Uzman Sistemlerin Programlanması

C , FORTRAN gibi geleneksel programlama dilleri prosedürsel olarak bilginin işlemsel açıdan değişikliğe uğratılabilmesi için geliştirilmiş ve optimize edilmiş dillerdir. (Sayılar veya diziler gibi ..) Bununla birlikte genelde insanlar kompleks problemleri soyut ve sembolik yöntemlerle ve geleneksel diller kullanılarak , tam olarak ifade edilemeyecek yollardan çözerler. Soyut bilgi bu dillerde modellenebilmesine rağmen işlemsel programlama dizileri ile kullanılabilecek biçime bilgiyi dönüştürmek için hayli fazla bir programlama eforu kaydetmek gerekir. Yapay zeka alanında yapılan araştırmaların sonuçlarından biri de soyutlandırmanın yüksek seviyelerinde bilginin modellenmesini sağlayan tekniklerin geliştirilmesi olmuştur. Kullanılan araçlarla veya dillerle iyice somutlaştırılan bu teknikler programların insan lojiğine daha benzer olarak yaratılmasını ve daha kolay geliştirilip ilerletilmesini sağlarlar. İşte , iyi tanımlanmış problem alanlarındaki insan tecrübesini emüle eden programlara *Uzman Sistemler* denir. Uzman sistem araçlarından kullanışlı olanlarının ortaya çıkması ile de uzman sistemlerin oluşturulmasında harcanan efor ve maliyeti azaltmıştır.

Temel Diller

Kural Tabanlı Diller

Kural-Tabanlı programlama dilleri uzman sistemleri geliştirme aşamasında yoğun olarak kullanılan tekniklerden biridir. Kurallar “if” ve “then” olmak üzere iki deyim bütününden oluşur. Bu programlama yönteminde kurallar verilen bir durum için yapılması gereken işlerin kümesi olarak tanımlanan , keşif veya buluşları temsil etmek için kullanılır. Bunlar tamamen insan deneyimine dayanır. Kuralın “if” kısımı kuralın uygulanabilir olmasını sağlayan olayı yada veriyi tanımlayan deyim dizileridir. Olayları kalıplara ilişkilendirme işlemine deyim ilişkilendirme denir. Uzman sistem araçları , sonuç çıkarama mekanizması içerirler. Bu mekanizma otomatik olarak olaylarla kalıpları ilişkilendirip buna bağlı olarak hangi kuralların uygulanabilir olduğunu belirlerler. Kuralın “then” kısmı ise kural uygulanabilir olduğunda yürütülecek olan işlemleri tanımlayan kısımdır. Uygulanabilir kuralların bu işlemleri sonuç çıkarım mekanizması çalışmaya başlatıldığında yürütülür. Sonuç çıkarım mekanizması önce bir kural seçer ve sonra seçilen bu kuralın işlemleri yürütülür. Sonra başka bir kural seçilir ve bu kuralın işlemleri yürütülür. Bu işlem uygulanabilir bir kural kalmayıncaya kadar sürdürülür.

Nesne Yönelimli Diller

Nesne Yönelimli Dillerde veriler, prosedürler ve ilişkilerini ifade etmenin başka bir yoludur. Tasarımda tanımlayıcı ve prosedürsel özellikleri inceler. Veri bağımlılığı problemini ortadan kaldırır. Yapa zekada nesnelere çerçeve (frame) denir.

Çerçeveler özlü ve yapısal bir bilgi sunumu sağlar. Çerçeve içindeki bilgi oluk (slot) denilen parçalara bölünür. Bir oluk , tanımlayıcı veya prosedürsel bilgiyi tanımlar. Bir çerçeve özel bir nesne, olay, yer, durum veya başka bir element hakkında bilgi taşır. Çerçeveler iyi bilinen bilgileri ifade etmede kullanılırlar.

Bilgiler karakteristikleri ve özellikleri ile beraber oluklarda saklanırlar. Bir çerçeve oluklar ve yüzlerden (facet) oluşur. Yüz , oluk içindeki bir kısım bilgi veya oluğun özelliğini tanımlayan prosedürlerden ibarettir.

Prosedürsel Diller

Prosedürsel diller veri temsil için esnek ve güçlü teknikler kullanırlar ve veri soyutlamasına izin verirler. Uzman sistem dilleri ise bilginin temsili için esnek ve güçlü teknikler kullanırlar ve bilgi soyutlamasına izin verirler.

FORTRAN, COBOL, PL/I, PASCAL , C gibi bütün geleneksel diller sayısal veri işleme üzerinde algoritma geliştirmek için tasarlanmışlardır. Şimdi ise uzman sistemlerin çoğu veri tabanlarına erişimi içerecek ve geleneksel prosedürsel kodlamayı kullanabilecek şekilde tasarlanmaktadır. Bu nedenle "prerecording tool" ların geleneksel dillerde yazılması fikri vurgulanmıştır. PASCAL 'ın uzman sistemlerdeki yapılar gibi özellikleri ve bazı tasarım özellikleri mevcuttur.

Bazı uzman sistem geliştirme araçları geleneksel dillerle yazılırlar. Örneğin TIMM FORTRAN 'da, INSIGHT2 PASCAL 'da , EXSYS ve CLIPS C ' de yazılmışlardır. Bunun nedeni ise bu dillerin donanım için uygun olması ve uzman sistemlerin kişisel bilgisayarlarda çalışıyor olmasıdır. Ayrıca bu şekilde yazılan uzman sistemler daha hızlı çalışırlar ve veri tabanı erişimi yazılımı bu şekilde daha kolay olur.

Bu yüzden uzman sistemler önce LISP, PROLOG gibi yapay zeka dillerinde yazılır daha sonra program kodu PASCAL, C, FORTRAN gibi dillerin koduna dönüştürülür. FORTRAN yapay zeka veri tiplerinin ancak küçük bir aralığında verimli olarak çalışabilirler.

Yapay zeka programlamada devamlı olarak kurallar oluşur veya parçalanır. Çok büyük miktarlarda sonuç bilgileri oluşur. Bu durumda bilgisayar belleği dolar. Yapay zeka dillerinde bellek otomatik olarak bir proses tarafından temizlenir. Ancak PASCAL 'da bunu yapabilmek için ayrıca kod yazmak gerekir.

Yüksel Seviyeli Yapay Zeka Dilleri

LISP ile yeni fonksiyon oluşturmanın kolaylığı uzman sistem tasarımında çok önemlidir. Bu durum LISP 'in yüksek seviyeli dilleri için bir taban oluşturmasını sağlamıştır. XLMS LISP 'in bir uzantısıdır. Bu dille açık, özlü ifadeler elde edilebilmektedir. LOOPS da başka bir yüksek seviyeli dildir.

Genel Amaçlı Bilgi Mühendisliği Dilleri

Bu tip diller bilgi mühendisliği için geliştirilmişlerdir. Bu tip diller daha esnektir. Fakat bu diller bazı giriş/çıkış kolaylıklarını yok edebilirler. Daha geniş amaçlı işler için kullanılırlar. Programlama ortamları kabul sistemleri kadar geniş değildir.

Kabuk programlarında olduğu gibi belli bir uygulama ile sınırlı değillerdir. Çok sayıda kontrol yapılarına sahiptirler. Bu sayede uygulanmaları kabuk sistemlerinden daha zor olduğu halde kullanıldıkları alanlar daha geniştir. HEARSAY-3,ROSIE,OPS5 VE RLL gibi diller bu gruptandır.

Kabuk Diller

Bir uzman sistem 6 parçadan oluşur. Bunlar bilgi kazanç alt sistemleri, sonuç çıkarım motoru, açıklama yeteneği, arabirim alt sistemleri ve bilgi temelli yönetim sistemleridir. İlk beş alt sistem bir uzman sistem kabuğunu oluşturur. İlk beş programın her uygulamada programlanmasına gerek yoktur. Kabuk bir kere oluşturulduktan sonra bir çok uygulama için kullanılabilir. Böylece uzman sistemler daha hızlı oluşturulabilir ve ihtiyaç duyulan programlama tekniği azalmış olur. Kabuk kavramı özellikle kural tabanlı sistemlerde

kullanılır. EXSYS , NEXPERT , IMPACT , TIMM , JESS kural tabanlı kabuklara örnek olarak verilebilir.

Uzman Sistemlerde Kullanılan Yapay Zeka Dilleri

Yapay zeka , nesnelerini bu tip dillerle ifade ermek verimli bir yoldur. En önemli iki tanesi PROLOG ve LISP 'tir. Bu dillerle programlama ve hata ayıklama işlemleri hızlı bir şekilde yapılabilir. Bundan başka CLIPS , OPS5 , ART , ART-IM , ECLIPSE ve ILOG bu dillere örnek teşkil eder.

PROLOG (Programming Logic)

Lojik üzerine temellendirilmiştir. Temeli "first order predicate calculus" a dayanır. Bu da PROLOG 'u anlaşılır ve düzgün sentakslı bir hale getirir. Kural tabanlı bir dildir, ve bir kuralın doğru olması için bilgisayarın bütün kuralların buna uyup uymadığını kontrol etmesi gerekir. Bir işin nasıl yapılması gerektiği değil, yapılması için neye ihtiyaç duyulduğu ve neyin doğru olduğu bilgisi çıkarılabilir.

PROLOG programlamada;
Nesneler hakkındaki olaylar (fact) ve aralarındaki ilişkiler tanımlanır.
Nesneler ve ilişkileri hakkındaki kurallar tanımlanır.
Nesneler ve ilişkileri hakkındaki sorular sorulur.

PROLOG ilişkisel ve tanımlayıcı olarak tanımlanabilir. Bu iki kavram da bir prolog olaylar (fact) grubunu ifade eder. Birinin başarıya ulaşmak istediği şeyi tanımlaması bakımından tanımlayıcı olarak düşünülür.

Örneğin "sort([5,3,7,1],Answer)!" prosedürünün sonucunda "Answer=[2,3,5,7]" döndürülür.

Bir prolog programı bir dizi ifadeden oluşur. Bunlara olaylar veya kurallar denir.

Bir ifadenin en genel şekli HEAD:-BODY., şeklindedir. HEAD tek başına bir yapıdır. BODY ise 0 veya daha fazla yapıdan oluşabilir. Bu yapılara "subgoal" denir ve virgüllerle birbirinden ayrılırlar.

Bir yapının en genel şekli ise FUNCTOR(TERM1,...TERMn) şeklindedir. TERM bir sabit, bir değişken veya başka bir yapı olabilir.

FUNCTOR' lar belirleyici semboller, operatörler veya ilişki isimleri olabilir. Bir belirleyici sembol true veya false değerini alabilir. <=(2,4)'ün değeri true' dur. Bazı operatörler PROLOG da sayı aritmetiğinde kullanılırlar. Örneğin X+Y+Z +(+(X,Y),Z) olarak, X+Y*Z de +(+(X,Y),Z) olarak yazılabilir.

PROLOG da sabitler küçük harflerle yazılır ve "underscore" karakterinden başka bir karakter içeremezler. Aksi takdirde iki tek tırnak arasında yazılırlar

Sabitler bir atomda olabilirler. Atom işaretlerin birleşmesiyle oluşur ve özel anlamları vardır .Örneğin ":-" if olarak,"?-"ise bir sorgu ifadesi olarak kullanılırlar. İşaretler, +,-,*,/,\,^,<,>,~,:,-,?,@,#,$,& karakterlerinden oluşur. Bir ilişkinin ismi bir atom da olabilir.

Örneğin <(2,4) teki '<' veya has(tom,beer) daki 'has' bir ilişki ismidir.

"Variable"ler ise büyük harfle veya "underscore" karakteri ile başlar.

?-has(tom,_) , has ilişkisini sağlayan bütün atomları dödürür.

Prolagda liste yapılarının kullanımı rahattır.[et,süt,patates] bir dizidir. Uzunluğu belli olmayan diziler de kullanılabilir. Ör:[et,patates[X]]

is_a(owns(tommy,tiger),yellow,cat) bir veri tabanında olan fact ise

?-is_a(owns(tommy,X),yellow,cat) bir sorgudur ve X yerine tiger üretir.
% işareti prolog da yorumlar için kullanılır.

hayvan(X):-köpek(X) X bir hayvandır,eğer X bir köpek ise
dog(Lassie) Lşasie bir köpektir.
?-animal(Lassie) Lassie bir köpektir.

:-P P kanıtlanacak olan amaç
P. P bir açıklama veya bir facttir
P:-Q,R,S Q,R,S P'yi gerektirir.

CLIPS

CLIPS Dilinin Gelişimi

C Language Integrated Production System adlı CLIPS NASA 'nın Johnson Space Center adlı merkezinde 1984 yılında ortaya atıldı. Bu sürede NASA ' da Arifical Intelligence Section da uzman sistemler için günün donanım ve yazılım tekniğini kullanarak bir çok prototip uygulama geliştirmişti. Uzman sistemlerin geniş demostrasyonları olmasına rağmen bunların çok azı düzgün ve kurallı bir kullanıma geçirilebilmişti. Bu hata NASA 'da neredeyse tüm uzman sistem araçlarında geliştirilmesinde LISP dilinin temel olarak kullanılmasına bağlanılabilir. NASA 'daki uzman sistem yazılım araçlarının LISP temelli olması 3 problemi ortaya çıkarttı. LISP 'ın çok çeşitli bilgisayar sistemlerinin bulunduğu bir ortamda az sayıda sistemde kullanılabilmesi , LISP araçlarının ve yazılımlarının fiyatlarının yüksek olması ve yaygın olan diğer dillerle LISP 'ın bütünlüğünün az olması.

Bunun üzerine NASA 'daki Artificial Intelligence Section C dili gibi geleneksel dillerin uzman sistemler için kullanılmasının birçok problemi gidereceğini karar vererek geleneksel dilleri kullanımını yaygınlaştırmaya başladılar. Bunun üzerine uzman sistemler için yazılım geliştiren yazılımcılar programlarının C diline çevirmeye başladılar. Fakat bu da yüksek maliyet ve uzun zaman getiriyordu. Bunun üzerinde Artificial Intelligence Section kendi C temelli uzman sistem aracını geliştirdi. CLIPS dilinin ilk prototip 1985 yılında ortaya atıldı. Bundan sonra CLIPS üzerinde çalışmalar sürdürülerek en son versiyon 6.1 ile 1998 yılında C++ uyumluluğu ve fonksiyonları da dile eklendi.

CLIPS düşük maliyeti , genişletilebilir olması , yetenekleri , taşınabilirliği ile geniş bir kullanıcı alanına kendisini kabul ettirmiştir. Bir çok askeri , endüstriyel ve akademik çalışmada kullanılmıştır. CLIPS 5000 'den fazla kullanıcı tarafından kullanılmaktadır. CLIPS şu anda programın yazılımcı sahipleri tarafından herkese açık bir yazılım olarak topluma sunulmaktadır.

CLIPS Dilinin Özellikleri

CLIPS kural yada nesne tabanlı uzman sistemin tasarımı için tam bir ortam sağlayan ve verimli olarak geliştirilebilecek ve dağıtılabilecek bir uzman sistem aracıdır. CLIPS NASA , Askeri birimler , üniversiteler , devlet kurumları gibi dünyadaki bir çok toplulukta sayısız kullanıcılar tarafından kullanılmaktadır.
CLIPS dilinin özelliklerine şöyle bir bakarsak ;

- **Bilgi Gösterilimi :** Kural tabanlı , Nesne yönelimli ve Prosedürsel olmak üzere 3 ayrı programlama yönteminde de destek vererek geniş çeşitlilikte olabilen bilgiyi kontrol altına alan uyumlu bir araç sağlar. Bilindiği gibi kural tabanlı diller , verilen durum için gerçekleşecek olan işlemler kümesi olarak tanımlanan temel kurallar ile bilginin gösterilmesine izin verirler . Nesne yönelimli diller kompleks sistemlerin modüler bileşenlerle modellenmesine izin verirler. CLIPS tarafından sağlanan prosedürsel programlama da C , PASCAL , LISP gibi diğer dillerde bulunabilen yeteneklerin benzeridir.
- **Taşınabilirlik :** CLIPS taşınabilirlik ve hız kazanılabilmesi için C dilinde yazılmış ve hiçbir kod değişikliği yapılmadan bir çok farklı bilgisayar sistemlerine yüklenmiştir. CLIPS dilinin test edildiği bilgisayar sistemleri DOS ve Windows 95 işletim sistemi ile çalışan IBM PC ile MacOS ve Mach ile çalışan Macintosh sistemidir. CLIPS ANSI C derleyicisi olan herhangi bir sisteme taşınabilir. CLIPS kullanıcın özel ihtiyaçları nedeniyle eklenebilecek veya değiştirilebilecek tüm kaynak kodu ile gelir.
- **Bütünlük ve Genişletilebilirlik :** CLIPS kodları alt rutin adı verilen prosedür kodların içine gömülebilir ve C , FORTRAN , ADA dilleriyle bütünlük içindedir. Herhangi bir kullanıcı tarafından rahatlıkla birkaç iyi tanımlanmış protokol kullanılarak genişletilebilir.
- **Interaktif Gelişim :** CLIPS 'in standart versiyonu , hata ayıklamaya yarayan araçları , yardım dosyaları ve bütünleşik editörü içeren interaktif ve tekst bazlı geliştirme ortamı sağlar. Aşağı inip çıkabilen menüler , ek editörler , Machintosh Windows 95 ve X Windows tarafından geliştirilen çoklu pencereler desteğiyle ortamdaki ara yüzleri için değişik ek özellikler de sağlanır.
- **Doğrulama ve Onaylama :** Kural deyimlerinin semantik analizinin yapılması , fonksiyon argümanlarının ve boş değerlerin statik ve dinamik kısıtlarının kontrol edilmesi, bilgi tabanının modüler tasarımı ve bölümlenmesinin sağlanması gibi destekleri içeren birçok doğrulama ve onaylama özelliği içerir.
- **Tam Dokümantasyon :** CLIPS kullanıcı kılavuzunu ve referans el kitabını içeren geniş bir dokümantasyon ile gelir.

OPS5

OPS5 (Official Production System, Version 5), Charles Forgy (Carnegie-Mellon University) liderliğinde ki bir grup araştırmacı tarafından geliştirilen ve uzman sistemlerde yaygın olarak kullanılan bir üretim dilidir. Ops5 üç ana bölüm içerir: 1- kurallar kümesi 2– bu kurallarla çalışan gerçekler kümesi 3- bu kuralların doğru uygulamasını belirleyen çıkarım makinesı. OPS5 gibi üretim dillerinin diğer programlama dillerinden en büyük farkı sıralı olarak çalıştırılan bir koda sahip değillerdir. Çalışma belleği tarafından karşılanan koşullara göre kurallar ateşlenir.
Her kural iki bölümden oluşur : LHS(left hand side) ve RHS(right hand side). Eğer LHS'ta ki koşul karşılanırsa RHS'taki olay gerçeklenir. Bir kural çalıştırıldıktan sonra diğer kurallar çalışma belleğinde yeniden değerlendirilir.Bu döngü çalışma belleğinde o andaki veri ile eşlenecek bir LHS'a sahip kural kalmayana kadar devam eder.
OPS5 basit veritipleri ve bileşik veritiplerine sahiptir. Basit veritipleri integers, floating points ve strings'tir. Bileşik veritipleri ise kullanıcıların tanımladığı veritipleridir. OPS5, verileri çalışma belleğinde (working memory) saklar. If-then kuralları ise üretim belleğinde (production memory) saklanır. OPS5'teki kurallar tamamen birbirlerinden bağımsızdır ve

üretim belleğinde herhangi bir sırada yeralabilir. Gerçekler ve kurallar OPS5'te bilgi temsili (knowledge representation) için kullanılır.
OPS5 hızlı, verimli ve kural tabanlı bir programlamadır (LISP kullanılarak yazıldığı için).

Ops5'in terminolojisi diğer üretim sistemleri için kullanılan Poplog terminolojisinden biraz farklıdır:

Poplog terminology	OPS5 terminology
Database	Working memory
Fact	Working memory element
Rulebase	Production memory
Rule	Production
Conditions	LHS
Actions	RHS
Set of applicable rules	Conflict set
Non-repeating	Refractoriness

Working Memory
Ops5'in çalışma belleği form elemanları topluluğundan oluşur:
(<type> <field1> <field2> ...)
Her bir alan bir isme ve içeriğe sahip olabilir:
(<type> ^<field1-name> <field1-contents>
^<field2-name> <field2-contents> ...)
'^' sembolü bir alan ismi olduğunu belirtir. Alan içerikleri ise bir LISP atomu olabilir ancak liste olamaz.

OPS5 Sentaksı

```
1   (p start-stack
2      (firstblock)
3      {(block ^size <s>)     <b>}
4      - (block ^size {> <s>})
5      -->
6         (remove 1)
7         (modify <b> ^instack yes)
8         (make stacking))
```

Satır 1: (p start-stack
'p' üretim kuralının başladığını belirtir. Devamında ise kuralın ismi yer alır. Burada kuralın adı start-stack.
Satır 2: (firstblock)
Burası kuralın ilk koşulu.Koşullar bir gerçek ile eşlenen modellerdir. Bundan dolayı ilk koşul 'firstblock' tipinde bir gerçek tanımlar.
Satır 3: {(block ^size <s>) <b>}
Bu da kuralın ikinci koşuludur. İki ilginç karakteristiğe sahiptir. İlki, model süslü parantez ve <b> ile guplanır. <b> model ile eşlenen WM element'in adıdır. İkincisi ise model bir field name (^size) ve değişken (<s>) içerir. Modelin ismi ^'ile verilir. Değeri ise <> ile verilir.
Satır 4: - (block ^size {> <s>})
Kuralın üçüncü koşuludur. '-' tümleyen anlamında kullanılmıştır. Tüm koşul şu şekilde okunabilir: " Size'ı <s>'ten daha büyük olan blok yok."

Satır 5: -->
Kuralın koşullarını kuralın gövdesinden ayırır. Kuralın gövdesinde olaylar(actions) yer alır.
Satır 6: (remove 1)
Kuralın ilk olayı. Kuralın ilk koşulu ile eşlenen WM element'i çıkarır.
Satır 7: (modify <b> ^instack yes)
Kuralın ikinci olayıdır. <b> ile gösterilen WM element'i ^instack fieldının değeri olan yes ile değiştirir.
Satır 8: (make stacking))
Bu kuralın son olayıdır. 'satcking' tipinde Fieldsız yeni bir WM element yaratır.

Bir OPS5 kuralının her bir koşulu bir WM element ile eşlenebilecek bir modeldir. Bir model "type" ile başlayan bir listedir. Modelin geri kalanı sıralı field değişkenlerinden yada değerlerinden oluşur.

OPS5 derleyicisini çağırmak için komut satırından aşağıdaki komut çağrılır:

OPS5 filename
Başarıli bir derlemeden sonra komu satırından **RUN FILENAME** çalıştırılır**.** Filename sizin OPS5 dosyasının adıdır. OPS5'ten çıkmak için **EXIT** yada Ctrl<Z> kullanılır.
HI.OPS adlı bir OPS5 programının derlenme ve çalıştırılma aşamaları aşağıdaki gibidir:
$ **ops5 hi**
%OPSCOMP-I-ENDCOMPILE, End of compilation 4-Dec-1991 16:32:14.89
%OPSCOMP-I-LINESREAD, Compiled 22 lines
%OPSCOMP-I-NOERRORS, No errors detected
%OPSCOMP-I-TIMEUSED, Time used was 0.78 seconds
creating executable image..
$ **run hi**

OPS5>**run**
Lütfen adınızı ve soyadınızı giriniz: **erhan cetintas**

MERHABA ERHAN CETINTAS

%OPSRT-I-HALTED, HALT -- right-hand-side action
OPS5>**exit**
$

LISP

LISP (List Processing language) Stanford'tan John McCarthy tarafından MIT 'de ki çalışmaları sırasında geliştirilmiş bir çok yönüyle kolay ve fonksiyonel bir dildir. LISP aynı zamanda yorumlanabilen bir dildir. LISP uygulamaları etkileşimli, esnek ve yinelemelidir ve programları derlemeye yada link etmeye gerek yoktur. LISP programları liste yapısı ile temsil edilir ve temel işlemleri listeler üzerindeki işlemlerdir. LISP, programın çalışması sırasında otomatik bellek yönetimi ve veri alanı ayırabilme imkanı sağlayan bir dildir. Bu özelliği sayesinde boşlukları ayarlama işlemini çok verimli olarak gerçekleştirir ve programcıyı karmaşık ve esnek programlar yaratmada özgür bırakır. LISP çok az sayıda veri yada veri tipi tanımını gerektirir ve hatasız bir LISP programı matematiksel olarak çok kolay sağlanır.

Bir LISP programı fonksiyon tanımları kümesini içerir ve bir fonksiyondan başka bir fonksiyonun çağrılması çok kolaydır. Bir LISP programcısı kolayca ve sık sık değişiklikler yapabilir ve her seferinde tüm programı derlemek zorunda kalmadan bunları deneyebilir. LISP 'te yeni fonksiyonları yaratılmasının çok kolay olması uzman sistem dizaynında çok önemlidir. LISP programları kendi kendinin değiştirebilme yeteneğine sahiptir. LISP aynı zamanda pahalı donanıma sahip makinalar gerektirir.

Bir çok orijinal uzman sistem kabuğu LISP ile yazılır. Çünkü :

- Temel veri yapısı listedir.
- Programlar ayrıca liste yapıları ile temsil edilebilir.
- Temel işlemleri listeler üzerindeki işlemlerdir.

Ağırlıklı olarak yapay zeka uygulamalarında kullanılır. Ancak uzman sistem uygulamalarında da kullanılabilen bir dildir. Bir LISP programı basitçe sembolik ifadelerin sıralanması olarak ifade edilebilir.

Sembolik ifadelerin iki sınıfı vardır : *atoms* ve *lists* . Atoms kendi içerisinde iki sınıfa ayrılır : *numbers* ve *symbols*.

Numbers tahmin edeceğiniz gibi tamsayı, ondalık sayı ve rasyonel sayılardır (3, 3.5, ¾). Symbols ise karakter sekanslarıdır (ball,+,A vb.). List ise bağımsız varlıklar sekansıdır.Bir list başka bir list'i içerebilir. Eğer *a and b* bir sembolik ifade ise a ve b'yi içeren listeyi *(a b)* şeklinde gösteririz. LISP yorumlayıcısını kullanırken komut satırından girilen sembolik ifadeler yorumlayıcı tarafından yazdığımız gibi değerlendirilecektir.

```
>3
3
>(+ 2 5)
   7
```

Yorumlayıcı aynı zamanda hesap makinesi olarak ta kullanabiliriz.
Yorumlayıcı sembolik ifadeleri şu şekilde değerlendirir:

- Eğer sembolik ifade bir sayı ise sembolik ifadenin değeri sayının kendisidir.
- Eğer sembolik ifade bir liste ise bu listenin hesaplanan değeri arguman olarak kullanılır.

```
>(* 1/4 (+ 1/2 1/4))
3/16
```

Burada ¼ sembolik ifade; (+ ½ ¼) ise listedir.
Değer atama işlemleri *setf* fonksiyonu ile yapılır:

```
>(setf A 3)
  3
```

Artık komut satırından A ifadesi yazıldığında 3 değeri döndürülür.

```
>A
 3
```

Quote bir argumanı alır ve değerlendirmeden geri döndürür :

```
>(quote (+ 3 5))
(+ 3 5)
```

Bu fonksiyon genelde şu şekilde kullanılır:

```
>'(+ 3 5)
(+ 3 5)
```

Bir liste kolayca ayrılabilir yada birleştirilebilir. Listenin ilk argumanını almak için *first*, ilk argumandan sonrakiler için *rest* fonksiyonları kullanılır.

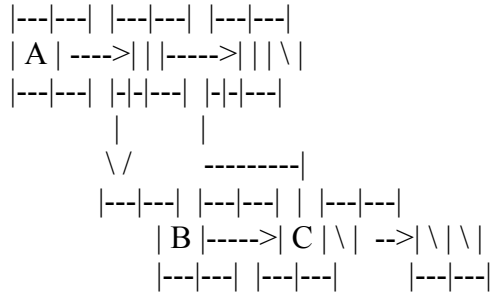
```
>(first '(a b c))
A
```

```
>(rest '(a b c))
(B C)
```

Listeleri birleştirmek için ise *cons*, *append* ve *list* fonksiyonları kullanılır. Bunlara ilişkin örnekler şunlardır:

```
>(cons '(a b) '(c d))
((A B) C D)
>(append '(a b) '(c d))
(A B C D)
>(list '(a b) '(c d))
((A B) (C D))
```

Bir listeyi ifade etmek için diyagramlar da kullanılabilir. Örneğin (a (b c) ()) listesi diyagramla şu şekilde ifade edilir:



Şimdi bir örnek yapalım. Rastgele dilbilgisel İngilizce cümleler üreten bir program yazalım.

```
The tree sleeps.
The dog eats the red dog.
Italy sleeps.
Bill Clinton eats.
A ball examines Bill Clinton.
A ball eats.
Stanford University eats a terrible pretty dog.
A ball hates the dog.
A pretty tree eats Stanford University.
Stanford University hits a terrible tree.
```

Yukarıdaki cümleler şu şekilde üretilebilir:

```
'((Subject (Noun Intransitive-Verb) (Noun Transitive-Verb Noun))
  (Noun (Proper-Noun) (Article Simple-Noun))
  (Proper-Noun ("Bill Clinton") ("Stanford University")
                  ("Italy"))
  (Simple-Noun (Adjective Simple-Noun) ("dog") ("tree") ("ball"))
  (Adjective ("red") ("pretty") ("terrible"))
  (Article ("a") ("the"))
  (Intransitive-Verb ("eats") ("sleeps"))
  (Transitive-Verb ("hates") ("hits") ("examines") ("eats")))
```

JESS (Java Expert System Shell)

JESS bir programlama kütüphanesidir. Bu kütüphane kendisi de Java dilinde yazılmıştır. Bu kütüphane diğer bir dil için tercüman olarak davranır. CLIPS uzman sistem komut kabuğuna da benzer. JESS iki yoldan kullanılabilir. İlk olarak , veriye kurallar uygulayan özel bir tür program şeklindeki bir kural mekanizması olarak düşünülebilir. Kural tabanlı bir program yüzlerce yada binlerce kurala sahiptir ve JESS de bilgi tabanının biçimine bağlı olarak bu kuralları veriye uygular.

Genelde kurallar insan deneyimlerinin keşifsel bilgisinden oluşmakta olup bilgi tabanı da gelişen durumun o anki halini temsil eder. Bu durumda tüm bu kurallar ve bilgi tabanı uzman sistemi meydana getirir. Uzman sistemler bir çok alanda geniş olarak kullanılırlar. İkinci olarak da JESS dili genel amaçlı programlama dili olarak kullanılabilmekle beraber tüm java sınıflarına ve kütüphanelerine de direkt erişim sağlar. Bunun sonucunda JESS sık olarak dinamik script programlama ve hızlı uygulamaların geliştirilmesinde de etkin olarak kullanılabilir. Java kodu genelde çalıştırılmadan önce derleneme ihtiyacı duyarken bir JESS kodu satırı yazılır yazılmaz hızlıca yürütülebilir. Bu programcıya Java API' leri ile interaktif olarak çalışma ve büyük programları adım adım oluşturabilme imkanı sağlar. JESS geniş alanlı uygulamalarda kullanışlıdır. JESS Internet uygulamalarında kullanılmak üzere applet desteği ile tam donatılmamıştır. JESS 'in boyutu büyük olması çok hızlı LAN 'lar dışında Internet bazlı applet olarak kullanımını engeller. Bunun dışında browser ortamında bazı JESS özellikleri de kaybolur. Genelde JESS kullanılarak WEB tabanlı uygulama yapılmak isteniyorsa JESS sunucu tarafında çalışan Java Servlet olarak kullanılmalıdır.

JESS dilinin kural mekanizması çok bilinen ve kuralları bilgi tabanıyla ilişkilendiren RETE algoritmasıyla oluşturulmuştur. JESS ,büyük problemler ve algoritma kalitesi söz konusu olduğunda birçok C dilinde yazılmış uzman sistem komut kabuklarından daha hızlıdır.

Rete algoritmasından dolayı JESS 'in kullandığı bellek önemsiz boyutta değildir. Ama JESS içinde performanstan biraz fedakarlık ederek bellek kullanımını düzetebilecek bazı komutlar vardır. JESS komut satırı uygulamalarında da kullanılabilir. GUI uygulamalarında , servlet olarak ve applet olarak kullanılabilir.