YILDIZ TEKNİK ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

ETKİLEŞİMLİ BİR SORU-CEVAP TABİ DİL ARAYÜZÜ PROGRAMI TASARIMI

Bilgisayar Müh. Hakan ACU

F.B.E. Bilgisayar Bilimleri Mühendisliği Anabilim Dalında hazırlanan

YÜKSEK LİSANS TEZİ

Tez Danışmanı : Prof. Dr. M.Yahya KARSLIGİL

İSTANBUL, 1995

İÇİNDEKİLER :

TEŞEKKÜR :

Projenin başından sonuna kadar yardımlarını esirgemeyen ve yapıcı eleştirileriyle beni yönlendiren Bilgisayar Bilimleri ve Mühendisliği Bölüm Başkanımız , Sayın Prof. Dr. M.Yahya KARSLIGİL'e sonsuz teşekkürlerimi sunarım.

ÖZET :

Tabi dillerin işlenmesi , yapay zekanın doğurduğu alt bilim dallarından biridir. Bu bilim dalının temel uğraşım alanı , insan konuşma dillerinin belirli bir model çerçevesinde bilgisayara algılattırılması, kullanıcı ile bilgisayar arasındaki iletişimin tabi konuşma dili havasında sağlanmasıdır. Tabi dillerin işlenmesi yada analizi alanında geliştirilen uygulamalardan tabi dil arayüz programları üzerinde en fazla çalışılan ve uygulama geliştirilen konudur. Sorgulama amaçlı oluşturulan büyük veri tabanları ve uzman sistem programlarında etkileşimli bir soru-cevap ortamı yaratmak için hep bu arayüz programları kullanılmaktadır.

Bu gerçekten yola çıkarak , bu tez çalışmasında tabi dil arayüzüne sahip bir veritabanı tasarımı yapılmıştır. Projede ağırlıklı olarak arayüz programının tasarımı ön planda kalmış olup uygulamanın doğruluğunu gösterebilmek içinde orta büyüklükte bir veritabanı oluşturulmuştur. Projenin amacı herhangi bir alanda kullanıcıların günlük konuşma dili havası içinde öğrenmek istedikleri konuları bilgisayardan sorgulayabilmeleridir. Uygulama için kullanılan veritabanını İstanbul ile ilgili değişik konularda topladığım bilgilerden oluşturdum. Bundaki amacım özellikle bu ve ileride buna benzer olarak geliştirilecek programların turizm sektörlerinde kullanılabileceğini göstermektedir. Nitekim bu alanda yurtdışında kullanılan pek çok program mevcuttur.

Geliştirilen programın karakteristik özelliği; kullanıcının, giriş sorularını İngilizce soru cümleleri halinde sisteme verebilmesidir. Şu anda programın veritabanında aşağıdaki konularda bilgiler mevcuttur.

İstanbul'un genel özellikleri (nüfus, yüzölçümü, komşuluk bilgileri), ilçeleri (nüfus, yüzölçümü, komşuluk bilgileri), coğrafi bilgiler (dağlar, nehirler, göller), üniversiteler, oteller (5, 4, 3 ve 2 yıldızlı oteller), tarihi yapıtlar (camiler, türbeler, çeşmeler, kuleler, müzeler), valiler, belediye başkanları.

Bu bilgiler sayfalarca süren detaylı bilgiler olmayıp, sadece soru-cevap mekanizmasına uygun hazırlanmış birer ikişer cümlelik bilgilerdir.

Program tabi dil işleme metodlarından informal metodlar sınıfına giren **fark çiftlerinin ayrıştırılması (difference pair parsing)** tekniği ile yazılmıştır. Sistem kullanıcının girdiği sorgu ifadesinde önceden belirlenmiş sözcük şablonları yada kalıpları arar. Eğer böyle bir şablona rastlarsa oluşturduğu bir sorgulama cümlesiyle veritabanından istenen kıstaslara uyan bilgileri arayıp çıkarır. Bu sözcük şablonları anahtar sözcük arama tekniğinin bir adım ötesidir. Sistem, kullanılan programlama dilinin özelliğinden ötürü oluşturulan veritabanını belleğe almakta ve her türlü sorgulama işlemini bellekte gerçekleştirmektedir. Böylece, kullanılan tekniğinde özelliğinden ötürü sorgulama işlemleri çok hızlı bir şekilde gerçekleşmektedir.

ABSTRACT :

NLP (Natural Language Processing) is one part of artificial intelligence. The main study area of NLP is getting computers to understand human natural languages within frame of a certain model and thus, establishing the interaction between user and computer in something like everyday speech. Study of natural interfaces, one class of applied systems in NLP, is the most prevalent area of research. They are used in querying databases and expert system programs to build interactive question-answer sessions.

Bearing this fact in mind, in this thesis, a database with a natural language interface has been designed. The particular emphasis has been given to interface program design and development. To verify the methods used in program, a middle-scale database has been formed. The main purpose of this project is providing the users to query a database and learn what they wanted in like an everyday speech. I gathered mostly various information related to İstanbul and used them in forming the database aiming at this program and the programs that will be developed in future might serve in tourism sector. As a matter of fact, programs that serve for tourism purposes are available abroad.

The characteristic feature of the program developed in this thesis, is that users can give their queries as plain English sentences. For the present, there are information about following areas available in the database of the program.

General information about İstanbul (population, area, neighborhood information), counties (population, area, neighborhood information), geographical information (mountains, rivers, lakes), universities, hotels (five, four, three and two star hotels), historical monuments (mosques, turbes, fountains, towers), governors, mayors.

The information about above topics are not arranged as long and detailed, but simple and maximum one or two sentences long which is suitable for question-answering mechanism.

The program uses difference pair parsing technique which is widely used in natural language interface programs. System accepts user queries and after stripping out non-essential words out of query, looks for certain templates determined before. Templates are the high level patterns that carry information about the specifics of the question being asked and the form of the question. If it encounters such a template, it arranges a query and extract information that meet the criteria from the database. Templates are one step ahead of keyword skimmer idea. System, a feature of Prolog programming language, loads the database into memory at the start of the program execution and all query processes are performed in memory very quickly.

I. GENEL GİRİŞ

1.1. Tabi Dil Uygulamalarının Ortaya Çıkışı ve Kısa Bir Tarihçe

İlk bilgisayarlar sadece sayıları işleyebilen, dört temel matematiksel işlemi yapabilen basit hesap makinelerinden ibaretti. Bugün anladığımız anlamda bilgisayarların icadını 1940'lı yıllar olarak kabul edersek aradan geçen yarım asırlık sürede elektronik teknolojisinin gelişmesi dev bilgisayarların küçülmesine ve sınırlı kullanışından kitlesel yani bireysel kullanıma kadar uzanan olanaklar doğurmuştur. Ancak tüm değişimler içinde değişmeyen tek şey bilgisayarların hala sayı sistemlerine göre çalışmalarıdır. Bugünün bilgisayarları çok karmaşık ve çeşitlilikte verileri işleyebilme yeteneğine sahiptirler. Ancak bu verilerin hepsi sayı sistemleriyle temsil edilmektedir. Konuşma dillerine özgü olan objelerin en basiti sesler, heceler ve kelimeler de sayı sistemleriyle temsil edilmektedir. Örnek olarak GO kelimesini ele alalım. Birçok bilgisayarda bu kelime 2 ardışıl byte'la temsil edilir. (01000111=71) ve (01001111=79) sayıları G ve O'yu temsil eden ASCII kodlarıdır. Bilgisayara alfabetik olarak sıralanmak üzere bir isimler listesi verdiğimizi kabul edelim. Bu listede George isminin Olga isminden önce geldiğini görürüz. Ancak bu, birçoğumuzun ilk anda düşüneceği gibi G harfinin alfabede O harfinden önce gelmesinden değil G'yi temsil eden ASCII kodu 71 in O yu temsil eden ASCII kodu 79 dan küçük olmasından ötürüdür.

Bugün anladığımız anlamda olmasa da bilgisayarların tabi dillerin işlenmesi ve analizi alanında ilk kullanıldığı uygulamalar uzun yazı metinlerinde verilen bir kelimenin veya kelimeler topluluğunun aranması idi. Bu çalışmaların kullanılma amaçlarından birisi yazarı belli olmayan şüpheli yazı ve metinlerde the, upon, he, she, a, an v.s. gibi kelimelerin ne sıklıkta kullanıldığına dair istatistiki bilgilerin çıkarılması ve bunların yazarı belli olan metinlerden elde edilen aynı türdeki bilgilerle karşılaştırılarak kimliği belirsiz yazıların kimin tarafından yazılmış olacağına dair gerçekçi ve somut yorumlar yapabilmekti. Komputasyonel linguistik denen bu faaliyet alanında bilgisayarlardan faydalanılan bir başka uygulamada dış dünyadan optik okuyucular, scannerler veya başka yollarla bilgisayara "okutulan" yazılara ilişkin konu indekslemelerinin yapılması çalışmalarıydı. Günümüzde sözü edilen çalışmalar aynı derecede olmasa da devam

etmektedir. Fakat teknikler o kadar geliştirildi ki artık sıradan kelime işlemci programlarında bile indeksleme fonksiyonunu yerine getiren özel modüller bulunmaktadır. Bu da bilgisayarlara komputasyonel linguistik alanında duyulan eski rağbeti azalttı.

Tabi dilleri işlemedeki pekçok gelişme aslında bilgisayarların yazılım ve donanımlarındaki gelişmelerle ve kullanılma amaçlarındaki çeşitliliklerle paralel gitmiştir. Sayısal işlemlerde bilgisayarlar oldukça iyi olmalarına karşın onları yine de genel sembol işleme makineleri olarak düşünmek yerinde olur. Bilgisayarların işleyebilecekleri semboller sayılar olduğu gibi kelimeler, cümleler, ağaçlar yada ağlar gibi kompleks objeler de olabilir.

Tabi dilleri işleme alanında kaydedilen dönüm noktalarından birisi hiç kuşkusuz 1971 yılında Winograd adlı bilim adamının geliştirdiği SHRDLU adlı Lisp'te yazılan programdır. Winograd yapay zeka çalışmalarına özellikle de tabi dillerin analizi alanında önemli katkıları bulunmuş bir bilim adamıdır. SHRDLU programı çok kısıtlı ve ilkel bir çerçeve içinde de olsa bazı soruları, ifadeleri, ve komutları yorumlama , sonuçlar çıkarma yeni kelimeler öğrenebilme v.s. gibi yeteneklere sahip olup bu o zamana kadar bir bilgisayar programında görülmemiş şeylerdi.

Yapay zeka ve onun bir alt bilim dalı olan tabi dillerin analiziyle ilgili geliştirilen programlama dilleri klasik-yapısal programlama dillerinden bazı farklılıklar arzeder. Bu farklılıklardan bazıları şunlardır. Öncekinde problemin çözümü için gerekli olan her türlü veriler fact ve rule (gerçek ve kural) adı verilen iki ayrı bölümde programın başında tanımlanır. Daha sonra sistem tarafından herbir olası durum denenerek çözümler elde edilir. Oysa bildiğimiz anlamda yapısal dillerde (Pascal, C v.s.) belirli bir problemi çözmek için gerekli olan komutlar adım adım bilgisayara verilir. Problemin çözümü için formüller ve algoritmalar kullanılır. Oysa yapay zeka dillerinde böyle bir metod kullanılmaz.. Kurala bağlı sistem tasarımlarında özellikle uzman sistem programlarında bu mekanizmaya göre çalışan programlama dilleri ve programlar önemli ölçüde başarılı olmuşlardır. Prolog, OPS5 gibi deklaratif (tanımsal) diller geliştirilmiştir. Tabi dil işlemede geliştirilen uygulamalarda programcı kullanacağı

grameri tıpkı bir dilbilimcisinin bir grameri tanımladığı gibi programın başında tanımlar, kurallarını, kıstaslarını belirler ve işin bundan sonraki kısmını bilgisayara bırakır. Bilgisayar bu gramerin kurallarına göre yeni cümleler üretir, girilen cümlelerin bu gramere göre doğruluklarını inceler.

1.2. Tabi Dillerin İşlenme Amaçları

Tabi dillerin işlenmesi kavramı 1960'ların ilk yarısında yapay zeka çalışmalarının başlaması ile ortaya çıkmıştır. Bazen tabi dillerin analizi, linguistik analiz, otomatik dil analizi v.s. gibi adlarla anılmışsa da tabi dillerin işlenmesi en çok tercih edilenidir. Tabi dillerin işlenmesindeki temel araştırma sahaları ve amaçlar şunlardır.

- Linguistik sistemlerin modellenmesi ve geliştirilmesi.
- Geliştirilen modellerin pratik hayatta uygulamaya geçirilmesi.
- Bu sistemlerin insan-makine ilişkisi açısından değerlendirilmesi.

İlk çalışma sahasındaki amacın gerçekleştirilmesi için genel dil kuralları ve bunlarla ilgili geliştirilmiş teorilerin bilgisayar ortamına çok iyi bir adaptasyonu gereklidir. Çünkü, bu teoriler geliştirilirken bunların bir gün bilgisayar ortamına taşınıp uygulamaya konacağı düşünülmemiştir. Dolasıyla sözü edilen teoriler bilgisayar ortamındaki tabi dil uygulamalarına yönelimli değildirler. Bu güçlüğe çözüm olarak öne sürülen yaklaşımlardan birisi, tabi diller ile programlama dilleri arasındaki benzerlikten yola çıkarak , programlama dillerindeki iyi oluşturulmuş söz dizimlerinin analizi için kullanılan metodların kullanılmasıdır. Kuşkusuz tabi dillerin kendine has özellikleri bu kısıtlı metodlara ekleme ve yeni geliştirmeler yapma zorunluluğunuda beraberinde getirir.

Bir tabi dil uygulamasının (arayüzler, makine çevrimi, bilgisayar destekli öğrenme v.s.) getirdiği güçlüklerden biri de mimarinin seçimidir. Tabi dil ve programlama dili arasındaki ilişkide oldukça önemlidir. Özellikle bilginin temsil edilmesinde kullanılan teori ve teknikler tabi dillerin anlamsal yönündeki araştırmaları derinden etkilemektedir. Kullanılacak bilgisayarın ve programlama dilinin seçimi de ayrı bir önem arzetmektedir.

Son olarak, bir değerlendirme safhası bize oluşturulan sistemin kalitesi, güvenirliliği ve kullanıcıya dost olması gibi konularda karar vermemizi sağlayacaktır.

1.3. Tabi Diller ve Yapay Dillerin (Programlama Dillerinin) Bir Karşılaştırılması

Yapay dillerin , oldukça ilkel bir düzeyde de olsa , ortaya çıkışını mantık biliminin doğuşuyla birlikte kabul edersek bunların yüzyıllardan beri tabi dillerle beraber var olduğu sonucuna varırız. Yapay diller sonuç itibarıyla insanoğlu tarafından geliştirildikleri için tamamıyla insan kontrolu altındadırlar. Burada söz konusu ettiğimiz yapay diller matematik, hesaplama, enformasyon bilimlerinde kullanılan sembollerdir. Yapay dillerin en genel anlamda amaçları oldukça kısıtlı bir kelime haznesi ve nispeten basit yapılarla belirli bir işi veya olayı tasvir etmektir. Bir yapay dil, hedeflenen işi kuramsal ve soyut bir seviyede temsil etmeye ve ardından modellemeye müsade eder.

Tabi diller ile yapay diller arasında büyük farklılıklar vardır. Bu farklılıklar sadece her iki gruptaki dillerin yapılarındaki teknik özelliklerden ötürü değil tarihi ve sosyal faktörlerden dolayı da oluşmaktadır. Ayrıca bu farklılıklar şu gerçeği de göstermektedir ki, tabi dillerin yapısal ve anlamsal bakımdan analizleri ile yapay dillerin aynı alandaki analizleri bakımından benzerlikler olmasına rağmen olayın diğer yönleri tamamıyla ayrılıklar arzetmektedir. O sebeble, bir tabi dilin kısıtlı bir bölümünü bile incelemek için oldukça kurallı, metodlu ve kompleks teknikler ile algoritmalara ihtiyaç vardır. Tabi diller ile yapay diller arasındaki genel farklılıkları ana hatları ile şöyle belirtmek mümkündür.

1.3.1. Tabi Dillerde Cümle Çeşitliliği

Bir tabi dil ile yapay dil arasındaki en büyük fark tabi dilin düzgün oluşturulmuş cümlelerden meydana gelmesi ve bu cümlelerin sonlu bir sayıda tüm yönleriyle karakterize edilememesidir. Bu özellik recursive fonksiyon içeren yapay dillerde kısmen bulunsa da tabi dillerdeki belirsizlik dilin pek çok seviyesinde mevcuttur.

- Bir tabi dildeki kelime sayısı tam olarak bilinemez. Çünkü dil dinamik bir yapıya sahiptir. Her geçen gün özellikle teknoloji ile ilgili yeni terimler dilin kelime haznesine dahil olmaktadır.
- Bir dilin grameri ile ilgili yapılarıda kesinlik arzetmez.
- İnsanlar genellikle günlük hayatlarında anlatmak istedikleri duygu ve düşüncelerini kelimelere tam olarak yansıtamazlar. Bu da kelimelerin kullanıldığı yere göre farklı anlamlarla yüklenmesine yol açar.

Yukarıda sıralanan bu güçlüklerin bir sonucu olarak tabi dil uygulamaları bir tabi dilin ancak anlamsal olarak iyi tanımlanmış kısıtlı bir bölümünü modelleyebilirler. Bu nedenle bir tabi dile ait eksiksiz bir modelleme kurma eğilimi bugünkü şartlar altında tamamen bir hayalden ibarettir. Dilin dinamik bir yapıya sahip olduğundan bahsederken dildeki her kelimenin bir yaşam peryodundan bahsetmek fazla abartı sayılmaz. Çünkü dil gelişim süreci içinde bir taraftan toplumsal, sosyal ve teknolojik gelişimlerin etkisiyle bünyesine yeni kelimeler katarken diğer yandan da modası geçen veya az kullanılan kelimeleri otomatik olarak yapısından çıkarır. Özellikle teknolojinin hızlı gelişim gösterdiği günümüzde bir teknik terimin en fazla popüler olduğu süre, uzmanlar tarafından birkaç yıl olarak verilmektedir.

1.3.2. Tabi Dillerde Hataların Durumu

Tabi ve yapay diller arasındaki önemli farklardan birisi de kavramların anlaşılabilirliği üzerinde bir esnekliğin bulunup bulunmadığı hususudur. Bilgisayar bilimlerinde bu kavrama **robustness** adı verilir. Tabi konuşma dilinde, söylenen bir cümle gramatik yönden hatalı olsa bile pekala anlam ifade edebilir. Oysa yapay bilgisayar dillerinde, bırakın gramatik hatayı, en küçük bir noktalama hatası dahi program derlenmesi esnasında sistem tarafından hata mesajı veya mesajlarının verilmesine yol açar. Tabi dillerde bir cümlenin anlam ifade edebilmesi, belirli bir eşik seviyesinde gramatik olarak sözdiziminin doğru yapılmasına bağlıdır. Eğer bu seviye geçilirse o cümle hiçbir anlam ifade etmez . Eşik seviyesinin altında kalan bir sözdizimi ise mükemmel olmasa da bir anlam teşkil eder. Bu seviye cümlenin amacına, karmaşıklığına, formuna hatta geçtiği yere göre değişkenlik gösterir.

1.3.3. Tabi Dillerdeki Belirsiz İfadeler (Ambiguities)

Tabi dillerde bulunan belirsizlikler varolan pekçok karmaşık problemin kaynağını teşkil etmektedir. Bu belirsizlikler dilin değişik seviyelerinde görülmekle beraber insanlar tarafından pek farkedilmezler. Çünkü insanın geniş hafiza kapasitesi bu belirsizlikleri doğru biçimde ve cümlede geçtiği yere göre yorumlama yeteneğine sahiptir. Fakat cümlelerin bir makine tarafından analizine gelince, olayın boyutu değişmektedir. Çünkü problemin kapsamı ve olasılıklar çok geniş ve önceden tahmin edilemediğinden buna yetecek bilgi tabanı ve verileri oluşturmakta imkansızdır. Ayrıca cümlelerde oluşan belirsizlikler çoğu zaman soyut faktörler, tercihler, inançlar v.s. gibi etmenlerin etkisi altında çıkmaktadır. Dolayısıyla bu belirsizliklerin % 100 kesinlikte bir sınıflamaya tabi tutulması da imkansızdır. Ancak mevcut tabi dil işleme sistemlerinde inceleme kapsamına alınan belirsizlikler lexical ve syntactic olmak üzere iki düzeyde ele alınmaktadır. Şimdi bunları örneklerle incelemeye çalışalım.

"that" kelimesi İngilizcede çoğu zaman bir bağlaç ve belirteç olarak kullanılır. "That grass" örneğinde "that" bir bağlaç olmasına rağmen "That grass is green is a well-known fact" cümlesinde that bir bağlaç görevi görür. Bu tip belirsizlikler genellikle yapısal analiz safhasında giderilmektedir. Çünkü bir cümlede bağlaç ve belirteçler birbirlerinden çok farklı yerlerde bulunduğundan, bu ayrım sistem tarafından somut bir şekilde yapılabilmektedir.

Bir başka lexical belirsizlik "homograph" denilen yazılışları aynı fakat birbirlerinden tamamıyla farklı iki anlam arzeden kelimeler ile ilgilidir. Örneğin "saw" kelimesi İngilizcede testere anlamına geldiği gibi see-görmek fiilinin geçmiş zaman halidir. Aşağıdaki cümlede saw kelimesi her iki anlamıyla iki ayrı yerde kullanılmıştır.

He saw the man with a saw.

Syntactic (yapısal) belirsizlik daha ziyade isim ve fiil tamlamalarında görülür. Bazı cümlelerde tamlayan kelimelerin ismi mi yoksa yüklemi mi tamladığını kestirmek çok güçtür. Aşağıdaki örnekleri inceleyelim.

The man saw the girl with a telescope.

John put the book on the table in his pocket.

İkinci örnekte biz biliyoruz ki insanlar ceplerine masa koyamazlar. Dolayısıyla bu cümleyi "John cebindeki kitabı masanın üzerine koydu." şeklinde anlarız. İlk örnek ise cümlenin geçtiği metni veya konuşmayı bilmiyorsak tamamıyla belirsizlik arzeder. Çünkü bu cümle "Adam kızı bir teleskopla gördü veya Adam teleskoplu bir kız gördü" şeklinde yorumlanabilir. Bazen lexical belirsizlikler de, cümlede yapısal belirsizliklerle karışık olarak görülebilirler.

He saw her shaking hands.	"shaking" bir fiil de, sıfat ta olabilir.
He saw her trembling hands.	"trembling" bir fiil de, sıfat ta olabilir.

Son olarakta bir cümle üzerinde daha durmak istiyoruz. Olumsuz bir cümle olan bu cümleden üç farklı yorum çıkartmak mümkündür.

Edith is not going by train to Paris.

1. yorum : Paris'e giden Edith değildir.
2. yorum : Edith Paris'e trenle değil başka bir ulaşım aracıyla gidiyor.
3. yorum : Edith'in trenle gittiği yer Paris değil fakat başka bir yer.

1.3.4. Tabi Dillerde İma Yoluyla Referans Edilen Yapılar (Implicit Elements)

Tabi dillerle yapay diller arasında sözünü edebileceğimiz son farklılık , tabi dillerde cümlede ilk defa bahis edilen bir sözcüğe daha sonra gelen cümlelerde this, that these, those gibi kişi zamirleri ile referans edilmesidir. Böylece cümlelerde aynı öznenin birden fazla tekrar edilmesi önlenmiş olur. Ancak bu referanslar esnasında bazen kullanılan referans sözcüğünün cümlenin görünen öznesinin yerine kullanılıp kullanılmadığı belirsiz kalabilir. Örneğin,

Anne promised that she would be on time.

cümlesinde "she" zamiri Anne'in yerine kullanıldığı gibi Anne'nin sorumluluğu yada gözetimi altında bulunan başka bir bayanın yerine de kullanılmış olabilir. Kimi zaman referans eden sözcük bir isim tamlaması da olabilir. Örneğin,

John fell in the water. The poor chap was soaked to the skin.

cümlesindeki gibi referanslara "anaforik referanslar" denir. Yukarıdaki cümlede "the poor chap" ile bir önceki cümlenin öznesi olan "John" kastedilmiştir. Anaforik referansların belirlenmesi daha karmaşık işlemler gerektirir. Anaforik referanslardan türeyen bir başka belirsizlik de "homonomi" dir. Bu belirsizlik türünde aynı özneye birden fazla sözcük yada sözcük grubuyla referans edilir. Örneğin,

MR MAJOR arrived in France today. THE PRIME MINISTER will meet the President tomorrow. THE CONSERVATIVE LEADER will then travel to Moscow where HE will meet Mr Gorbachev. Mrs Major will join HER HUSBAND in Russia, where THIS SON OF A CIRCUS ARTIST is a relatively unknown figure.

metninde İngiltere başbakanından bahsederken beş farklı ifade kullanılmıştır.

Tabi dillerde karşılaşılan belirsizliklerden biriside "ellipsis" denen bir cümlede kullanılan ifadenin bir sonraki cümlede kullanılırken tekrardan kaçınmak için değiştirilmesidir.

Örneğin, "John is having dinner with Mary tomorrow night, and Paul with Susan." cümlesinde herkes kolaylıkla cümlenin virgülden sonraki kısmının aslında, "Paul is having dinner with Susan tomorrow night". anlamında söylendiğini anlayabilir.

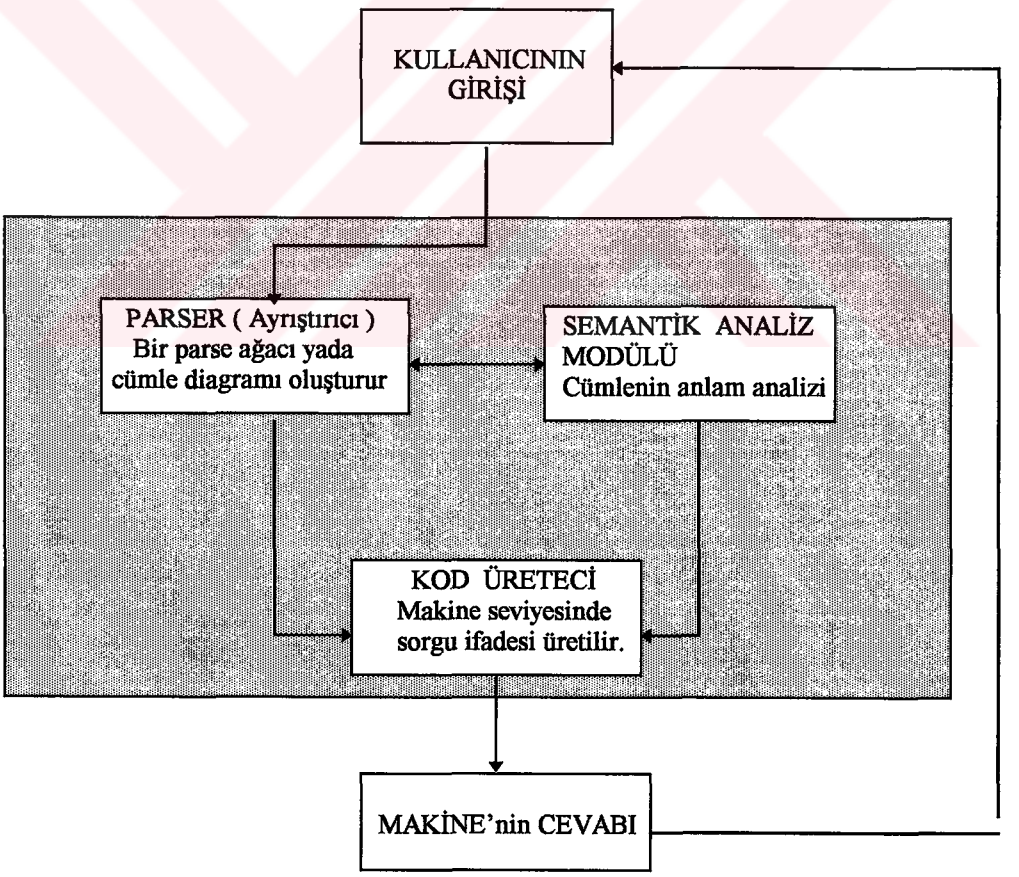
1.4. Tabi Dil Programları

Tabi dil programları, kullanıcılarına karmaşık bilgisayar komutları yerine, konuşma dilinin (İngilizce, Fransızca veya diğer diğer dillerde) düz ve basit cümle yada sözcüklerini kullanma fırsatı veren yapay zeka programlarıdır. Tabi dil arayüz programları sayesinde kullanıcılar, çeşitli konularda oluşturulmuş veri tabanlarına ulaşıp

sorgulama yaparak istedikleri bilgileri kolayca elde edebilmektedirler. Hangi alanda yazılmış olursa olsun tüm tabi dil arayüz programları iki ana kategoride toplanırlar.

- Kullanıcıların ekranda ard arda gelen menülerden seçtikleri kelimelerin birleşmesiyle, bilgisayarın anlayabileceği bir sorgulama ifadesini oluşturan tabi dil programları.
- Yada kullanıcıların herhangi bir menü yapısı kullanmadan sorgulama cümlesini doğrudan girdiği programlar.

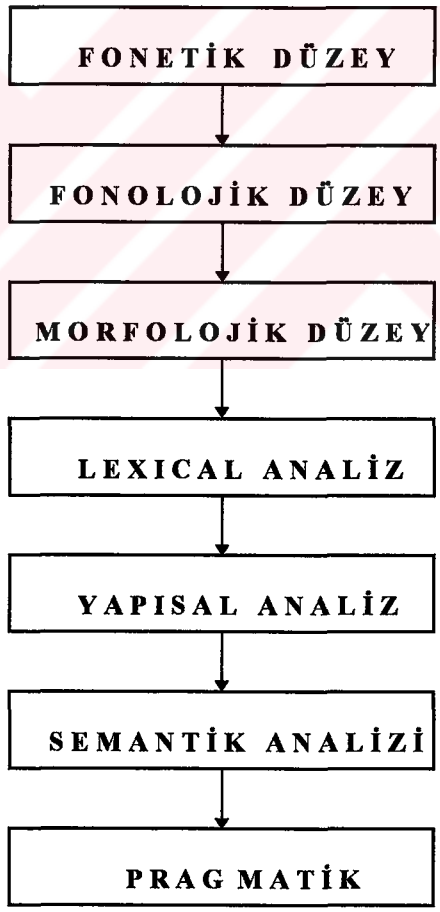
İlk yazılan tabi dil programları birinci kategoriye girmekte olup tasarım ve geliştirimleri ikinci kategoriye giren tabi dil programlarına nazaran daha kolaydır. Hangi gruba dahil olursa olsun bir tabi dil programı yada arayüzü aşağıda gösterildiği gibi temelde üç ana modülden oluşur.



Şekil 1.1. Bir Tabi Dil Arayüz Programının Modülleri

İlk modül "ayrıştırıcı" yada "parser" adını alıp, sözlü yada yazılı bir cümlenin elemanlarını (kelimeler ve noktalama işaretleri) alıp bunları, o dilin gramer kurallarına uygun bir şekilde isim, yüklem v.s. gibi kelime gruplarına ayrıştırarak bir cümle diagramı oluşturur. İkinci modül ise "semantik analiz modülü" olup oluşturulan cümledeki kelimelerin anlamsal analizlerini yapar. Bunun için önceden oluşturulmuş bir kelime bankasını kullanır. Böylece sorgulamaya giren her cümlenin kelime bankasında ki varlığı ve doğru kullanılıp kullanılmadığı kontrol edilir. Genellikle ayrıştırıcı ve semantik modülü paralel çalışır. Bir tabi dil programındaki son modül ise "kod üreteci" adını alıp, doğru oluşturulmuş kullanıcı sorgu cümlesini makinenin veritabanından sorgulamayı yapabilecek bir hale getirir. Bu aşamada programlama dilinin düşük seviyeli dosya erişim fonksiyonları kullanılır.

1.5. Tabi Dillerin İşlenmesinde Takip Edilen Safhalar



Şekil 1.2. Bir Tabi Dil Uygulamasında Takip Edilen Safhalar

Tabi dillerin işlenmesinin en genel manada yukarıda gösterildiği gibi yedi aşaması vardır. Bu safhaların birbirleriyle ilişkisi yada biri diğerine bağımlılık gösterse de genelde biribirlerini takip sırası yukarıdaki gibidir. Bu sıra, bugünkü tabi dil işleme uygulamalarının % 90 nında takip edildiği için adeta bir standart haline gelmiştir. Şimdi sırasıyla bu aşamaları ve her birinde neler yapıldığını kısaca açıklamaya çalışalım.

1.5.1. Fonetik Düzey

Fonetik, kelimenin sözlük anlamından tahmin edilebileceği gibi , seslerin uyumlu bir kompozisyonda birleşerek önce hecelerin daha sonra kelime ve cümlelerin nasıl meydana geldiğini inceleyen bir ses bilimidir. Akustik ve fizik beraber çalıştığı diğer yardımcı bilim dallarıdır. Günümüzde geliştirilen ve uygulanan teknikler sayesinde hangi konuşma sinyallerinin hangi seslere karşılık geldiği büyük bir doğrulukla belirlenebilmektedir. Fonetik düzeyi içeren tabi dil işleme uygulamaları çok azdır.

1.5.2. Fonolojik Düzey

Seslerin sözcük, sözcük grupları ve cümleleri nasıl oluşturduğu ile ilgilenir. Ayrıca yazılı bir metinde kelimelerin nasıl heceleneceği ve nasıl telaffuz edileceği ile ilgili uygulamalar da bu safhada yapılır.

1.5.3. Morfolojik Düzey

Bir cümledeki kelimelerin aldıkları biçimlerle ilgilenir. Yüklem eşlenekleri, isimlerin çoğul ve tekil halleri v.s. Bu düzeyde ayrıca bir kelime grubunda başka bir kelime grubuna ait sözcüklerin nasıl oluşturulacağına dair çalışmalar yapılır. Örneğin,

isimden sıfat türetme (courage - courageous)
yüklemlerden isim türetme (paint - painter)

İngilizcede kelime türetiminde suffix ve prefix dediğimiz sonek ve önek'ler çokça kullanılır. Örnek,

tidy - untidy

force - enforce

hope - hopeless, hopeful.

Bazen de sözcük türetilerek elde edilen sözcüğün türediği sözcükle anlam ilişkisi tamamen yok olur. Örnek,

revolve - revolver.

1.5.4. Lexical Düzey

Bir tabi dildeki mevcut tüm kelime gruplarına "lexicon" adı verilir. Lexical düzeyde bir dilin bilinen kelimeleri sınıflandırılarak isim, sıfat, fiil, zarf, edat, bağlaç, zamir v.s. gibi kategoriler oluşturulur. Bu kelime grupları dilden dile ufak ayrıntılar gösterseler de hemen hemen tüm dillerde aynı sınıflandırmaya tabi kalırlar. Sözünü ettiğimiz kelime gruplarından isim, sıfat, fiil ve zarflar bir dildeki kelime toplamının % 80 - % 90 arasında bir bölümünü oluşturduğu için bu gruplara "açık-sınıf kelime grupları" adı verilir. Bağlaçlar, edatlar ve zamirler ise azınlık grubunu teşkil ettikleri için bunlara da "kapalı-sınıf kelime grupları" denir. Bir dilde, kapalı-sınıf kelime grubuna dahil tüm kelimeleri listelemek mümkündür. Ancak aynı şeyi açık-sınıf grupları için söylemek mümkün değildir.

1.5.5. Yapısal Düzey

Yapısal yada gramer düzeyi, doğru cümleler kurmak için kelimelerin ne şekilde birleştirilmesi gerektiği ile ilgilenir. Her dilde cümlelerin oluşturulmasına ait yazılı gramer kuralları vardır. Gramer düzeyi tabi dil işleme uygulamalarında üzerinde en fazla araştırma yapılan safhadır.

1.5.6. Semantik Düzey

Kelime, kelime grupları ve cümlelerin anlamsal yönden incelendiği safhadır. Pragmatik düzeyle oldukça içiçedir.

1.5.7. Pragmatik Düzey

Bu düzeyde bir cümle veya bir kelime grubunun anlamı geçtiği yere göre, ki bu bir yazılı metin veya dialog olabilir, değerlendirilir ve anlamsal bütünlüğü bu çerçeveye göre incelenir.

Bu safhalar birbirlerinden bağımsız olarak ele alınabileceği gibi birbirleriyle bütünleştirilerek de incelenebilirler. Dolayısıyla her düzey için kullanılan uygulama stratejisi, heuristik, mimariler ve algoritmalar farklılık gösterir. Tabi dil işleme uygulama ve sistemlerinde bu düzeylerin hepsinin kullanılacağına dair kesin bir kural yoktur.

II. TABİ DİL ARAYÜZÜ GELİŞTİRMEDE KULLANILAN YÖNTEMLER

2.1. Informal Yöntemler

Tabi dil arayüzü geliştirmede kullanılan informal yöntemler cümle yapıları ile ilgilli teorilere dayanmazlar. Bunlar kullanıcının bilgisayara bir işi yaptırmak için verdiği komutlardaki bazı anahtar kelimeleri yakalamak suretiyle kullanıcının talebini karşılamaya çalışırlar. Biz burada iki metodu ele alıp örneklerle açıklamaya çalışacağız. İlk metod bilgisayara verilen sorgulama cümlesinde öyle bir anahtar kelimenin bulunmasını öngörür ki , sistem bu kelimenin yardımıyla varolan veritabanındaki istenen tüm bilgileri versin. İkinci metod ise daha karmaşık olup, veri tabanı kullanan soru cevap sistemleri için uygundur. Belirgin soruların analizleri yapıldığında bu metodun pek çok ilginç özellikler yansıttığı görülür. Bu özelliklerden birisi, iki veya daha fazla nicelik arasında daima bir ilişki bulunmasıdır. Aşağıdaki örnekleri inceleyelim.

manager_of_department(X,Y) X=manager ilişkisi, Y=department ilişkisi
length_of_river(X,Y) X=length ilişkisi, Y=river ilişkisi
cost_of_part(X,Y) X=cost ilişkisi, Y=part ilişkisi

Tüm örneklerde görüldüğü gibi X ve Y aralarında ilişki kurulan niceliklerdir.
Bu metotta rastlanan birbaşka önemli özellikte şudur. Eğer sistem nicelikler arasında herhangi bir ilişkiye ait olası bütün çözümleri verebiliyorsa, sorgu cümlelerinde daima aranan çözümleri bulmaya yarayan anahtar sözcükler yada sözcük kalıpları bulunur. Bu sözcük yada kalıplarının soruda bulunması bir filtreleme işleminin yapılmasına müsade eder. Şöyle bir örnek vermek suretiyle söylediklerimizi daha anlaşılır hale getirebiliriz.

Diyelim ki bilgisayardan İstanbul il sınırları içerisinde bulunan ve 50 km'den uzun olan nehirleri listelemesini isteyelim. Burada takip edilecek yollardan birisi önce İstanbul'daki tüm nehirlerin bulunup daha sonra aralarından 50'km den kısa olanlarını elemek olabilir. Örnekte görüldüğü gibi önce tüm çözümlerin veritabanında varlıkları araştırılıp, daha sonra sorgu cümlesindeki kıstasa uyanlar filtreden geçirilip çözüm

olarak sistem tarafından sunuluyor. Bu oldukça güçlü bir yaklaşımdır, ama cevap sayısı çok olunca sistemde belirgin ağırlaşmalara rastlamak mümkün olabilir.
Soru cevap sistemleri sorgulama ifadelerinden kullanıcının talebini karşılayacak kriter nitelikteki sözcük şablonlarını çıkarmaya dayanırlar. Her sorgu ifadesi en az bir şablon içerir. Bu sözcük şablonları sorgulamanın tipi ve sorunun hangi niceliklerle ilgili olduğu hususundaki bilgileri içerir. Bu şablonların sorgu cümlelerinden doğru olarak filtrelenip çıkarılması halinde veritabanından istenen bilgilerin alınması mümkün olur.
Örnek olarak aşağıdaki sorgu ifadesini ele alalım.

What is the population of Avcılar ?

Bu örnekte "population of Avcılar" aradığımız sözcük şablonudur. What, is, the, ? sözcükleri filtreden geçerken elenen sözcükler olup bunların Avcılar'ın nüfusu bilgisinin veritabanından çıkarılmasına hiçbir katkıları yoktur.

2.2. Anahtar Sözcüklerden Anlam Çıkarılması

En basit tabi dil arayüz programları anahtar sözcüklerin sorgu ifadesinden bulunup çıkarılması esasına göre geliştirilmişlerdir. Böylece herhangi bir anahtar sözcüğü içeren bir cümle, soru cümlesi veya sadece sözcüğün kendisi sorgulama ifadesi olarak bilgisayara verildiğinde, bu sözcüğün varlığı bir anahtar sözcük tarayıcısı tarafından aranır ve cümledeki diğer sözcükler atılır. Bundan sonraki işlem ise veritabanından anahtar kelimenin belirttiği bilgileri çıkarmaktır.

2.3. Anahtar Sözcüklerden Anlam Çıkarılması Yöntemine Göre Geliştirilmiş Bir Uygulama: Dökümantasyon Bilgi Sistemi (DOCSYS)

Anahtar sözcüklerden anlam çıkarılması yöntemine göre geliştirilmiş bir program olan DOCSYS aslında bir dökümantasyon bilgi sistemidir. Böyle bir programın geliştirilmesi şu ihtiyaçtan doğmuştur. Bilindiği gibi değişik alanlarda faaliyet gösteren büyük şirketlerin pekçoğu ürettikleri ürünleri içeren tanıtıcı ve bilgilendirici kataloglar hazırlarlar. Bu özellikle bilgisayar endüstrisinde büyük bir problemdir. Çünkü her

yazılım ürününün bir veya birden fazla kullanım kitabı vardır. Ayrıca bu yazılımlarda yapılan en küçük bir terfi veya upgrade işlemleri otomatik olarak daha önceden yazılan kullanım kitaplarınında yeniden düzenlenmesini gerektirir. Bir örnek vermek gerekirse bilgisayar dünyasının devi IBM'in her yıl, çıkardığı yeni ürünler için veya daha önceden piyasaya sürdüğü ürünlerde yaptığı değişiklikleri yansıtan 100,000 sayfalık dökümantasyon çıkardığı istatistiklerde belirtilmiştir.

DOCSYS'in arkasındaki temel fikir basittir. Bütün kullanım kitaplarındaki ilgili konular ve referans numaraları bilgisayara yüklenir. Her referans numarası DOCSYS programından ilgili bir veri grubuna karşı gelir. Örneğin kullanıcı bilgisayara bir yazı editörü olan EDLIN hakkında bilgi istediğini aşağıdaki şekillerde sorsun.

Tell me about EDLIN (EDLIN hakkında bana konuş)
Do you have any information about EDLIN ? (EDLIN hakkında bilgin var mı)
Please help me, where can I find a referance concerning EDLIN ?
(Lütfen bana yardım et, EDLIN'le ilgili bir referansı nerede bulabilirim. ?)

Sistem bu sorgu ifadelerinden birini aldığı zaman anahtar kelime tarayıcısını devreye sokar. EDLIN'i bulduğu zamanda ne hakkında bilgi istendiğini anlar. Daha sonra veritabanında tüm dizinlere bakarak EDLIN'le ilgili referansları arar. Bu referansları buluncada bunları birazda tabi konuşma dili havasında kullanıcıya sunar. Mesela yukarıdaki soru karşısında sistemin yanıtı pekala şu olabilir

EDLIN can be found in referance BOOK1 on pages ["41-46","57"]
EDLIN can be found in referance BOOK2 on pages ["100-102"]
EDLIN can be found in referance BOOK3 on pages ["100-110"]

2.4. DOCSYS'de Kullanılan Veri Yapıları

DOCSYS programının temel işlevi veritabanından istenen kıstaslara uygun bilgilerin alınması olduğuna göre önce Prolog'da bunu gerçekleştirmek için verinin hangi şekillerde organize edilmesi gerektiğine bakmak gerekir.

En temel veriler index verileridir. Bu programda 3 temel parametre argüman olarak kullanılmıştır. İlk parametre yayının ismi, ikincisi konu ismi, son parametrede o yayının kaçıncı sayfalarda bulunduğunu gösteren sayfa numaralarıdır. İlk iki parametre string tipinde son parametre ise string listesi tipindedir. Şayet bir konu birden fazla yayında geçiyorsa bu konuya ait birden fazla index verisi vardır. Aşağıda DOCSYS programından alınan birkaç satırı inceleyelim.

```
ind("BOOK1","LOGOFF", ["99"]).
ind("BOOK1","EDLIN", ["41-46","57"]).
ind("BOOK2","EDLIN", ["100-102"]).
ind("BOOK7","EDLIN", ["100","110"]).
ind("BOOK1","EXE", ["14-18"]).
ind("BOOK1","ECHO", ["35","146"]).
ind("BOOK7","BNF", ["51","55-56"]).
ind("BOOK7","BNF", ["30-31"]).
ind("BOOK8","BNF", ["109","130-148"]).
ind("BOOK4","RECURSION", ["56-78"]).
ind("BOOK7","RECURSION", ["119-125"]).
```

Bazen kullanıcı belirli bir maddeden bahsederken o maddeyi belirten ifadenin yerine onun çok kullanılan bir eşanlamlısını kullanabilir. Bu önceden tahmin edilebilirse sistem tarafından ayrı bir veritabanında eşanlamlı sözcükler tutularak, sorgulama ifadelerinde bu sözcüklerin anlaşılabilmesi sağlanır. Örneğin veritabanında LOGOFF konusuna ilişkin bir referans bulunduğunu kabul edelim. Ama kullanıcı bu konuya ısrarla LOGOUT maddesinden erişmeye kalkabilir. Eğer bu iki kelimenin eşanlamlı olduğu bilgisi daha önceden sisteme verilirse, sistemde bunları aynı anlamlı kabul edip aynı konuya farklı iki isim altından erişmeyi mümkün kılar. Aynı şekilde LOGIN ve LOGON kelimeleri de eşanlamlı olup bunların da aynı anlamlı kelimeler olduğu sisteme verilerek aynı konuya farklı iki kelimeden ulaşım sağlanır.

```
dsyn("LOGOUT","LOGOFF")
dsyn("LOGIN","LOGON")
```

Çoğu zaman bilgisayara girilen sorgulama ifadelerinde tamamıyla tabi dil havası vermek için çok sık kullanılan sözcük grupları vardır. Zarflar, belirteçler, soru sözcükleri gibi. Bunların anahtar sözcüklerle herhangi bir ilgisi olmadığı için çoğu zaman dikkate alınmaz ve ihmal edilirler. Bu kelime gruplarından en çok kullanılanları şunlardır.

reject("HOW")	reject("INFORMATION")
reject("PLEASE")	reject("GO")
reject("ON")	reject("CAN")
reject("WOULD")	reject("OFF")
reject("LISTINGS")	reject("COULD")
reject("REFERENCE")	reject("SHOW")
reject("REFERENCES")	reject("ALL")
reject("LISTING")	reject("FINDING")
reject("INDEX")	reject("LISTING")
reject("YOUR")	reject("IS")
reject("INFO")	reject("ARE")
reject("I")	reject("ANYTHING")
reject("FIND")	reject("ANY")
reject("YOU")	reject("OUT")
reject("DO")	reject("KNOW")
reject("HAVE")	reject("THE")
reject("CAN")	reject("A")
reject("WHY")	reject("SOME")
reject("ABOUT")	reject("ME")
reject("GET")	reject("TO")
reject("HELP")	reject("WHERE")
reject("UP")	reject("WANT")
reject("LOOK")	reject("TELL")
reject("NEED")	reject("WHAT")
reject("KIND")	reject("DO")
reject("EVERYTHING")	reject(",")
reject(";")	reject(".")

reject(":") reject("!")
reject("?") reject(""")

2.5. DOCSYS'de Kullanılan Programlama Stratejileri

DOCSYS'deki veri yapılarını inceledikten sonra şimdi de programın can alıcı bazı kodlarını yakından incelemeye çalışalım. Sistemin bütün çalışması ve yaptıkları aşağıdaki kodla özetlenir. Detaylar ise bu koddaki prosedürlere dallanılarak gerçeklenir.

```
docdriver if repeat, nl, qetquery(X), findref(X,Y),
        produceans(Y),fail.
```

Sistemin bütün işlevi repeat ve fail predicateleri arasındaki prosedürlere dallanmakla gerçeklenir. Repeat .... fail yapısı Turbo Prolog da sonsuz döngü oluşturmak için kullanılır.

getquery predicate i, kullanıcının sorusunu alıp cevapları bir semboller listesi halinde üretir. Bu predicate in detaylı yapısı şöyledir.

```
getquery(Z) if write("Please ask your question."),
        nl,readln(Y),upper_lower(Y1,Y),
        changeform(Y1,Z).
```

readln klavyeden girilen her türlü bilgiyi okuyan standart bir Prolog fonksiyonudur. upper-lower ise okunan stringteki tüm küçük harfleri büyük harfe dönüştürür. Bu da Prologun kütüphanesinde olan bir fonksiyondur. Böylece sistemin istenen bilgileri veritabanında ararken ortaya çıkabilecek olan küçük büyük harf uyuşmazlığı ortadan kaldırılmış olur. Veritabanında tüm veriler büyük harfle tutulur. Klavyeden okunan ve bir stringler listesi halindeki soru ifadesini tokenlarına ayırıp liste yapısına dönüştürmek gerekir. Bunun içinde changeform fonksiyonu kullanılır. Bu fonksiyonu aşağıdaki biçimde Prologda kodlamak mümkündür.

```
changeform(S,[H|T]) if fronttoken(S,H,S1),!,
                        changeform(S1,T).
changeform(_,[]).
```

Bu kod parçasında kullanılan fronttoken bir Prolog fonksiyonudur. İşlevi ise bir semboller listesinin ilk elemanını listeden koparmaktır. Örneğin,

```
changeform("TELL ME ABOUT EDLIN",[TELL,ME,ABOUT,EDLIN]).
```

changeform fonksiyonu "TELL ME ABOUT EDLIN" ifadesini tokenlarına ayırarak [TELL,ME,ABOUT,EDLIN] şekline dönüştürür. getquery fonksiyonu işlevini tamamladıktan sonra , sıra sorgu ifadesinde anahtar kelimenin bulunup çıkarılmasına gelir. Bu ise findref fonksiyonunun görevidir. Findref parametre olarak aldığı ilk listeden sırayla sembolleri alır ve bunları ihmal edilecek olan sözcük grupları ile karşılaştırır. Şayet ihmal edilecek sözcükler arasında olmayan bir sözcüğe rastlanırsa bu, aranan anahtar sözcüktür ve fonksiyondan geri dönüş değeri olarak gönderilir.

```
findref([TELL,ME,ABOUT,EDLIN],Y).
```

findref fonksiyonu yukarıdaki şekilde çağrılırsa, EDLIN anahtar sözcük olarak Y değişkenine atanarak fonksiyondan geri gönderilir. Anahtar kelime bulunduktan sonraki aşama ise bunu produceans fonksiyonuna parametre olarak gönderip fonksiyonu çağırmaktır. Bu fonksiyon aşağıdaki gibi 4 ayrı yapıda çağrılabilir.

```
produceans(X) if ind (X1,X,Y), putflag,
                    write(X, "can be found in the referance ",
                     X1, " on pages ",Y, "."), nl.

produceans(X) if syn(X,Z),  ind(X1,Z,Y), putflag,
                    write(X, "can be found in the referance ",
                     X1, " on pages ",Y, "."), nl.
```

```
produceans(_) if not(flag),
                write("We have no information in the referance",nl.

produceans(_) if remflag.
```

Buraya kadar verilen örneklerde ve açıklamalarda bazı önemli noktalar vardır ki bunları açıklamakta fayda vardır. İlk olarak temel program kodundaki 4 fonksiyonun aşağıda gösterildiği gibi bir repeat - fail döngüsü arasında yeraldığı görülmektedir.

```
repeat, nl, getquery(X), findref(X,Y), produceans(Y), fail.
```

Kontrol, döngü içinde produceans fonksiyonuna gelince backtracking mekanizması devreye girerek birinci cümle için tüm olası çözümleri araştırır. Eğer bir başarı kaydedilirse ikinci, sonrada üçüncü ve dördüncü cümleler için aynı işlemler gerçekleştirilir. Produceans fonksiyonunun ilk iki biçimi sorgulamanın cevabını bulup ekrana çözümleri yazarlar. Üçüncü yapı ise şayet sorgulama sonunda istenen bilgiler bulunamadıysa bunu bir mesajla ekrana çıkarır.

Eğer backtracking mekanizması tüm bu fonksiyonlardan geçerse doğru fonksiyonların doğru zamanlarda kullanıldığından emin olmak için bir yola ihtiyacımız vardır. Bu da flag denen bir bilginin veritabanına ekstradan eklenmesi ve bunun değerine göre bir cevabın bulunup bulunmadığına karar verilmesidir. İlk iki kuraldan geçilirse flag bilgisi veritabanına yazılır. Bunun anlamı sorgulama sonucunda bir cevabın bulunduğudur. Üçüncü formda ise bu flagın değerine göre istenen cevabın bulunamadığı mesajı ekrana yazdırılır. Produceans kuralının son biçimi ise flag bilgisini veritabanından siler. Böylece bir sonraki sorgulama için veritabanı önceki haline döner.

Bu flag bilgisiyle ilgili iki düşük seviyeli erişim fonksiyonu kullanılmıştır. Bunlar putflag ve remflag fonksiyonlarıdır. Putflag veritabanına bir flag bilgisini ekler, remflag de bu bilgiyi şayet varsa siler.

```
putflag if not(flag), assert(flag),!.
```

```
purflag.
remflag if flag, retract(flag),!.
remflag.
```

Veritabanından cevap üretme mekanizmasıyla ilgili bir başka özellikte eşanlamlı sözcüklerin nasıl kullanılıp değerlendirildiğidir. Bir eşanlamlı sözcük ile ilgili bazı genel kurallar vardır. Sözgelimi A sözcüğü bir başka B sözcüğünün eşanlamlısı ise B de A'nın eş anlamlısıdır.

```
syn(Y,X) if dsyn(X,Y).
syn(Y,X) if dsyn(Y,X).

dsyn(Y,X) if concat(X,"S",Y).
dsyn(Y,X) if concat (X,"ES",Y).
dsyn(Y,X) if concat (X,"'S",Y ).
```

Bu kuralların dışında sistemin tekil ve çoğul sözcüklerin birbirlerinin eşanlamlı olduklarını algılayabilmesi içinde kurallar vardır.

III. ETKİLEŞİMLİ BİR TABİ DİL ARAYÜZ PROGRAMI : İSTANBUL REHBERİ

İSTANBUL REHBERİ (İR) tabi dil arayüzüne sahip bir veritabanı programıdır. Programın amacı herhangi bir alanda kullanıcıların günlük konuşma dili havası içinde öğrenmek istedikleri konuları bilgisayardan sorgulayabilmeleridir. Programın tasarımını yaparken bunun daha ziyade seyahat edenler için (özellikle yabancı turistler için) faydalı olabileceğini düşündüm ve veritabanını İstanbul ile ilgili değişik konularda topladığım bilgilerden oluşturdum.

İSTANBUL REHBERİ (İR) programının karakteristik özelliği kullanıcının giriş sorularını İngilizce soru cümleleri halinde verebilmesidir. Şu anda programın veritabanında aşağıdaki konularda bilgiler mevcuttur.

İstanbul'un genel özellikleri,

- Nüfus, yüzölçümü, komşular bilgileri.
- İlçeleri,
  - . Nüfus, yüzölçümü, komşular bilgileri
- Coğrafi bilgiler (dağlar, nehirler, göller)
- Üniversiteler
- Oteller
  - . 5, 4, 3 ve 2 yıldızlı oteller.

Tarihi yapıtlar,

- Camiler
- Türbeler
- Çeşmeler
- Kuleler
- Müzeler

İdari yapı,

- Valiler
- Belediye başkanları

Bu bilgiler sayfalarca süren detaylı bilgiler olmayıp, sadece soru-cevap mekanizmasına uygun hazırlanmış birer ikişer cümlelik bilgilerdir.

İR programı tabi dil işleme metodlarından informal metodlar sınıfına giren **fark çiftlerinin ayrıştırılması (difference pair parsing)** tekniği ile yazılmıştır. Sistem kullanıcının girdiği sorgu ifadesinde önceden belirlenmiş sözcük şablonları yada kalıpları arar. Eğer böyle bir şablona rastlarsa oluşturduğu bir sorgulama cümlesiyle veritabanından istenen kıstaslara uyan bilgileri arayıp çıkarır. Bu sözcük şablonları anahtar sözcük arama tekniğinin bir adım ötesidir. Aşağıda bu sözcük şablonlarından birkaç örnek verilmiştir.

- population of county (ilçenin nüfusu)
- height of mountain (dağın yüksekliği)
- address of hotel (otelin adresi)
- county border county (ilçeyle sınırlı ilçe)
- opening date of museum (müzenin açılış tarihi)
- length of river (nehrin uzunluğu)
- ...

3.1. İR 'nin Veritabanının Temel Yapısı

İR 'nin veritabanında veriler temel olarak şu gruplarda organize edilmişlerdir. (Bu gruplar istenildiği takdirde çoğaltılarak veritabanının genişletilmesi sağlanabilir.)

city	museum
mountain	tower
river	tower
lake	fivehotel
county	fourhotel
university	threehotel
palace	twohotel
mosque	hotel

turbe mayor
fountain governor

Yukarıdaki veri gruplarının herbirine veritabanından örnekler verelim.

```
city("İstanbul",5712,7.309e3,["Tekirdağ","Kocaeli","Bursa"])
mountain("Kayışdağ","Kadıköy",438)
river("Riva","Ömerli Dam Lake",100)
lake("Küçükçekmece Lake",16)
county("Avcılar",50,150000,["Büyükçekmece","Küçükçekmece"])
university("Yıldız Teknik Üniversitesi","Yıldız")
palace("Dolmabahçe Sarayı","Dolmabahçe",1854)
mosque("Kılıç Ali Paşa Camii","Tophane",1580)
turbe("Beyazıd II Türbesi","Beyazıd",1512)
fountain("Ahmet III Çeşmesi-Ayasofya","Sultanahmet",1728)
museum("Deniz Müzesi","Beşiktaş",1897)
tower("Galata Kulesi","Galata",507)
fivehotel("Conrad İstanbul Hotel","Conrad İstanbulHotel -Yıldız
        Caddesi.P.K.200, 80700 Beşiktaş",2273000,20)
fourhotel("Color Hotel","Millet Cad: 82, Fındıkzade",6312020,12)
threehotel("Astor Hotel","Laleli Cad: 12, Laleli",5186470,16)
twohotel("Akyıldız Hotel","M.Kemalpaşa Cad: Langa Bostanları Sok: 6,
        Aksaray",5306960,15)
hotel("Akgün İstanbul Hotel","Adnan Menderes Bulvarı",5344879,10)
mayor("Ahmet İsvan",1973,1977)
governor("Lütfi Kırdar",1938,1949)
```

Bütün ilgili bilgiler yukarıdaki veri gruplarına aşağıdaki argümanlarla endekslenmişlerdir. Şöyleki,

```
city(<isim>,<yüzölçümü>,<nüfus>,<komşular>)
mountain(<isim>,<bulunduğu yer>,<yükseklik>)
```

river(<isim>,<döküldüğü yer>,<uzunluk>)
lake(<isim>,<yüzölçümü>)
county(<isim>,<yüzölçümü>,<nüfusu>,<komşular>)
university(<isim>,<bulunduğu yer>)
palace(<isim>,<bulunduğu yer>,<açılış tarihi>)
mosque(<isim>,<bulunduğu yer>,<inşa tarihi>)
turbe(<isim>,<bulunduğu yer>,<inşa tarihi>)
fountain(<isim>,<bulunduğu yer>,<inşa tarihi>)
museum(<isim>,<bulunduğu yer>,<inşa tarihi>)
tower(<isim>,<bulunduğu yer>,<inşa tarihi>)
fivehotel(<isim>,<adres>,<telefon numarası>,<havalanından uzaklığı>)
fourhotel(<isim>,<adres>,<telefon numarası>,<havalanından uzaklığı>)
threehotel(<isim>,<adres>,<telefon numarası>,<havalanından uzaklığı>)
twohotel(<isim>,<adres>,<telefon numarası>,<havalanından uzaklığı>)
hotel(<isim>,<adres>,<telefon numarası>,<havalanından uzaklığı>)
mayor(<isim>,<göreve geldiği tarih>,<görevden ayrıldığı tarih>)
governor(<isim>,<göreve geldiği tarih>,<görevden ayrıldığı tarih>)

Örneğin county (ilçe) grubunda ilçe adı, yüzölçümü, nüfus ve komşuluk bilgileri bulunur. Eğer county grubuna başka bir veri eklemek gerekirse bu kolaylıkla bir virgül ile en son veriden sonra ilave edilebilir. Ayrıca ilave edilen bu yeni veri ile ilgili sorgu cümlelerinin de şablonlarının gramere ve parser'a ilave edilmesi gerekir ki sistem buna ilişkin sorgu cümlelerini algılayabilsin.

Tabi dil arayüz programlarının karakteristik özelliği kullanıcı ile bilgisayar arasındaki soru cevap "dialoğunun" günlük konuşma dilinden seçilen cümleler ile olması idi. Öyle ki tamamen konuşma dili havası versin diye sorgu cümlelerine eklenen bu sözcüklerin esas sorgulama işlemini gerçekleştirecek sözcük yada sözcük şablonlarıyla hiç bir ilgisi yoktur. Dolayısıyla bu kelimelerin daha ayrıştırma işleminin başında bir filtreleme işleminden geçerek elenmesi gerekir. Bunun içinde, ihmal edilecek bu sözcük gruplarının en çok kullanılanlarının sisteme tanıtılması gerekmektedir. Böylece sistem filtreleme işlemini yaparken kullanıcının verdiği sorgu cümlesini önce tokenlarına ayırıp

daha sonra oluşturduğu sözcük listesindeki her bir sözcüğü, bu ihmal edilecek sözcükler grubundaki sözcüklerle teker teker karşılaştırır. İhmal edilecek sözcükler grubuna dahil edilmesi gereken bir başka grupta noktalama işaretleri grubudur. Şayet girilen sorgu cümlesinde ihmal edilmesi gereken bir sözcük var fakat bu ihmal edilecek sözcükler grubuna dahil edilmemişse sistem tarafından "unknown word - <word>" şeklinde bir hata mesajı verilir. Bu sözcük daha sonra ihmal edilecek sözcükler grubuna alınırsa bir sonraki sorgu işleminde aynı sözcük kullanıldığı takdirde bu kez sistem aynı sözcüğü anlayarak eleme işlemine tabi tutacaktır. İR programında da aşağıdaki kelimeler ihmal edilecek sözcükler grubu olarak teşkil edilmiştir.

ihmalet("which")	ihmalet("is")	ihmalet("are")
ihmalet("the")	ihmalet("tell")	ihmalet("list")
ihmalet("give")	ihmalet("me")	ihmalet("what")
ihmalet("as")	ihmalet("that")	ihmalet("please")
ihmalet("to")	ihmalet("serve")	ihmalet("served")
ihmalet("how")	ihmalet("from")	ihmalet("airport")
ihmalet("star")	ihmalet("many")	ihmalet("live")
ihmalet("lives")	ihmalet("do")	ihmalet("does")
ihmalet("number")	ihmalet("a")	ihmalet("and")
ihmalet("an")	ihmalet("any")	ihmalet(".")
ihmalet(":")	ihmalet(",")	ihmalet(";")
ihmalet("!")	ihmalet("?")	.....

Bazen kullanıcı sorgu cümlesinde sistemin algılayabileceği bir niceliği direk olarak cümlesinde kullanmayabilir. Bu tip sözcükler önceden tahmin edilebilirse veritabanında bir eşanlamlı sözcükler grubu oluşturulur ve filtreleme işlemi esnasında şayet bir sözcük ihmal edilecek sözcükler grubunda bulunmamışsa bu kez eşanlamlı sözcükler grubuna bakılır. Eğer orada bir karşılık varsa o zaman bu kelime eş anlamlısı ile değiştirilir ve parser'a giderken bu yeni karşılığı ile gönderilir. İR nin veritabanında tuttuğum bir kaç eşanlamlı sözcük grubuna şunları örnek vermek mümkün.

esanlam("people","population")

esanlam("city","istanbul")
esanlam("far","distance")
esanlam("five",5)
esanlam("four",4)
.....

3.2. İR 'nin Genel Çalışma Prensibi

İR'nin soru cevap mekanizması sisteme verilen soruda mevcut anahtar şablonlardan birinin bulunup bulunmadığı araştırır. Böyle bir şablon bulur bulmazda veritabanından istenen kıstasa uygun bilgileri çıkarır. Programın temel olarak 3 önemli işlevi yerine getiren modülü vardır. Şimdi sırasıyla bu modülleri açıklamaya çalışalım. Kullanıcının sisteme verdiği soru cümlesi önce bir semboller listesi haline dönüştürülür.

Soru > What is the largest county of İstanbul ?

sorusunun kullanıcı tarafından klavyeden girildiğini kabul edelim. Bu soru ["What","is","the","largest","county","of","İstanbul","?"] şeklinde tokenlarına ayrılarak semboller listesi oluşturulur.

Programın ilk modülü bir **filtreleme** işlemi gerçekleştirir. Bu işlem esnasında sorgulamaya hiçbir katkısı olmayan , ihmal edilecek olan sözcükler grubuna ait kelimeler ve noktalama işaretleri listeden çıkarılır. Böylece örneğimizdeki semboller listesi ["largest","county","of","İstanbul"] şekline dönüşür.

Normalize edilmiş bu liste daha sonra parsing işlemi gerçekleştiren bir modüle aktarılır. Programın en can alıcı ve karmaşık modülü budur. **Parsing** modülü veritabanında sorgulamayı gerçekleştirecek olan bir sorgu ifadesi oluşturur.

Oluşturulan bu sorgu ifadesi **computation** adı verilen bir modüle gönderilir. Bu modül veritabanına ulaşıp kıstaslara uyan cevapları çıkarır. Computation fonksiyonunun çalışması kısaca şöyledir. Soru-cevap sistemi içinde her önemli ilişki için düşük seviyede

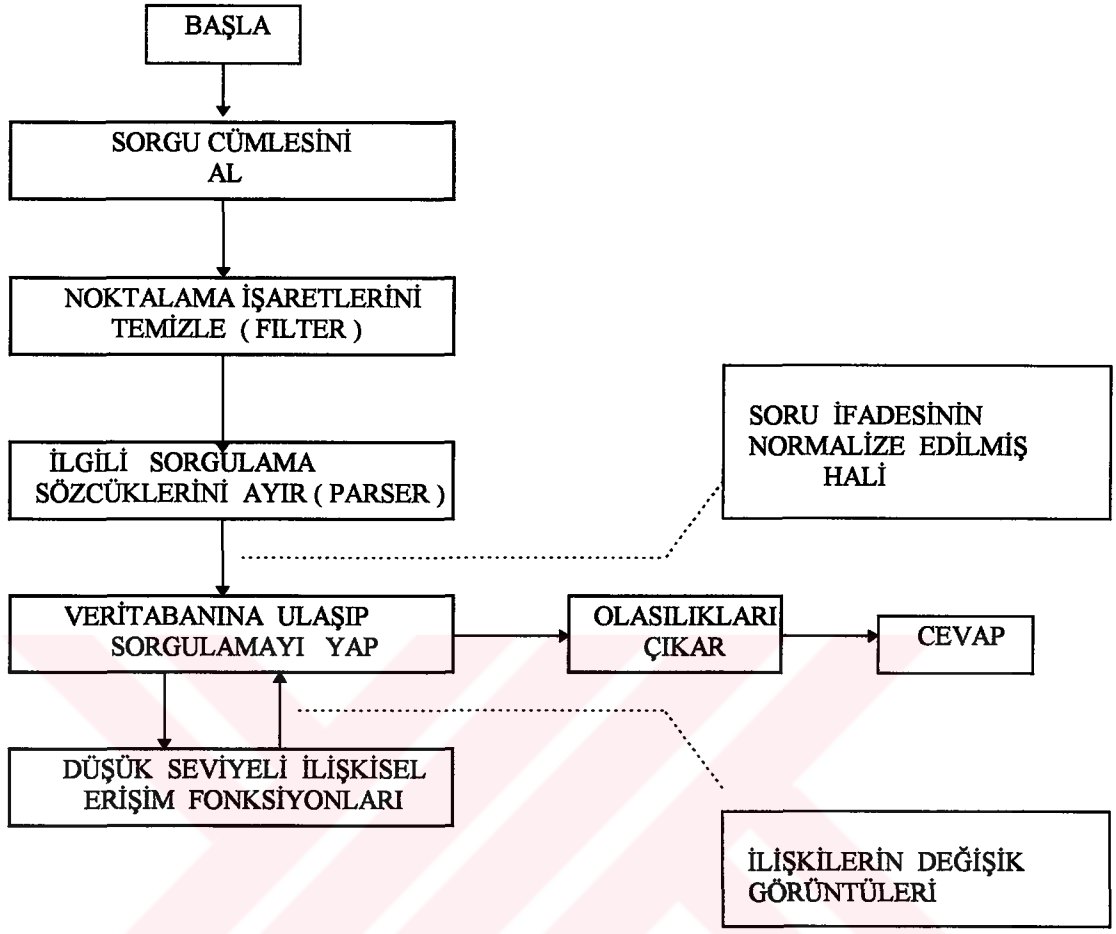
bir erişim fonksiyonu vardır. Örneğin İstanbul'daki herhangi bir kulenin adresi bilgisinin sorgulandığını farzedelim. Kuleler ile ilgili bilgiler tower(<name>, < location>, <dateofbuild>) veri grubu ile temsil edilmektedir. Adres bilgisine ilişkin düşük seviyedeki erişim fonksiyonu aşağıdaki gibidir.

```
db(location,of,tower,LOCATION,TOWER):-tower(TOWER,LOCATION,_).
```

Computation fonksiyonu her çağrıldığında, mevcut çözümlerden sadece birini üretir. Bu yüzden bu fonksiyon bir findall predicate'i içine alınarak birden fazla çözümü (şayet varsa) veritabanından çıkarıp bir çözüm listesi haline getirir. Programın bundan sonraki kısımları ise çözüm listesini belirli bir format dahilinde ekrana getirir. İR'nin en genel anlamda çalışması aşağıdaki psedo kodla belirtilebilir. Program her bir ana prosedürü çağırdığında onların altında bulunan diğer alt prosedürlere dallanmakta ve işlevini yerine getirmektedir.

```
İR :- write("Soru:"),
        readln(Query),
        convert(Query,List),
        filter(List,PurifiedList),
        parser(PurifiedList,E,Q),
        findall(Ans,computation(Q,Ans),AnswerList),
        writeanswer(AnswerList,E).
```

Yukarıdaki yapıyı aşağıdaki gibi bir bilgi akış diagramında belirtebiliriz.



Şekil 3.1. İstanbul Rehberi Tabi Dil Arayüz Programında Bilgi Akışı

3.3. Parser Modülü

Parser modülünün çalışmasını programdan aldığım örnek giriş ve çıkışlarla açıklamaya çalışacağım.

parser(InputData,E,Q),

Bu modülün 3 argumanı vardır. İlk arguman InputData, filtrelenmiş semboller listesidir. Q argumanı ise veritabanında sorgulamayı yapacak olan sorgu ifadesi olup aynı

zamanda fonksiyonun geri dönüş değeridir. E ise ara modüllerde kullanılan bir yardımcı argumandır. Şimdilik bunu fazla dikkate almayacağım.

Örnek :

Soru: Tell me the counties ?

What counties do you have ?

Yukarıdaki iki sorgulamadan herhangi birini sisteme verdiğimizi kabul edelim. İlk aşamada sorgulamaya katkısı olmayacak sözcükler ve noktalama işaretleri elenir. Parser modülüne gelene kadar sadece "counties" kelimesi kalır. Modül bu kelimeyi ilk argümanı - InputData olarak alır.

Parser("counties",E,Q). Fonksiyon işlevini tamamladığında
E="county"
Q=q_e("county") şeklinde E ve Q argumanları değer almış olurlar.

Örnek :

Soru: How far is the Akgün Hotel from airport ?

Bu sorgu sisteme verildiğinde parser modülüne gidecek olan semboller listesi ["distance","Akgün Hotel"] dir. Burada hemen dikkati çeken nokta distance kelimesinin orjinal sorgu cümlesinde bulunmadığıdır. Ancak sistem filtreleme işlemini yaparken her kelimenin veritabanında bir eş anlamının olup olmadığını kontrol eder . Far kelimesinin de eş anlamlı yada onun yerine kullanılacak kelimeler listesinde karşılığı distance alınmıştır. Parser'dan çıkan argümanlar ise,

E="distance"
Q=q_eaec("distance","of","hotel","Akgün","Hotel") değerlerini alırlar.

Oluşturulan cümle şablonu bir nicelik (distance), bir bağlantı sözü (of), bir başka nicelik ve sabit bir terim (Akgün Hotel) den oluşmaktadır. Aynı soruyu sisteme aşağıdaki şekilde sorduğumuzda da sistem aynı sorgulama ifadesini oluşturur.

Soru: What is the distance of Akgün Hotel from airport ?

Örnek:

Soru: Give me the biggest lake in İstanbul ?

Parser'a girecek olan semboller listesi ["biggest","lake","in","İstanbul"] olup, modülden çıkan argumanlarda,

E="lake"

Q=q_max("lake",q_eaec("lake","in","city","İsatanbul")) değerlerini alır.

Burada içiçe geçmiş iki tane sorgulama ifadesi görmekteyiz. Sistem önce İstanbul'da bulunan göllerin listesini veritabanından sorgulayıp daha sonra bulunan göllerden yüzölçümü en büyük olanını seçiyor. Bu, fark çiftlerinin ayrıştırılması tekniğinin karakteristik bir özelliğidir.

Örnek:

Soru: Give me the shortest river in İstanbul ?

Bu örnek de yapı olarak bir önceki örneğe benzemektedir. Parser'a girecek olan semboller listesi ["smallest","river","in","İstanbul"] olup modülden çıkan argümanlarda

E="river"

Q=q_min("river",q_eaec("river","in","city","İstanbul")) değerlerini almaktadır.

Dikkat edilirse önceki sorgulamada en dıştaki sorgu ifadesi q_max olup birden fazla çözümün olduğu sorgu ifadelerinde en büyük niceliği seçmekte kullanılmaktaydı. Burada ise en dıştaki sorgu ifadesi q_min olup yine çoklu çözümlerde en küçük niceliği seçmede kullanılmaktadır.

Örnek:

Soru: List the mayors that served between 1956 and 1967 ?

Yukarıdaki soru sisteme verildiği zaman gerekli filtreleme işlemlerinin yapılmasının ardından ["mayor","between",1956,1967] listesi elde edilir ve parsere ilk argüman olarak girer.

Parser modülünden çıkan argümanlar ise aşağıdaki değerleri alarak çıkarlar.

E="mayor"
Q=q_serv(mayor, between, 1956, 1967)

Veritabanında idari yapıyla ilgili olarak vali ve belediye başkanlarının göreve geliş ve ayrılışlarıyla ilgili sorgulamalarda bu sorgu ifadesi kullanılmaktadır. Yukarıdaki soruyu benzer şekilde,

Soru : List the governors/mayors that served after/before 1945 ? biçiminde sisteme verdiğimizde program yine yine aynı sorgu ifadesini kullanacaktır. O zaman parser'dan çıkan argümanlarda

E="governor/mayor"
Q= Q=q_serv(governor/mayor, before/after, 1945, 1945)

değerlerini alacaklardır. Görüldüğü gibi yukarıdaki sorgu ifadesinde 1945 iki kez tekrar edilmiştir. Bu, aynı sorgu ifadesini tek nicelik karşılaştırmalarında da kullanmak ve bu tip sorgular için yeni bir sorgu şablonu oluşturmamak için soruda sanki iki değer varmış gibi bir izlenim vermek için özellikle kullanılmıştır.

3.4. Veritabanında Sorgulama İşleminde Kullanılan Düşük Seviyeli Erişim Fonksiyonları

İR 'nin veritabanında sorgulamayı yapan üç temel düşük seviyeli erişim fonksiyonu vardır. Kullanıcı düzeyindeki tüm işlemler bu üç fonksiyondan geçip, veritabanına direk olarak müdahale de bu fonksiyonlar yardımıyla yapılmaktadır. Sistemin tanıdığı veri grupları ise sınırlı sayıdadır.

İlk erişim fonksiyonumuz **ent**, 2 parametrelidir. Veri grubunun tipi (county, city, lake v.s.) ilk argüman ve bu tipe uygun veritabanındaki bilgilerde ikinci argümandır. Örneğin county veri grubunu ele alırsak, fonksiyon her çağrılışında ardışık olarak

```
ent(county, Adalar)
ent(county, Avcılar)
ent(county,Bahçelievler)
.....
```

değerlerini veritabanından çıkarır. ent fonksiyonunun her veri grubu tipi için karşılıkları şöyledir.

```
ent(county,NAME):- county(NAME,_,_,_).
ent(mountain,NAME):- mountain(NAME,_,_).
ent(river,NAME):- river(NAME,_,_).
ent(lake,NAME):- lake(NAME,_).
ent(university,NAME):- university(NAME,_).
ent(palace,NAME):- palace(NAME,_,_).
ent(mosque,NAME):- mosque(NAME,_,_).
ent(turbe,NAME):- turbe(NAME,_,_).
ent(fountain,NAME):- fountain(NAME,_,_).
ent(museum,NAME):- museum(NAME,_,_).
ent(tower,NAME):- tower(NAME,_,_).
ent(fivehotel,NAME):- fivehotel(NAME,_,_,_).
ent(fourhotel,NAME):- fourhotel(NAME,_,_,_).
ent(threehotel,NAME):- threehotel(NAME,_,_,_).
ent(twohotel,NAME):- twohotel(NAME,_,_,_).
ent(hotel,NAME):- hotel(NAME,_,_,_).
ent(mayor,NAME):- mayor(NAME,_,_).
ent(governor,NAME):- governor(NAME,_,_).
```

ent fonksiyonu belirli bir tipteki veri grubuna ilişkin tüm bilgileri veritabanından çıkarır. Aynı şekilde ikinci erişim fonksiyonumuz **db**de bir adım ileriye gidilerek veritabanından iki farklı gruptaki veriler arasında kurulan değişik ilişkileri sağlayan bilgilere ulaşılabilmektedir. Fonksiyonun 5 parametresi vardır. İlk üç parametre iki ilişki sözcüğü ve bunlar arasındaki bir bağlantı sözcüğü son 2 parametre de bu ilişkiye uyan veritabanındaki bilgilerdir. Böyle bir yöntemin amacı veritabanında mümkün olduğu kadar değişik olasılıkta ilişkilere ait ayrı veri grupları bulunduğu görünümünü verdirmektir. Aşağıda db erişim fonksiyonunun değişik ilişkileri için tanımlanmış karşılıkları örnek olarak sunulmuştur.

```
db(area,of,city,AREA,CITY):-city(CITY,AREA1,_,_),str_real(AREA,AREA1).
db(population,of,city,POPUL,CITY):-city(CITY,_,POPUL1,_),
                                   str_real(POPUL,POPUL1).
db(city,border,city,CITY1,CITY2):-city(CITY2,_,_,LST),member(CITY1,LST)
db(neighbours,of,city,CITY1,CITY2):- city(CITY2,_,_,LST),
                                     member(CITY1,LST).
db(area,of,county,AREA,COUNTY):-county(COUNTY,AREA1,_,_),
                                str_real(AREA,AREA1).
db(length,of,river,LENGTH,RIVER):-river(RIVER,_,LENGTH1),
                                  str_real(LENGTH,LENGTH1).
db(height,of,mountain,HEIGHT,MOUNTAIN):-mountain(MOUNTAIN,_,H),
                                        str_int(HEIGHT,H).
```

Son erişim fonksiyonumuz **db2** ise db nin bir benzeridir. Fonksiyonun 6 parametresi vardır. İlk üç parametre iki ilişki sözcüğü ve bunlar arasındaki bir bağlantı sözcüğü son 3 parametre de bu ilişkiye uyan veritabanındaki bilgilerdir. Bu erişim fonksiyonu sadece iki tarih aralığı içeren sorgulamalarda kullanılmaktadır. Aşağıda db2 erişim fonksiyonunun değişik ilişkileri için tanımlanmış karşılıkları örnek olarak sunulmuştur.

```
db2(servetime,of,mayor,DATE1,DATE2,MAYOR):-
                                  mayor(MAYOR,DATESR,DATEST),
                                  str_int(DATE1,DATESR),
                                  str_int(DATE2,DATEST).
```

```
db2(servetime,of,governor,DATE1,DATE2,GOVERNOR):-
                                    governor(GOVERNOR,DATESR,DATEST),
                                str_int(DATE1,DATESR),str_int(DATE2,DATEST).
```

Böyle bir yöntemin amacı veritabanında mümkün olduğu kadar değişik olasılıkta ilişkilere ait ayrı veri grupları bulunduğu görünümünü verdirmektir. Aslında dikkatlice bakılırsa veritabanından esas sorgulamayı yapan fonksiyonlar, bu üç fonsiyondur.

3.5. İR 'nin Soru Cevap Mekanizması

Parser modülünün son argümanı kullanıcının sorusuna uygun olarak oluşturulan sorgulama ifadesi idi. Bu sorgulama ifadesi daha sonra computation modulüne parametre olarak gönderilir. Computation modülünün 2 parametresi vardır. İlk parametre parser'den gelen sorgu ifadesi, ikinci parametre de sorgulama sonunda oluşturulan cevap listesidir. Örneğin aşağıdaki soruyu sisteme verdiğimizi kabul edelim.

How long is the Riva ?

Ayrıştırma (parsing) işleminden sonra "q_eaec("length","of","river","Riva")"şeklinde bir sorgulama ifadesi oluşur. Bu , computation fonksiyonuna parametre olarak gönderilir.

computatiton(q_eaec("length","of","river","Riva"), ANSWER).

computation fonksiyonu işlevini tamamladığında ANSWER , ikinci arguman, aşağıdaki değeri alır.

ANSWER =computation(q_eaec("length","of","river","Riva"), 100)

Bazen sorgulama sonucunda birden fazla cevabın oluştuğu durumlara rastlanır. Bu cevapların hepsini veritabanından bir anda bulup çıkarmak için findall fonksiyonu kullanılır. Findall standart bir Prolog fonksiyonu olup gördüğü işlev ise bir sorgulama sonunda birden fazla cevap oluşursa bunları bir listede toplamaktır. Programın bu kısmı aşağıdaki gibi görünür.

findall(A,computation(Q,A),L). L burada cevapların toplandığı listedir.

Computation fonksiyonunu değişik sorgu cümlelerine göre ne karşılıklar aldığı programın çalışmasından alınan örneklerle açıklayalım.

Örnek :

What counties do you have ? soru cümlesini tekrar ele alalım. Daha önce gördüğümüz gibi bu cümle parserdan geçtikten sonra q_e("county") şeklinde bir sorgulama ifadesi oluşur. Bu sorgunun çözümünü verecek olan computation kuralı aşağıdaki biçimi alır.

computation(q_e(E),ANS) :- ent(E,ANS).

Bunun anlamı şayet veritabanında ent("county",ANS) şeklinde bir veri gurubu bulabilirsek computation işleminin yapılabileceğini göstermektedir.

Örnek :

How far is the Akgün Hotel from airport ?

Yukarıdaki soru cümlesi biraz daha komplike bir yapıya sahip olup, ilgili computation fonksiyonu bu kez
computation(q_eaec(E1,A,E2,C),ANS):-db(E1,A,E2,ANS,C). karşılığını alır. Burada db daha önce gördüğümüz gibi veritabanına ulaşmada kullanılan düşük seviyeli erişim fonksiyonlarından birisi idi. Bu sorgulama için db aşağıdaki parametreleri alır.

db(distance,of,hotel,ANS,C)

Bazen veritabanlarından düz bir sorgulama yerine belirli bir kritere uygun çözüm yada çözümlerin bulunması istenebilir. Örneğin,

What is the largest county in İstanbul ? soru cümlesinin normalize edilmiş sorgu ifadesi şöyledir.

q_max("county",q_eaec("county","in","city","İstanbul"). Buna ilişkin gelen computation fonksiyonuda

```
computation(q_max(ENT,TREE),ANS):-
        findall(X,computation(TREE,X),L),
        entitysize(ENT,ATTR),
        sel-max(ENT,ATTR,-1," ",ANS,L).
```

TREE değişkeni q_eaec sorgusuna karşılık gelmektedir. Findall fonksiyonu sadece bu sorgu ifadesini ardışıl olarak değerlendirerek bulunan çözümleri L listesinde tutar. L listesi İstanbul'da bulunan ilçelerin tamamını göstermektedir. Sonraki ifade ise nüfus niceliğini belirlemektedir.

(large ile population nicelikleri arasında bir ilişki kurulmuştur) Computation fonksiyonunun tanımındaki son cümle ise L listesindeki tüm ilçelerin nüfuslarına bakarak, nüfusu en büyük olan ilçeyi seçip çözüm olarak belirler.

TARTIŞMA ve SONUÇLAR :

Tabi dil arayüzlerinin tümü ya tanım yada soru tiplerine göre özeldir. Bir tabi dil arayüzü geliştirilmeden önce ne tip sorular sorulacağı üzerinde uzun araştırmalar yapılır. Sorular mümkün olduğu kadar sade ve açıklayıcı biçimlere getirilip daha sonra bu soruları algılayıp gerekli çözümleri üretebilecek olan ayrıştırıcılar (parser) yazılır. Hangi konuda yazılırsa yazılsınlar tüm tabi dil arayüz programlarının can alıcı modülleri ayrıştırıcı (parser) dediğimiz bölümleridir.

Parser yazarken takip edilen metodlar iki sınıfa ayrılmaktadır. Formal metodlar la yazılan ayrıştırıcılar cümlenin hem gramer hem de semantik analizini yapmakta olup böylece sorgulama için sisteme verilen cümlenin gramatik ve anlamsal olarak sözdiziminin de doğru yapılmasını öngörmektedirler. Bu yöntemin dezavantajlarından sözetmeden önce bugün geliştirilmiş ve geliştirilmekte olan tabi dil arayüz programlarında formal metodların çok az kullanıldığı gerçeğini vurgulamak gerekmektedir. Bu metodların sisteme getirdiği kısıtlardan en önemli iki tanesini şöyle belirtmek mümkündür.

- Bir dilin gramer kuralları kati sınırlarla çizilemediği için, programın veritabanına yeni tür bilgiler eklendiğinde, bu, yeni sorgu cümleleri ve dolayısıyla yeni gramer kurallarının program tarafından algılanmasını gerektirecektir. Bu da parser'ın kaynak kodunu yeniden düzenlemek ve eklemeler anlamına geleceğinden parser'ın çalışma performansı ve süratini olumsuz yönde etkileyecektir.
- Tabi dil arayüz programları insanların bilgisayardan faydalanmalarını sağlarken karmaşık bilgisayar komutlarının yerine günlük konuşma dilinde çok kullanılan basit cümleleri kullanmalarını amaçlıyordu. Ve biz biliyoruz ki insanlar günlük hayatlarında hiçbir zaman konuşma dillerinin gerektirdiği grameri doğru kullanmazlar. Dolayısıyla cümlelerin gramer ve semantik analizlerine sıkı bağlı tabi dil arayüz programları kullanıcılara sıkıcı geldiği için çok tercih edilmezler.

Parser geliştirirken kullanılan ikinci sınıf yöntemler informal yöntemler adını almaktadır. İnformal yöntemlerin çok tercih edilmelerinin sebebi adından tahmin edileceği üzere sorgulamalarda tabi dilin gramerine fazla bağımlı kalmamalarıdır. Bu yüzden informal yöntemlerle yazılan parser'lar gramer kurallarını dikkate almadıkları için daha hızlı çalışmakta ve yüksek performans göstermektedirler. Dolayısıyla kullanıcıların gramer yönünden doğru olmayan sorgu cümleleri sistem tarafından kabul görmektedir. Ayrıca bu yöntemle yazılan programlarda cümlede oluşan yapısal belirsizlikler de otomatik olarak bertaraf edilmektedir. Örneğin: "Give me the counties around airport having population less than 100000" şeklindeki bir sorgu cümlesinde "around airport" bir belirsizlik ifadesidir. Şayet "around" ve "airport" sözcüklerinin kullanımı önceden tahmin edilip ihmal edilecek kelimeler grubuna dahil edilirse, bu kelimeler daha filtrelenme işlemi esnasında eleneceğinden parser modülüne ulaşamaz ve sistem tarafında bir belirsizliğe yol açamaz.

İşte bu ikinci yöntemin ilk yönteme göre daha üstün tarafları ve tabi dil arayüzü programlarının amacına daha uygun düşeceğini görerek, İR programında kullandığım ayrıştırıcıyı da son bahsedilen yönteme göre yazdım. Bu yöntemi tercih etmemim birbaşka sebebide yukarıdaki avantajlarına ek olarak Prolog dilinin bu yöntemin kullanımına yapı olarak çok uygun düşmesi idi. Gerçektende Prolog bir yüksek seviyeli programlama dili olarak yapay zeka uygulamaları ve onun doğurduğu alt bilim dallarının vazgeçilmez programlama dilidir. Backtracking mekanizması sayesinde problemler şaşılacak derecede kısa program koduyla çözülebilmektedir.

İR programının parser'ının, ayrıştırma işlemi sonunda oluşturduğu sorgu cümlesi aslında kullanıcının sisteme girdiği sorgu ifadesinin normalize edilmiş bir halidir. Yani sorgu ifadesi filtreleme işleminden geçtikten sorgu cümlesinin niteliğine göre önceden oluşturulmuş sorgu kalıplarından uygun olanına yerleştiriliyor. Örneğin kullanıcının
"What is the population of county Avcılar ?" şeklindeki bir soruyu sisteme verdiğini kabul edelim. Gerekli filtreleme ve parsing (ayrıştırma) işlemlerinden geçtikten sonra bu soru

q_eaec("population","of","county","Avcılar")

biçimine getiriliyor. Dikkat edilirse bu ifadenin kullanıcının sisteme verdiği sorudan farkı "What","is","the","?" gibi sorgulamaya herhangi bir katkısı olmayan ve ihmal edilecek sözcük gruplarına ait sözcüklerin çıkarılmış olmasıdır (soru sözcükleri, belirteçler, yardımcı fiiller, noktalama işaretleri v.s.)

Programda sorgulama ifadeleri direkt olarak herhangi bir programlama tekniği kullanılarak oluşturulmamıştır. Ayrıştırıcı (parser) kendi içinde bir takım alt prosedürlere ayrılmıştır. Bu prosedürler en düşük seviyeden en yüksek seviyeye kadar sorgulama cümlesinin oluşturulmasına katkıda bulunurlar. Her bir prosedür sisteme girilen soruyu alıp gereksiz olan herhangi bir sözcüğü sorudan çıkarır veya normalize ederek bir üst seviyedeki prosedüre iletir. Soru en üst seviyedeki prosedürden çıktıktan sonra artık sistemin anlayacağı sorgu cümlesi oluşturulmuş olur. İşte sisteme verilen sorunun farklı kademelerden geçerek nihayi sorgu cümleciğinin oluşturulması yöntemine **"fark çiftlerinin ayrıştırılması-difference pair parsing"** adı verilir. Bu yöntemde cümlenin herhangi bir gramer veya semantik analizi yapılmaz. Fark çiftlerinin ayrıştırılması tekniğinde bir nicelik, ki bu ekseriya bir string yada stringler listesi olur, iki ayrı nicelik halinde tanımlanmaya çalışılır. Ve her yeni yapılan tanımlamada bu iki nicelikten ilki daha küçük iki niceliğe bölünür. Bu işlem niceliklerden birisi boş liste haline gelen kadar devam eder. Fark çiftlerinin ayrıştırılması tekniği kullanılarak yazılan parserlarda verimliliğin, nicelikleri bütün olarak ele alıp inceleyen tekniklere oranla 5 ila 50 oranında fazla olduğu tesbit edilmiştir. Bu yöntemin bir dezavantajı, kuralların deklaratif özelliklerini yitirmelerine yol açmasıdır. Öyleki ilk bakışta kurallara bakarak bunların arasında bir ilişkilendirme kurmak ve kuralın ne işe yaradığını kestirmek oldukça güç bir hal almaktadır.

İR programının ayrıştırıcısı da tamamıyla karmaşık çiftlerin ayrıştırılması metoduna göre geliştirildiği için parser'ın çalışma mantığını anlamak biraz zordur. Ancak bu sağlanırsa programın veritabanına yeni veri grupları eklemek ve kodunda modifikasyonlar yapmak suretiyle sistemin yeni soruları algılaması kolaylıkla sağlanabilir.

KAYNAKLAR :

1) Aho, Alfred V., and Ullman, Jeffrey D., 1972. The Theory of Parsing, Translation, and Compiling. Prentice Hall:Englewood Cliffs, NJ: 143-165
2) Appelt, Douglas E., 1985. Planning English Sentences. Cambridge University Press, Cambridge: 23-26
3) Briscoe, Edward J., and Bogureev, Branimir K., 1988. Computational Lexicography for Natural Language Processing. Longman/Wiley, London / New York: 65-80
4) Bolc, Leonard, ed., 1980. Natural Language Question Answering Systems. Hanser, Munich: 78-89
5) Borland International, 1988. Turbo Prolog User's Guide 2.0. Borland International Press, USA: 1-477
6) Clark, H.H., 1975. Theoretical Issues in Natural Language Processing.Cambridge:
7) Dahlgreen, Kathleen., 1988. Naive Semantics for Natural Language Understanding. Kluwer Academic Publishers, USA: 25-45
8) Gal, A., Lapalme, G., Saint-Dizier, P., and Somers, H., 1989. Prolog for Natural Language Processing. John Wiley & Sons, USA: 1-230
9) Gazdar, Gerald and Mellish, Chris., 1989. Natural Language Processing in Lisp - An Introduction to Computational Linguistics. Addison Wesley, USA: 1-30
10) Grishman, Ralph., 1989. Computational Linguistics, An Introduction. Cambridge University Press, Great Britain : 223-225
11) K. Krulee, G., 1989. Computer Processing of Natural Language. Prentice Hall , USA: 213-230
12) R.Dowty, David, and Karttunen, Lauri and Zwicky, M., 1988. Natural Language Parsing, psychological, computational and theoritical perspectives. Cambridge University Press, USA: 230-238
13) Reyle, U., and C.Rohrer, 1988. Natural Language Parsing and Linguistic Theories. E.Reidel Publishing Company, The Netherlands: 212-234
14) Sager, Naomi., 1981. Natural Language Information Processing, A Computer Grammer of English and Its Applications. Addison Wesley, USA: 234-265
15) Vlach, F., 1981. The Semantics of the Progressive. In Syntax an Semantics 14. Academic Press, New York: 124-130

ÖZGEÇMİŞ :

Doğum tarihi : 30 Aralık 1969

Doğum yeri : İstanbul

Bitirdiği okullar : 1984 - 1987 İstanbul 50. Yıl Avcılar İNSA Lisesi

1987 - 1992 Y.T.Ü. Bilgisayar Bilimleri ve Mühendisliği Bölümü

1992 - Y.T.Ü. Fen Bilimleri Enstitüsü - Bilgisayar Bilimleri Mühendisliği Ana Bilim Dalı - Y. Lisans