanda ikinci koşulun nesnesine ait değer *University0* olarak belirtilmiştir. Bu nedenle sorgunun kök düğüme aşağıdaki ilişkiler ve kısıtlar eklenir.

$$1. \Omega_{ResearchGroup}.s = \Omega_{subOrganizationOf}.s$$

$$2. \Omega_{subOrganizationOf}.o = "http://www.University0.edu"$$

Yukarıda anlatılan aşamalar tamamlandıktan sonra Şekil 5.6'da gösterilen sorgu ağacı elde edilir. Bu sorgu ağacı oluşturulurken geçiş aşamasında bir işlem yapılmamaktadır, çünkü birinci katmanda karmaşık bir sınıf düğümü yoktur.



Şekil 5.6. Örnek Sorgu 2 ağacı.

Sorgu ağacının oluşturulmasının ardından sıra sorgunun yazılması ve cevapların alınması aşamasına gelmektedir. Bölüm 4.3.3'te verilen sorgu karşılıkları kullanılarak sorgu ağacının SQL karşılığı elde edilir. *ResearchGroup* düğümü ($\Omega_{ResearchGroup}$) için karşılık gelen sorgu aşağıda gösterilmektedir:

SELECT resource FROM contains WHERE extension =
 $\epsilon_{ResearchGroup}$

subOrganizationOf nitelik düğümü ($\Omega_{subOrganizationOf}$) için karşılık gelen sorgu aşağıda gösterilmektedir:

SELECT subject, object FROM statements WHERE predicate =
 $subOrganizationOf$

İki düğümü birleştiren kısıtlar da eklendiği zaman sorgunun Sorgu 5.6'da gösterilen son hali elde edilmiş olur.

?X	rg01	rg02	dep01	dep02	dep03
----	------	------	-------	-------	-------

Çizelge 5.25. Örnek Sorgu 2 sonucu.

Sorgu 5.6 Örnek Sorgu 2'nin SQL karşılığı.

```

SELECT T0.resource FROM
(SELECT resource
FROM contains
WHERE extension = cextOfResearchGroup) AS T0,
(SELECT subject , object
FROM statements
WHERE predicate = subOrganizationOf) AS T1
WHERE T0.resource = T1.subject
AND T1.object = "http://www.University0.edu"

```

Çizelge 5.25, Sorgu 5.6'da gösterilen sorgunun sonucunu göstermektedir.

5.5.3 Örnek Sorgu 3

Örnek Sorgu 3, LUBM uygulamasına ait 12 numaralı sorgudur. Aşağıda bu sorguya ait dört koşul gösterilmektedir;

1. (*?X type Chair*)
2. (*?Y type Department*)
3. (*?X worksFor ?Y*)
4. (*?Y subOrganizationOf "http://www.University0.edu"*)

Örnek verilerinde, *Chair* sınıfına ait belirtik bir örnek yoktur. Fakat, *Department* sınıfı *Chair* sınıfına *headOf* niteliği ile bağlanmıştır. Bu nedenle, *Department* sınıfına ait bir örneğin *headOf* niteliği ile bağlı olduğu sınıf örneklerinin *Chair* sınıfının örnekleri olduğu çıkarılmalıdır.

Bu sorguda iki adet sınıf koşulu vardır. Bu nedenle iki sınıf düğümü yaratılır ve bu düğümlere ait *memory* tabloları aşağıdaki sorguların yanıtları ile doldurulur;

hasClassExtension(Chair, ?ε)

hasClassExtension(Department, ?ε)

İlk sorgunun cevabı olarak ε_{Chair} , ε_{α_0} gelecektir. *Chair* sınıf düğümünün *memory* tablosuna bu değerler eklenir. İkinci sorgunun cevabı olarak $\varepsilon_{Department}$ gelecektir. *Department* sınıf düğümünün *memory* tablosuna da bu değer eklenir.

Diğer iki koşul nitelik koşullarıdır. Bu nedenle iki yeni nitelik düğümü oluşturulur. İlk nitelik düğümüne ait *memory* tablosu aşağıdaki iki sorgunun yanıtları ile doldurulur. İlk sorgu *worksFor* niteliğine ait düz uzantıları bulurken, ikincisi ters uzantıları elde etmektedir;

$hasPropertyExtension(worksFor, ?\varepsilon)$

$hasInversePropertyExtension(worksFor, ?\varepsilon)$

İlk sorgunun cevabı $\mathcal{P}_{worksFor}$ ve $\mathcal{P}_{headOf}$ 'dur. İkinci sorgunun cevabı ise boş dönecektir, çünkü *worksFor* niteliğine ters bağlı bir uzantı yoktur. İkinci nitelik düğümüne ait *memory* tablosu, aşağıdaki iki sorgunun yanıtları ile doldurulur.

$hasPropertyExtension(subOrganizationOf, ?\varepsilon)$

$hasInversePropertyExtension(subOrganizationOf, ?\varepsilon)$

İlk sorgunun cevabı $\mathcal{P}_{subOrganizationOf}$ 'dur. İkinci sorgunun cevabı ise boş dönecektir, çünkü *subOrganizationOf* niteliğine ters bağlı bir uzantı yoktur.

Her koşula karşılık gelen düğümler oluşturulduktan sonra bu koşullar arasındaki bağlantıların kök düğüme eklenmesi gerekmektedir. Bu sorguda, birinci ve ikinci koşulların öznelerinin aynı olması gerekmektedir. Aynı zamanda ikinci koşulun nesnesinin de *University0* olması gerekmektedir. Bu nedenle sorgunun kök düğümüne aşağıdaki bağlantı kısıtları eklenir:

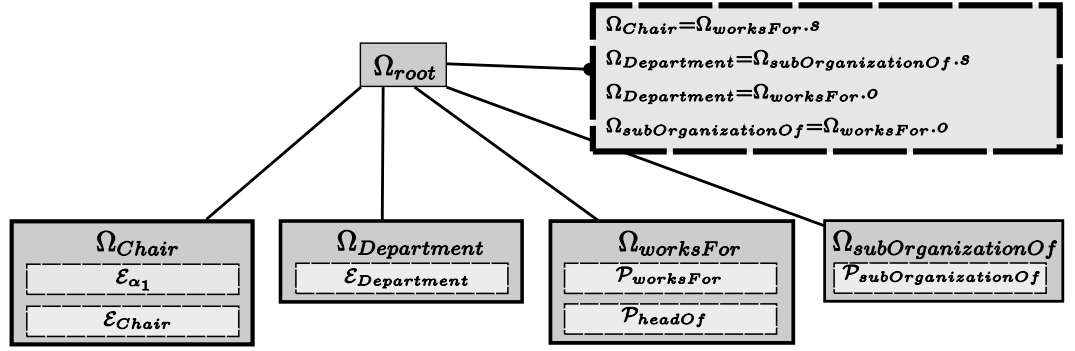
$$1. \Omega_{Chair} = \Omega_{worksFor.S}$$

$$2. \Omega_{Department} = \Omega_{subOrganizationOf.S}$$

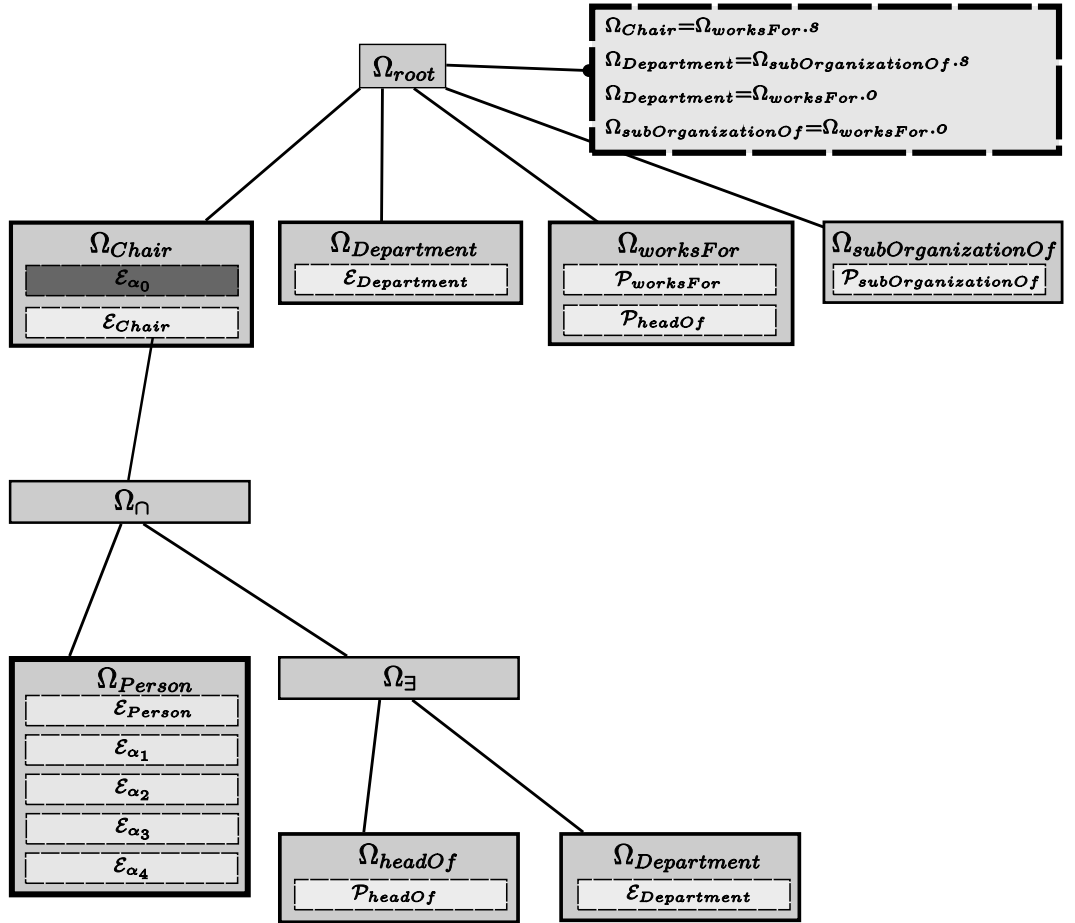
$$3. \Omega_{Department} = \Omega_{worksFor.O}$$

$$4. \Omega_{subOrganizationOf} = \Omega_{worksFor.O}$$

Bağlantı kısıtlarının eklenmesinin ardından Şekil 5.7'de gösterilen sorgu ağacı elde edilir. Bu aşamadan sonra sorgulamanın geçiş aşamasına geçilmektedir. Başlangıç ağacında karmaşık uzantıya sahip tek düğüm olan *Chair* sınıf düğümünün açılımı yapılır. Sorgu ağacının son hali Şekil 5.8'de gösterilmektedir. Bu sorgu ağacından elde edilen sorgu ise Sorgu 5.7'de verilmiştir. Son olarak Çizelge 5.26'da sorgunun çalıştırılması sonucu elde edilen değerler gösterilmektedir.



Şekil 5.7. Örnek Sorgu 3 hazırlık aşaması.



Şekil 5.8. Örnek Sorgu 3 geçiş aşaması.

?X	rg01	rg02	dep01	dep02	dep03
----	------	------	-------	-------	-------

Çizelge 5.26. Örnek Sorgu 3 sonucu.

Sorgu 5.7 Örnek Sorgu 3'ün SQL karşılığı.

```

SELECT T0.resource
FROM
  (( QueryPerson
  INTERSECT
  (SELECT T0.resource FROM
  (SELECT resource FROM contains
  WHERE extension = cextOfDepartment) AS T0,
  (SELECT subject,object FROM statements
  WHERE predicate = headOf) AS T1
  WHERE T1.object = T0.resource))
  UNION
  (SELECT resource FROM contains
  WHERE extension = cextOfChair)) AS T2,
  (SELECT resource FROM contains
  WHERE extension = cextOfDepartment) AS T3,
  (SELECT subject,object FROM statements
  WHERE predicate = worksFor OR predicate = headOf) AS T4,
  (SELECT subject,object FROM statements
  WHERE predicate = subOrganizationOf) AS T5
  WHERE T2.resource = T4.subject
  AND T3.resource = T5.subject
  AND T3.resource = T4.object
  AND T5.subject = T4.object

```

5.5.4 Örnek Sorgu 4

Örnek Sorgu 4, LUBM uygulamasına ait 13 numaralı sorgudur. Aşağıda bu koşula ait iki sorgu gösterilmektedir;

1. ($?X \text{ type Person}$)
2. ($?X \text{ hasAlumnus "http : //www.University0.edu"}$)

hasAlumnus niteliği *degreeFrom* niteliğinin tersi olarak tanımlanmıştır. *degreeFrom* niteliğinin üç altniteliği vardır; *undergraduateDegreeFrom*, *mastersDegreeFrom* ve *doctoralDegreeFrom*. Örnek verilerinde, bir kişinin bir üniversitenin mezunu olduğu, *hasAlumnus* ilişkisi kullanılarak değil, *hasAlumnus* niteliğine ait üç altnitelik kullanılarak belirtilmektedir. Bu nedenle, *University0*'ın mezunlarını hesaplayan bu sorgu, mezunları bulmak için *hasAlumnus* niteliğinin tersi olarak tanımlanan *degreeFrom* örnekleri, dolayısıyla *degreeFrom* niteliğine ait üç altniteliğin örnek uzantıları üzerinde çıkarsama yapmalıdır;

Bu sorguda bir tane sınıf koşulu vardır. Bu nedenle, bir sınıf düğümü yaratılır ve bu düğüme ait *memory* tablosu aşağıdaki sorgunun yanıtları ile doldurulur;

$hasClassExtension(Person, ?\epsilon)$

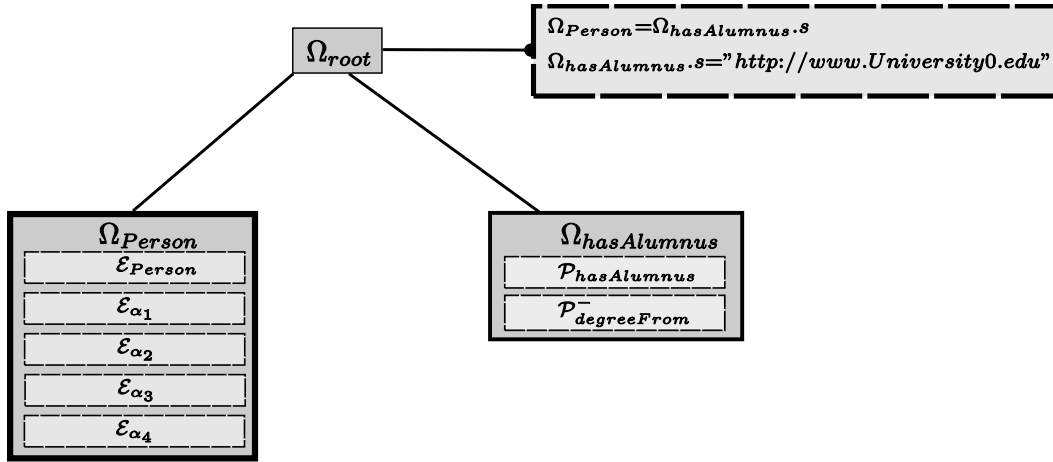
Bu sorgunun cevabı olarak ϵ_{Person} , ϵ_{α_1} , ϵ_{α_2} , ϵ_{α_3} , ϵ_{α_4} gelecektir. Sınıf düğümünün *memory* tablosuna bu değerler eklenir. Diğer koşul bir nitelik koşuludur. Bu nedenle, bir nitelik düğümü yaratılır ve bu düğüme ait *memory* tablosu aşağıdaki sorguların yanıtları ile doldurulur;

$hasPropertyExtension(hasAlumnus, ?\epsilon)$

$hasInversePropertyExtension(hasAlumnus, ?\epsilon)$

Örnekte, ilk sorgunun cevabı $\mathcal{P}_{hasAlumnus}$, ikinci sorgunun cevabı ise $\mathcal{P}_{degreeFrom}^-$ 'dur. Her koşula karşılık gelen düğümler oluşturulduktan sonra bu koşullar arasındaki bağlantıların kök düğüme eklenmesi gerekmektedir. Bu sorguda, birinci ve ikinci koşulların öznelere aynı olması gerekmektedir. Aynı zamanda ikinci koşulun nesnesinin de *University0* olması gerekmektedir. Bu nedenle sorgunun kök düğümüne aşağıdaki bağlantı kısıtları eklenir.

Bağlantı kısıtlarının eklenmesinin ardından Şekil 5.9'da gösterilen sorgu ağacı elde edilir. Bu aşamadan sonra sorgulamanın geçiş aşamasına geçilmektedir.



Şekil 5.9. Örnek Sorgu 4 hazırlık aşaması.

Sorgu 5.8 Örnek Sorgu 4'ün SQL karşılığı.

```

SELECT QueryPerson.resource FROM
((SELECT subject, object FROM statements
WHERE predicate = hasAlumnus)
UNION
(SELECT object AS subject, subject AS object FROM statements
WHERE predicate = degreeFrom)) AS T0, QueryPerson
WHERE QueryPerson.resource = T0.subject
AND T0.object = "http://www.University0.edu"

```

Sorgu ağacındaki karmaşık uzantıya sahip olan düğüm Person düğümünün açılımı yapıldıktan sonra sorgu ağacı tamamlanır. Bu sorgu ağacından elde edilen sorgu ise Sorgu 5.8'de verilmiştir. Son olarak Çizelge 5.27'de sorgunun çalıştırılması sonucu elde edilen değerler gösterilmektedir.

?X	rg01	rg02	dep01	dep02	dep03
----	------	------	-------	-------	-------

Çizelge 5.27. Örnek Sorgu 4 sonucu.

6 Uzantı Tabanlı Bilgi Modelinin Analizi ve Doğrulanması

6.1 Ontoloji Ölçütlerini Kullanarak Faydalanma Oranını Tahminleme

Bu bölümde, farklı karakteristiklere sahip bilgi tabanları ile modelden faydalanma oranının tahminlenmesine yönelik gözlemler sunulacaktır. Bilgi tabanlarının karakteristiklerini belirlemek için OntoQA ölçütleri (Tartir et al., 2005) kullanılacaktır. OntoQA ölçütleri, şema ölçütleri ve örnek ölçütleri olmak üzere iki grupta incelenir. Şema ölçütleri ontolojideki kavram tanımlamaları ile ilgili ölçütlerdir. Örnek ölçütleri ise ontoloji popülasyonu ve popülasyonun kavramlar bazında dağılımı ile ilgili ölçütlerdir.

Modelden faydalanma oranı, örneksel verinin şematik veriye oranı ile doğru orantılıdır. Başka bir deyişle, örneksel verinin boyutu şematik veriden az olduğunda model negatif bir etki gösterir. Fakat, bilindiği üzere somut ve kullanımda olan ontolojilerin neredeyse tamamında örnek verisinin boyutu şematik verinin boyutuna göre çok fazladır ve güncellemeler yoluyla sürekli artış göstermeye meyillidir. OntoQA ölçütleri örneksel verinin oranıyla ilgili iki ölçüt belirlemektedir:

- *Ortalama Popülasyon* (örneklerin kavramlar bazında dağılımı): Bu ölçüt örnek sayısını, sınıf sayısı ile karşılaştırır.
- *İlişki Zenginliği*: Her bir sınıfa ait nitelikler içinde kaçının örnek verisi seviyesinde kullanıldığını gösteren önemli bir ölçüttür.

Modelden faydalanma oranını etkileyen bir diğer faktör, sınıf ve nitelik hiyerarşilerinin derinliğidir. İleriye doğru çıkarsama algoritmasını kullanan herhangi bir çıkarsama motorunun performansı sınıf ve nitelik hiyerarşisinin derinliği ile doğru-
dan bağlantılıdır. Hiyerarşi derinleştikçe çıkarsama motorunun performansı düşer. Bunun nedeni, bir altkavrama (altsınıf/altnitelik) ait her bir örneğin, çıkarsama sonucu altkavrama ait her bir üstkavramla ilişkilendirilmesidir. Altsınıf/altnitelik ilişkilerinin geçişken olarak hesaplanması sonucunda, örneğin n tane örneğe, m

tane üstsınıfa sahip bir sınıf, $m \times n$ tane yeni "type" ilişkisi türetmektedir. Diğer bir deyişle, çıkarsanan "type" ilişkisi sayısı, sınıfın ait olduğu hiyerarşinin derinliği ile lineer olarak artış göstermektedir. Uzantı tabanlı bilgi modeli dönüşümü sonrası, altkavrama ait her bir örneği üstkavramla ilişkilendirmek yerine, yalnızca altkavrama ait uzantı üstkavramla ilişkilendirilir. Bu nedenle, çıkarsamanın performansı kavram hiyerarşisinden etkilenmez.

Kavram hiyerarşisinin derinliği ile ilgili olarak OntoQA aşağıdaki ölçütü tanımlar:

- *Kalıtım Zenginliği*: ontolojik bilginin farklı kategorilere ve alt kategorilere gruplanma başarımını belirtir.

Son olarak, tez kapsamında, OntoQA ölçütlerine ek olarak, yeni bir ölçüt sunulmaktadır:

- *Kavram Tanımı Karmaşıklığı*: bu ölçüt anonim sınıfların ve herhangi bir OWL nitelik karakteristiğine sahip niteliklerin sayısını belirtir.

Kavram tanımı karmaşıklığındaki artış, sorgu yanıtlama performansında düşüğe neden olur. Fakat, geriye doğru çıkarsama algoritması kullanan bir çıkarsama motoruyla karşılaştırıldığında, sorgu yanıtlama performansındaki düşüş daha az olmaktadır (*bkz. Bölüm 7*).

6.2 Faydalanma Oranının Hesaplanması

Bu bölümde, sunulan dönüşümün sağladığı depolama yeri kazanımını analiz etmektedir. Bu bölüm, metodun mantıksal temelini anlaşılmaması ve olası gelecek iyileştirmelerin belirlenebilmesi açısından önem taşımaktadır.

$\mathcal{O}$ bir ontoloji, $n(\mathcal{O})$ üçlü sayısı, $\mathcal{R}_{\mathcal{O}}^{type}$ "type" ilişkilerinin sayısı, $\mathcal{R}_{\mathcal{O}}^{\mathcal{P}}$ "type" ilişkisi hariç tüm niteliklere ait toplam örnek sayısı, $\mathcal{C}_{\mathcal{O}}$ sınıf sayısı ve $\mathcal{P}_{\mathcal{O}}$ "type" hariç tüm niteliklerin sayısı olmak üzere, ontolojideki tüm üçlü sayısı $n(\mathcal{O})$ aşağıdaki şekilde hesaplanır:

$$n(\mathcal{O}) = \mathcal{R}_{\mathcal{O}}^{type} + \mathcal{R}_{\mathcal{O}}^{\mathcal{P}}$$

Ontoloji ayrıştırıldıktan sonra her üçlü veritabanında "statements" tablosuna yazılır. Ontolojideki anonim veya anonim olmayan her sınıf için, bu sınıfı ilgili

uzantı ile bağlayan *hasExplicitClassExtension* ilişkisi oluşturulur. Diğer bir deyişle, dönüşümden sonra ontolojiye eklenen *hasExplicitClassExtension* örneklerinin sayısı sınıf sayısına ($\mathcal{C}_O$) eşittir. Daha sonra, her bir uzantı, ilgili sınıfa "type" ilişkisi ile bağlı her bir sınıf örneği ile *contains* ilişkisi üzerinden ilişkilendirilir. Diğer bir deyişle, dönüşümden sonra *contains* ilişkilerinin sayısı $\mathcal{R}_O^{type}$ olarak hesaplanabilir. Ayrıca, "type" hariç her bir nitelik için, bu niteliği ilgili uzantılarla bağlayan *hasExplicitSubjectExtension*, *hasExplicitObjectExtension* ve *hasExplicitPropertyExtension* ilişkileri tanımlanır. Diğer bir deyişle, dönüşümden sonra ontolojiye nitelik sayısının üç katı kadar daha ($3 \times \mathcal{P}_O$) yeni ilişki eklenir. "type" ilişkileri hariç her bir nitelik örneğine ait özne ilgili nitelik öznesi uzantısı ile *contains* ilişkisi üzerinden ilişkilendirilir ve ontolojiye nitelik örneği sayısı kadar ($\mathcal{R}_O^P$) yeni üçlü eklenmiş olur. Benzer şekilde, "type" ilişkileri hariç her bir nitelik örneği nesnesi ilgili nitelik nesnesi uzantısı ile *contains* ilişkisi üzerinden ilişkilendirilir ve ontolojiye nitelik örneği sayısı kadar ($\mathcal{R}_O^P$) yeni üçlü eklenmiş olur. Sonuç olarak, dönüşümden sonra veritabanındaki üçlü sayısı $n(\mathcal{O}^t)$ aşağıdaki şekilde hesaplanır:

$$n(\mathcal{O}^t) = n(\mathcal{O}) + \mathcal{C}_O + \mathcal{R}_O^{type} + 3 \times \mathcal{P}_O + 2 \times \mathcal{R}_O^P$$

Dönüşümden sonra, veritabanındaki üçlü sayısındaki artış Δ_1 aşağıdaki şekilde hesaplanır:

$$\Delta_1 = \mathcal{C}_O + \mathcal{R}_O^{type} + 3 \times \mathcal{P}_O + 2 \times \mathcal{R}_O^P$$

Dönüşümden sonra çıkarsama sürecine katılan üçlü sayısı azalmaktadır. Çıkarsama sürecine yalnızca şema bilgisi ve gruplama yapıtaşları (*hasExplicitClassExtension*, *hasExplicitPropertyExtension*, *hasExplicitSubjectExtension*, *hasExplicitObjectExtension*) ile ilişkilendirilmiş üçlüler katılır. Böylelikle yalnızca şema seviyesinde çıkarsama yapılır. Diğer bir deyişle, çıkarsamaya katılan ve çıkarsama sonucu elde edilen üçlülerin sayısında büyük bir düşüş görülür. Örnek seviyesinde yapılan çıkarsamaların tümü iptal edilmiş olur, yalnızca gruplama yapıtaşları ile ilgili ekstra çıkarsama sonuçları türetilir. Fakat bu üçlülerin sayısı da ihmal edilebilecek kadar azdır.

Sonuç olarak, dönüşüm sonrası veritabanındaki üçlü sayısında bir artış olur. Fakat dönüştürülen bilginin yalnızca küçük bir kısmı çıkarsamaya katıldığından, çıkarsamanın depolama ve zaman gereksinimleri önemli ölçüde azalır. Örnek

verileri çıkarsama sürecine katılmadığından, örnek verisinin boyutu artsa bile çıkarsamanın depolama ve zaman gereksinimi sabit kalır. Bu sonuçlara göre, modelden faydalanma oranını yükselten faktörler aşağıdaki gibi listelenebilir:

- örnek verisindeki artış
- sınıf ve nitelik hiyerarşilerinin derinliğindeki artış (özellikle örnek verisinin hiyerarşinin alt seviyelerinde yoğunlaştığı durumlar)
- önalın/artalan ilişkilerindeki artış

6.3 Sınama ve Doğrulama

Bu bölümde OWL Sınama Durumları belgesinde bulunan sınama durumları arasından pD* dilinin desteklediği sınama durumlarının sonuçları verilerek pD* dilinin üzerinde yalnızca sözdizimsel bir dönüşüm yapıldığı, sonuçların eksiksiz ve geçerli olduğu gösterilecektir.

OWL Sınama Durumları belgesi (Carroll and Roo, 2004) World Wide Web Consortium tarafından OWL dilini kullanan, üzerinde çıkarsama yapan uygulamalar geliştiren yazılımcılar için çıkarılmıştır. Belgenin amacı hem “OWL Semantics” belgesinde (Patel-Schneider et al., 2004) açıklanan dilin semantiğini örnekler vererek açıklamak hem de bu gereksinimlerin herkesin üzerinde anlaşabileceği bir biçime dönüştürerek paylaşımını kolaylaştırmaktır. Bu belgede verilen sınama durumlarını tümünü geçen uygulamaların OWL dili için eksiksiz ve geçerli bir çıkarsama yaptığı kanıtlanmış olur.

OWL dilinin kural tabanlı sistemler için kısıtlanmış bir hali olan pD* dili içinde kullanılan OWL yapıları dikkate alınırsa 37 sınama durumunu içeren bir alt kümeyi desteklemesi gereklidir. Fakat, Bölüm 2’de bahsedilen kural tabanlı sistemlerin çıkarsamadaki yetersizlikleri nedeniyle bu sınama durumlarının yalnızca 18 tanesinin pD* dilini destekleyen bir çıkarsama motoru tarafından geçilmesi beklenebilir (Li et al., 2006). Bu nedenle tez kapsamında önerilen algoritmanın bu sınama durumlarını geçmesi, algoritmanın kural tabanlı sistemlerde çıkarsamanın eksiksizliğini ve geçerliliğini etkilemediğini göstermeye yeterlidir.

Çizelge 6.1 algoritmanın sınama sonuçları başarımını göstermektedir. + ile belirtilen sınama durumları pD* diline ait kuralların desteklediği ve algoritmanın

doğru olarak hesapladığı durumlardır. x ile belirtilen sınama durumları ise pD* diline ait kuralların desteklemediği sınama durumlarını göstermektedir.

Gerektirme Sınama Durumu	001	002	003	004	005	006	007
<i>owl : FunctionalProperty</i>	+	+	x	x	x		
<i>owl : InverseFunctionalProperty</i>	+	+	x	x			
<i>owl : Restriction</i>						x	
<i>owl : SymmetricProperty</i>	+	x	x				
<i>owl : TransitiveProperty</i>	+	x					
<i>owl : allValuesFrom</i>	+						
<i>owl : differentFrom</i>	x	x					
<i>owl : disjointWith</i>	+	+					
<i>owl : equivalentClass</i>	+	+	+	x		x	x
<i>owl : equivalentProperty</i>	+	+	+	x	x	+	
<i>owl : inverseOf</i>	+						
<i>owl : sameAs</i>	+						
<i>owl : someValuesFrom</i>	x	x	x				

Çizelge 6.1. Sınama sonuçları.

Tez kapsamında geliştirilen uygulama bu sınama durumlarının tamamını geçmiştir. Bu nedenle tez kapsamında önerilen yöntemin, çıkarsamanın eksiksizliğini ve geçerliliğini etkilemediği söylenebilir. pD* dili dışında diller için kural tabanlı sistemlerin başarımını kanıtlayan bir yayın olmadığı için bu diller tez kapsamı dışındadır.

owl:FunctionalProperty

Sınama Durumu 001

Sınama Durumu 001'e göre, örneğin *prop* niteliği *owl:FunctionalProperty* olarak tanımlanmışsa ve *prop* niteliğine ait iki farklı örneğin özneleri aynı ise, bu durumda bu nitelik örneklerinin nesnelerinin de aynı olduğu sonucuna varılır. *Sınama Durumu 001*, *P* gerektirme kurallarından *rdfp1* kuralına karşılık gelir.

Sorgu 6.1 *owl:FunctionalProperty 001* numaralı sınaama durumu için öncül ontoloji.

```
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:owl="http://www.w3.org/2002/07/owl#"
  xmlns:first="http://www.w3.org/2002/03/owl#FunctionalProperty/premises001#"
  xml:base="http://www.w3.org/2002/03/owl#FunctionalProperty/premises001" >

  <owl:FunctionalProperty rdf:ID="prop"/>

  <rdf:Description rdf:ID="subject">
    <first:prop rdf:resource="#object1" />
    <first:prop rdf:resource="#object2" />
  </rdf:Description>
</rdf:RDF>
```

Sorgu 6.2 *owl:FunctionalProperty 001* numaralı sınaama durumu için sonuç.

```
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:owl="http://www.w3.org/2002/07/owl#"
  xml:base="http://www.w3.org/2002/03/owl#FunctionalProperty/conclusions001" >

  <rdf:Description rdf:about="premises001#object1">
    <owl:sameAs rdf:resource="premises001#object2" />
  </rdf:Description>
</rdf:RDF>
```

Sınaama Durumu 002

Sınaama Durumu 002'ye göre, örneğin *prop* niteliği *owl:FunctionalProperty* olarak tanımlanmışsa ve *prop* niteliğine ait iki farklı örneğin özneleri aynı ise, bu nitelik örneklerinin nesneleri de aynıdır. Bu nedenle bu nesnelerin biri hakkındaki bir ifadenin, diğer nesne için de doğru olduğu sonucuna varılır. *Sınaama Durumu 002*, *P* gerektirme kurallarından *rdfp1*, *rdfp5a* ve *rdfp11* kurallarına karşılık gelir.

Sorgu 6.3 *owl:FunctionalProperty 002* numaralı sinama durumu için öncül ontoloji.

```
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:owl="http://www.w3.org/2002/07/owl#"
  xmlns:eg="http://www.example.org/"
  xmlns:first="http://www.w3.org/2002/03/owl/FunctionalProperty/premises002#"
  xml:base="http://www.w3.org/2002/03/owl/FunctionalProperty/premises002" >

  <owl:FunctionalProperty rdf:ID="prop" />

  <rdf:Description rdf:ID="subject">
    <first:prop>
      <rdf:Description rdf:ID="object1">
        <eg:prop2>value</eg:prop2>
      </rdf:Description>
    </first:prop>
    <first:prop rdf:resource="#object2" />
  </rdf:Description>
</rdf:RDF>
```

Sorgu 6.4 *owl:FunctionalProperty 002* numaralı sinama durumu için sonuç.

```
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:owl="http://www.w3.org/2002/07/owl#"
  xml:base="http://www.w3.org/2002/03/owl/FunctionalProperty/conclusions001" >

  <rdf:Description rdf:about="premises001#object1">
    <owl:sameAs rdf:resource="premises001#object2" />
  </rdf:Description>
</rdf:RDF>
```

owl:InverseFunctionalProperty

Sinama Durumu 001

Sinama Durumu 001'e göre, örneğin *prop* niteliği *owl:InverseFunctionalProperty* olarak tanımlanmışsa ve *prop* niteliğine ait iki farklı örneğin nesneleri aynı ise, bu durumda bu nitelik örneklerinin öznelere de aynı olduğu sonucuna varılır. *Sinama Durumu 001*, *P* gerektirme kurallarından *rdfp2* kuralına karşılık gelir.

Sorgu 6.5 *owl:InverseFunctionalProperty 001* numaralı sınama durumu için öncül ontoloji.

```
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:owl="http://www.w3.org/2002/07/owl#"
  xmlns:first="http://www.w3.org/2002/03/owlt/InverseFunctionalProperty/premises001#"
  xml:base="http://www.w3.org/2002/03/owlt/InverseFunctionalProperty/premises001" >

  <owl:InverseFunctionalProperty rdf:ID="prop"/>

  <rdf:Description rdf:ID="subject1">
    <first:prop rdf:resource="#object" />
  </rdf:Description>

  <rdf:Description rdf:ID="subject2">
    <first:prop rdf:resource="#object" />
  </rdf:Description>
</rdf:RDF>
```

Sorgu 6.6 *owl:InverseFunctionalProperty 001* numaralı sınama durumu için sonuç.

```
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:owl="http://www.w3.org/2002/07/owl#"
  xml:base="http://www.w3.org/2002/03/owlt/InverseFunctionalProperty/conclusions001" >

  <rdf:Description rdf:about="premises001#subject1">
    <owl:sameAs rdf:resource="premises001#subject2" />
  </rdf:Description>
</rdf:RDF>
```

Sınama Durumu 002

Sınama Durumu 002'ye göre, örneğin *prop* niteliği *owl:InverseFunctionalProperty* olarak tanımlanmışsa ve *prop* niteliğine ait iki farklı örneğin nesneleri aynı ise, bu nitelik örneklerinin özneleri de aynıdır. Bu nedenle bu öznelerin biri hakkındaki bir ifadenin, diğer özne için de doğru olduğu sonucuna varılır. *Sınama Durumu 002*, *P* gerektirme kurallarından *rdfp2*, *rdfp5b* ve *rdfp11* kurallarına karşılık gelir.

Sorgu 6.7 *owl:InverseFunctionalProperty 002* numaralı sınama durumu için öncül ontoloji.

```
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:owl="http://www.w3.org/2002/07/owl#"
  xmlns:eg="http://www.example.org/"
  xmlns:first="http://www.w3.org/2002/03/owl#InverseFunctionalProperty/premises002#"
  xml:base="http://www.w3.org/2002/03/owl#InverseFunctionalProperty/premises002">

  <owl:InverseFunctionalProperty rdf:ID="prop"/>

  <rdf:Description rdf:ID="subject1">
    <first:prop rdf:resource="#object"/>
    <eg:prop2>value</eg:prop2>
  </rdf:Description>

  <rdf:Description rdf:ID="subject2">
    <first:prop rdf:resource="#object"/>
  </rdf:Description>
</rdf:RDF>
```

Sorgu 6.8 *owl:InverseFunctionalProperty 002* numaralı sınama durumu için sonuç.

```
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:eg="http://www.example.org/"
  xml:base="http://www.w3.org/2002/03/owl#InverseFunctionalProperty/conclusions002">

  <rdf:Description rdf:about="premises002#subject2">
    <eg:prop2>value</eg:prop2>
  </rdf:Description>
</rdf:RDF>
```

owl:SymmetricProperty

Sınama Durumu 001

Sınama Durumu 001'e göre, örneğin *path* niteliği *owl:SymmetricProperty* olarak tanımlanmışsa ve *Ghent*'ten *Antwerp*'e bir *path* bulunmakta ise, *Antwerp*'ten *Gent*'e de bir *path* bulunduğu sonucuna varılır. *Sınama Durumu 001*, *P* gerektirme kurallarından *rdfp3*'e karşılık gelir.

Sorgu 6.9 *owl:SymmetricProperty 001* numaralı sınama durumu için öncül ontoloji.

```
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns:owl="http://www.w3.org/2002/07/owl#"
  xmlns:first="http://www.w3.org/2002/03/owlt/SymmetricProperty/premises001#"
  xmlns:second="http://www.w3.org/2002/03/owlt/SymmetricProperty/conclusions001#"
  xml:base="http://www.w3.org/2002/03/owlt/SymmetricProperty/premises001" >

  <rdf:Description rdf:about="premises001#Ghent">
    <first:path rdf:resource="premises001#Antwerp"/>
  </rdf:Description>

  <owl:SymmetricProperty rdf:about="premises001#path"/>
</rdf:RDF>
```

Sorgu 6.10 *owl:SymmetricProperty 001* numaralı sınama durumu için sonuç.

```
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns:owl="http://www.w3.org/2002/07/owl#"
  xmlns:first="http://www.w3.org/2002/03/owlt/SymmetricProperty/premises001#"
  xmlns:second="http://www.w3.org/2002/03/owlt/SymmetricProperty/conclusions001#"
  xml:base="http://www.w3.org/2002/03/owlt/SymmetricProperty/conclusions001" >

  <rdf:Description rdf:about="premises001#Antwerp">
    <first:path rdf:resource="premises001#Ghent"/>
  </rdf:Description>
</rdf:RDF>
```

owl:TransitiveProperty

Sınama Durumu 001

Sınama Durumu 001'e göre, örneğin *path* niteliği *owl:TransitiveProperty* olarak tanımlanmışsa ve *Ghent*'ten *Antwerp*'e, *Antwerp*'ten *Amsterdam*'a birer *path* bulunmakta ise, *Ghent*'den *Amsterdam*'a bir *path* bulunduğu sonucuna varılır. *Sınama Durumu 001*, *P* gerektirme kurallarından *rdfp4*'e karşılık gelir.

Sorgu 6.11 *owl:TransitiveProperty 001* numaralı sınama durumu için öncül ontoloji.

```
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns:owl="http://www.w3.org/2002/07/owl#"
  xmlns:first="http://www.w3.org/2002/03/owl#TransitiveProperty/premises001#"
  xmlns:second="http://www.w3.org/2002/03/owl#TransitiveProperty/conclusions001#"
  xml:base="http://www.w3.org/2002/03/owl#TransitiveProperty/premises001" >

  <rdf:Description rdf:about="premises001#Antwerp">
    <first:path rdf:resource="premises001#Amsterdam" />
  </rdf:Description>

  <rdf:Description rdf:about="premises001#Ghent">
    <first:path rdf:resource="premises001#Antwerp" />
  </rdf:Description>

  <owl:TransitiveProperty rdf:about="premises001#path" />
</rdf:RDF>
```

Sorgu 6.12 *owl:TransitiveProperty 001* numaralı sınama durumu için sonuç.

```
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns:owl="http://www.w3.org/2002/07/owl#"
  xmlns:first="http://www.w3.org/2002/03/owl#TransitiveProperty/premises001#"
  xmlns:second="http://www.w3.org/2002/03/owl#TransitiveProperty/conclusions001#"
  xml:base="http://www.w3.org/2002/03/owl#TransitiveProperty/conclusions001" >

  <rdf:Description rdf:about="premises001#Ghent">
    <first:path rdf:resource="premises001#Amsterdam" />
  </rdf:Description>
</rdf:RDF>
```

owl:allValuesFrom

Sınama Durumu 001

Sınama Durumu 001'e göre r sınıfı, a kısıt sınıfının bir alt sınıfı ise, a kısıt sınıfı için p niteliğine ait tüm değerler c sınıfından gelmekte ise ve r sınıfına ait i örneğinin p değeri o olarak belirtilmişse, o örneğinin c sınıfına ait olduğu sonucuna varılır. *Sınama Durumu 001*, P gerektirme kurallarından *rdfp16*'ya karşılık gelir.

Sorgu 6.13 *owl:allValuesFrom 001* numaralı sinama durumu için öncül ontoloji.

```

<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns:owl="http://www.w3.org/2002/07/owl#"
  xmlns:first="http://www.w3.org/2002/03/owlt/allValuesFrom/premises001#"
  xml:base="http://www.w3.org/2002/03/owlt/allValuesFrom/premises001" >

  <owl:Class rdf:ID="r">
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty rdf:resource="#p"/>
        <owl:allValuesFrom rdf:resource="#c"/>
      </owl:Restriction>
    </rdfs:subClassOf>
  </owl:Class>

  <owl:ObjectProperty rdf:ID="p"/>

  <owl:Class rdf:ID="c"/>

  <first:r rdf:ID="i">
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#Thing"/>
    <first:p>
      <owl:Thing rdf:ID="o"/>
    </first:p>
  </first:r>
</rdf:RDF>

```

Sorgu 6.14 *owl:allValuesFrom 001* numaralı sinama durumu için sonuç.

```

<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:first="http://www.w3.org/2002/03/owlt/allValuesFrom/premises001#"
  xmlns:owl="http://www.w3.org/2002/07/owl#"
  xml:base="http://www.w3.org/2002/03/owlt/allValuesFrom/conclusions001" >

  <first:c rdf:about="premises001#o">
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#Thing"/>
  </first:c>

  <owl:Class rdf:about="premises001#c"/>
</rdf:RDF>

```

owl:equivalentClass

Sinama Durumu 001

Sinama Durumu 001'e göre iki sınıf eşitse, herhangi birine ait bir örnek aynı zamanda diğerine de aittir. *Sinama Durumu 001*, D^* gerektirme kurallarından *rdfs9*'a ve *P* gerektirme kurallarından *rdfp12a* ve *rdfp12b*'ye karşılık gelir.

Sorgu 6.15 *owl:equivalentClass 001* numaralı sınama durumu için öncül ontoloji.

```

<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns:owl="http://www.w3.org/2002/07/owl#"
  xmlns:first="http://www.w3.org/2002/03/owlt/equivalentClass/premises001#"
  xml:base="http://www.w3.org/2002/03/owlt/equivalentClass/premises001" >

  <owl:Class rdf:ID="Car">
    <owl:equivalentClass>
      <owl:Class rdf:ID="Automobile"/>
    </owl:equivalentClass>
  </owl:Class>

  <first:Car rdf:ID="car">
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#Thing" />
  </first:Car>

  <first:Automobile rdf:ID="auto">
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#Thing" />
  </first:Automobile>
</rdf:RDF>

```

Sorgu 6.16 *owl:equivalentClass 001* numaralı sınama durumu için sonuç.

```

<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns:owl="http://www.w3.org/2002/07/owl#"
  xmlns:first="http://www.w3.org/2002/03/owlt/equivalentClass/premises001#"
  xml:base="http://www.w3.org/2002/03/owlt/equivalentClass/conclusions001" >

  <first:Car rdf:about="premises001#auto">
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#Thing" />
  </first:Car>

  <first:Automobile rdf:about="premises001#car">
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#Thing" />
  </first:Automobile>

  <owl:Class rdf:about="premises001#Car"/>

  <owl:Class rdf:about="premises001#Automobile"/>
</rdf:RDF>

```

Sınama Durumu 002

Sınama Durumu 002'ye göre iki sınıf eşitse, bu iki sınıf aynı zamanda birbirinin alt sınıfıdır. *Sınama Durumu 002*, *P* gerektirme kurallarından *rdfp12a* ve *rdfp12b*'ye karşılık gelir.

Sorgu 6.17 *owl:equivalentClass 002* numaralı sınama durumu için öncül ontoloji.

```
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns:owl="http://www.w3.org/2002/07/owl#"
  xml:base="http://www.w3.org/2002/03/owl#equivalentClass/premises002" >

  <owl:Class rdf:ID="Car">
    <owl:equivalentClass>
      <owl:Class rdf:ID="Automobile"/>
    </owl:equivalentClass>
  </owl:Class>
</rdf:RDF>
```

Sorgu 6.18 *owl:equivalentClass 002* numaralı sınama durumu için sonuç.

```
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns:owl="http://www.w3.org/2002/07/owl#"
  xml:base="http://www.w3.org/2002/03/owl#equivalentClass/conclusions002" >

  <owl:Class rdf:about="premises002#Car">
    <rdfs:subClassOf>
      <owl:Class rdf:about="premises002#Automobile">
        <rdfs:subClassOf rdf:resource="premises002#Car" />
      </owl:Class>
    </rdfs:subClassOf>
  </owl:Class>
</rdf:RDF>
```

Sınama Durumu 003

Sınama Durumu 003'e göre, iki sınıf birbirinin alt sınıfı ise, bu iki sınıf aynı zamanda birbirine eşittir. *Sınama Durumu 003*, *P* gerektirme kurallarından *rdfp12c*'ye karşılık gelir.

Sorgu 6.19 *owl:equivalentClass 003* numaralı sınama durumu için öncül ontoloji.

```
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns:owl="http://www.w3.org/2002/07/owl#"
  xml:base="http://www.w3.org/2002/03/owl#equivalentClass/premises003" >

  <owl:Class rdf:about="conclusions003#Car">
    <rdfs:subClassOf>
      <owl:Class rdf:about="conclusions003#Automobile">
        <rdfs:subClassOf rdf:resource="conclusions003#Car" />
      </owl:Class>
    </rdfs:subClassOf>
  </owl:Class>
</rdf:RDF>
```

Sorgu 6.20 *owl:equivalentClass 003* numaralı sınama durumu için sonuç.

```
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns:owl="http://www.w3.org/2002/07/owl#"
  xml:base="http://www.w3.org/2002/03/owlt/equivalentClass/conclusions003" >

  <owl:Class rdf:ID="Car">
    <owl:equivalentClass>
      <owl:Class rdf:ID="Automobile"/>
    </owl:equivalentClass>
  </owl:Class>
</rdf:RDF>
```

owl:equivalentProperty

Sınama Durumu 001

Sınama Durumu 001'e göre örneğin *hasHead* niteliğinin *hasLeader* niteliğine eşit olduğu belirtilmişse, *hasLeader* niteliğine ait bir örnek aynı zamanda *hasHead* niteliğine de aittir. *Sınama Durumu 001*, D^* gerektirme kurallarından *rdfs7x*'e ve P gerektirme kurallarından *rdfp13b*'ye karşılık gelir.

Sorgu 6.21 *owl:equivalentProperty 001* numaralı sınama durumu için öncül ontoloji.

```
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns:owl="http://www.w3.org/2002/07/owl#"
  xmlns:first="http://www.w3.org/2002/03/owlt/equivalentProperty/premises001#"
  xml:base="http://www.w3.org/2002/03/owlt/equivalentProperty/premises001" >

  <owl:ObjectProperty rdf:ID="hasHead">
    <owl:equivalentProperty>
      <owl:ObjectProperty rdf:ID="hasLeader"/>
    </owl:equivalentProperty>
  </owl:ObjectProperty>

  <owl:Thing rdf:ID="X">
    <first:hasLeader>
      <owl:Thing rdf:ID="Y"/>
    </first:hasLeader>
  </owl:Thing>
</rdf:RDF>
```

Sorgu 6.22 *owl:equivalentProperty 001* numaralı sinama durumu için sonuç.

```

<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns:owl="http://www.w3.org/2002/07/owl#"
  xmlns:first="http://www.w3.org/2002/03/owlt/equivalentProperty/premises001#"
  xml:base="http://www.w3.org/2002/03/owlt/equivalentProperty/conclusions001" >

  <owl:Thing rdf:about="premises001#X">
    <first:hasHead>
      <owl:Thing rdf:about="premises001#Y"/>
    </first:hasHead>
  </owl:Thing>

  <owl:ObjectProperty rdf:about="premises001#hasHead"/>
</rdf:RDF>

```

Sinama Durumu 002

Sinama Durumu 002'ye göre iki nitelik eşitse, bu iki nitelik aynı zamanda birbirinin altniteliğidir. *Sinama Durumu 002*, *P* gerektirme kurallarından *rdfp13a* ve *rdfp13b*'ye karşılık gelir.

Sorgu 6.23 *owl:equivalentProperty 002* numaralı sinama durumu için öncül ontoloji.

```

<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns:owl="http://www.w3.org/2002/07/owl#"
  xmlns:first="http://www.w3.org/2002/03/owlt/equivalentProperty/premises002#"
  xml:base="http://www.w3.org/2002/03/owlt/equivalentProperty/premises002" >

  <owl:ObjectProperty rdf:ID="hasHead">
    <owl:equivalentProperty>
      <owl:ObjectProperty rdf:ID="hasLeader"/>
    </owl:equivalentProperty>
  </owl:ObjectProperty>
</rdf:RDF>

```

Sorgu 6.24 *owl:equivalentProperty 002* numaralı sinama durumu için sonuç.

```
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns:owl="http://www.w3.org/2002/07/owl#"
  xmlns:first="http://www.w3.org/2002/03/owlt/equivalentProperty/premises002#"
  xml:base="http://www.w3.org/2002/03/owlt/equivalentProperty/conclusions002" >

  <owl:ObjectProperty rdf:about="premises002#hasHead">
    <rdfs:subPropertyOf rdf:resource="premises002#hasLeader"/>
  </owl:ObjectProperty>

  <owl:ObjectProperty rdf:about="premises002#hasLeader">
    <rdfs:subPropertyOf rdf:resource="premises002#hasHead"/>
  </owl:ObjectProperty>
</rdf:RDF>
```

Sinama Durumu 003

Sinama Durumu 003'e göre, iki nitelik birbirinin altniteliği ise, bu iki nitelik aynı zamanda birbirine eşittir. *Sinama Durumu 003*, *P* gerektirme kurallarından *rdfp13c*'ye karşılık gelir.

Sorgu 6.25 *owl:equivalentProperty 003* numaralı sinama durumu için öncül ontoloji.

```
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns:owl="http://www.w3.org/2002/07/owl#"
  xml:base="http://www.w3.org/2002/03/owlt/equivalentProperty/premises003" >

  <owl:ObjectProperty rdf:about="conclusions003#hasHead">
    <rdfs:subPropertyOf rdf:resource="conclusions003#hasLeader"/>
  </owl:ObjectProperty>

  <owl:ObjectProperty rdf:about="conclusions003#hasLeader">
    <rdfs:subPropertyOf rdf:resource="conclusions003#hasHead"/>
  </owl:ObjectProperty>
</rdf:RDF>
```

Sorgu 6.26 *owl:equivalentProperty 003* numaralı sinama durumu için sonuç.

```
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns:owl="http://www.w3.org/2002/07/owl#"
  xml:base="http://www.w3.org/2002/03/owlt/equivalentProperty/conclusions003" >

  <owl:ObjectProperty rdf:ID="hasHead">
    <owl:equivalentProperty>
      <owl:ObjectProperty rdf:ID="hasLeader"/>
    </owl:equivalentProperty>
  </owl:ObjectProperty>
</rdf:RDF>
```

Sinama Durumu 006

Sinama Durumu 006'ya göre örneğin *hasHead* niteliğinin *hasLeader* niteliğine eşit olduğu belirtilmişse, *hasHead* niteliğine ait bir örnek aynı zamanda *hasLeader* niteliğine de aittir. *Sinama Durumu 006*, D^* gerektirme kurallarından *rdfs7x*'e ve P gerektirme kurallarından *rdfp13a*'ya karşılık gelir.

Sorgu 6.27 *owl:equivalentProperty 006* numaralı sinama durumu için öncül ontoloji.

```
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns:owl="http://www.w3.org/2002/07/owl#"
  xmlns:first="http://www.w3.org/2002/03/owlt/equivalentProperty/premises006#"
  xml:base="http://www.w3.org/2002/03/owlt/equivalentProperty/premises006" >

  <rdf:Description rdf:ID="hasHead">
    <owl:equivalentProperty>
      <rdf:Description rdf:ID="hasLeader"/>
    </owl:equivalentProperty>
  </rdf:Description>

  <rdf:Description rdf:ID="X">
    <first:hasLeader>
      <rdf:Description rdf:ID="Y"/>
    </first:hasLeader>
  </rdf:Description>
</rdf:RDF>
```

Sorgu 6.28 *owl:equivalentProperty 006* numaralı sinama durumu için sonuç.

```
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:first="http://www.w3.org/2002/03/owlt/equivalentProperty/premises006#"
  xml:base="http://www.w3.org/2002/03/owlt/equivalentProperty/conclusions006" >

  <rdf:Description rdf:about="premises006#X">
    <first:hasHead rdf:resource="premises006#Y"/>
  </rdf:Description>
</rdf:RDF>
```

owl:inverseOf

Sinama Durumu 001

Sinama Durumu 001'e göre örneğin *hasBrother* niteliği *isBrotherOf* niteliğinin ters rolü olarak olduğu belirtilmişse, *hasBrother* niteliği ile bağlanmış bir özne-nesne ikilisinin, aynı zamanda nesne-özne ikilisi olarak *isBrotherOf* niteliği

ile de bağlanmış olduğu sonucuna varılır. *Sinama Durumu 001*, *P* gerektirme kurallarından *rdfp8ax*'e karşılık gelir.

Sorgu 6.29 *owl:inverseOf 001* numaralı sinama durumu için öncül ontoloji.

```
<rdf:RDF
  xmlns:my="http://example.net/myVocab#"
  xmlns:eg="http://example.net/vocab#"
  xmlns:owl="http://www.w3.org/2002/07/owl#"
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xml:base="http://www.w3.org/2002/03/owlt/inverseOf/premises001"
  xmlns:your="http://example.net/yourVocab#">

  <rdf:Description rdf:about="http://example.net/myVocab#hasBrother">
    <owl:inverseOf rdf:resource="http://example.net/yourVocab#isBrotherOf"/>
  </rdf:Description>

  <rdf:Description rdf:about="http://example.net/vocab#joe">
    <my:hasBrother rdf:resource="http://example.net/vocab#bob"/>
  </rdf:Description>
</rdf:RDF>
```

Sorgu 6.30 *owl:inverseOf 001* numaralı sinama durumu için sonuç.

```
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xml:base="http://www.w3.org/2002/03/owlt/inverseOf/conclusions001"
  xmlns:your="http://example.net/yourVocab#">

  <rdf:Description rdf:about="http://example.net/vocab#bob">
    <your:isBrotherOf rdf:resource="http://example.net/vocab#joe"/>
  </rdf:Description>
</rdf:RDF>
```

owl:sameAs

Sinama Durumu 001

Sinama Durumu 001'e göre iki sınıf örneği birbirine eşitse, birine ait açıklama diğeri için de geçerlidir. *Sinama Durumu 001*, *P* gerektirme kurallarından *rdfp11*'e karşılık gelir.

Sorgu 6.31 *owl:sameAs 001* numaralı sınama durumu için öncül ontoloji.

```

<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns:owl="http://www.w3.org/2002/07/owl#"
  xmlns:first="http://www.w3.org/2002/03/owlt/sameAs/premises001#"
  xml:base="http://www.w3.org/2002/03/owlt/sameAs/premises001" >

  <owl:Class rdf:ID="c1">
    <owl:sameAs>
      <owl:Class rdf:ID="c2"/>
    </owl:sameAs>
    <first:annotate>description of c1</first:annotate>
  </owl:Class>

  <owl:AnnotationProperty rdf:ID="annotate" />
</rdf:RDF>

```

Sorgu 6.32 *owl:sameAs 001* numaralı sınama durumu için sonuç.

```

<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns:owl="http://www.w3.org/2002/07/owl#"
  xmlns:first="http://www.w3.org/2002/03/owlt/sameAs/premises001#"
  xml:base="http://www.w3.org/2002/03/owlt/sameAs/conclusions001" >

  <owl:Class rdf:about="premises001#c2">
    <first:annotate>description of c1</first:annotate>
  </owl:Class>

  <owl:AnnotationProperty rdf:about="premises001#annotate" />
</rdf:RDF>

```

7 Deneyler

Tez kapsamında sunulan çıkarsama yaklaşımının işlevsel doğruluğunu ispatlamak ve performansını değerlendirmek üzere birtakım deneyler gerçekleştirilmiştir. Bu deneylerde, LUBM (Lehigh University Benchmark) (Guo et al., 2004) ve UOBM (University Ontology Benchmark) (Ma et al., 2006) adlı iki test uygulaması kullanılmıştır. Bölüm 7.2 ve Bölüm 7.3, bu iki test uygulamasına ait veri setlerini ve sorguları anlatmakta ve iki uygulama arasındaki farkları özetlemektedir. Bölüm 7.1, deneylerde kullanılan Rete tabanlı standart (eniyilenmemiş) çıkarsama motorunun detaylarını anlatmaktadır.

Deneylerde 2 GB RAM ve Core 2 Duo T7700@2.4 GHz işlemciye sahip dizüstü bilgisayarlar kullanılmıştır. Bölüm 7.4, deneylerde gözönünde bulundurulan başarımların değerlendirme ölçütlerini tanımlamaktadır. Bölüm 7.5'te deney sonuçları verilmekte ve bu sonuçlar değerlendirilmektedir.

7.1 Standart Çıkarsama Motoru

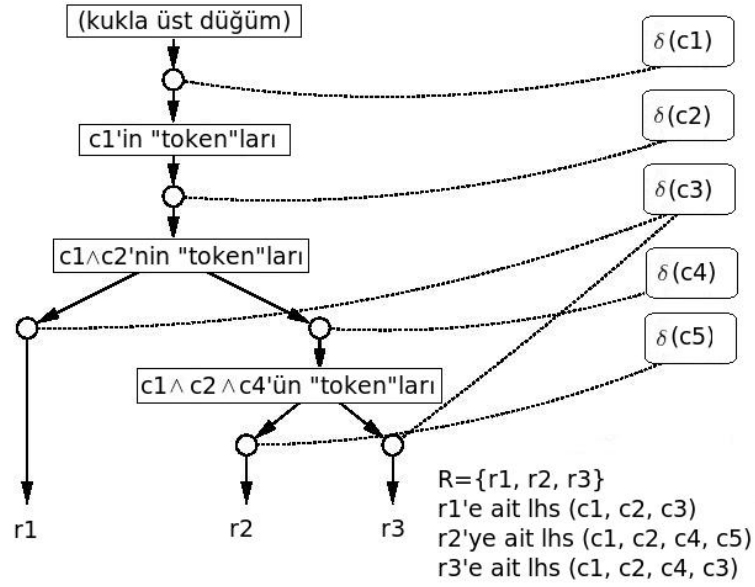
Bu tezde kullanılan standart çıkarsama motoru, OWL ontolojilerindeki örtük bilginin, kurallar aracılığı ile belirtik hale getirilmesi için Rete algoritmasını (Forgy, 1982) kullanmaktadır. Rete, eniyilenmiş bir ileriye doğru çıkarsama algoritmasıdır. İleriye doğru çıkarsama algoritması, kuralları uygulayarak yeni olgular elde eder ve ontolojiye bir olgu eklendiği veya ontolojiden bir olgu çıkartıldığı anda, algoritma neredeyse bir önceki döngüdekiler ile aynı sonuçları üretmek üzere en baştan başlar. Rete, önceden bulunmuş sonuçları hatırlayarak, bunların tekrar hesaplanmasını önler. Rete yalnızca silinen ve yeni eklenen olguları kurallarla test ederek performansı önemli ölçüde artırır.

Rete algoritması, Rete ağının çıkarsama sürecine dayalıdır. İzleyen tanımlar, Rete ağındaki kavramların biçimsel gösterimlerini sunmaktadır. U sabitler kümesi, $W = \{w \mid w = (s, p, o) \wedge s, p, o \in U\}$ ontolojideki tüm olguların kümesi, R ontolojiye ait tüm kurallar kümesi ve $O = (W, R)$ ontoloji ve $w \in W$ olmak üzere her bir w , diğer bir deyişle her bir olgu, bir özne (s), bir yüklem (p) ve bir nesneden (o) oluşur. $r \in R$ ve lhs ve rhs atom listeleri olmak üzere, $r = (lhs, rhs)$ 'dir. Bir

atom, $at = (a, i, v)$, bir öznitelik a , bir tanıtıcı i ve bir değişkenden v oluşur ve $a, i, v \in T$ ve V değişkenler kümesi olmak üzere $T = U \cup V$ 'dir. Bir lhs atomu koşul olarak adlandırılır. C , tüm kurallara ait tüm koşulların bulunduğu kümedir. Bir ontolojiye ait Rete ağında $\Omega(O) = (\alpha, \beta)$, α "alpha" ağını, β ise "beta" ağını belirtir. $\alpha = \delta(c) \mid c \in C$ ve $D = \{x \mid x \subseteq W\}$ olmak üzere, $\delta : C \rightarrow D$ bir fonksiyondur. $\delta(c)$, c koşulu ile eşleşen tüm olguları içeren kümeyi, diğer bir deyişle $\{w \mid w = (s, p, o) \wedge c = (a, i, v) \wedge ((s = a) \vee (a \in V)) \wedge ((p = i) \vee \neg (i \in V)) \wedge ((o = v) \vee (v \in V))\}$ kümesini döndürür. β ağı "beta" belleklerden ve "join" düğümlerden oluşur. "Beta" bellekler "token" adı verilen, kuralların kısmi gerçekleşmelerini saklarken, "join" düğümler kural koşulları arasındaki değişken bağlamalarının tutarlılığını test ederler. Şekil 7.1, ağdaki kavramlar arası ilişkileri göstermektedir. Şekilde, dikdörtgenler "beta" bellek düğümlerini, ovaler "alpha" bellekleri ve daireler "join" düğümleri göstermektedir.

$$S = \{x \mid x = (c_1 \wedge \dots \wedge c_n) \wedge r = (lhs, rhs) \wedge lhs = (c_1, \dots, c_t) \wedge (1 \leq n \leq t) \wedge r \in R\},$$

ve $\varphi : S \rightarrow I$ bir fonksiyon olmak üzere, $PI = \{\varphi(s) \mid s \in S\}$ bütün "beta" bellekleri içeren kümedir. $\varphi(s)$, s ile eşleşen olguların bağlamalı setlerini döndürür. Rete ağının sonundaki gerçekleşmeler, "production" düğümleri, diğer bir deyişle, "p-node", olarak ele alınırlar. $P = \{\varphi(s) \mid s = (c_1 \wedge \dots \wedge c_t) \wedge r = (lhs, rhs) \wedge lhs = (c_1, \dots, c_t) \wedge r \in R\}$ kümesi, tüm "p-node"ları içeren kümedir. Bir yayılımın Rete ağının sonuna varması, diğer bir deyişle bir "p-node"un aktive edilmesi, bir kuralın tüm koşullarının tamamıyla eşleştiğini ve kuralın sağ el tarafındaki (rhs) atomların ontolojiye eklenecek yeni bir olgu ürettiğini gösterir. Ontolojiye yeni bir olgunun eklenmesi, Rete ağını tetikler ve önceden bulunmuş olguların tekrar hesaplanmasını gerektirmeden, bu yeni eklenen olgulara bağlı yeni olgular çıkarır.



Şekil 7.1. Örnek bir Rete ağı.

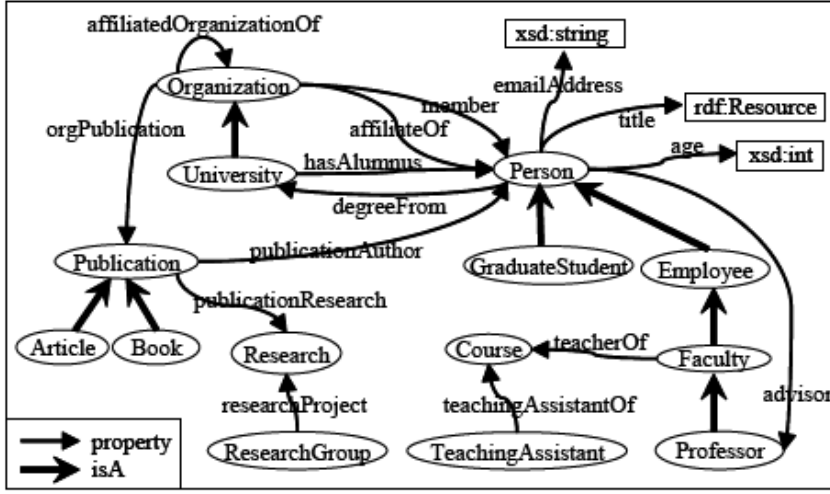
Çıkarsama aracına bir sorgu gönderildiğinde, bu sorgu Rete ağına, sağ el tarafı (*lhs*) boş olan bir "p-node" olarak eklenir. Bu "production" düğümünün elementleri, kullanıcıya sorgu sonucu olarak döndürülürler. Sonuç döndürüldükten sonra, bu "production" düğümü ve "production" düğümünün ebeveyn düğümleri Rete ağından silinir.

7.2 LUBM Değerlendirme Uygulaması

LUBM değerlendirme uygulaması, OWL ontoloji depolarının performanslarını değerlendirmek için sunulmuştur ve aşağıdaki bileşenlerden oluşur:

- **univ-bench (ub) adı verilen bir OWL şeması.** Şekil 7.2'de bu şemanın bir kısmı görülmektedir. Bu şema üniversiteleri, bölümleri ve üniversitelerde yürütülen etkinlikleri modellemek için ortaya konmuştur.
- **ub şemasına ait yapay örnek verileri üreten bir bileşen.** Deneylerde $LUBM(x, 0)$ ifadesi, ub şemasının, x adet üniversiteye ait örnek verisiyle popülasyonu sonucu oluşan veri setini göstermektedir. Çizelge 7.1, deneylerde kullanılan 1, 5, 10, 20 ve 50 üniversiteye ait veri setlerinin boyutlarını üçlülerin sayısı cinsinden göstermektedir.

- *SPARQL* ile ifade edilmiş ondört sorgudan oluşan bir sorgu seti.



Şekil 7.2. Univ-Bench şeması.

1	5	10	20	50
100880	625105	1273260	2689157	6656867

Çizelge 7.1. 1, 5, 10, 20 ve 50 üniversiteye ait veri setlerinin boyutları.

Aşağıda LUBM değerlendirme uygulamasına ait 14 sorgunun analizi sunulmaktadır. Bu sorgular 5 kriter gere göre analiz edilmektedir:

Girdi boyu Sorguyla ilişkili sınıf örneklerinin, değerlendirme uygulamasındaki toplam sınıf örneği sayısına oranıdır. Oran %5'ten büyük olduğunda girdi boyunun büyük olduğu, diğer durumlarda girdi boyunun küçük olduğu belirtilmiştir.

Seçicilik Sorgu kriterlerini karşılayan sınıf örneklerinin oranını belirtir. Bu değer %10'dan küçük olduğunda seçicilik yüksek, düşük olduğunda seçicilik azdır.

Karmaşıklık Sorguyla ilişkili sınıf ve nitelik sayısını belirtir.

Hiyerarşi bilgisi gereksinimi Sorgu yanıtının eksiksiz olması için sınıf ve nitelik hiyerarşisi bilgisine gereksinim duyulup duyulmadığını belirtir.

Çıkarıma gereksinimi Sorgu yanıtının eksiksiz olması için çıkarılamaya gereksinim duyulup duyulmadığını belirtir.

Sorgu-1: Girdi boyu büyük, seçiciliği yüksek bir sorgudur. Yalnızca bir sınıf ve bir nitelik ile ilişkilidir. Hiyerarşi bilgisine veya çıkarsamaya gereksinim duymaz.

Sorgu 7.1 LUBM Sorgu 1.

```
(?X type GraduateStudent)
(?X takesCourse http://www.Department0.University0.edu/GraduateCourse0)
```

Sorgu-2: Bir önceki sorguya göre daha karmaşıktır. 3 sınıf ve 3 nitelik ile ilişkilidir.

Sorgu 7.2 LUBM Sorgu 2.

```
(?X type GraduateStudent ?X)
(?Y type University ?Y)
(?Z type Department ?Z)
(?X memberOf ?Z)
(?Z subOrganizationOf ?Y)
(?X undergraduateDegreeFrom ?Y)
```

Sorgu-3: Bu sorgu, *Sorgu-1*'e benzemektedir fakat *Publication* sınıfı geniş bir hiyerarşiye sahiptir.

Sorgu 7.3 LUBM Sorgu 3.

```
(?X type Publication)
(?X publicationAuthor http://www.Department0.University0.edu/AssistantProfessor0)
```

Sorgu-4: Girdi boyu küçük, seçiciliği yüksek bir sorgudur. *Professor* sınıfı ve alt-sınıfları arasındaki hiyerarşi bilgisine gereksinim duyar. *Professor* sınıfı geniş bir hiyerarşiye sahiptir. Bu sorgunun başka bir özelliği tek bir sınıfa ait birçok niteliği sorgulamasıdır.

Sorgu 7.4 LUBM Sorgu 4.

```
(?X type Professor)
(?X worksFor http://www.Department0.University0.edu)
(?X name ?Y1)
(?X emailAddress ?Y2)
(?X telephone ?Y3)
```

Sorgu-5: Bu sorgu *Person* sınıfı ve alt sınıfları arasındaki hiyerarşiye ve *memberOf* ve alt nitelikleri arasındaki hiyerarşiye gereksinim duyar. *Person* sınıfı derin ve geniş bir hiyerarşiye sahiptir.

Sorgu 7.5 LUBM Sorgu 5.

```
(?X type Person)
(?X memberOf http://www.Department0.University0.edu)
```

Sorgu-6: Bu sorgu yalnızca tek bir sınıfla ilişkilidir. *UndergraduateStudent* ve *Student* sınıfları arasındaki dolaysız *subClassOf* ilişkisi ile *GraduateStudent* ve *Student* arasındaki dolaylı *subClassOf* ilişkisine gereksinim duyar. Büyük girdi boyu ve düşük seçiciliği olan bir sorgudur.

Sorgu 7.6 LUBM Sorgu 6.

```
(?X type Student)
```

Sorgu-7: *Student* sınıfı ile ilişkili olmasından dolayı *Sorgu-6*'ya benzeyen fakat daha karmaşık ve seçiciliği yüksek bir sorgudur.

Sorgu 7.7 LUBM Sorgu 7.

```
(?X type Student)
(?Y type Course)
(http://www.Department0.University0.edu/AssociateProfessor0 teacherOf ?Y)
(?X takesCourse ?Y)
```

Sorgu-8: *Sorgu-7*'ye göre daha karmaşık bir sorgudur. Çünkü ilişkili olduğu nitelik sayısını bir fazladır.

Sorgu 7.8 LUBM Sorgu 8.

```
(?X type Student)
(?Y type Department)
(?X memberOf ?Y)
(?Y subOrganizationOf http://www.University0.edu)
(?X emailAddress ?Z)
```

Sorgu-9: İlişkili olduğu sınıf ve nitelik sayısı fazla olan, diğer bir deyişle, karmaşıklığı yüksek bir sorgudur.

Sorgu 7.9 LUBM *Sorgu 9.*

```
(?X type Student)
(?Y type Faculty)
(?Z type Course)
(?X advisor ?Y)
(?X takesCourse ?Z)
(?Y teacherOf ?Z)
```

Sorgu-10: Bu sorgu, *Sorgu-6*, *Sorgu-7*, *Sorgu-8* ve *Sorgu-9*'dan farklı olarak, yalnızca *GraduateStudent* ve *Student* sınıfları arasındaki dolaylı *subClassOf* bilgisini gerektirir. Örneğin, *UndergraduateStudent* ve *Student* sınıfları arasındaki dolaysız *subClassOf* ilişkisine gereksinim duymaz.

Sorgu 7.10 LUBM *Sorgu 10.*

```
(?X type Student)
(?X takesCourse http://www.Department0.University0.edu/GraduateCourse0)
```

Sorgu-11: Sorgu 11, 12 ve 13, sistemdeki belirli *OWL* çıkarsama yeteneklerini sınamak üzere sunulmuşlardır. *subOrganizationOf* niteliği geçişken bir nitelik olarak tanımlanmıştır. *ResearchGroup* sınıfı *Department* sınıfı ile *subOrganizationOf* niteliği ile ilişkilendirildiğinden ve *Department* sınıfı *University* sınıfı ile *subOrganizationOf* niteliği ile ilişkilendirildiğinden, bu sorguyu yanıtlayabilmek için *ResearchGroup* ve *University* sınıflarına ait örnekler arası *subOrganizationOf* ilişkisinin çıkarsanması gerekmektedir. Bu sorgunun girdi boyu küçüktür.

Sorgu 7.11 LUBM *Sorgu 11.*

```
(?X type ResearchGroup)
(?X subOrganizationOf http://www.University0.edu)
```

Sorgu-12: Test uygulaması tarafından üretilen örnek verilerinde *Chair* sınıfına ait bir örnek yoktur. *Department* sınıfı *Chair* sınıfına *headOf* niteliği ile bağlanmıştır. Bir *Department* örneğinin *headOf* niteliği ile bağlı olduğu sınıf örneklerinin *Chair* sınıfına ait örnekler olduğu çıkarsamasına gereksinim duyulur. Bu sorunun da girdi boyu küçüktür.

Sorgu 7.12 LUBM Sorgu 12.

```
(?X type Chair)
(?Y type Department)
(?X worksFor ?Y)
(?Y subOrganizationOf http://www.University0.edu)
```

Sorgu-13: Test uygulamasına ait şemada, *hasAlumnus* niteliği *degreeFrom* niteliğinin tersi olarak tanımlanmıştır. *degreeFrom* niteliğinin üç altniteliği vardır: *undergraduateDegreeFrom*, *mastersDegreeFrom* ve *doctoralDegreeFrom*. Test uygulaması bir kişinin bir üniversitenin mezunu olduğunu, *hasAlumnus* ilişkisi yerine bu üç altniteliği kullanarak belirtmektedir. Bu nedenle bu sorgu, *degreeFrom* niteliğine ait üç altniteliğin de gözönünde bulundurularak, *owl:inverseOf* yapıtaşıyla ilgili çıkarsamanın yapılmasını gerektirmektedir.

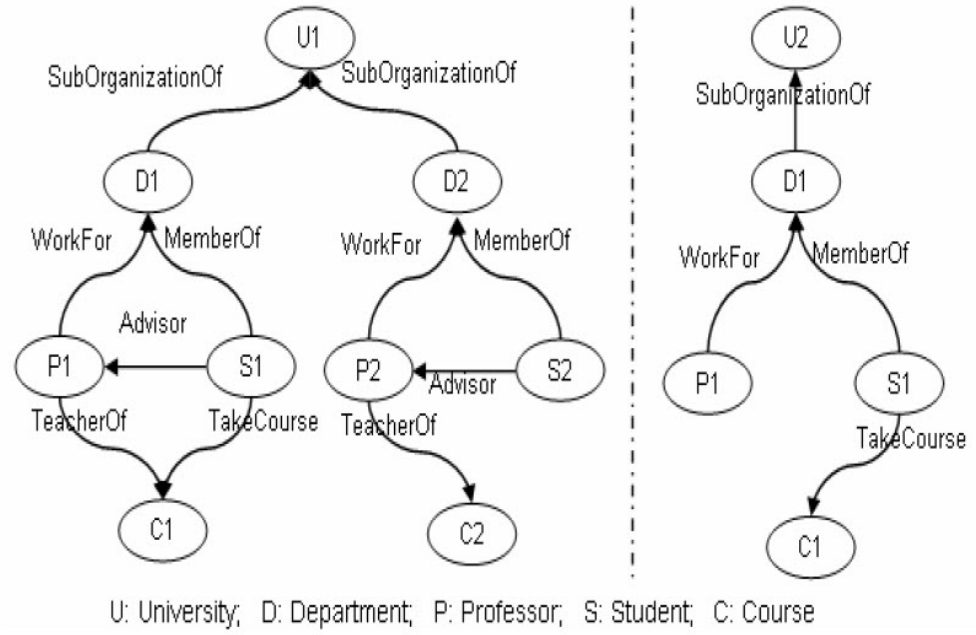
Sorgu 7.13 LUBM Sorgu 13.

```
(?X type Person)
(http://www.University0.edu hasAlumnus ?X)
```

Sorgu-14: Bu sorgu, değerlendirme uygulamasındaki en basit sorgudur. Büyük girdi boyuna ve düşük seçiciliğe sahiptir. Hiçbir hiyerarşi bilgisi ve çıkarsama gereksinimi yoktur.

Sorgu 7.14 LUBM Sorgu 14.

```
(?X type UndergraduateStudent)
```

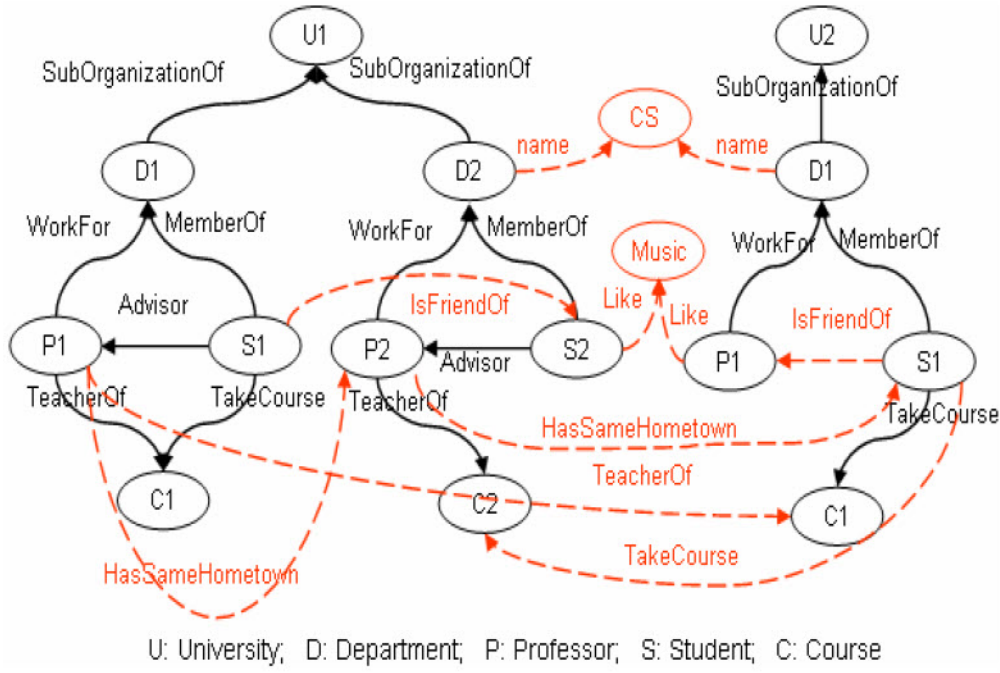


Şekil 7.3. LUBM şeması.

7.3 UOBM Değerlendirme Uygulaması

LUBM test uygulamasını, çıkarsama ve ölçeklendirme açılarından daha eksiksiz hale getirerek, varolan ontoloji sistemlerini daha iyi değerlendirebilmek için UOBM test uygulaması geliştirilmiştir (Ma et al., 2006). Şekil 7.3 ve Şekil 7.4, LUBM ve UOBM test uygulamalarına ait şemaların, dolayısıyla şemalara ait örnek verilerinin, aralarındaki farkları göstermektedir.

Çizelge 7.2’de LUBM ve UOBM test uygulamaları, sınıf, nitelik ve üniversite başına örnek sayısı açısından değerlendirilmektedir. Örnek tanımlamak üzere oluşturulan sınıfların ve niteliklerin sayıları parantez içerisinde belirtilmektedir. Çizelgeden de anlaşılacağı üzere, UOBM test uygulamasında 131 sınıf ve nitelik, yalnızca TBox’taki sınıf ve nitelik hiyerarşilerini tanımlamak için kullanılmaktadır. Çizelgeden okunan başka bir sonuç, UOBM test uygulamasının daha büyük ve karmaşık bir örnek çizgesi oluşturabildiğidir. UOBM’nin LUBM’den diğer bir farkı da bir veri kümesi üretme uygulamasının bulunmayışındır. Başka bir deyişle, UOBM’nin veri kümeleri 1, 5 ve 10 üniversiteden oluşur ve sabittir.



Şekil 7.4. UOBM şeması.

	LUBM	UOBM
Sınıf Sayısı	43 (22)	51 (41)
Datatype Nitelik sayısı	7 (3)	9 (5)
Objecttype Nitelik sayısı	25 (14)	34 (24)
Üniversite başına nitelik örneği sayısı	90000-110000	210000-250000
Üniversite başına örnek sayısı	8000-15000	10000-20000

Çizelge 7.2. LUBM ve UOBM test uygulamalarına ait sınıf, nitelik ve üniversite başına örnek sayıları.

Aşağıda UOBM test uygulamasına ait sorgular listelenmektedir. UOBM test uygulamasına ait sorguları değerlendirirken sorguların tek tek özelliklerini incelemek yerine LUBM sorgularından farklı olan kısımlarını vurgulamak yerinde olacaktır, çünkü sorgular LUBM sorguları temel alınarak üretilmiştir. UOBM sorgularının iki temel farkı bulunmaktadır:

1. Sorgular birden fazla üniversiteyi kapsayacak şekilde genişletilmiştir. Böylece, değerlendirme uygulamasının sonuçlarının ölçeklenebilirlik açısından daha anlamlı olmasına çalışılmıştır. Bunun gerçekleştirilebilmesi için veri kümesinde üniversiteler arasındaki bağlantılar arttırılmıştır.

2. Her bir sorgunun farklı bir OWL çıkarsamasını sınaması sağlanmıştır.

Sorgu-1:

Sorgu 7.15 UOBM *Sorgu 1.*

```
(?X type UndergraduateStudent)
(?X takesCourse http://www.Department0.University0.edu/Course0)
```

Sorgu-2:

Sorgu 7.16 UOBM *Sorgu 2.*

```
(?X type Employee)
```

Sorgu-3:

Sorgu 7.17 UOBM *Sorgu 3.*

```
(?X type Student)
(?X isMemberOf http://www.Department0.University0.edu)
```

Sorgu-4:

Sorgu 7.18 UOBM *Sorgu 4.*

```
(?X type Publication)
(?X publicationAuthor ?Y)
(?Y type Faculty)
(?Y isMemberOf http://www.Department0.University0.edu)
```

Sorgu-5:

Sorgu 7.19 UOBM *Sorgu 5.*

```
(?X type ResearchGroup)
(?X subOrganizationOf http://www.University0.edu)
```

Sorgu-6:

Sorgu 7.20 UOBM *Sorgu 6.*

(?X type Person)
 (http://www.University0.edu hasAlumnus ?X)

Sorgu-7:

Sorgu 7.21 UOBM *Sorgu 7.*

(?X type Person)
 (?X hasSameHomeTownWith http://www.Department0.University0.edu/FullProfessor0)

Sorgu-8:

Sorgu 7.22 UOBM *Sorgu 8.*

(?X type SportsLover)
 (http://www.Department0.University0.edu hasMember ?X)

Sorgu-9:

Sorgu 7.23 UOBM *Sorgu 9.*

(?X type GraduateCourse)
 (?X isTaughtBy ?Y)
 (?Y isMemberOf ?Z)
 (?Z subOrganizationOf http://www.University0.edu)

Sorgu-10:

Sorgu 7.24 UOBM *Sorgu 10.*

(?X isFriendOf http://www.Department0.University0.edu/FullProfessor0)

Sorgu-11:

Sorgu 7.25 UOBM *Sorgu 11.*

(?X type Person)
 (?X like ?Y)
 (?Z type Chair)
 (?Z isHeadOf http://www.Department0.University0.edu)
 (?Z like ?Y)

Sorgu-12:

Sorgu 7.26 UOBM *Sorgu 12.*

(?X type Student)
 (?X takesCourse ?Y)
 (?Y isTaughtBy http://www.Department0.University0.edu/FullProfessor0)

Sorgu-13:

Sorgu 7.27 UOBM *Sorgu 13.*

(?X type PeopleWithHobby)
 (?X isMemberOf http://www.Department0.University0.edu)

7.4 Başarım Değerlendirme Ölçütleri

Bir çıkarsama motorunun başarımının değerlendirme ölçütleri temel olarak dört maddede toplanabilir:

- *Yükleme süresi*
- *Çıkarsama süresi*
- *Bellek ve disk kullanımı*
- *Sorgu süresi*

Yükleme süresi bir ontolojinin herhangi bir biçimdeki dosyadan okunarak bellekteki veri yapısına aktarılması sırasında geçen zamandır. Yüklem başarıımı ve süresi, çıkarsama motorunun kendisi dışında kalan bileşenlere de bağlıdır. Ontoloji dosyası ayrıştırıcısı, ontolojinin bellekte saklanması için kullanılan veri yapısı bunlar arasında sayılabilir.

Çıkarsama süresi, çıkarsama motoruna kuralların ve olguların yüklenmesi ve bunlar üzerinde çıkarsamanın tamamlanması için gereken sürenin tamamıdır. Burada başarıımı etkileyen temel faktör tek bir üçlünün çıkarsama motoruna eklenme hızıdır. Birçok testte *yükleme süresi* ve *çıkarsama süresinin* toplamına *açılış süresi* denmektedir. Bunun nedeni başarımın arttırılabilmesi için yapılan eniyilemelerin,

yükleme ve çıkarsama aşamaları arasındaki ayrımı ortadan kaldırmasıdır. Bu tezde de açılış süresinin, *yükleme süresi* ve *çıkarsama süresinin* toplamına eşit olduğu kabul edilmektedir.

Ontoloji açılış performansını etkileyen üç neden çıkarsamaya katılan üçlülerin sayısı (η_{tpi}), çıkarsanan üçlülerin sayısı (η_{it}) ve toplam üçlü sayısı (η_t) olarak listelenebilir. Bu nedenle, bu deneyde iki ölçüt kullanılacaktır: η_{tpi}/η_t ve η_{it}/η_t . Açılış başarımını değerlendirmek için kullanılabilecek bir diğer ölçüt saniyede yüklenen üçlü sayısıdır (η_{tps}) (Kiryakov, 2008).

Bellek ve disk kullanımı çıkarsama motorunun yükleyebileceği ontolojilerin miktarını belirleyeceğinden, başarımlar açısından önemlidir. Ontolojiler büyüdükçe bu ontolojileri açabilmek için gereken bellek ve/veya disk kullanımı da aynı oranda hatta daha yüksek bir oranda artmaktadır. Bu nedenle, daha az bellek kullanarak veya aynı zamanda disk üzerinde ontolojileri açarak çıkarsama yapabilmek giderek daha önemli bir hale gelmektedir.

Bellek ve disk kullanımını etkileyen faktörler, açılış performansını da etkilemektedir. Bu nedenler, çıkarsamaya katılan üçlülerin sayısı (η_{tpi}), çıkarsanan üçlülerin sayısı (η_{it}) ve toplam üçlü sayısının (η_t) yanısıra bu üçlülerin üzerinde yaratılan indekslerin boyutunun da eklenmesiyle oluşan veritabanının boyutu (η_{vtb}) ve kullanılan bellek boyutu (η_{bb}) olarak sıralanabilir.

Sorgu süresi, çıkarsama motorunun sorguları ne kadar zamanda çalıştırdığı ile ilgilidir. Anlamsal web için uygulama geliştirirken, kullanıcıların yaptığı sorgulamaların kabul edilebilir bir zaman aralığında cevap dönmesi önemlidir. Sorgu süresi için kullanılabilecek temel birim *sonuç başına düşen sorgu işletme süresidir* (q_{tpr}). Aynı zamanda bunun ontoloji boyutuna olan oranı da kullanılabilir.

7.5 Deney Sonuçları

Bu bölümde, Bölüm 7.4'te verilen ölçütler kullanılarak üç farklı değerlendirme deneyine ait sonuçlar verilmektedir. İlk deney (Bölüm 7.5.1) yaklaşımı uygulamadan önce (standart çıkarsama motoru) ve yaklaşımı uyguladıktan sonra (eniyeleştirilmiş çıkarsama motoru) elde edilen performansları değerlendirmektedir. Bu deneyde amaç, çıkarsama motorunun salt başarımını göstermekten çok yöntemin uygulanmasının çıkarsama motorunun başarımına olan etkilerini göstermek-

tir. Bu deneyde LUBM(1,0) veri seti ve bu veri setinin alt kümeleri kullanılmaktadır. Bunun nedeni standart çıkarsama motorunun bundan daha büyük veri setlerini açamamasıdır. Standart çıkarsama motorunun başarımı Jess (Friedman-Hill, 2002) ile paralellik göstermektedir.

İlk deney, standart ve eniyilenmiş uygulama arasındaki başarımların farklarını açılış süresi (yükleme süresi + çıkarsama süresi), bellek ve disk kullanımı ve sorgu süresi başarımlarını değerlendirme ölçütleri doğrultusunda incelemektir. Eniyilenmiş yöntem, standart yönteme ek olarak model dönüşüm işlemi gerçekleştirmektedir. Dönüştürme işlemi çıkarsamaya yardımcı bir işlem olduğu için, dönüştürme süresi çıkarsama süresine eklenmiştir.

İkinci deneyde ise yöntemin ölçeklenebilirliğini görmek açısından, LUBM değerlendirme verisi 1, 5, 10, 20 ve 50 üniversitelerle eniyilenmiş çıkarsama motoru kullanılarak açılmıştır. Ayrıca, bu bölümde LUBM değerlendirme verisine ait tüm sorgular çalıştırılarak, eksiksiz ve doğru sorgu sonuçları elde edilmiştir.

Üçüncü deney, UOBM değerlendirme uygulamasına ait farklı veri setleri (bir üniversiteye ait 4, 8 ve 12 bölüme ait veri setleri), standart ve eniyilenmiş çıkarsama motoru kullanılarak açılmıştır. Bu deneyin amacı yöntemin daha karmaşık ontoloji şemalarındaki davranışını incelemektir.

Dördüncü ve son deney ise, UOBM değerlendirme uygulamasına ait 1, 5 ve 10 üniversitelerle eniyilenmiş çıkarsama motorunun performansını değerlendirmektedir. UOBM değerlendirme uygulaması 1, 5 ve 10 üniversitelerle sabit veri setleri içerdiği için, bu deneyde yalnızca bu veri kümeleri kullanılmıştır.

7.5.1 Birinci Deney

Standart uygulamanın açılış süreleri Çizelge 7.3'te verilirken, eniyilenmiş uygulama için açılış süreleri Çizelge 7.4'te verilmiştir. Her iki deneyde de aynı ayrıştırıcı kullanılmıştır. Ayrıştırma başarımındaki fark, yöntemlerin ayrıştırılan veriyi saklama mantıkları arasındaki farktan kaynaklanmaktadır. Standart yöntem veriyi çıkarsama motorunun kullandığı veri yapısında saklarken, eniyilenmiş yöntem veritabanı teknolojilerinden faydalanmaktadır.

Her iki çizelgeye ait değerler arasındaki fark, yöntemin açılış başarımına

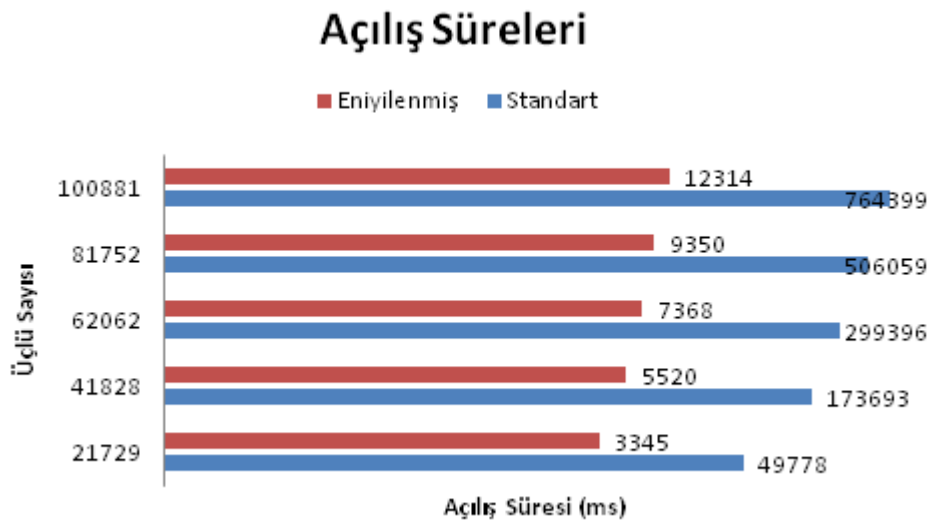
	Toplam Üçlü Sayısı				
	21729	41828	62062	81752	100881
Ayrıştırma	3.756	4.349	4.847	6.283	7.893
Çıkarsama	46.022	169.344	294.549	499.776	756.506
Toplam	49.778	173.693	299.396	506.059	764.399

Çizelge 7.3. Standart uygulama için ontoloji açılış süreleri (ms).

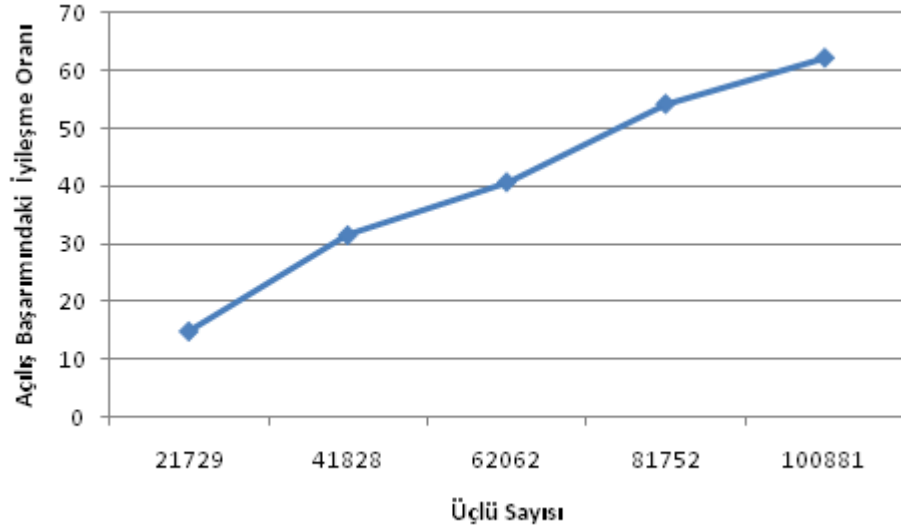
olan olumlu etkisini açıkça göstermektedir. Şekil 7.5 ve Şekil 7.6’da iyileşme grafiksel olarak gösterilmiştir. Artan ontolojik veri miktarı ile yöntemin açılış başarımına olan etkisi de artmaktadır (Şekil 7.6).

	Toplam Üçlü Sayısı				
	21729	41828	62062	81752	100881
Ayrıştırma	1.834	3.439	4.599	5.774	7.934
Dönüştürme	1.127	1.776	2.464	3.267	4.083
Çıkarsama	0.384	0.305	0.305	0.309	0.297
Toplam	3.345	5.520	7.368	9.350	12.314

Çizelge 7.4. Eniyilenmiş uygulama için ontoloji açılış süreleri (ms).

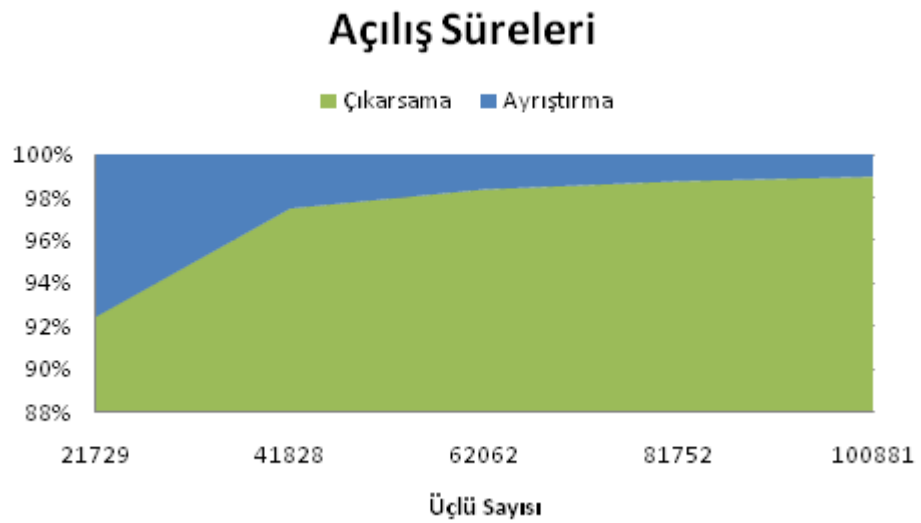


Şekil 7.5. Standart ve eniyilenmiş uygulamaya ait ontoloji açılış sürelerinin karşılaştırması.



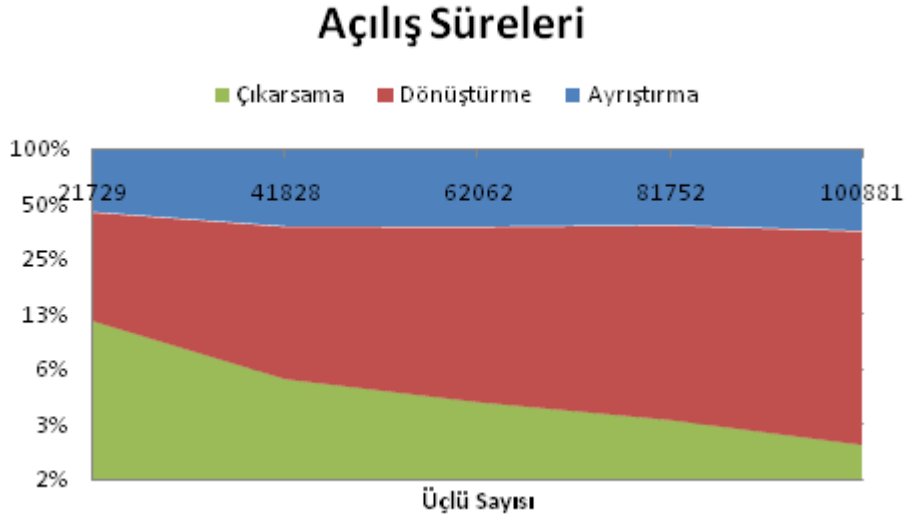
Şekil 7.6. Standart ve eniyilenmiş uygulama arasındaki açılış başarımının iyileşme oranı.

Şekil 7.7 ve Şekil 7.8’de açılış süreleri gösterilmektedir. Mavi alan ayrıştırma için, kırmızı alan dönüştürme için (yalnızca eniyilenmiş uygulamada), yeşil alan ise çıkarsama için gereken süreleri belirtmektedir. Bu şekillerden çıkarılabilecek sonuç, yöntemin uygulanması ile çıkarsamanın, toplam açılış süresindeki ağırlığının çok azaldığıdır. Standart yöntemle, bir üniversitelik veri kümesi için çıkarsama yükü toplam sürenin %98’ine yaklaşırken, eniyilenmiş yöntem ile bu oran daima %50’nin altında kalmıştır.



Şekil 7.7. Standart uygulama ontoloji açılış süreleri.

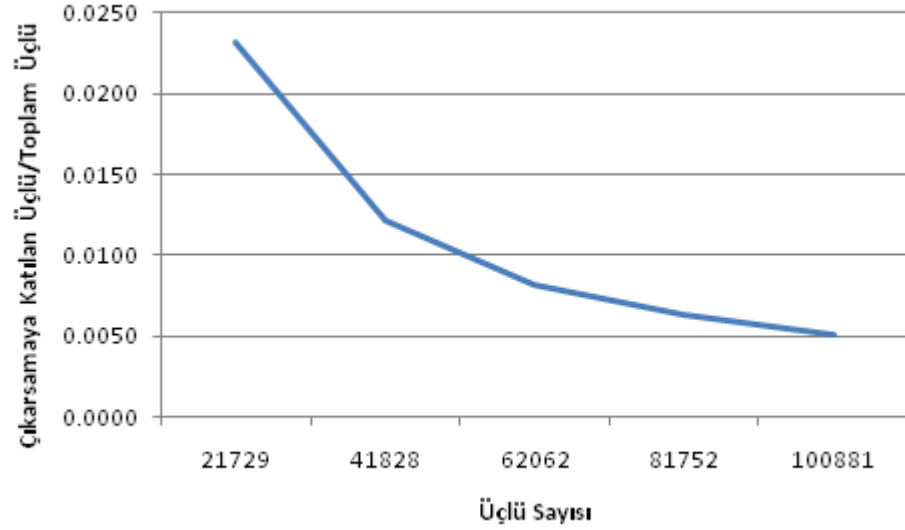
Bu oranlar da aslında sistemin başarıma olan etkisinin nereden kaynaklandığını göstermektedir. Standart uygulamada, çıkarsamanın tamamı çıkarsama motoru üzerinde ve dönüştürülmemiş bir ontoloji üzerinde gerçekleştirildiği için çıkarsama motorunun başarıımı artan veri miktarı ile birlikte kötüleşmektedir. Eniyilenmiş uygulamada ise çıkarsama süresi artan veri miktarı ile sabit hale gelmektedir.



Şekil 7.8. Eniyilenmiş uygulama ontoloji açılış süreleri.

Çıkarsama yükünün artan veri miktarı ile sabit bir hale getirilebilmesi için çıkarsamaya katılması gereken üçlü sayısının da sabit olması gereklidir. Şekil 7.9’da çıkarsamaya katılan üçlü sayısının ontolojideki toplam üçlü sayısına oranı gösterilmektedir.

Şekil 7.10, ontoloji açılış performansını değerlendirmek için kullanılacak bir diğer ölçü olan saniyede yüklenen üçlü sayısı (η_{tps}) ölçütünü veri boyutunun bir fonksiyonu olarak göstermektedir. Şekilden de görüleceği üzere, uzantı tabanlı modele dönüştürme için harcanan sürenin performansa olumsuz bir etkisi olmamaktadır. Dönüşüm bedeli, çıkarsama sürecinde sağlanan zaman tasarrufunun yanında ihmal edilebilecek boyutta kalmaktadır. η_{tps} değeri, artan örnek verisi boyutu ile neredeyse sabit kalmaktadır. Fakat, standart çıkarsama motorunun saniyede yüklediği üçlü sayısı artan veri miktarı ile belirgin bir şekilde azalmaktadır. Bunun nedeni (Guo et al., 2004)’de anlatılan çıkarsama motorlarından farklı olarak, çıkarsama süresinin, örnek verisindeki değişikliklerden etkilenmemesidir.

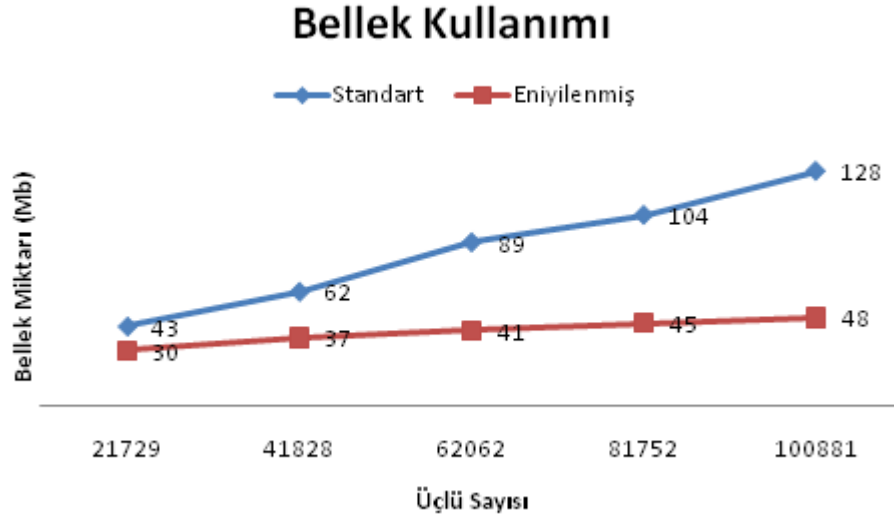


Şekil 7.9. Çıkarsamaya katılan üçlü sayısının toplam üçlü sayısına oranının artan veri miktarı ile değişimi.

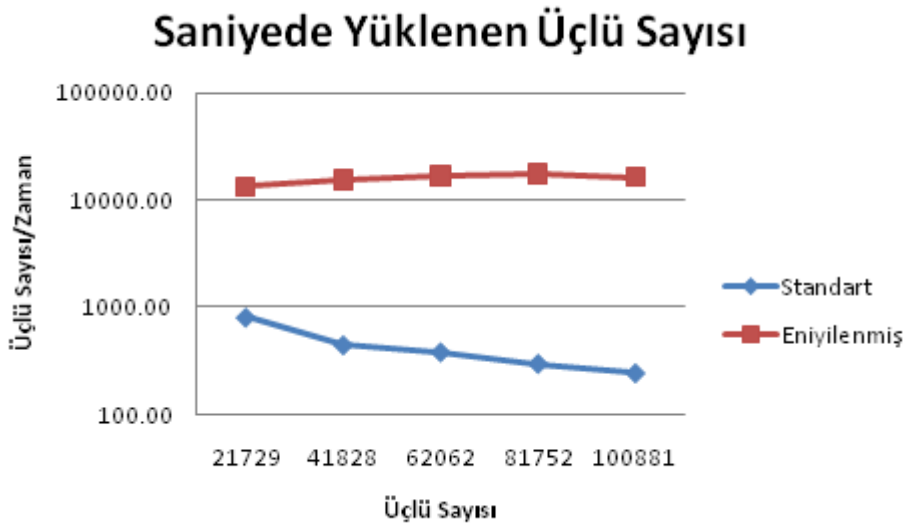
Yöntemin ölçeklenebilirliğe olan katkısının görülebileceği diğer bir alan da bellek kullanımıdır. Şekil 7.11’de görülebileceği gibi, yöntem bellek kullanımında %60’a varan iyileştirmeler sağlamaktadır.

	Toplam Üçlü Sayısı				
	21729	41828	62062	81752	100881
Standart	806	443	380	296	241
Eniyilenmiş	13417	15308	16781	17439	16287

Çizelge 7.5. Standart ve eniyilenmiş uygulama için saniyede açılan üçlü sayısı.

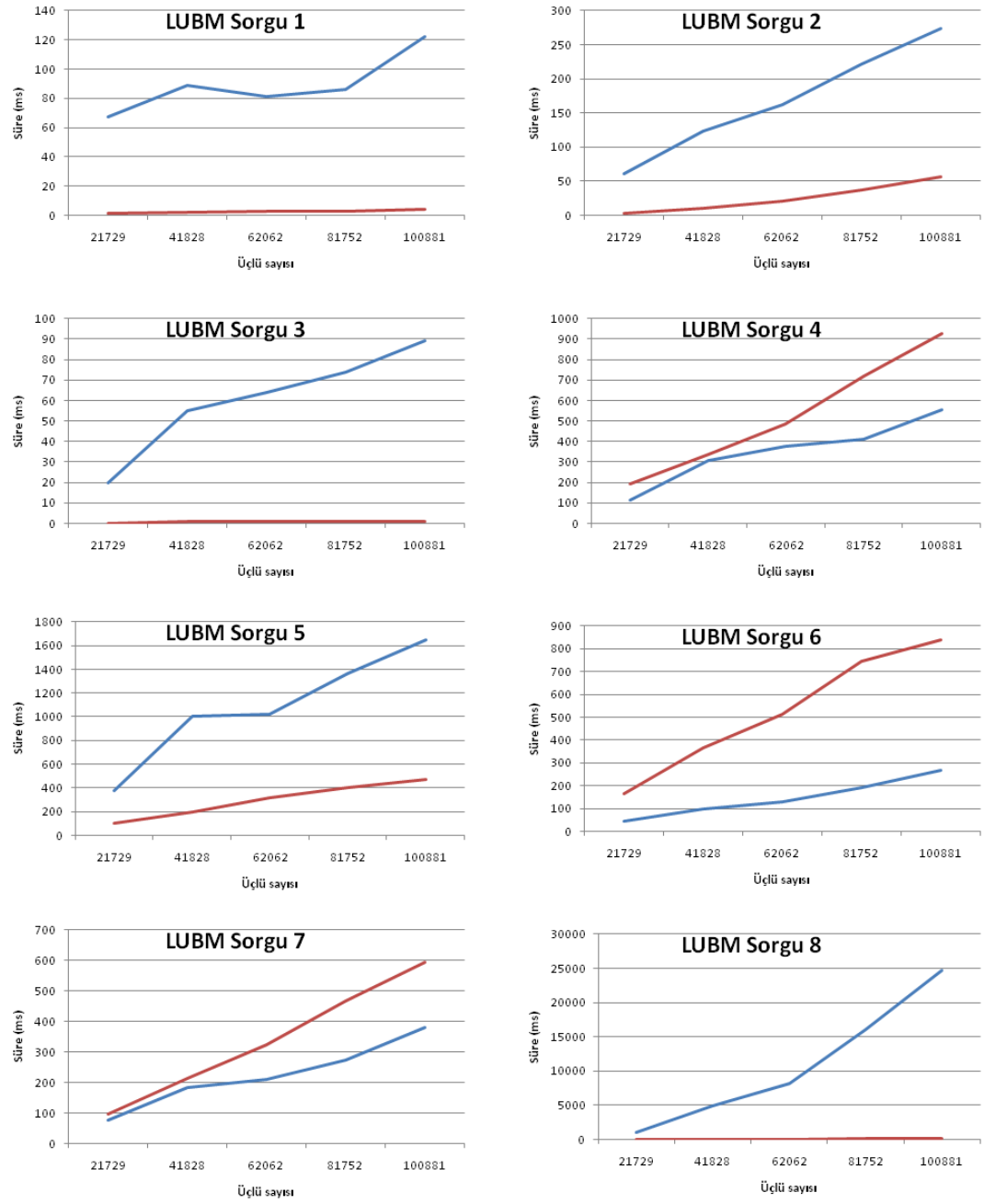


Şekil 7.11. Standart ve eniyilenmiş uygulamanın bellek kullanım miktarları.

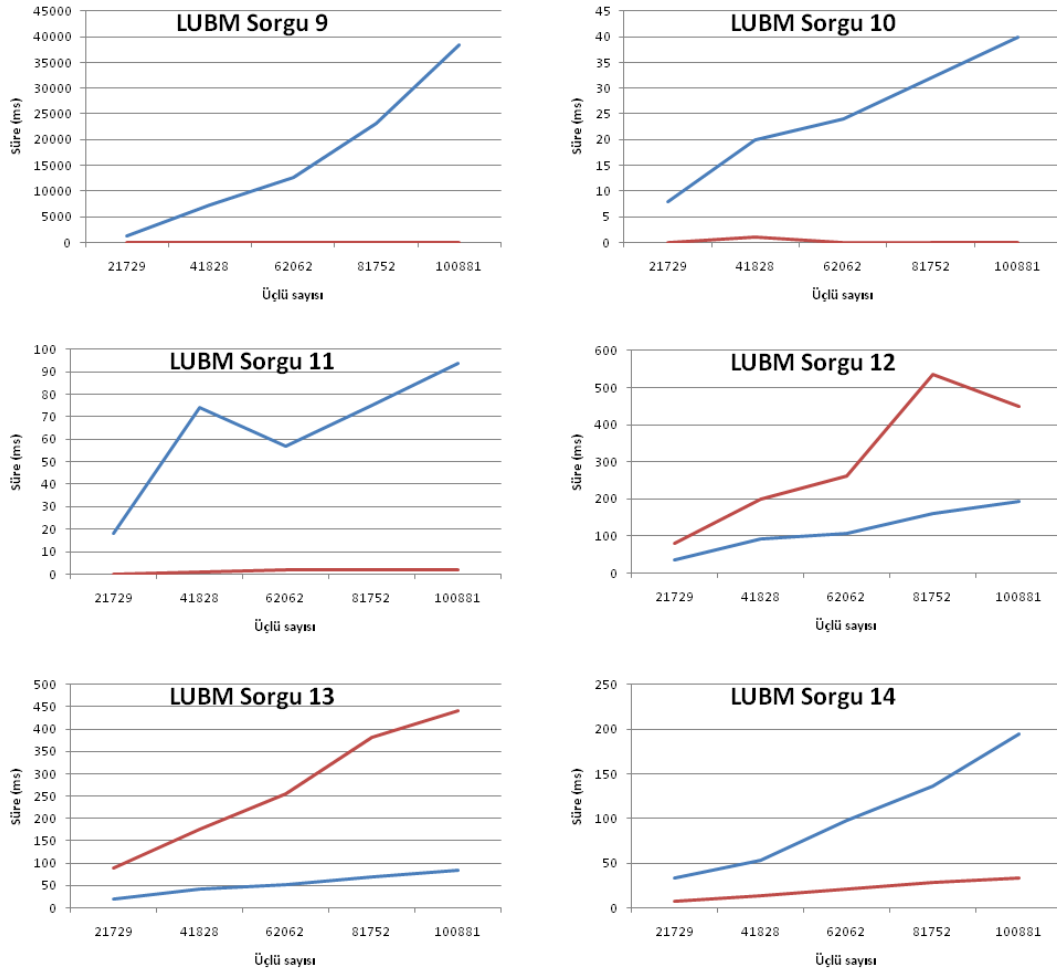


Şekil 7.10. Saniyede yüklenen üçlü sayısının artan veri miktarıyla değişimi.

Kabul edilebilir ve ölçeklenebilir sorgu performansı, Anlamsal Web için anahtar gereksinimlerden biridir. Şekil 7.12 ve Şekil 7.13'te standart ve eniyilenmiş uygulamaların, artan veri miktarı ile LUBM değerlendirme uygulamasındaki ondört sorguya ait başarımları karşılaştırılmaktadır. Çizelge 7.6 ve Çizelge 7.7'de ise sorgu süreleri verilmektedir.



Şekil 7.12. Standart ve eniyilenmiş çıkarsama motorunun LUBM değerlendirme uygulaması için sorgu sonuçları.



Şekil 7.13. Standart ve eniyilenmiş çıkarsama motorunun LUBM değerlendirme uygulaması için sorgu sonuçları (Devamı).

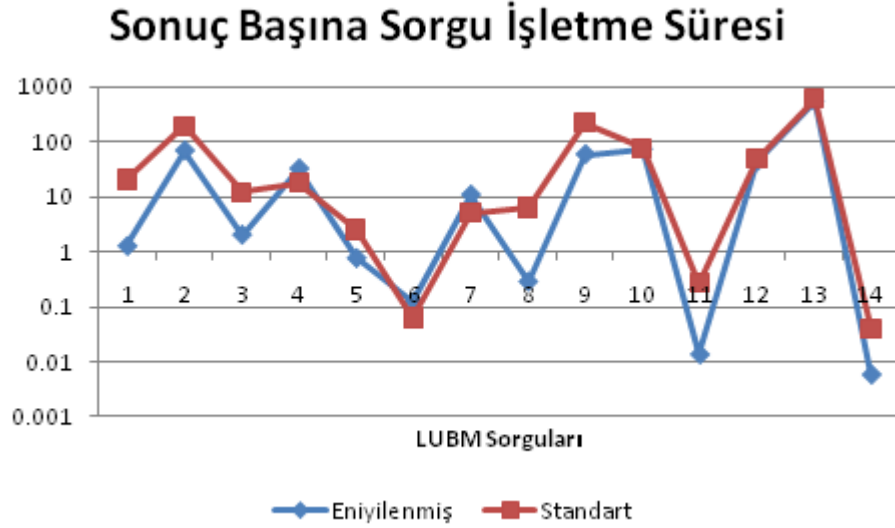
	Üçlü Sayısı				
	21729	41828	62062	81752	100881
1	67	89	81	86	122
2	61	124	162	222	273
3	20	55	64	74	89
4	114	309	379	413	555
5	376	1000	1025	1362	1651
6	44	100	131	192	267
7	76	183	210	273	379
8	1032	4870	8144	16701	24717
9	1344	7270	12561	23109	38456
10	8	20	24	32	40
11	18	74	57	75	94
12	35	92	107	159	194
13	19	43	53	69	83
14	33	53	98	136	194

Çizelge 7.6. Standart çıkarsama motoru için sorgu çalışma süreleri (ms).

	Üçlü Sayısı				
	21729	41828	62062	81752	100881
1	1	2	3	3	4
2	3	11	21	37	56
3	1	1	1	1	1
4	193	337	486	721	926
5	102	194	321	406	474
6	165	366	512	746	839
7	95	213	323	466	593
8	22	43	60	84	99
9	13	28	38	54	67
10	1	1	1	1	1
11	1	1	2	2	2
12	81	198	261	535	448
13	89	177	256	382	441
14	7	14	21	29	33

Çizelge 7.7. Eniyilenmiş çıkarsama motoru için sorgu çalışma süreleri (ms).

Sonuçlardan görüldüğü gibi yöntemin sorgulamaya da olumlu etkisi olmaktadır. Ayrıca, Şekil 7.14, tez kapsamında önerilen yaklaşımın, LUBM değerlendirme uygulamasına ait 14 sorgu üzerindeki etkilerini göstermektedir. Şekilden de anlaşılacağı üzere, dördüncü, altıncı ve yedinci sorgular dışındaki tüm sorgularda bir hızlanma olmuştur. Yaklaşımın temel mantığı düşünüldüğünde, geriye doğru zincirleme algoritmasının kullanılması sonucunda tüm sorguların performanslarında genel bir düşüş beklenebilir. Fakat bu düşüş, uzantı tabanlı modelin ve veritabanı teknolojisinin kullanılması ile avantaja dönüştürülmüş ve sorgu performanslarında genel olarak bir iyileşme gözlemlenmiştir. Yavaşlayan sorgular incelendiğinde, bu sorguların tamamının geniş ve derin sınıf hiyerarşilerini sorguladığı görülebilir. Diğer bir deyişle, yavaşlamanın sebebi sorgu ağacı karmaşıklığındaki artıştır.



Şekil 7.14. Standart ve eniyilenmiş çıkarsama motorunun LUBM değerlendirme uygulaması sorguları için sonuç başına sorgu işletme süresi değerleri.

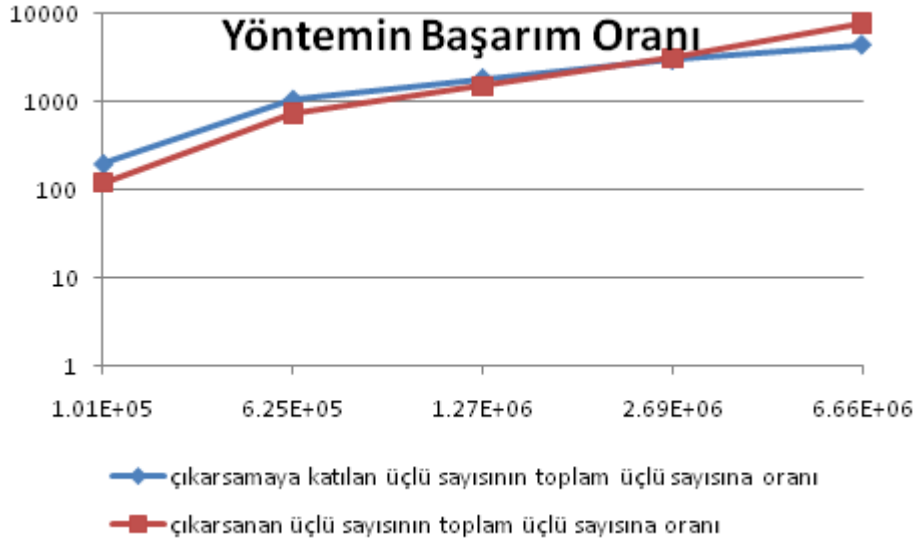
Uzantı tabanlı modelin kullanılması, geriye doğru zincirleme sürecinde oluşturulan sorguların karmaşıklığını azaltmaktadır. Olgun ve verimli veritabanı teknolojisi büyük miktarlarda veriyi başarılı bir şekilde destekleyebilmiş ve bu veri üzerinde ölçeklenebilir sorgular çalıştırılmasına olanak sağlamıştır. Birkaç sorguda gözlemlenen performans kaybının nedeni, eniyilenmemiş *SQL* sorgularıdır. Bölüm 4.3'te anlatılan sorgu ağacı üzerinde yapılabilecek budama işlemleri, oluşturulan *SQL* sorgularının karmaşıklığını azaltacaktır. Sorgu algoritmasının, bu ve benzeri yaklaşımlarla eniyilenmesi gelecek çalışmalar arasında yer almaktadır.

7.5.2 İkinci Deney

Standart uygulama ile bir üniversiteden fazlasının açılması mümkün olmazken, eniyilenmiş uygulamayla açılabilen üniversiteye sayısı 50'ye kadar çıkarılmıştır. Bu da önerilen model dönüşümünün bellek kullanımını ve çıkarsama gereksinimini azaltmasının doğal sonuçlarından biridir. Bu nedenle, ikinci deneyde yalnızca eniyilenmiş uygulamanın LUBM(1,0), LUBM(5,0), LUBM(10,0), LUBM(20,0) ve LUBM(50,0) veri setleri ile başarıyı değerlendirilmektedir.

Şekil 7.15 toplam üçlü sayısının, çıkarsamaya katılan ve çıkarsanan üçlülere

oranlarını, veri boyutunun fonksiyonları olarak göstermektedir. Faydalanma oranı, veri boyutu ile logaritmik olarak artış göstermektedir. Veri boyutundaki altmış katlık bir artış, faydalanma oranında bin katlık bir artışa neden olmaktadır. Sonuç olarak, yaklaşımın artan veri boyutu ile çok daha fazla fayda sağladığını söyleyebiliriz.

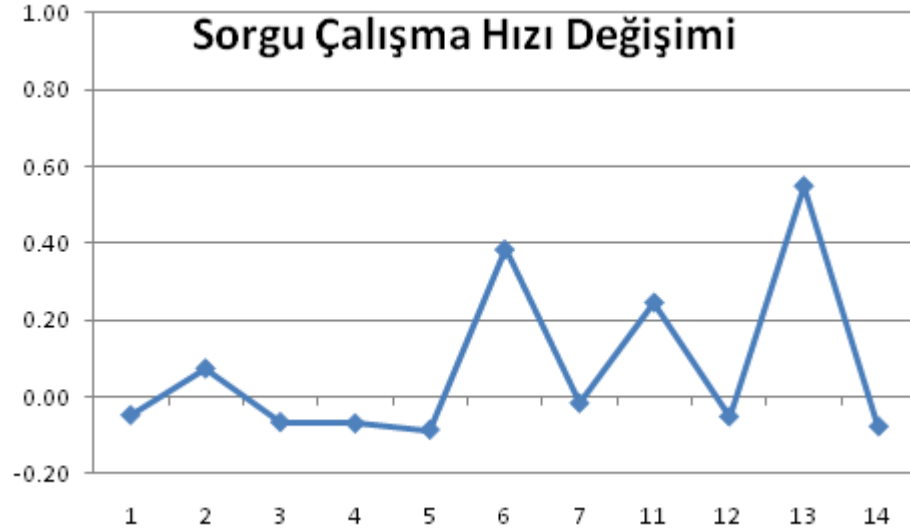


Şekil 7.15. Dönüşümün başarımlarının LUBM 1-50 üniversite veri kümesi ile değişimi.

Bir sistemdeki sorgu yanıtlatma süresi, artan veri boyutuyla orantılı olarak yükselmektedir. Bu bölümde sunulacak son ölçüt olan Γ_{di} , sorgu yanıtlatma süresindeki artış (d_{qrt}) ile veri boyutundaki artış (i_{ds}) arasındaki ilişkiyi belirtmektedir ve aşağıdaki eşitliğe göre hesaplanmaktadır.

$$\Gamma_{di} = 1 - \frac{d_{qrt}}{i_{ds}}$$

Bu ölçüt yapılan deney için $[-1, 1]$ aralığında değerler almaktadır. 0 değeri $d_{qrt} = i_{ds}$ ifadesinin doğruluğunu gösterir. $[-1, 0)$ aralığındaki değerler $d_{qrt} > i_{ds}$ ifadesinin doğruluğunu, $(0, 1]$ aralığındaki değerler ise $d_{qrt} < i_{ds}$ ifadesinin doğruluğunu belirtmektedir. Şekil 7.16, farklı sorgulara ait Γ_{di} değerlerini göstermektedir. Şekilden de görülebileceği gibi tüm sonuçlar 0 değerine çok yakın veya $(0, 1]$ aralığındadır. Bu sonuçlar, sorgu motorunun artan veri boyutu ile ölçeklenebilir olduğunu göstermektedir.



Şekil 7.16. LUBM(50,0) veri kümesi için sorgu başarımının değişimi.

7.5.3 Üçüncü Deney

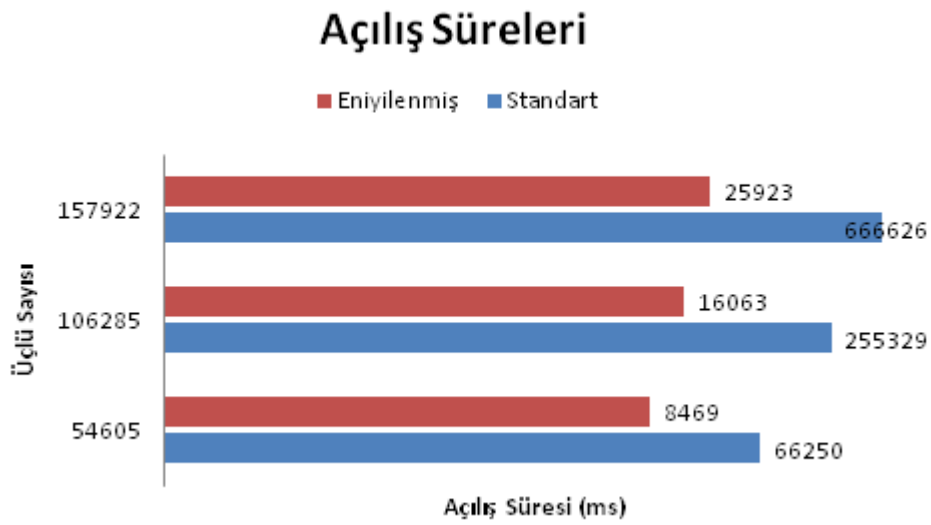
Bu deneyde UOBM değerlendirme uygulaması için bir üniversitenin alt kümesi olan 4, 8 ve 12 bölümlük veri kümeleri kullanılmıştır. Bunun nedeni standart çıkarsama motorunun bundan daha büyük veri kümelerini açamamasıdır. UOBM sonuçları incelendiğinde LUBM ile paralel sonuçlar verdiği gözlemlenmiştir. Bu nedenle, bu bölümde LUBM ile aynı sırada olmak üzere yalnızca çizelgeler ve şekiller verilecek ayrıca bir değerlendirme yapılmayacaktır.

	Toplam Üçlü Sayısı		
	54605	106285	157922
Ayrıştırma	5.241	10.452	16.722
Çıkarsama	61.009	244.877	649904
Toplam	66.250	255.329	666626

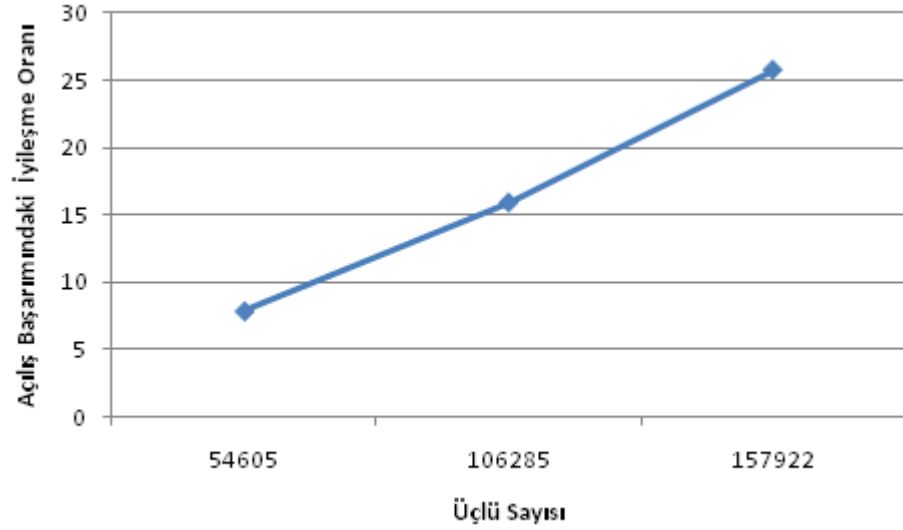
Çizelge 7.8. Standart uygulama için ontoloji açılış süreleri (ms).

	Toplam Üçlü Sayısı		
	54605	106285	157922
Ayrıştırma	4.414	8.314	14.131
Dönüştürme	2.558	5.539	8.983
Çıkarsama	1.497	2.210	2.809
Toplam	8.469	16.063	25.923

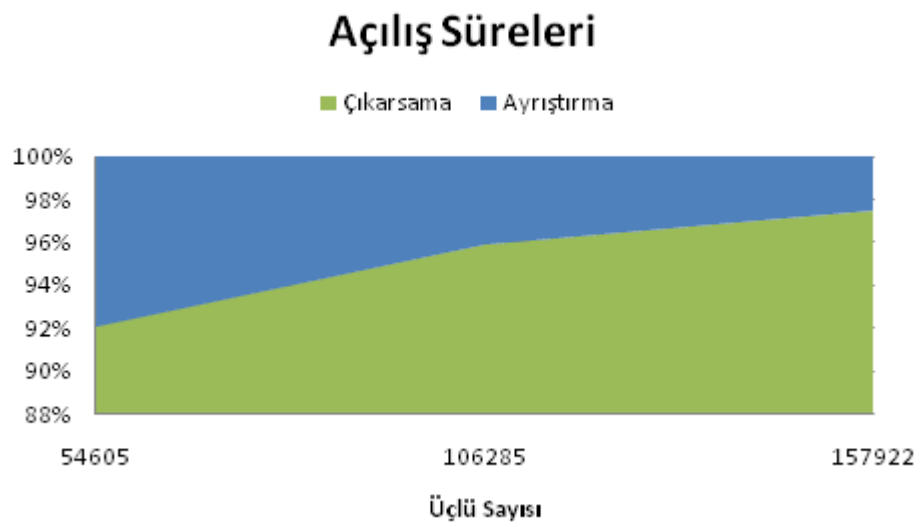
Çizelge 7.9. Eniyilenmiş uygulamaya için ontoloji açılış süreleri (ms).



Şekil 7.17. Standart ve eniyilenmiş uygulamaya ait ontoloji açılış sürelerinin karşılaştırması.

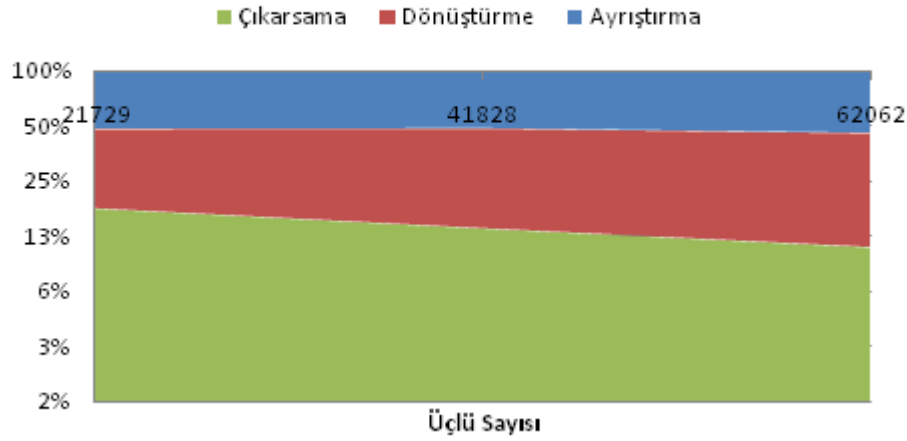


Şekil 7.18. Standart ve eniyilenmiş uygulama arasındaki açılış başarımının iyileşme oranı.

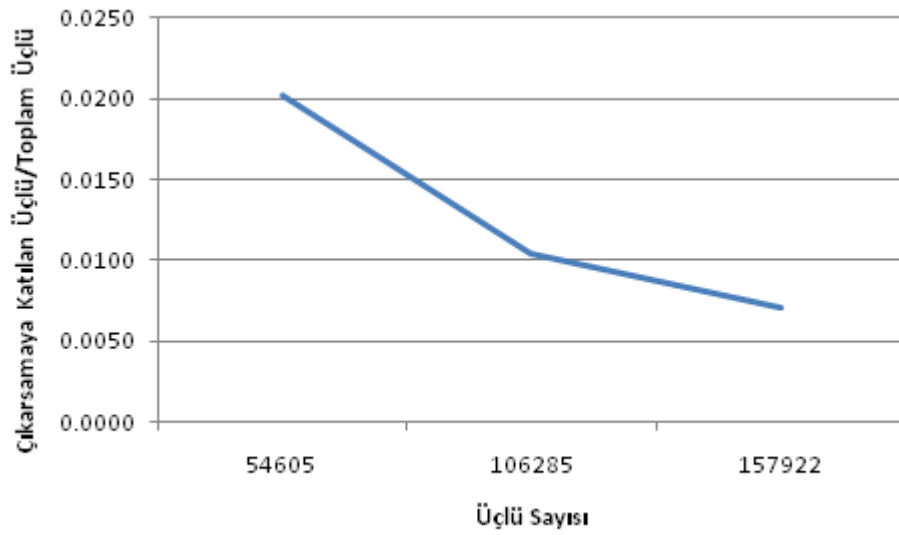


Şekil 7.19. Standart uygulama ontoloji açılış süreleri (ms).

Açılış Süreleri



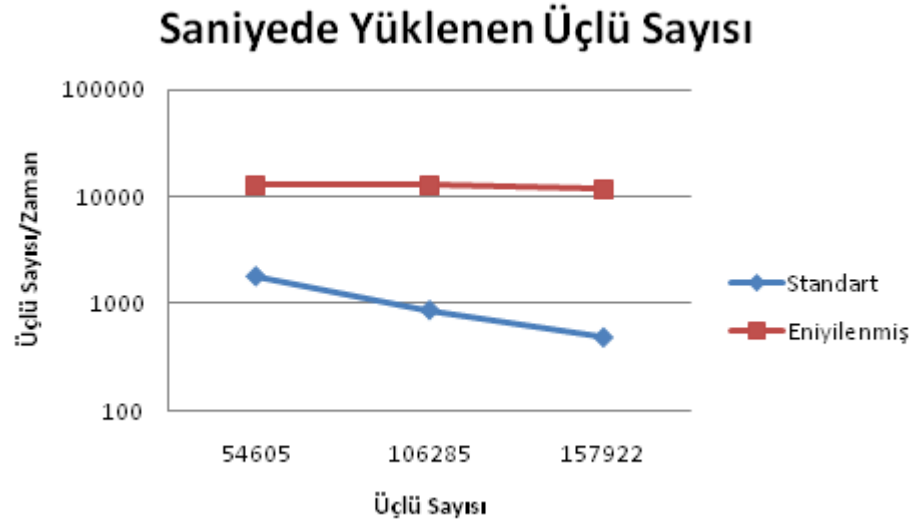
Şekil 7.20. Eniyilenmiş uygulama ontoloji açılış süreleri (ms).



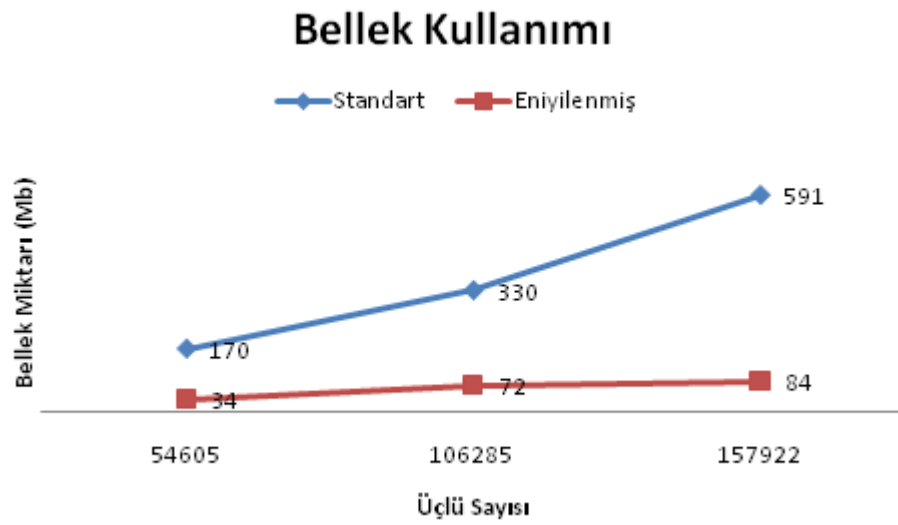
Şekil 7.21. Çıkarsamaya katılan üçlü sayısının toplam üçlü sayısına oranının artan veri miktarı ile değişimi.

	Toplam Üçlü Sayısı		
	54605	106285	157922
Standart	1791	873	489
Eniyilenmiş	12857	12852	11709

Çizelge 7.10. Standart ve eniyilenmiş uygulama için saniyede açılan üçlü sayısı.



Şekil 7.22. Saniyede yüklenen üçlü sayısının artan veri miktarıyla değişimi.



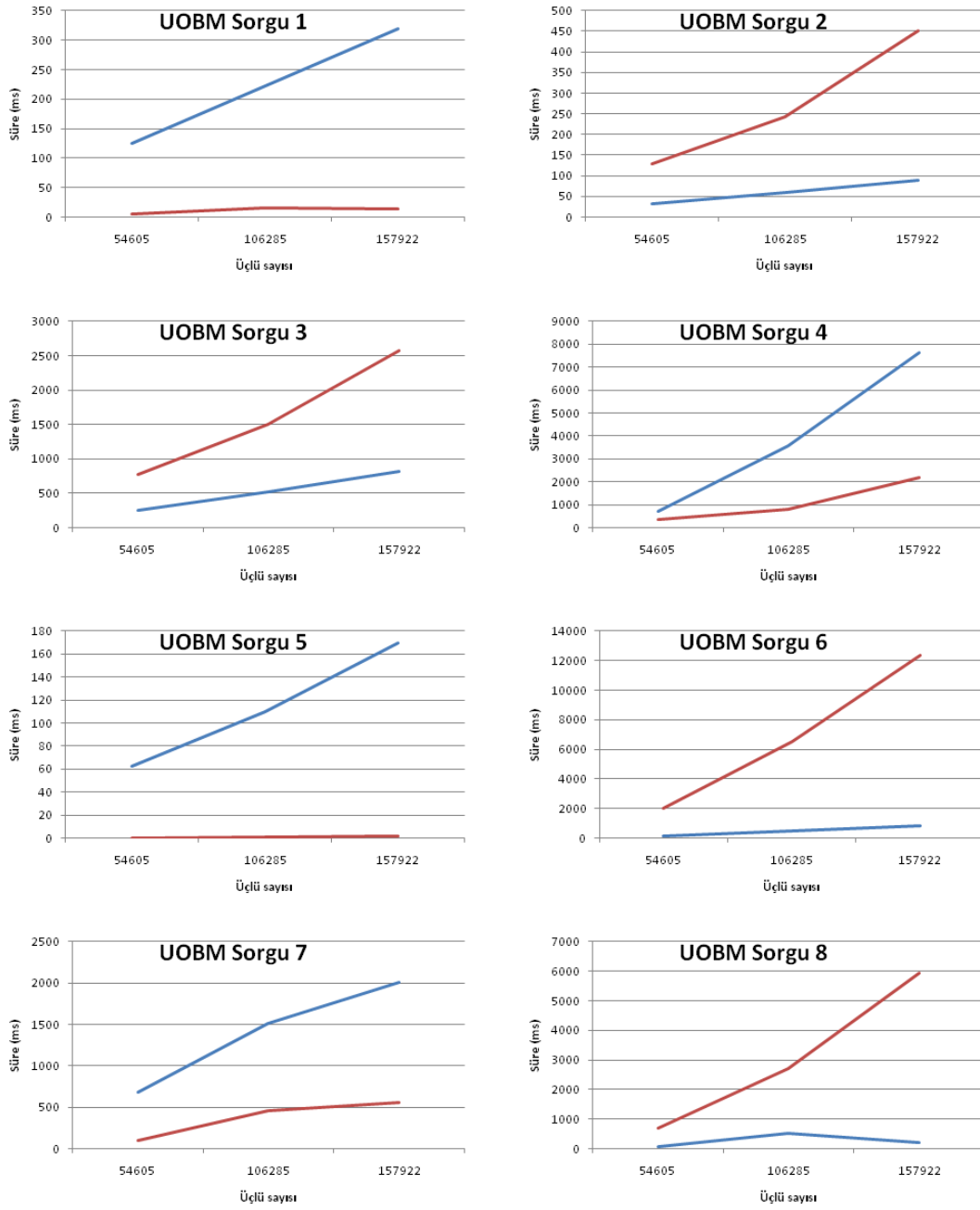
Şekil 7.23. Standart ve eniyilenmiş uygulamanın bellek kullanım miktarları.

	Üçlü Sayısı		
	54605	106285	157922
1	124	222	320
2	31	60	90
3	249	520	810
4	702	3568	7650
5	62	110	170
6	156	472	850
7	93	454	560
8	62	512	200
9	109	233	320
10	31	50	90
11	1762	9027	24466
12	2730	12160	38782
13	62	93	160

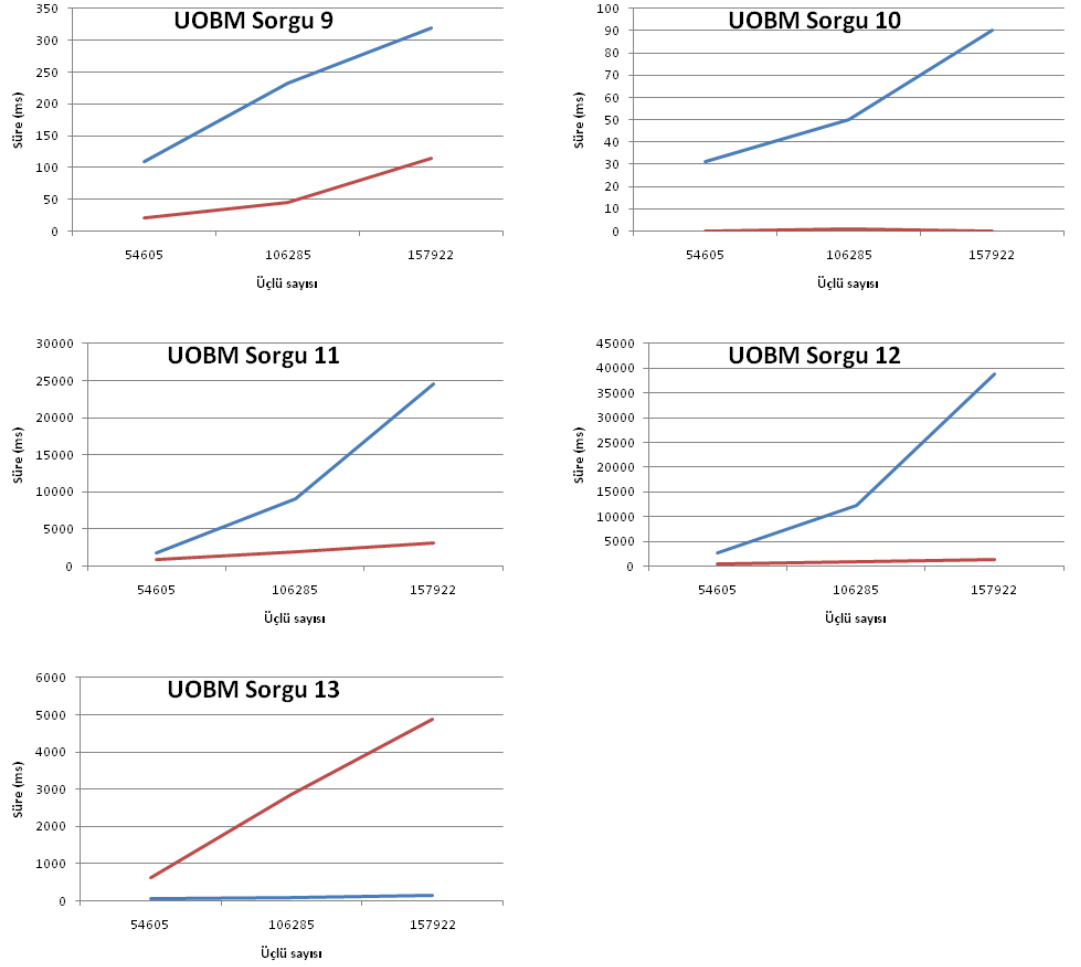
Çizelge 7.11. Sorgu çalışma süreleri (ms).

	Üçlü Sayısı		
	54605	106285	157922
1	5	15	14
2	128	243	452
3	768	1498	810
4	337	805	2181
5	0	1	2
6	2039	6520	12323
7	676	1517	2007
8	701	2715	5937
9	21	45	114
10	0	1	2
11	952	1864	3105
12	387	839	1218
13	622	2855	4890

Çizelge 7.12. Sorgu çalışma süreleri (ms).



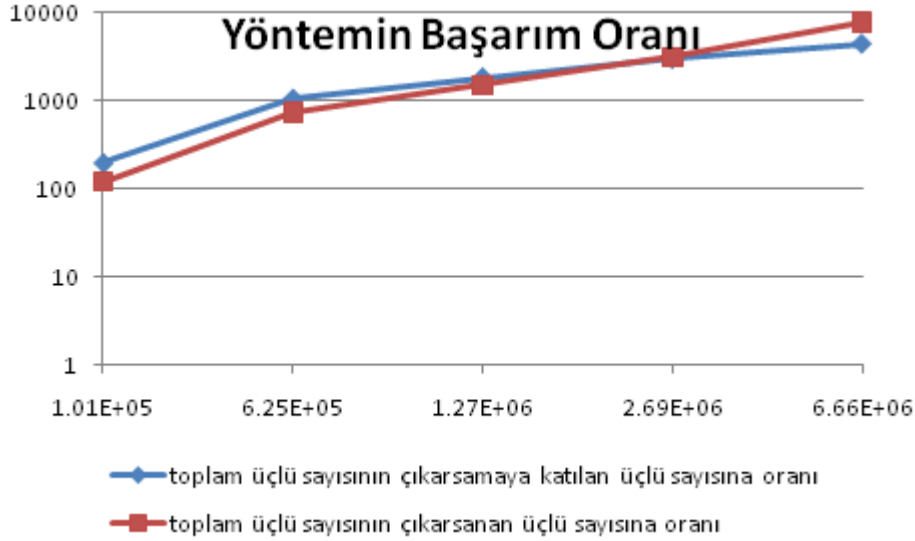
Şekil 7.24. Standart ve eniyilenmiş çıkarsama motorunun LUBM değerlendirme uygulaması için sorgu sonuçları.



Şekil 7.25. Standart ve eniyilenmiş çıkarsama motorunun UOBM değerlendirme uygulaması için sorgu sonuçları (Devamı).

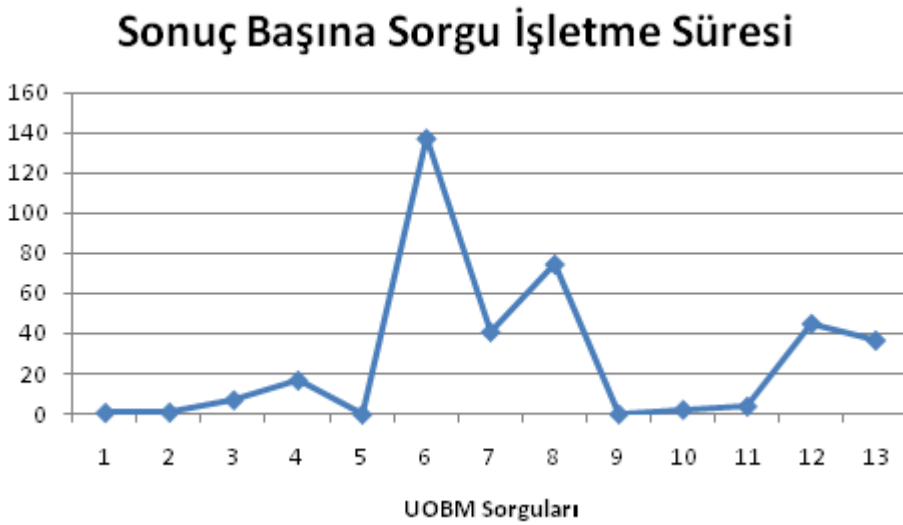
7.5.4 Dördüncü Deney

Bu deneyde UOBM değerlendirme uygulaması verisinde yöntemin başarımının sınaması yapılmıştır. Bu deneyin yapılmasının amacı, yöntemin daha karmaşık şemaya sahip ontolojilerdeki başarımını incelemektir. UOBM deneyi daha önceki deneylerle benzer sonuçları vermiştir. Şekil 7.26’da görülebileceği gibi model dönüşümünün başarımı büyüyen veri kümeleri için yine artan oranda başarımla göstermektedir.



Şekil 7.26. UOBM değerlendirme uygulaması için model dönüşümünün başarımının artan veri kümesi ile değişimi.

Şekil 7.27’de UOBM sorgularının sonuç başına sorgu işletme süresindeki değişim gösterilmektedir. UOBM değerlendirme uygulamasında sorguların üzerinde çalıştığı ontolojinin karmaşıklığı artmasına karşın sorgu başarımı kabul edilebilir düzeyde kalmıştır. Yalnızca *Sorgu 6* diğer sorgulara göre daha yavaş çalışmaktadır. Bunun nedeni, bu sorgunun sınıf sıradüzeninin temeli olan *Person* sınıfını içermesi ve bu nedenle oluşan sorgu ağacının karmaşıklığının artmasıdır.



Şekil 7.27. UOBM değerlendirme uygulaması sorgu başarımının artan veri kümesi ile değişimi.

8 Varolan Sistemlerle ve Tekniklerle Karşılaştırma

Tezin bu bölümünde, tez kapsamında gerçekleştirimi yapılan yöntemin varolan çalışmalarla olan farklılıkları ve benzerlikleri incelenecektir. Tüm çalışmaların ortak yönü çıkarsama ve sorgulamayı anlamsal web ontolojilerinin gereksinimleri doğrultusunda özelleştirmeye çalışmalarıdır. Bunu yaparken izledikleri yöntemler açısından birbirlerinden farklılaşmaktadırlar. İncelenen yaklaşımların her biri ontolojilerin depolanması, üzerinde çıkarsama yapılması ve sorgulanması açısından farklılıklar taşımaktadırlar. Bu bölümün devamında ilk olarak tüm sistemler incelenmekte ve Bölüm 8.7’de bu sistemlerin tez çalışması ile karşılaştırılması yapılmaktadır. Bölüm 8.8’de ise seçilen bir grup sistemin başarımı tez çalışması ile karşılaştırılmaktadır.

8.1 SHER

SHER (Scalable Highly Expressive Reasoner) (Ölçeklenebilir, Yüksek Anlatımlı Çıkarsama Motoru) ontoloji ABox’ının bir özetini çıkararak veritabanında saklar. Bu saklanan özete ve özeti çıkarma yöntemine “*Summary ABox*” (ABox Özeti) adı verilmektedir (Fokoue et al., 2006). ABox özeti aynı kavramın örneklerinin biraraya toplanması yöntemi ile elde edilir. Çıkarsama ve sorgulama bu özetlenmiş ABox üzerinde gerçekleştirilir.

Depolama

SHER depolama için hem belleği hem de veritabanlarını kullanmaktadır. Bunun nedeni çıkarsama algoritmasının bellekte çalışmasının zorunlu olmasıdır. SHER’in kullandığı veritabanı şeması ile ilgili bilgi yayınlarda verilmemiştir. Bunun nedeni veritabanının sistemin başarımında bir etkisinin olmamasıdır.

Çıkarsama

SHER ontolojilerin yüklenmesi aşamasında çıkarsama gerçekleştirmez, bu nedenle sık değişen verilerle göreceli olarak daha verimli çalışmaktadır.

Fakat, çıkarsamayı sorgu cevaplama aşamasında yaptığı için sorgu cevaplama yavaşlama olabilmektedir. SHER'in çıkarsama başarımı temel olarak özet ABox yöntemine dayanmaktadır. Örnek verisininin sıkıştırılmış hali olarak tanımlanabilecek bu indeksleme tekniği ile veritabanında bulunan örnek verisi çıkarsamada daha verimli olarak kullanılır.

Sorgulama

SHER sorgulama aşamasında yalnızca özet veriyi kullanarak cevapları bulur, fakat cevaplayabileceği sorgular sınırlıdır ve sorgu başarımı kötüdür Fokoue et al. (2006). Sorguların cevaplanması sırasında sorgu yeniden yazılır ve çıkarsama motorundan gelen özet ABox bilgileri ile veritabanındaki ontoloji kullanılarak sorgunun cevabı elde edilir. Fakat, sorgulama için DL çıkarsama motoru kullanılmasının doğal bir sonucu olarak performans kötüdür. Bunun nedeni hem DL çıkarsama motorlarının ABox sorgulamadaki genel kötü başarımları hem de SHER'in sorguları cevaplayabilmek için çıkarsama gerektirmesi gösterilebilir.

8.2 Minerva

Minerva (Zhou et al., 2006), büyük boyutlu OWL ontolojileri için bir depolama ve çıkarsama sistemidir. Minerva, veritabanlarını kullanarak OWL ontolojileri üzerinde çıkarsama ve sorgulama yapar. Minerva'nın farklı olan yönü, betim mantığı çıkarsama araçlarını şema üzerinde çıkarsama yapmak için kullanması ve örnekler üzerinde çıkarsama yapmak için kural tabanlı sistemlerden faydalanmasıdır.

Depolama

Minerva veritabanı şeması çıkarsamayı destekleyecek şekilde geliştirilmiştir. Şemanın en önemli özelliği DLP kurallarında yer alan her yükleme karşılık gelen bir veritabanı tablosunun olmasıdır. Bu nedenle, bu kurallar kolaylıkla ilişkisel cebir operasyonlarına çevrilebilirler. Örneğin, aşağıda gösterilen kuralda üç farklı yükleme sahip dört koşul vardır;

$$Type(x, c) \leftarrow Rel(x, Y, y) \wedge Type(y, D) \wedge SomeValuesFrom(C, R, D)$$

Bu kuralı işletmek için ilişkili üç tablo (*RelationshipInd*, *TypeOf* ve *SomeValuesFrom*) üzerinde basit *SQL select* ve *join* işlemlerini çalıştırmak yeterli olacaktır.

Çıkarsama

Minerva’da çıkarsama öncelikle *subClassOf* ve *subPropertyOf* ilişkilerinin bir DL reasoner ile elde edilmesi ile başlar. Çıkarsama sürecinde yalnızca TBox verisi kullanılarak, sınıf ve nitelik sıradüzeni çıkarsanır. Çıkarsanan verilerin de yardımıyla, ABox çıkarsama kuralları veritabanı üzerinde çalışacak sorgulara dönüştürülür ve bu sorguların çalıştırılmasıyla çıkarsama tamamlanır.

Sorgulama

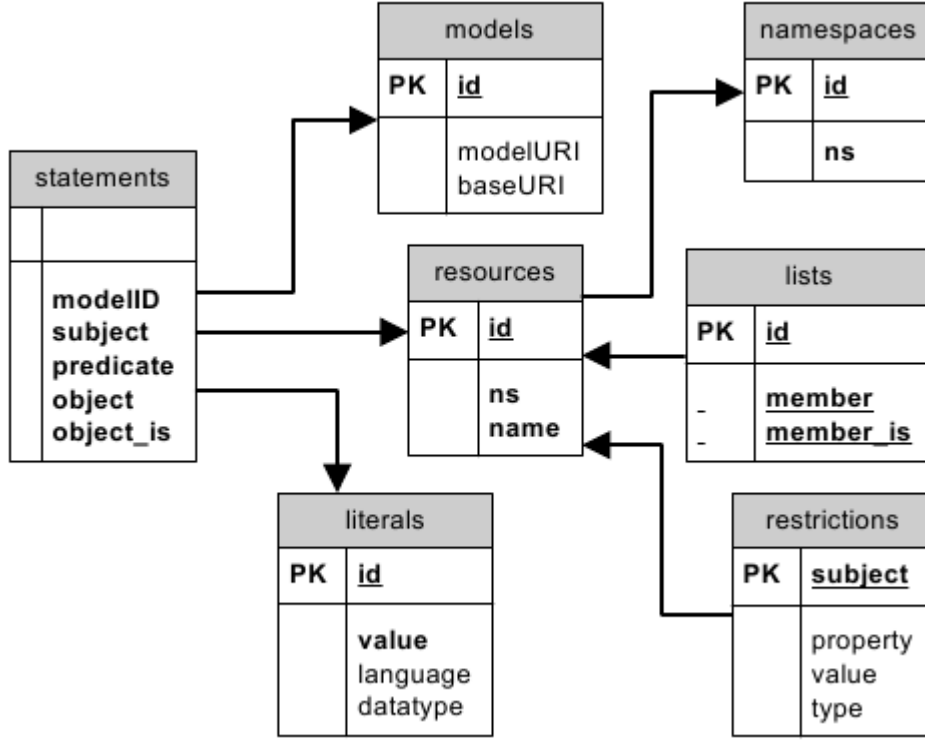
Minerva da diğer çalışmalarda olduğu gibi belli bir biçimde gelen sorguyu alıp onu kendi desteklediği şemaya uygun bir SQL cümlesine dönüştürüp, sorguyu çalıştırarak dönen cevabı kullanıcıya döner. Minerva sorgu zamanında çıkarsama gerçekleştirmemektedir.

8.3 OWLDB

OWLDB (Auer and Ives, 2007) ontolojileri ve veritabanlarını birleştirmeye çalışan bir yaklaşımdır. OWLDB tüm çıkarsama kurallarını veritabanı sorgularına dönüştürmüştür. Tezde önerilen yöntemin bu iki yöntemden farkı uzantı tabanlı bilgi modelinin kullanılması nedeniyle çıkarsamanın bellekte yapılabilir hale getirilmesi, bellekte yapılamayan çıkarsamanın ise sorgu zamanında gerçekleştirilmesidir.

Depolama

OWLDB’nin kullandığı veritabanı şeması standart veritabanı şemasıdır. Tüm üçlüler *statement* tablosunda saklanırken, sorguların daha hızlı çalıştırılabilmesi ve çıkarsamanın hızlandırılması için bazı ek tablolar kullanılır. *Restrictions* tablosunda *minCardinality*, *maxCardinality*, *cardinality*, *hasValue*, *someValuesFrom*, *allValuesFrom* ile tanımlanmış kısıt sınıflarına ait tanımları saklanmaktadır.



Şekil 8.1. OWLDB şeması

Çıkarsama

OWLDB, OWL dilinin getirdiği kuralların her birini işleterek, bir ya da birden fazla SQL sorgusuna dönüştürür. Bu sorguların çalıştırılması sonucu elde edilen sonuçlar da veritabanı şemasındaki uygun tablolara eklenirler.

Sorgulama

OWLDB tam cisimleştirme gerçekleştirdiği için sorgulama sırasında herhangi bir ek işleme gerek duymadan gelen sorguları SQL sorgularına dönüştürerek çalıştırır.

8.4 DLDB

DLDB, bir veritabanı yönetim sistemi (Microsoft Access) ve bir DL çıkarsama motorunun (FaCT DL, (Horrocks, 1998)) birleşiminden oluşan bir bilgi tabanı sistemidir. DLDB'nin geliştiricilerinin temel varsayımı anlamsal web üzerinde çalışacak sorguların şemadan çok şema örnekleri ile ilgili ola-

cağıdır. Bu nedenle veritabanlarının çıkarsama yeteneklerine kavuşacak şekilde genişletilmelerinin anlamsal web için daha uygun bir çözüm olacağını savunmaktadır.

DLDB bir veritabanı yönetim sistemini çıkarsama yapabilmek için ek yeteneklerle genişletir (Pan and Heflin, 2003). Bu sistemin temel amacı, betim mantığı çıkarsama mekanizmasının bir ilişkisel veritabanı sistemiyle nasıl birleştirilebileceğini incelemektir. Desteklediği ontoloji tanımlama dili OWL-DL'dir. Ontolojiler Access veritabanında saklanırlar. Her sınıf ve nitelik için bir tablo yaratılır. Sınıf hiyerarşisine veritabanı görünümüleri yardımıyla ulaşılır. Bir sınıfın bütün örnekleri, sınıfın kendine ait örneklerin ve alt sınıflarına ait örneklerinin ilgili görünümünden toplanması ile hesaplanır. Nitelik örneklerinin hesaplanması aşamasında da aynı yöntem izlenir. Sistem orta boy (yüzlerce sınıf ve nitelikten oluşan) ontolojiler için verimli çalışmaktadır. Çıkarsama için FaCT çıkarsama motorunu kullanmaktadır. FaCT yalnızca TBox çıkarsamayı desteklediği için DLDB yalnızca sınıflar arası kapsama problemine indirgenebilen çıkarsamaları yapabilmektedir. FaCT kullanarak sınıfların veritabanı görünümünü oluşturmak için gereken sınıf ve nitelik hiyerarşileri hesaplanır. Bu nedenle *inverseOf*, *equivalentClass*, *hasValue* gibi yapılarla çıkarsama desteklenmez.

Depolama

DLDB, depolama için veritabanlarını kullanmaktadır. DLDB'nin veritabanı şeması "horizontal class" ve "property table" yaklaşımlarının birleşimidir. "Horizontal class" yaklaşımında her bir sınıf için veritabanında bir tablo yaratılmaktadır. "Property table" yaklaşımında ise her bir nitelik için bir tablo yaratılmaktadır. Başka bir deyişle, DLDB'nin veritabanı şeması ontolojide tanımlı bulunan sınıflara ve nitelikleri göre değişken bir yapıdadır, sabit bir şeması bulunmamaktadır.

Çıkarsama

DLDB çıkarsamayı iki farklı seviyede gerçekleştirmektedir. RDF(S) seviyesindeki çıkarsama, veritabanında görünüm tanımlarının saklanması ve değiştirilmesi yoluyla gerçekleştirilirken, OWL seviyesinde yapılan çıkarsama FaCT kullanılarak yapılır. FaCT'in çıkarsama işlemini tamamlamasından sonra oluşan XML

belgesindeki veriye göre veritabanı görünümünün tanımları yapılır ve tablolar üzerinde gerekli olan değişiklikler gerçekleştirilir.

Sorgulama

DLDB’de her bir sorgu sisteme ulaşmasından sonra bir SQL cümlesine dönüştürülerek veritabanı üzerinde çalıştırılır ve dönen sonuç kullanıcıya gönderilir. DLDB’de kullanılan yöntemde sorgulama sırasında çıkarsama gerçekleştirilmez.

8.5 Jena

En yaygın anlamsal web geliştirme arayüzlerinden olan Jena diğer çalışmalardan farklı olarak depolama, çıkarsama ve sorgulama için farklı seçenekler sunmaktadır. Tez kapsamında, Jena’nın sunduğu tüm seçeneklerin incelenmesi yerine yalnızca tez çalışmasıyla benzerlik gösterenler seçilmiştir.

Jena, Java üzerinde RDF çizgelerini yönetebilmek için geliştirilmiş bir API’dir (McBride, 2001). RDF ontolojileri sistem içinde çizgeler olarak kontrol edilir. Jena RDF/XML çözümleyici, sorgulama dili ve saklama modülü gibi bileşenler içerir. Jena2, Jena’nın ikinci neslidir ve ontoloji depolama ile ilgili sorunları çözmüştür (Wilkinson et al., 2003a). Jena’da ontolojiler kullanıcının tercihiyle bağlı olarak bellekte ya da veritabanında saklanabilir.

Depolama

Jena’daki veritabanı şeması tez çalışmasındaki şema ile benzerlik göstermektedir. Kullanılan şemada her bir üçlü *statement* tablosunda bir kayda karşılık gelmektedir. Jena’nın diğer yaklaşımlardan farkı, tek bir ontoloji için birden fazla *statement* tablosu oluşturulmasına izin vermesidir.

Çıkarsama

Jena, melez bir çıkarsama motoruna sahiptir. Jena’nın çıkarsama motorunun melez olarak adlandırılmasının nedeni ileriye ve geriye zincirleme algoritmalarını birlikte kullanmasıdır. İleriye zincirleme motoru çalışarak yeni ifadeler çıkarsar ve

bunları saklar. Bazı ileriye zincirleme kurallarının çalıştırılması geriye zincirleme kurallarını çıkarsar ve bu kurallar geriye zincirleme motoruna geçirilir. İleriye zincirleme ile çıkarsanmış geriye zincirleme kurallarının sorgu zamanında çalıştırılması ile sorgu cevabı hesaplanır.

Sorgulama

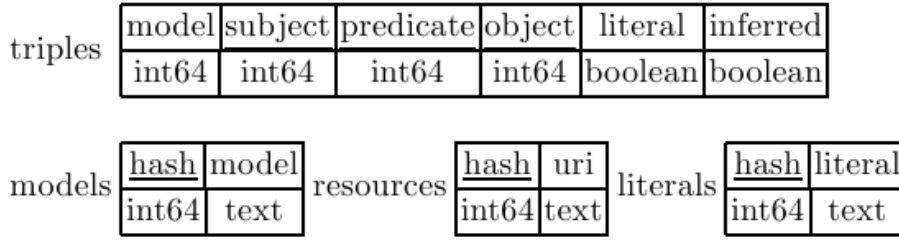
Jena'da sorgulama dili olarak RDQL desteklenmektedir. Jena2 harici bir çıkarsama motoru kullanılmasına izin verdiği gibi varsayılan çıkarsama motorlarını da içermektedir. Varsayılan çıkarsama motorları, "Transitive reasoner", "RDFS reasoner" ve "Generic rule reasoner" olarak listelenebilir. "Transitive reasoner" bileşeni sınıfların ve niteliklerin geçişkenlik ve simetri ilişkilerini hesaplar. "RDFS reasoner", tüm RDFS kurallarını destekler. "Generic rule reasoner" ise RDF çizgeleri üzerinde kural tabanlı çıkarsamayı destekler.

8.6 3Store

3Store (Harris and Gibbins, 2003) melez çıkarsama mekanizmalarını destekleyen bir çıkarsama motorudur. Çıkarsama kurallarını sınıflandırarak daha az çıkarsamaya neden olan kuralları ileriye zincirleme yöntemiyle çalıştırır. Diğer kurallar ise geriye zincirleme sırasında çalıştırır. 3Store, RDF(S) dilini desteklemektedir.

Depolama

3Store ontoloji üçlülerini veritabanında saklar. SQL veritabanı şemasındaki tablolar ve bu tablolar arasındaki ilişkiler Şekil 8.2'de gösterilmektedir. Kaynaklar ve sabitler ayrı tablolarda saklanırlar. Bu tabloların birincil anahtarları, kaynak ve sabit değerlerinden belirli bir fonksiyona göre üretilen anahtarlardır. Kaynakların ve sabitlerin anahtarlanması sırasında aynı fonksiyon kullanıldığı için, üçlü tablolarında üçlüye ait nesnenin resource veya literal olduğunu belirten bir saha bulunur.



Şekil 8.2. 3store veritabanı şeması.

Çıkarsama

3Store melez bir çıkarsama yaklaşımını benimser. Daha az çıkarsamaya neden olacak kurallar üzerinde ileriye zincirleme algoritmasını uygular. Daha fazla çıkarsamaya neden olacak, fakat hesaplanma maliyeti düşük kurallar ise geriye zincirleme algoritması ve sorguların yeniden yazılması ile hesaplanırlar. 3Store *subPropertyOf* (*rdfs9*) ve *subClassOf* (*rdfs7x*) kurallarını geriye zincirleme yöntemi ile çalıştırır. Sorgulama anında sorguların yeniden yazımı ve ek koşullar eklenmesi ile sorgular cevaplanır. Örneğin, $(?x, rdf:type, c)$ koşulu $(?y rdf:subClassOf c)$ ve $(x rdf:type ?y)$ koşulları ile değiştirilir.

Sorgulama

3Store sorgulama sırasında sorgu yeniden yazımı yapmaktadır. RDQL biçiminde alınan sorgu veritabanı şeması üzerinde çalışabilecek SQL sorgularına dönüştürülmeden önce, ileriye zincirleme sırasında eksik kalan bilginin tamamlanması için sorguya gerekli koşullar eklenir. Sonuçta elde edilen SQL sorgusunun çalıştırılmasıyla cevap elde edilir.

8.7 Niteliksel Karşılaştırma

Tezde önerilen yaklaşımın, diğer yaklaşımlardan temel farkı uzantı tabanlı bilgi modeli ve bu model üzerinde sorgulama yapabilen algoritmadır. Bu model hem ileriye zincirlemenin hem de sorgu algoritmasının sorgu yazma karmaşıklığını ve yükünü azaltmaktadır.

SHER ve kullandığı "Summary ABox" yöntemi, tez çalışmasına en yakın yöntemdir. Uzantı tabanlı bilgi modeli ile dönüştürülen ontoloji, özet ABox'a ben-

zetilebilir. Ayrıca, çıkarsamanın dönüştürülen model üzerinde gerçekleştirilmesi de diğer bir benzer yöndür. Fakat, iki yaklaşımın özetleme mekanizmaları birbirinden farklıdır. Tez kapsamında önerilen yöntemin özetleme mekanizması daha basittir ve yalnızca TBox'da yapılan tanımlara bağlıdır. Buna karşın Summary ABox yaklaşımı tamamen ABox'a bağımlıdır, başka bir deyişle ABox'ın değişmesi durumunda özetleme işleminin de yenilenmesi gerekmektedir. Summary ABox yöntemiyle olan en temel farklılık ise bu yöntemin betim mantığı temelli olması ve tablo algoritmalarını kullanmasıdır.

Bu bölümde anlatılan tüm yaklaşımların ortak noktası ontolojilerin veritabanlarında saklanmasıdır. Depolama açısından farklılaştıkları nokta ise saklanan ontoloji üzerinde çıkarsama sırasında değişiklik yapılıp yapılmaması ve çıkarsama için ontolojinin belleğe aktarılmasının gerekip gerekmemesidir. Bu açıdan OWLDB tüm yöntemlerden farklı bir yerdedir, çünkü tüm çıkarsama veritabanı üzerinde SQL sorgularının çalıştırılmasıyla yapılmaktadır. Minerva ise özelleşmiş veritabanı şemasıyla farklılık göstermektedir. Bu özelleşmiş şema çıkarsama başarımı gözönünde tutularak tasarlanmıştır.

Cisimleştirme yöntemi de çalışmaların değerlendirilmesinde önemli bir ölçüttür. Tam cisimleştirme yapan çalışmalar çıkarsanması gereken tüm üçlülere çıkarsarken, kısmi cisimleştirme yapanlar tüm çıkarsamaları gerçekleştirmezler. Bu ölçüt, tez çalışmasının diğerleri ile olan temel farklılığı görülebilir ve tezin sunduğu temel katkılardan biridir. Tezde önerilen kısmi cisimleştirme yaklaşımı ile tüm üçlülerin cisimleştirilmesine gerek kalmaz. Üçlü özetleri ile gerekli çıkarsamaların tamamlanması ile ileriye ve geriye zincirleme aşamalarında performans artışı sağlanır.

	Veritabanı Şeması	Özetleme Mekanizması	Cisimleştirme Yöntemi	Temel Paradigma	Çıkarımın Yapıldığı Yer	İleriye Zincirleme	Geriyeye Zincirleme
SHER	standart	var	kısmi	DL	Bellek	yok	var
Minerva	özelleşmiş durağan şema	yok	tam	DL+Kural	Veritabanı	var	yok
OWLDB	standart	yok	tam	Kural	Veritabanı	var	yok
DLDB	özelleşmiş değişken şema	yok	tam	DL	Bellek	var	yok
Jena	standart	yok	tam	Kural	Bellek	seçimlik	seçimlik
3Store	standart	yok	kısmi	Kural	Bellek	var	var
Uzantı-tabanlı yaklaşım	standart	var	özel kısmi	Kural	Bellek	var (özet üzerinde)	var

Çizelge 8.1. Tez çalışmasının diğerleri ile karşılaştırılması.

8.8 Başarım Karşılaştırması

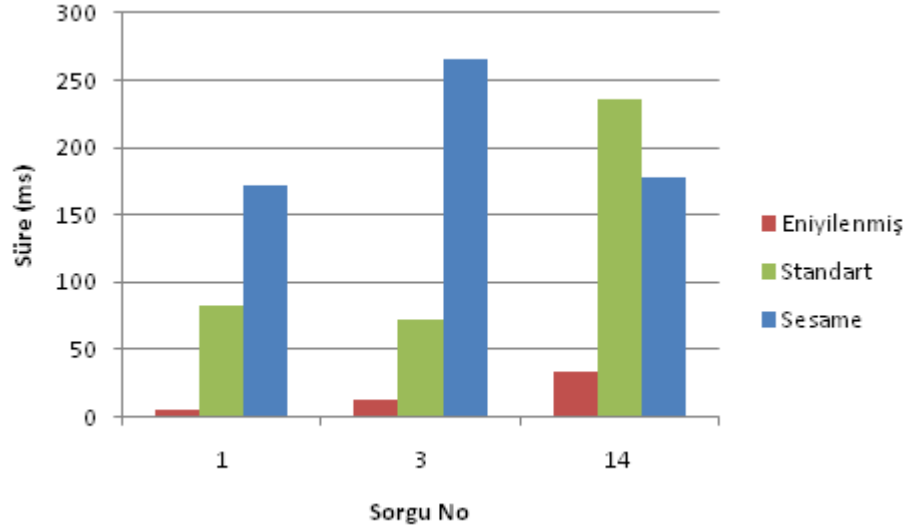
Bu bölümde, tez kapsamında önerilen yaklaşıma ait başarım, varolan kurallı tabanlı sistemlerle karşılaştırılmaktadır. Bu karşılaştırmalar için üç uygulama kullanılmaktadır: *Sesame*, *Jena* ve *OWLIM*. Takip eden alt başlıklar seçilen her bir uygulamanın ve önerilen yöntemin başarımını detaylı olarak karşılaştırmaktadır.

Bu başlıkta yeralan *Sesame* ve *Jena* deneylerinde ilgili uygulamaların başarım değerleri atıf yapılan yayınlardan alınmıştır (Kampman, 2006) (Moskal and Matheus, 2008). Bu değerlerin elde edildiği bilgisayarlar tez deneylerinde kullanılan bilgisayarlara çok yakın niteliktedir.

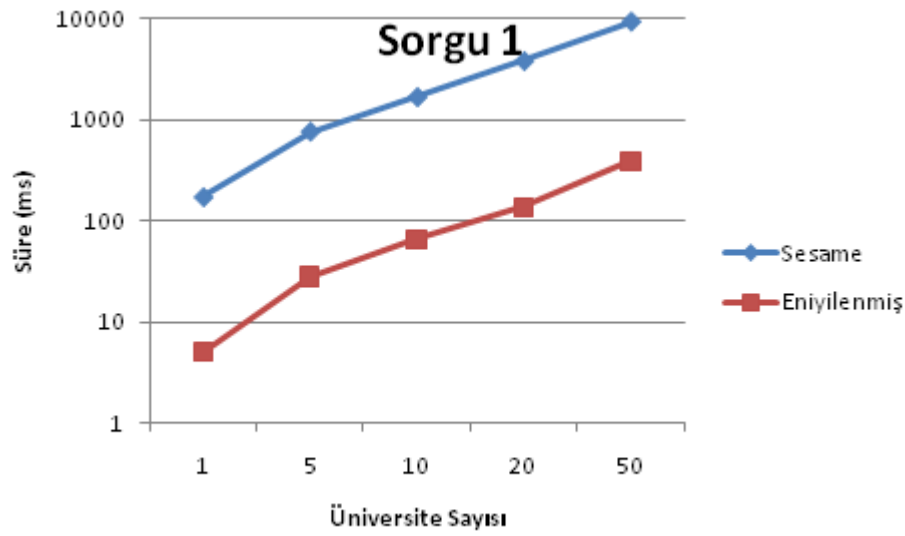
8.8.1 Sesame

Sesame (Broekstra et al., 2002) ontolojilerin saklanması, sorgulanması ve üzerinde çıkarsama yapılabilmesi için tasarlanmış bir çerçeve sistemdir. Çıkarsama seviyesi RDFS seviyesindedir ve ontoloji depoya eklenirken çıkarsanan bilgiler de depoya eklenir. Fakat, bu değerlendirmede çıkarsama yapılmadan ontolojiler yüklenmiştir (Kampman, 2006).

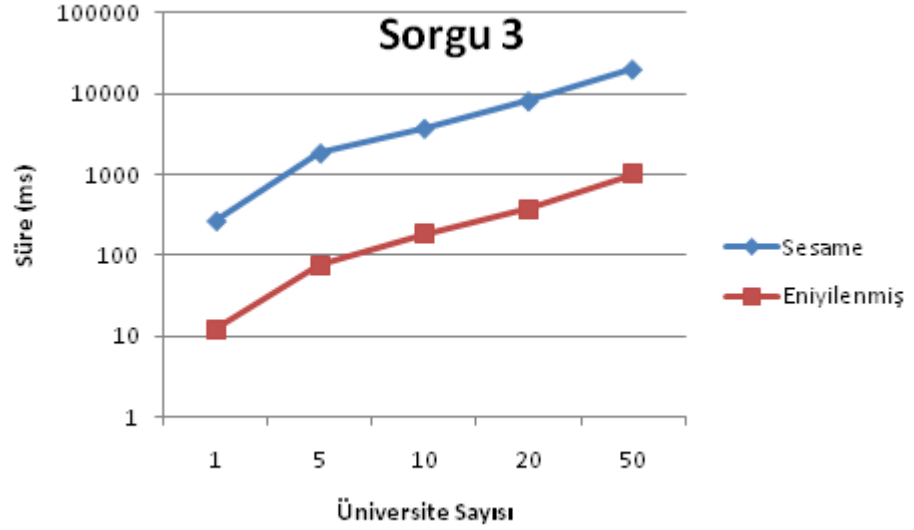
Sesame bu testlerde saf bir üçlü deposu olarak kullanılmaktadır ve hiç çıkarsama yapmayan saf bir üçlü deposunun başarımının tezde yapılan uygulamayla olan başarım farkı değerlendirilmektedir. Bu nedenle bu testte çıkarsama gerektirmeyen sorgular olan 1, 4 ve 14 numaralı sorgular seçilmiştir. Şekil 8.3'te verilen sorgu başarımlarına bakıldığında standart uygulamanın *Sesame* seviyesinde başarım gösterdiği görülmektedir. Bu test bir üniversite üzerinde gerçekleştirilmiştir. Şekil 8.4, Şekil 8.5 ve Şekil 8.6 ise 1, 5, 10, 20 ve 50 üniversitelik setler üzerinde, *Sesame* ve eniyilenmiş çıkarsama motorunun 1, 4 ve 14 numaralı sorgulara ait başarımlarını göstermektedir. Eniyilenmiş çıkarsama motoru, sorgu cevaplama aşamasında çıkarsama yapmasına karşın *Sesame*'den daha iyi bir başarım göstermektedir. Yalnızca, *Sorgu 14*'ün ölçeklenebilirliği (bkz. Şekil 8.6) *Sesame*'ye göre daha kötüdür.



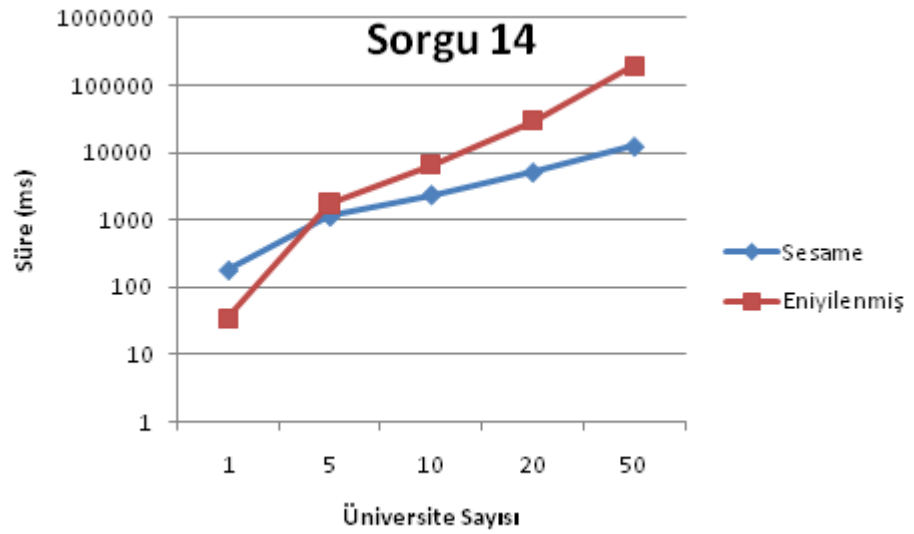
Şekil 8.3. Sesame, standart çıkarsama motoru ve eniyilenmiş çıkarsama motorunun 1, 4 ve 14 numaralı sorgu başarımları.



Şekil 8.4. Sesame ve eniyilenmiş çıkarsama motorunun artan veri miktarıyla 1 numaralı sorgu başarımları.



Şekil 8.5. Sesame ve eniyilenmiş çıkarsama motorunun artan veri miktarıyla 4 numaralı sorgu başarımları.



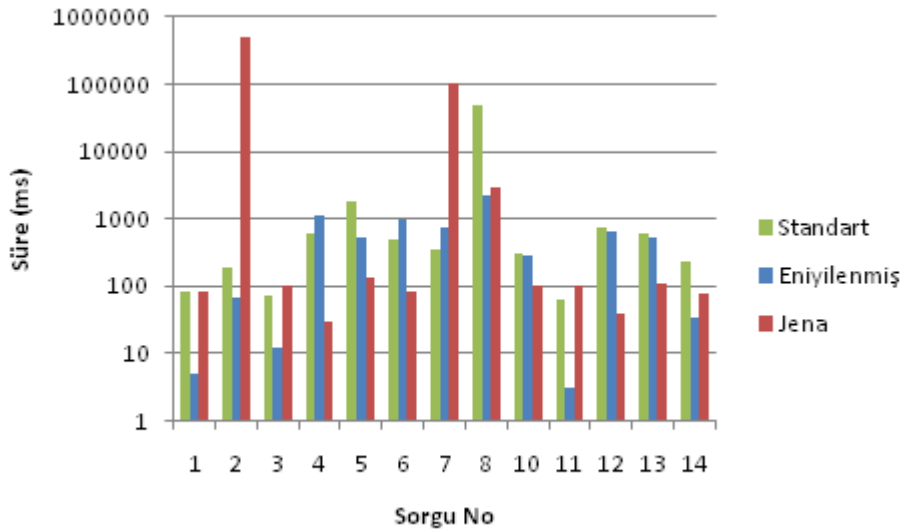
Şekil 8.6. Sesame ve eniyilenmiş çıkarsama motorunun artan veri miktarıyla 14 numaralı sorgu başarımları.

8.8.2 Jena

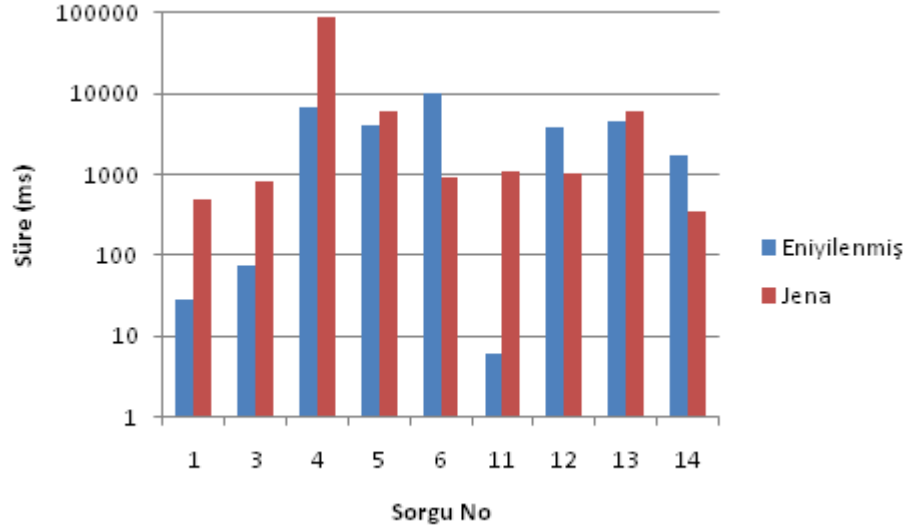
Jena (bkz. Bölüm 8.5) en bilinen kural tabanlı çıkarsama motorudur. Jena yalnızca bir ve beş üniversitelik veri kümelerini yükleyebilmektedir. 10 üniversitelik veri kümesini bellek yetersizliğinden dolayı yükleyememektedir (Moskal and

Matheus, 2008). Bu açıdan değerlendirildiğinde Jena'nın başarımı tezde önerilen sistemden kötüdür.

Sorgu başarımı karşılaştırmaları ise Şekil 8.7 ve Şekil 8.8'de gösterilmektedir. Şekil 8.7'de bir üniversitelik küme için sorgu başarımı karşılaştırması yapılmaktadır. Şekilde belirtilmesi gereken önemli noktalardan bir dokuz numaralı sorgunun Jena'da yine bellek yetersizliği nedeniyle çalışmamasıdır. Şekil 8.8'de ise beş üniversitelik küme için sorgu başarımı karşılaştırması yapılmaktadır. Bu testte ise 2, 7, 8, 9 ve 10 numaralı sorgular bellek yetersizliği nedeniyle çalışmamıştır.



Şekil 8.7. Jena, standart çıkarsama motoru ve eniyilenmiş çıkarsama motorunun bir üniversite için sorgu başarımları.

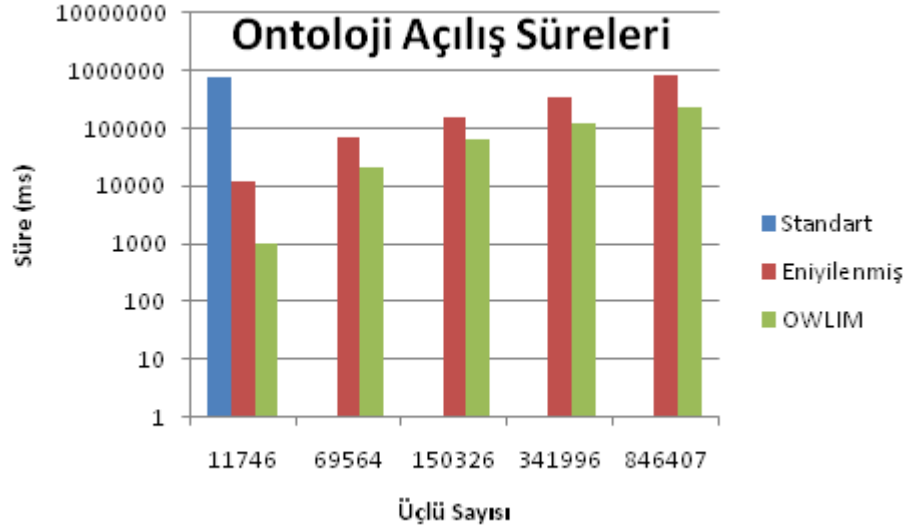


Şekil 8.8. Jena ve eniyilenmiş çıkarsama motorunun beş üniversite için sorgu başarımları.

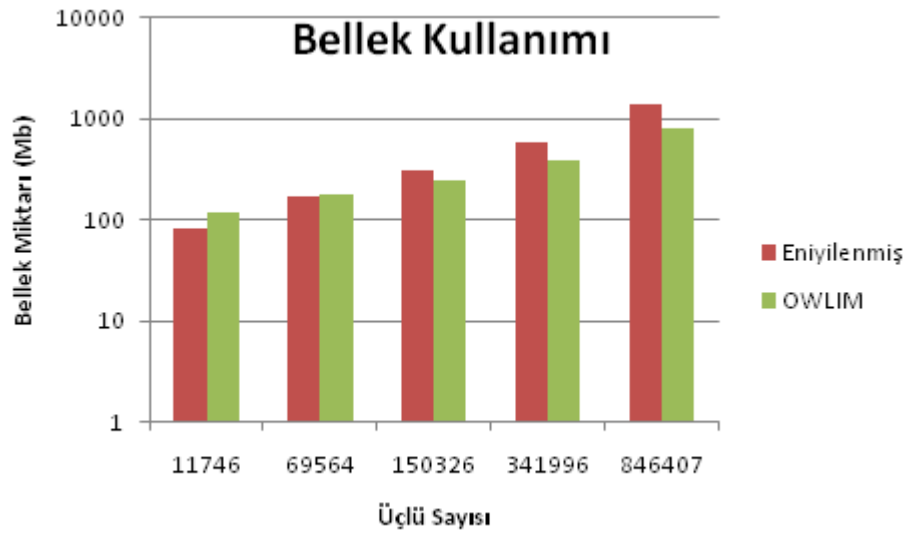
8.8.3 OWLIM

OWLIM (Kiryakov et al., 2005) yüksek performanslı bir ileriye zincirleme motorudur. Çıkarsama ve sorgulama bellekte gerçekleştirilir ve ontoloji dosyaları N-Triple dosyalarında saklanır. OWLIM çıkarsama motoru performansı arttırmak için bazı özel indeksleme mekanizmaları kullanır. Şu anda çıkarsama için en yüksek performansı sağlayan araçtır ve endüstri standardı olarak kabul edilmektedir.

Bu bölümde, OWLIM çıkarsama motorunun 1, 5, 10, 20, 50 üniversitelik kümeler için yapılmış olan testleri eniyilenmiş ve standart yöntemler ile karşılaştırılacaktır. Şekil 8.9'da tezde kullanılan standart çıkarsama motoru, eniyilenmiş çıkarsama motoru ve OWLIM'in ontoloji açılış süreleri karşılaştırılmaktadır. Şekil 8.10'da ise eniyilenmiş çıkarsama motoru ve OWLIM'in bellek kullanım karşılaştırılması verilmektedir.



Şekil 8.9. Standart çıkarsama motoru, eniyilenmiş çıkarsama motoru ve OWLIM’in ontoloji açılış süreleri.



Şekil 8.10. Eniyilenmiş çıkarsama motoru ve OWLIM’in bellek kullanımı.

9 Sonular

Bu tez alıřması, OWL ontoloji diline gre zelleřmiř kural tabanlı bir melez ıkarsama algoritması sunmaktadır. ıkarsama algoritmasının, iki nemli katkısı bulunmaktadır. İlki, uzantı tabanlı bilgi modelidir. Bu model, ontolojideki kavram rneklerine ait bilgiler zerinde bir eřit zetleme mekanizması gibi alıřmaktadır. Bylelikle, ileriye ve geriye doėru ıkarsama algoritmalarının daha verimli alıřması iin bir temel oluřturmaktadır. Ontolojilerin, uzantı tabanlı modele dnřtrlmesi iin harcanan sre ontolojilerin ayrıştırılması iin harcanan sre ile neredeyse eřittir. Bu sre, uzantı tabanlı modelin kullanılmasıyla ileriye doėru ıkarsama srecinde saėlanan zaman ve bellek kazancının yanında ihmal edilebilir boyutlarda kalmaktadır. Ayrıca uzantı tabanlı modelin kullanılması, ileriye doėru ıkarsama sonucu ıkarsanan l sayısında nemli miktarda dřře neden olmaktadır (bir niversite iin ıkarsanan l sayısında yz katlık bir dřř gzlemlenmektedir). Dřř oranı, rnek verisinin boyutu ile doėru orantılı olarak artmaktadır.

İkinci katkı, uzantı tabanlı modele dayalı melez ıkarsama yaklařımıdır. Bu yaklařım, ıkarsama srecinin iki kısma ayırmaktadır. İlk kısım, ileriye doėru ıkarsama algoritmasını uygulayarak, kavramlar (sınıflar ve nitelikler) arası hiyerarřik iliřkiler ve bu iliřkilere dayalı yeni kavram-uzantı iliřkileri ıkarsamaktadır. Bu seviyede yapılan ıkarsama RDFS seviyesindedir ve ıkarsanan kavram-uzantı iliřkileri, ikinci kısımda oluřturulan sorguların karmařıklıėının azalmasında ok nemli bir role sahiptir.

İkinci kısım, sorunun yeniden yazımına dayalı bir geriye doėru ıkarsama algoritması uygulamaktadır. Sorgu yeniden yazım mekanizmaları, uzantı tabanlı model ve OWL gerektirmeleri sonucu oluřturulmuř mekanizmalardır. Birinci kısımda gerekleřtirilen RDFS seviyesindeki ıkarsamayı, uzantı tabanlı modelin gereksinimleri doėrultusunda OWL seviyesine ıkarırlar. Her bir OWL yapıtařı ile ilgili ıkarsama, bir SQL sorgusu ile eřleřtirilmiřtir. Gerekli olduėunda, bu SQL sorgularının alıřtırılması ile OWL seviyesindeki ıkarsamalar, sorgu zamanında tamamlanmıř olur. Bu ařamada, sorgu kořullarında, rnek seviyesi yerine uzantı seviyesinde bileřenler kullanılması sorgu karmařıklıėını nemli lde dřrmekte

ve sorgu performansına katkıda bulunmaktadır.

Tez çalışmasında sunulan çıkarsama algoritması deneysel bir çıkarsama motoru üzerinde denenmiştir. Yaklaşımın uygulanması ile çıkarsama motorunun ontoloji açılış zamanları (*bkz. Bölüm 7.5*) ve bellek gereksinimi önemli ölçüde düşüş göstermiştir. Yaklaşımın temel mantığı düşünüldüğünde, geriye doğru zincirleme algoritmasının kullanılması sonucunda tüm sorguların performanslarında genel bir düşüş beklenebilir. Fakat bu düşüş, uzantı tabanlı modelin ve veritabanı teknolojinin kullanılması ile avantaja dönüştürülmüş ve sorgu performanslarında genel olarak bir iyileşme gözlemlenmiştir. Az sayıda sorgunun daha yavaş çalıştığı gözlemlense de, bu yavaşlama beklenmektedir.

Bu teknik önemli performans artışları sağlasa da, hala yapılabilecek eniyilemeler bulunmaktadır. Öncelikli gelecek çalışmalar listesinde yer alan eniyilemeler iki grupta incelenebilir. İlk grupta, veritabanı üzerinde yapılabilecek eniyilemeler yer almaktadır. Veritabanı şemasının özelleştirilmesi (Minerva sisteminde kullanılan şema ile benzer şekilde) ile veritabanındaki çıkarsama, saklama ve sorgulama performansları önemli ölçüde artacaktır. Bu yaklaşıma kolaylıkla uygulanabilecek bir başka teknik *dikey ayrıştırma* ("vertically partitioning" (Abadi et al., 2007)) tekniğidir. Bu tekniğe göre, *statements* tablosu n tane iki sütunlu tabloya ayrıştırılacaktır (n veri setindeki nitelik sayısı olmak üzere). Her bir tablonun ilk sütunu belirli bir niteliğe ait özneleri, ikinci sütunu ise aynı niteliğe ait nesneleri saklayacaktır. Her tablo öznelerine göre sıralı olarak saklanacaktır, böylelikle belirli bir özneye ait nitelik değerlerine kolaylıkla ulaşılabilecektir. Nesne sütunu isteğe bağlı olarak indekslenebilir. Anlaşılabileceği üzere, dikey ayrıştırma tekniği, sunulan yaklaşımla kolaylıkla entegre edilebilir ve sorgu performanslarını önemli ölçüde iyileştirebilir.

İkinci grupta yer alan iyileştirmeler sorgu yeniden yazım algoritması ile ilgilidir. Sorgu yeniden yazım algoritması ağırlıklı olarak uzantıların ve uzantılara ait elemanların toplanması ile ilgilidir. Bu aşamada, gereksiz yere boş uzantılara erişim sağlanmaktadır. Bu boş uzantıların elenmesi ile sorgu zamanları önemli ölçüde iyileştirilebilir. Boş uzantıların elenmesi "magic sets" adıyla bilinen eniyileme (Bancilhon et al., 1986) ile temel benzerlikler göstermektedir. Ayrıca, oluşturulan sorgu ağacında bir düğüm birden fazla yerde görülebilmektedir. Ortak düğümlerin pay-

laşılmasının (Rete algoritmasında (Forgy and Shepard, 1987; Doorenbos, 1995) olduğu gibi) zaman ve bellek gereksiniminde düşüş sağlayacağı görülmektedir. Bir başka olası iyileştirme, çalıştırılan sorgular arasındaki ortak görevlerin belirlenmesi ve yalnızca bir kez çalıştırılması prensibine dayanmaktadır (Sellis, 1988). Bunların yanısıra (Chaudhuri, 1998) ve (Heymans et al., 2008)'de sunulan çalışmalar da uygulanmaya aday eniyilemeler arasında yer almaktadır.

Kaynakça

- Abadi, D. J., A. Marcus, S. R. Madden, and K. Hollenbach: 2007, 'Scalable semantic web data management using vertical partitioning'. In: *VLDB '07: Proceedings of the 33rd international conference on Very large data bases*. pp. 411–422, VLDB Endowment.
- Antoniou, G. and F. van Harmelen: 2004, *A Semantic Web Primer (Cooperative Information Systems)*. The MIT Press.
- Auer, S. and Z. Ives: 2007, 'Integrating ontologies and relational data'. Technical Report MS-CIS-07-24, Computer and Information Sciences Department, School of Engineering and Applied Science, University of Pennsylvania.
- Baader, F., D. Calvanese, D. L. McGuinness, D. Nardi, and P. F. Patel-Schneider (eds.): 2003, *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press.
- Baader, F. and U. Sattler: 2000, 'An Overview of Tableau Algorithms for Description Logics'. *Studia Logica* **69**, 2001.
- Bancilhon, F., D. Maier, Y. Sagiv, and J. D. Ullman: 1986, 'Magic sets and other strange ways to implement logic programs (extended abstract)'. In: *PODS '86: Proceedings of the fifth ACM SIGACT-SIGMOD symposium on Principles of database systems*. New York, NY, USA, pp. 1–15, ACM.
- Berners-Lee, T., J. Hendler, and O. Lassila: 2001, 'The Semantic Web: Scientific American'. *Scientific American*.
- Broekstra, J., A. Kampman, and F. van Harmelen: 2002, 'Sesame: A generic architecture for storing and querying rdf and rdf schema'.
- Carroll, J. and J. Roo: 2004, 'OWL Web Ontology Language Test Cases. W3C Recommendation 10 February 2004. <http://www.w3.org/TR/2004/REC-owl-test-20040210/>'.

- Chaudhuri, S.: 1998, 'An overview of query optimization in relational systems'. In: *PODS '98: Proceedings of the seventeenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*. New York, NY, USA, pp. 34–43, ACM.
- Das, S., E. I. Chong, Z. Wu, M. Annamalai, and J. Srinivasan: 2008, 'A Scalable Scheme for Bulk Loading Large RDF Graphs into Oracle'. In: *ICDE*. pp. 1297–1306.
- Dean, M., G. Schreiber (eds.), F. van Harmelen, J. Hendler, I. Horrocks, D. McGuinness, P. Patel-Schneider, and L. Stein: 2003, 'OWL Web Ontology Language Reference'. W3C Recommendation 10 February 2004.
- Ding, L., P. Kolari, Z. Ding, S. Avancha, T. Finin, and A. Joshi: 2005, 'Using Ontologies in the Semantic Web: A Survey'. Technical report, UMBC.
- Dolby, J., A. Fokoue, A. Kalyanpur, A. Kershenbaum, E. Schonberg, K. Srinivas, and L. Ma: 2007, 'Scalable Semantic Retrieval through Summarization and Refinement'. In: *AAAI*. pp. 299–304.
- Doorenbos, R. B.: 1995, 'Production matching for large learning systems'. Ph.D. thesis, Pittsburgh, PA, USA.
- Fokoue, A., A. Kershenbaum, L. Ma, E. Schonberg, and K. Srinivas: 2006, 'The Summary Abox: Cutting Ontologies Down to Size'. In: *International Semantic Web Conference*. pp. 343–356.
- Forgy, C.: 1982, 'Rete: A Fast Algorithm for the Many Patterns Many Objects Match Problem'. *Artif. Intell.* **19**(1), 17–37.
- Forgy, C. L. and S. J. Shepard: 1987, 'Rete: a fast match algorithm'. *AI Expert* **2**(1), 34–40.
- Friedman-Hill, E.: 2002, *Jess in Action: Java Rule-Based Systems (In Action series)*. Manning Publications.
- Grosof, B. N., R. Volz, I. Horrocks, and S. Decker: 2003, 'Description Logic Programs: Combining Logic Programs with Description Logic'. In: *Proceedings of WWW2003*. pp. 48–57, ACM.

- Gruber, T. R.: 1995, 'Toward principles for the design of ontologies used for knowledge sharing'. Vol. 43. Duluth, MN, USA, pp. 907–928, Academic Press, Inc.
- Guo, Y., Z. Pan, and J. Heflin: 2004, 'An Evaluation of Knowledge Base Systems for Large OWL Datasets.'. In: *International Semantic Web Conference*. pp. 274–288.
- Haarslev, V. and R. Möller: 2001, 'RACER System Description'. In: *IJCAR '01: Proceedings of the First International Joint Conference on Automated Reasoning*. London, UK, pp. 701–706, Springer-Verlag.
- Harris, S. and N. Gibbins: 2003, '3store: Efficient Bulk RDF Storage'. In: *PSSS*.
- Hayes, P.: 2004, 'RDF Semantics'. W3C Recommendation, <http://www.w3.org/TR/rdf-mt/>, (Eriřim Tarihi : 15 Ekim 2009).
- Heymans, S., L. Ma, D. Anicic, Z. Ma, N. Steinmetz, Y. Pan, J. Mei, A. Fokoue, A. Kalyanpur, A. Kershenbaum, E. Schonberg, K. Srinivas, C. Feier, G. Hench, B. Wetzstein, and U. Keller: 2008, 'Ontology Reasoning with Large Data Repositories'. In: *Ontology Management*. pp. 89–128.
- Hitzler, P. and D. Vrandecic: 2006, 'Faster OWL Using Split Programs'. In: F. Bry, F. Fages, M. Marchiori, and H.-J. Ohlbach (eds.): *Principles and Practices of Semantic Web Reasoning*. Internationales Begegnungs- und Forschungszentrum fuer Informatik (IBFI), Schloss Dagstuhl, Germany. <<http://drops.dagstuhl.de/opus/volltexte/2006/480>> [date of citation: 2006-01-01].
- Horrocks, I.: 1998, 'The FaCT System'. In: *TABLEAUX '98: Proceedings of the International Conference on Automated Reasoning with Analytic Tableaux and Related Methods*. London, UK, pp. 307–312, Springer-Verlag.
- Horrocks, I., L. Li, D. Turi, and S. Bechhofer: 2004, 'The Instance Store: Description logic reasoning with large numbers of individuals'. In: *International Workshop on Description Logics (DL 2004)*. pp. 31–40.
- Ioannidis, Y. E.: 1996, 'Query optimization'. *ACM Comput. Surv.* **28**(1), 121–123.

- Kampman, A.: 2006, 'NativeStore LUBM benchmark results'. <http://www.openrdf.org/forum/mvnforum/viewthread?thread=829>. (Eriřim Tarihi: 13 Ekim 2009).
- Kiryakov, A.: 2008, 'Measurable Targets for Scalable Reasoning'. Technical report, The Large Knowledge Collider: a platform for large scale integrated reasoning and Web-search.
- Kiryakov, A., D. Ognyanov, and D. Manov: 2005, 'OWLIM - A Pragmatic Semantic Repository for OWL'. In: *WISE Workshops*. pp. 182–192.
- Li, H., Y. Wang, Y. Qu, and J. Z. Pan: 2006, 'A Reasoning Algorithm for pD*'. In: *ASWC*. pp. 293–299.
- Ma, L., Y. Yang, Z. Qiu, G. Xie, and Y. Pan: 2006, 'Towards A Complete OWL Ontology Benchmark'. In: *3rd European Semantic Web Conference (ESWC2006)*.
- McBride, B.: 2001, 'Jena: Implementing the RDF Model and Syntax Specification'. In: *SemWeb*.
- Meditkos, G. and N. Bassiliades: 2009, 'Rule-based OWL Reasoning Systems: Implementations, Strengths and Weaknesses'. In: *Handbook of Research on Emerging Rule-Based Languages and Technologies: Open Solutions and Approaches*. pp. 124–149, IGI Global.
- Mei, J., L. Ma, and Y. Pan: 2006, 'Ontology Query Answering on Databases'. In: *International Semantic Web Conference'06*. pp. 445–458.
- Moskal, J. and C. J. Matheus: 2008, 'Detection of Suspicious Activity Using Different Rule Engines – Comparison of BaseVISor, Jena and Jess Rule Engines'. In: *RuleML '08: Proceedings of the International Symposium on Rule Representation, Interchange and Reasoning on the Web*. Berlin, Heidelberg, pp. 73–80, Springer-Verlag.
- Motik, B., B. C. Grau, I. Horrocks, Z. Wu, A. Fokoue, and C. Lutz: 2008, 'OWL 2 Web Ontology Language: Profiles'. World Wide Web Consortium, Working Draft WD-owl2-profiles-20081202.

- Ontotext: 2009, 'RDF(S), Rules, and OWL Dialects'. www.ontotext.com/inference/rdfsrulesowl.html. (Eriřim Tarihi: 11 Ekim 2009).
- Özacar, T., Ö. Öztürk, and M. O. Ünalir: 2007, 'A Pragmatic Approach for RDFS Reasoning over Large Scale Instance Data'. In: *OTM Workshops (2)*. pp. 1155–1164.
- Öztürk, Ö., T. Özacar, and M. O. Ünalir: 2006, 'Reducing the Inferred Type Statements with Individual Grouping Constructs'. In: *International Semantic Web Conference*. pp. 573–582.
- Pan, Z. and J. Heflin: 2003, 'DLDB: Extending Relational Databases to Support Semantic Web Queries'. In: *PSSS*.
- Patel-Schneider, P. F., P. Hayes, and I. Horrocks: 2004, 'OWL Semantics'. <http://www.w3.org/TR/owl-semantics/>. (Eriřim Tarihi: 14 Ekim 2009).
- Patel-Schneider, P. F. and I. Horrocks: 2007, 'A comparison of two modelling paradigms in the Semantic Web'. *J. Web Sem.* **5**(4), 240–250.
- Schmidt-Schaubß, M. and G. Smolka: 1991, 'Attributive concept descriptions with complements'. *Artif. Intell.* **48**(1), 1–26.
- Sellis, T. K.: 1988, 'Multiple-Query Optimization'. *ACM Trans. Database Syst.* **13**(1), 23–52.
- Smith, M. K., C. Welty, and D. L. McGuinness: 2004, 'OWL Web Ontology Language Guide'. Technical report, OWL Web Ontology Language Guide.
- Stuckenschmidt, H. and J. Broekstra: 2005, 'Time - Space Trade-Offs in Scaling up RDF Schema Reasoning'. In: *WISE Workshops*. pp. 172–181.
- Tartir, S., I. B. Arpinar, M. Moore, A. P. Sheth, and B. Aleman-meza: 2005, 'On-toQA: Metric-based ontology quality analysis'. In: *In Proceedings of IEEE Workshop on Knowledge Acquisition from Distributed, Autonomous, Semantically Heterogeneous Data and Knowledge Sources*.

- ter Horst, H.: 2004, 'Extending the RDFS Entailment Lemma'. In: D. Plexousakis, S. McIlraith, and F. van Harmelen (eds.): *Proceedings of the 3rd International Semantic Web Conference (ISWC)*. Springer.
- ter Horst, H. J.: 2005, 'Completeness, decidability and complexity of entailment for RDF Schema and a semantic extension involving the OWL vocabulary'. *J. Web Sem.* **3**(2-3), 79–115.
- ter Horst, H. J.: 2006, 'Semantic Web Ontologies and Entailment: Complexity Aspects'. In: *Intelligent Algorithms in Ambient and Biomedical Computing*. pp. 215–242, Springer Netherlands.
- Weithöner, T., T. Liebig, M. Luther, S. Böhm, F. W. von Henke, and O. Noppens: 2007, 'Real-World Reasoning with OWL'. In: *ESWC*. pp. 296–310.
- Wilkinson, K., C. Sayers, H. Kuno, and D. Reynolds: 2003a, 'Efficient RDF Storage and Retrieval in Jena2'. Technical Report HPL-2003-266, Enterprise Systems and Data Management Laboratory, HP Laboratories Palo Alto.
- Wilkinson, K., C. Sayers, H. A. Kuno, and D. Reynolds: 2003b, 'Efficient RDF Storage and Retrieval in Jena2'. In: *SWDB*. pp. 131–150.
- Zhou, J., L. Ma, Q. Liu, L. Zhang, Y. Yu, and Y. Pan: 2006, 'Minerva: A Scalable OWL Ontology Storage and Inference System'. In: *ASWC*. pp. 429–443.

EkA TÜRKÇE-İNGİLİZCE TERİMLER SÖZLÜĞÜ

A

açık çevre varsayımı	open world assumption
alan karmaşıklığı	space complexity
altdil	sublanguage
altküme	subset
altnitelik	subproperty
altsınıf	subclass
anlamsal ağ	semantic network
anlamsal web	semantic web
anlatımlılık	expressivity
artalan	range
artbileşen	consequent
asıl	original
ayrıklık	disjointnessg
ayrıştırma	parsing

B

belirtik	explicit
betim mantığı	description logic
biçimsel	formal
biçimsel tanım	formal definition
bilgi gösterimi	knowledge representation
bilgi tabanı	knowledgebase
birinci basamak mantık	first order logic

C

cisimleşme	materialize
------------	-------------

Ç

çerçeve sistemi	frame based system
çevrimdışı	offline
çevrimiçi	online
çıkarsama	inference
çıkarsama motoru	inference engine
çizge	graph

D

dönüşlülük	reflexive
dönüşsüzlük	irreflexive
düğüm	node

E

eklemler mantığı	propositional logic
eksiksiz	complete
eksiksizlik	completeness
eniyileme	optimization
evrensel sınıf	top class

F

faydalanma oranı	utilization rate
fonksiyonel	functional

G

geçerli	sound
geçerlilik	soundness
geçişken	transitive
gerektirme	entailment
geriye zincirleme	backward chaining

İ

içlemsel	intensional
ifade	statement
ileriye zincirleme	forward chaining
ilksav	axiom

K

kalıcı	persistent
kapalı çevre varsayımı	closed world assumption
kapanış	closure
kaplamsal	extensional
karşılaştırmalı değerlendirme	benchmark
kaynak	resource
kısmi cisimleştirme	partial materialization
koşul	condition

L

listelenmiş sınıf	enumerated class
-------------------	------------------

M

melez	hybrid
-------	--------

N

nesne	object
nitelendirilmiş nitelik sayı kısıtı	qualified number restriction
nitelik sayı kısıtı	cardinality constraint

O

olgu	fact
------	------

Ö

ölçeklenebilirlik	scalability
-------------------	-------------

ölçüt	metric
önalın	domain
önbileşen	antecedent
önerme	statement
örnek	individual
örtük	implicit
özne	subject

P

pratik	pragmatic
--------	-----------

R

R-ardılı	R-successor
----------	-------------

S

sabit	literal
sağ el tarafı	right hand side
saha	field
sayı kısıtı	number constraint
sınıf örneği	class individual
sınırlı tikel niceleyici	existential quantifier
simetrik	symmetric
sol el tarafı	left hand side
sonlandırılabilirlik	decidability
sözde-kod	pseudo-code
sözdizimsel	syntactic

T

tablo algoritması	tableux algorithm
tam cisimleştirme	total materialization
ters fonksiyonel	inverse functional
ters roller	inverse roles

terslik	inverse
tikel niceleyici	full existential quantifier
tutarsızlık	inconsistency
tümdengelimli	deductive
tümdengelimli veritabanı sistemi	deductive database
tümevarımlı	inductive
tümel niceleyici	universal quantifier
tümleyen	complement

U

uslamlama	reasoning
uzantı	extension
uzman sistem	expert system

Ü

üçlü	triple
üçlü deposu	triple store

V

varlık	entity
--------	--------

Y

yapıtaşı	construct
yeniden yazım	rewriting
yığıt	stack
yorumlama	interpretation
yönlü kenar	directed arc
yüklem	predicate

Z

zaman karmaşıklığı	time complexity
--------------------	-----------------

Özgeçmiş

Adı Soyadı: Övünç Öztürk
Doğum Tarihi 13.11.1978
Doğum Yeri: İzmir
Uyruğu: T.C.
Görevi: Araştırma Görevlisi,
Ege Üniversitesi,
Bilgisayar Mühendisliği Bölümü.
Lise: Bornova Anadolu Lisesi, 1995.
Lisans: Ortadoğu Teknik Üniversitesi,
Bilgisayar Mühendisliği Bölümü, 2001
Yüksek Lisans: Ege Üniversitesi,
Bilgisayar Mühendisliği Bölümü, 2004
Dil: İngilizce, Almanca